
Transportation Network Analysis

Volume I: Static and Dynamic Traffic Assignment

Stephen D. Boyles
Civil, Architectural, and Environmental Engineering
The University of Texas at Austin

Nicholas E. Lownes
Civil and Environmental Engineering
University of Connecticut

Avinash Unnikrishnan
Civil and Environmental Engineering
Portland State University

Version 0.91 (Public beta)
Updated August 22, 2023

Preface

This book is the product of more than ten years of teaching transportation network analysis, at the The University of Texas at Austin, the University of Wyoming, the University of Connecticut, Portland State University, and West Virginia University. The project began during a sabbatical visit by the second author to The University of Texas at Austin, and has continued since. This version is being released as a *public beta*, with a final first edition hopefully to be released within a year. In particular, we aim to improve the quality of the figures, add some additional examples and exercises, and add historical and bibliographic notes to each chapter. We are also developing a companion set of lecture slides and assignments. A second volume, covering transit, freight, and logistics, is also under preparation.

Any help you can offer to improve this text would be greatly appreciated, whether spotting typos, math or logic errors, inconsistent terminology, or any other suggestions about how the content can be better explained, better organized, or better presented.

We gratefully acknowledge the support of the National Science Foundation under Grants 1069141/1157294, 1254921, 1562109/1562291, 1636154, 1739964, and 1826320. Travis Waller (University of New South Wales), Chris Tampère (Katholieke Universiteit Leuven), and Xuegang Ban (University of Washington) hosted visits by the first author to their respective institutions, and provided wonderful work environments where much of this writing was done.

Target Audience

This book is primarily intended for first-year graduate students, but is also written with other potential audiences in mind. The content should be fully accessible to highly-prepared undergraduate students, and certain specialized topics would be appropriate for advanced graduate students as well. The text covers a large number of topics, likely more than would be covered in one or two semesters, and would also be useful for self-paced learners, or practitioners who may want in-depth learning on specific topics. We have included some supplementary material in optional sections, marked with an asterisk, which we believe are interesting, but which can be skipped without loss of continuity.

The most important prerequisites for this book are an understanding of multivariate calculus, and the intellectual maturity to understand the tradeoffs involved in mathematical modeling. Modeling does not involve one-size-fits-all approaches, and dogma about the absolute superiority of one model or algorithm over another is scarce. Instead, the primary intent of this book is to present a survey of important approaches to modeling transportation network problems, as well as the context to determine when particular models or algorithms are appropriate for real-world problems. Readers who can adopt this perspective will gain the most from the book.

Appendix A covers the mathematics needed for the network models in this

book. Readers of this book will have significantly different mathematical backgrounds, and this appendix is meant to collect the necessary results and concepts in one place. Depending on your background, some parts of it may need only a brief review, while other parts may be completely new.

While this book does not explicitly cover how to program these models and algorithms into a computer, if you have some facility in a programming language, it is highly instructive to try to implement them as you read. Many network algorithms are tedious to apply by hand or with other manual tools (calculator, spreadsheet). Computer programming will open the door to applying these models to interesting problems of large, realistic scale.

Difficulty Scale for Exercises

Inspired by Donald Knuth's *The Art of Computer Programming*, the exercises are marked with estimates of their difficulty. The key reference points are:

- 0** : A nearly trivial problem that you should be able to answer without any pencil-and-paper work.
- 20** : A straightforward problem that may require a few minutes of effort, but nothing too difficult if you have given the chapter a good read.
- 40** : A typical problem requiring some thought, or a few attempts at solution in different ways, but the answer should yield after some dedicated effort.
- 60** : A problem of above-average difficulty, where the correct approach is not obvious. You may require a bit of scratch paper or computer work as you try out different approaches before settling on a solution.
- 80** : A highly challenging or involved problem, which may be appropriate as a course project or other long-term study.
- 100** : An open problem in the research literature, whose solution would be a substantial advance in the understanding of transportation networks.

The tens digit indicates the intellectual difficulty of the exercise, while the ones digit indicates the amount of calculation required. An exercise rated at 50 may require more cleverness and insight than one ranked at 49, but the ultimate solution is shorter. Of course, each student will find different problems more challenging than others.

Contents

I	Preliminaries	1
1	Introduction to Transportation Networks	3
1.1	Transportation Networks	3
1.2	Examples of Networks	5
1.3	The Notion of Equilibrium	6
1.4	Traffic Assignment	11
1.5	Static Traffic Assignment	13
1.5.1	Overview	13
1.5.2	Critique	15
1.6	Dynamic Traffic Assignment	16
1.6.1	Overview	17
1.6.2	Critique	18
1.7	Historical Notes and Further Reading	20
1.8	Exercises	22
2	Network Representations and Algorithms	25
2.1	Terminology	25
2.2	Acyclic Networks and Trees	28
2.3	Data Structures	30
2.4	Shortest Paths	33
2.4.1	Shortest paths on acyclic networks	36
2.4.2	Shortest paths on cyclic networks: label correcting	38
2.4.3	Shortest paths on general networks: label setting	40
2.4.4	Shortest paths from one origin to one destination	42
2.5	Historical Notes and Further Reading	44
2.6	Exercises	45
3	Mathematical Techniques for Equilibrium	53
3.1	Fixed-Point Problems	54
3.2	Variational Inequalities	56
3.3	Convex Optimization	59
3.3.1	One-dimensional problems	63
3.3.2	Bisection method	66
3.3.3	Multidimensional problems with nonnegativity constraints	68

3.3.4	Constrained problems	69
3.4	Historical Notes and Further Reading	71
3.5	Exercises	72
II Static Traffic Assignment		75
4	Introduction to Static Assignment	77
4.1	Notation and Concepts	77
4.1.1	Commentary	81
4.2	Principle of User Equilibrium	83
4.2.1	A trial-and-error solution method	85
4.3	Three Motivating Examples	89
4.4	Constructing a Static Traffic Assignment Network	96
4.5	Historical Notes and Further Reading	99
4.6	Exercises	100
5	The Traffic Assignment Problem	103
5.1	Mathematical Formulations	103
5.1.1	Multifunctions and application to network equilibrium (*)	107
5.2	Properties of User Equilibrium	110
5.2.1	Existence and link flow uniqueness	111
5.2.2	Path flow nonuniqueness, entropy, and proportionality . .	113
5.2.3	Aggregation by origins or destinations	120
5.3	Alternative Assignment Rules	123
5.3.1	System optimal assignment	124
5.3.2	Perception errors	125
5.3.3	Bounded rationality (*)	128
5.4	User Equilibrium and System Optimal	131
5.4.1	Externalities	132
5.4.2	Mathematical equivalence	134
5.4.3	Price of anarchy	136
5.5	Historical Notes and Further Reading	138
5.6	Exercises	138
6	Algorithms for Traffic Assignment	143
6.1	Introduction to Assignment Algorithms	144
6.1.1	Why do different algorithms matter?	144
6.1.2	Framework	145
6.1.3	Convergence criteria	147
6.2	Link-Based Algorithms	149
6.2.1	Method of successive averages	149
6.2.2	Frank-Wolfe	155
6.2.3	Small network example	157
6.2.4	Large network example	159
6.2.5	Conjugate Frank-Wolfe	161

6.3	Path-Based Algorithms	168
6.3.1	Gradient projection	171
6.3.2	Manifold suboptimization	175
6.4	Bush-Based Algorithms	177
6.4.1	Bush labels	179
6.4.2	Shifting flows on a bush	183
6.4.3	Improving bushes	191
6.5	Likely Path Flow Algorithms	192
6.5.1	Primal method	198
6.5.2	Dual method	201
6.5.3	Traffic assignment by paired alternative segments	204
6.6	Historical Notes and Further Reading	221
6.7	Exercises	221
7	Sensitivity Analysis and Applications	231
7.1	Sensitivity Analysis Preliminaries	231
7.2	Calculating Sensitivities	235
7.2.1	Changes to the OD matrix	236
7.2.2	Changes to a link performance function	239
7.3	Network Design Problem	241
7.4	OD Matrix Estimation	248
7.5	Historical Notes and Further Reading	253
7.6	Exercises	253
8	Extensions of Static Assignment	257
8.1	Elastic Demand	258
8.1.1	Demand functions	258
8.1.2	Gartner's transformation	260
8.1.3	Variational inequality formulation	261
8.1.4	Optimization formulation	261
8.1.5	Solution method	263
8.1.6	Example	265
8.2	Link Interactions	265
8.2.1	Formulation	267
8.2.2	Properties	268
8.2.3	Algorithms	273
8.3	Stochastic User Equilibrium	278
8.3.1	Discrete choice modeling	279
8.3.2	Logit route choice	281
8.3.3	(*) The Markov property and the logit formula	286
8.3.4	Stochastic network loading	292
8.3.5	Stochastic user equilibrium	299
8.3.6	Relationships between logit loading and most likely path flows (*)	305
8.3.7	Alternatives to logit loading	307
8.4	Historical Notes and Further Reading	308

8.5 Exercises	309
III Dynamic Traffic Assignment	317
9 Network Loading	319
9.1 Link Model Concepts	320
9.1.1 Sending and receiving flow	321
9.1.2 Point queue	323
9.1.3 Spatial queue	325
9.2 Node Model Concepts	327
9.2.1 Links in series	330
9.2.2 Merges	332
9.2.3 Diverges	334
9.3 Combining Node and Link Models	335
9.4 Elementary Traffic Flow Theory	337
9.4.1 Traffic state variables	339
9.4.2 Cumulative counts and conservation	341
9.4.3 The Lighthill-Whitham-Richards model	345
9.4.4 Characteristics and the Newell-Daganzo method	350
9.5 LWR-based Link Models	356
9.5.1 Cell transmission model	356
9.5.2 Link transmission model	361
9.5.3 Point and spatial queues and the LWR model (*)	366
9.5.4 Discussion	369
9.6 Fancier Node Models	371
9.6.1 Basic signals	371
9.6.2 Equal priority movements	372
9.6.3 Intersections with priority	378
9.7 Historical Notes and Further Reading	382
9.8 Exercises	383
10 Time-Dependent Shortest Paths	393
10.1 Time-Dependent Shortest Path Concepts	393
10.1.1 Time-expanded networks	395
10.1.2 Bellman's principle in time-dependent networks	397
10.2 Time-Dependent Shortest Path Algorithms	398
10.2.1 FIFO networks	399
10.2.2 Discrete-time networks, one departure time	400
10.3 Departure Time Choice	404
10.3.1 Artificial origins and destinations	404
10.3.2 Arrival and departure time preferences	406
10.4 Dynamic A^*	410
10.5 Historical Notes and Further Reading	410
10.6 Exercises	411

11 Dynamic User Equilibrium	415
11.1 Towards Dynamic User Equilibrium	415
11.1.1 Flow Representation	416
11.1.2 Travel time calculation	417
11.1.3 Determining splitting proportions	419
11.1.4 Principle of dynamic user equilibrium	421
11.2 Solving for Dynamic Equilibrium	423
11.2.1 General framework	423
11.2.2 Convex combinations method	425
11.2.3 Simplicial decomposition	426
11.2.4 Gradient projection	428
11.3 Properties of Dynamic Equilibria	431
11.3.1 Existence: competition at merges	431
11.3.2 Uniqueness: Nie's merge	433
11.3.3 Nie's merge redux	437
11.3.4 Efficiency: Daganzo's paradox	438
11.3.5 Conclusion	441
11.4 Historical Notes and Further Reading	441
11.5 Exercises	442
A Mathematical Concepts	447
A.1 Indices and Summation	447
A.2 Vectors and Matrices	450
A.3 Sets	455
A.4 Functions	460
A.5 Exercises	468
B Optimization Concepts	471
B.1 Components of an Optimization Problem	471
B.2 Index Notation	476
B.2.1 Single index notation	478
B.2.2 Multiple index notation	482
B.2.3 Additional information on index notation	484
B.3 Vector and Matrix Notations	485
B.4 Examples of Basic Optimization Problems	488
B.5 More Examples of Optimization Formulations	495
B.6 General Properties of Optimization Problems	501
B.6.1 Local and global solutions	502
B.6.2 Objective function transformations	503
B.6.3 Existence of solutions	504
B.6.4 Necessary and sufficient conditions	506
B.7 Exercises	506

C Optimization Techniques	511
C.1 More Line Search Algorithms for One-Dimensional Optimization	511
C.1.1 Golden section method	512
C.1.2 Newton's method	514
C.2 Linear Programming	517
C.2.1 Basic properties and graphical solution method	517
C.2.2 Standard form of a linear program	525
C.2.3 Simplex algorithm	527
C.3 Unconstrained Nonlinear Optimization	534
C.3.1 Optimality conditions	534
C.3.2 Solution framework	538
C.3.3 Convergence criteria	539
C.3.4 Descent direction	539
C.3.5 Step size	541
C.3.6 Examples	541
C.4 Constrained Nonlinear Optimization	546
C.4.1 Optimality conditions	546
C.4.2 Examples of applying KKT conditions	549
C.4.3 Algorithms for linear constraints	555
C.4.4 Convex combination methods	556
C.4.5 Gradient projection	559
C.4.6 Manifold suboptimization	562
C.5 Integer Programming	566
C.5.1 Branch and bound algorithm	571
C.5.2 Total unimodularity	574
C.6 Metaheuristics	575
C.6.1 Simulated annealing	577
C.6.2 Genetic algorithms	583
D Algorithms and Complexity	591
D.1 Algorithms	591
D.2 Pseudocode	594
D.3 Algorithmic Efficiency	596
D.4 Complexity Classes	602
D.4.1 Decision problems vs. optimization	603
D.4.2 The classes P , NP , NP -hard, and NP -complete	604
D.4.3 Examples demonstrating NP -completeness	608
Bibliography	613

Part I

Preliminaries

Chapter 1

Introduction to Transportation Networks

This introductory chapter lays the groundwork for traffic assignment, providing some overall context for transportation planning in Section 1.1. Some examples of networks in transportation are given in Section 1.2. The key idea in traffic assignment is the notion of equilibrium, which is presented in Section 1.3. The goals of traffic assignment are described in Section 1.4. Traffic assignment models can be broadly classified as static or dynamic. Both types of models are described in this book, and Sections 1.5 and 1.6 provide general perspective on these types of models.

1.1 Transportation Networks

Planning helps ensure that transportation spending and policies are as effective as possible. As transportation engineers and researchers, we support this process by developing and running models which predict the impact of potential projects or policies — for instance, what would be the impact on city traffic and emissions if an extra lane was added on a major freeway? If the toll on a bridge was reduced? If streetcar lines are installed downtown? In this way, the benefits of projects can be compared with their costs, and funding and implementation priorities established. Depending on the models used, a variety of measures of effectiveness can be considered, and one may want to know the impacts of a project on mobility, congestion, emissions, equity, toll revenue, transit ridership, infrastructure maintenance needs, or countless other metrics.

This is rather difficult. To predict ridership on a new transit line with complete accuracy would require knowing how many trips every single possible rider makes, and the decision process each one of these potential riders uses when deciding whether or not to use transit. Unlike trusses or beams, human beings can behave in ways that are impossible to predict, maddeningly inconsistent, and motivated by a variety of factors difficult to observe (many of which occur

at a subconscious level). Further, most transportation infrastructure lasts for decades, meaning that effective planning must also predict the impact of projects and policies decades into the future. And so far, we've only considered the pure engineering dimension: introduce local, state, and federal politics into the mix, other stakeholders such as neighborhood associations and transit agencies, and a public with a variety of priorities (is it more important to reduce congestion, increase safety, or have livable communities?), and the picture only grows more complicated. What to do?

Enter the mathematical model. The purpose of a mathematical model is to translate a complicated, but important, real-world problem into precise, quantitative language that can be clearly and unambiguously analyzed. By their nature, models cannot account for all of the possible factors influencing planning. As the famous statistician George Box once quipped, "All models are wrong, but some models are useful." A useful model is one which provides enough insight that good decisions can be made. To do this, a model must capture the most important characteristics of the underlying system; must not require more input data than what is available for calibration; and must not require more time and memory than what available hardware permits.

Further, just because a model is useful does not mean it cannot be improved. Indeed, this is the goal of transportation researchers around the world. The usual pattern is to start with a model which is simple, transparent, insightful... and also wrong. This simple model can then be improved in ways to make it more correct and useful, and this is the general pattern which will be seen in this book. The first network models you will see are such gross simplifications of reality that you may question whether they can truly be of value in practice. Perhaps they can, perhaps they can't; but in any case, they form a foundation for more advanced and realistic models which relax the assumptions made earlier on.

For this reason, as a student of transportation planning, you should always be looking for the assumptions involved in everything you see. All models make assumptions which are not entirely correct. The relevant questions are, how much does this assumption limit the applicability of the model, and how easy would it be to relax this assumption? If you're looking for a research topic, finding an existing model and relaxing an assumption is often a good approach. With this book, if you clearly understand all of the assumptions underlying each model, and how they differ from those made in other models introduced, you're 90% of the way there.

Networks are a type of mathematical model which are very frequently used in the study of transportation planning. This introductory chapter gives a very brief overview of transportation networks, and provides a sketch for the remainder of the book.

This book covers both *static* and *dynamic* network models. Static models assume that network conditions are at steady-state, while dynamic models represent changes in congestion and demand patterns over the course of several hours or a day. Static models were the first to be developed historically, and remain the most commonly-used in current transportation planning practice.

Dynamic models are more realistic in portraying congestion, but require more data for calibration and validation, and more computational resources to run. Solving and interpreting the output of dynamic models is also more difficult. As research progresses, however, more planners are using dynamic models, particularly for applications when travel conditions are changing rapidly during the analysis period. This chapter will present a balanced perspective of the advantages and disadvantages of dynamic traffic assignment *vis-à-vis* static assignment, but one of them is worth mentioning now: dynamic traffic assignment models are inherently mode-specific.

That is, unlike in static assignment (where it is relatively easy to build “multimodal” networks mixing roadway, transit, air, and waterway infrastructure), the vast majority of dynamic traffic assignment models have been specifically tailored to modeling vehicle congestion on roadways. In recent years, researchers have started integrating other modes into dynamic traffic assignment, and this area is likely to receive more attention in years to come. However, the congestion model for each mode must be custom-built. This is at once an advantage (in that congestion in different modes arises from fundamentally different sources, and perhaps ought to be modeled quite differently) and a disadvantage (a “generic” dynamic traffic assignment model can only be specified at a very high level). For this reason, this book will focus specifically on automobile traffic on roadway networks. This is not meant to suggest that dynamic traffic assignment cannot or should not be applied to other modes, but simply an admission that covering other modes would essentially require re-learning a new theory for each mode. Developing such theories would make excellent research topics.

1.2 Examples of Networks

Networks are fundamental to the study of large-scale transportation models representing an entire metropolitan area, a state, or multistate regions. They can be applied in many contexts, including alternatives analysis, developing congestion pricing plans, identifying bottlenecks and critical infrastructure, shipping and freight logistics, multimodal planning, and disaster evacuation planning, to name only a few. The reason network models are so useful, and so broadly applicable, is because a mathematical network is a simple, compact, and flexible way to represent a large, complicated system.

A network consists of *links* and *nodes*. In transportation applications, a link usually represents a means of travel from one point to another: a road segment between two intersections, a bus route between two stops, and so on, as seen in Figure 1.1. The nodes, in turn, are the endpoints of the links. Quite often, nodes are adjacent to multiple links, so a node representing an intersection may adjoin multiple links representing road segments. Nodes and links may also be more abstract; for instance, links in a multimodal network might represent a transfer from one transport mode to another. The level of detail in a network varies from application to application. For multistate freight models, major highways may be the only links, and major cities the only nodes. For a city’s

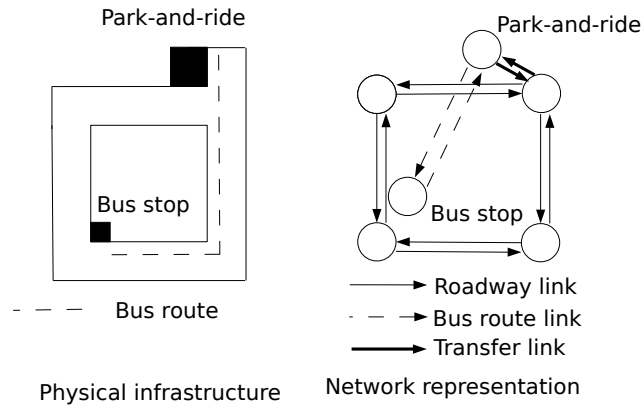


Figure 1.1: Nodes and links in transportation networks.

Table 1.1: Nodes and links in different kinds of transportation networks.

Network type	Nodes	Links
Roadway	Intersections	Street segments
Public transit	Bus or train stops	Route segments
Freight	Factories, warehouses, retailers	Shipping options
Air	Airports	Flights
Maritime	Ports	Shipping channels

planning model, all major and minor arterials may be included as well. For a more detailed model, individual intersections may be “exploded” so that different links represent each turning movement (Figure 1.2). Other examples of transportation networks are shown in Table 1.1.

1.3 The Notion of Equilibrium

The nature of transportation systems is that of multiple interacting systems. Congestion is determined by the choices travelers make: where, when, how often to travel, and by what mode. At the same time, these choices depend on congestion: travelers may choose routes or departure times to avoid congestion. These two “systems” (travel choices and system congestion) are thus interdependent and interrelated, with a circular or chicken-and-egg quality to their relationship. This interdependency lies at the root of transportation analysis. It is at once interesting, because of the complexity of transportation systems involving both humans and physical systems; challenging, because we must find a way to resolve this circular dependency; and frustrating, because obvious-looking policy interventions can actually be counterproductive. Some examples

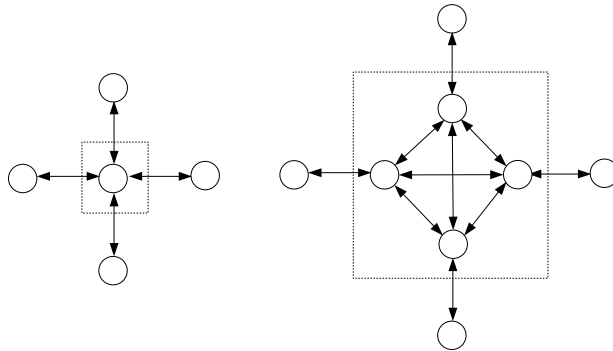


Figure 1.2: Two representations of the same intersection.

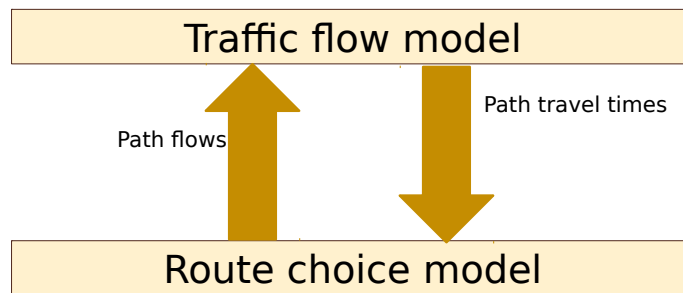


Figure 1.3: Traffic assignment as interacting systems.

of “paradoxical” effects will be seen in Chapters 4 and 11.

The schematic in Figure 1.3 shows the dependency between the choices made by travelers (sometimes called the *demand side*), and the congestion and delay in the system (sometimes called the *supply side*) in the basic traffic assignment problem. Each of these systems requires a distinct set of models. Demand-side models should be behavioral in nature, identifying what factors influence travel choices, and how. Supply-side models are often based in traffic flow theory, queueing theory, computer simulation, or empirical formulas describing how congestion will form.

It is not difficult to imagine other types of mutually-dependent transportation systems. Figure 1.4 shows how one might model traffic assignment in a region with a privately-operated toll road. Now, there are three systems. In addition to the demand side and supply side from before, the private toll operator can also influence the state of the network by choosing the toll in some way, such as maximizing toll revenue. But this choice is not made in isolation: as the toll is increased, drivers will choose alternate routes, suggesting that driver choices

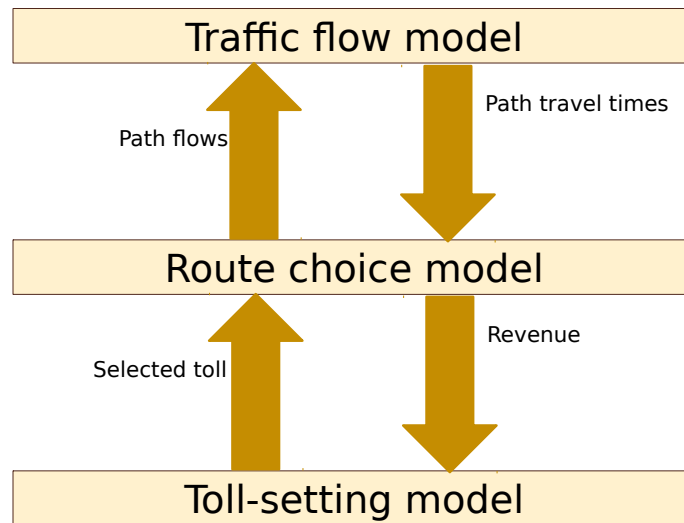


Figure 1.4: A more complicated traffic assignment problem, with tolls.

are affected by tolls just as the toll revenue is determined by driver choices. It is fruitful to think of other ways this type of system can be expanded. For instance, a government agency might set regulations on the maximum and minimum toll values, but travelers can influence these policy decisions through the voting process.

The task of transportation planners is to somehow make useful predictions to assist with policy decision and alternatives analysis, despite the complexities which arise when mutually-dependent systems interact. The key idea is that a good prediction is *mutually consistent* in the sense that all of the systems should “agree” with the prediction. As an example, in the basic traffic assignment problem (Figure 1.3), a planning model will provide both a forecast of travel choices, and a forecast of system congestion. These should be consistent in the sense that inputting the forecasted travel choices into the supply-side model should give the forecast of system congestion, and inputting the forecasted system congestion into the demand-side model should give the forecast of travel choices. Such a consistent solution is termed an *equilibrium*.

The word equilibrium is meant to allude to the concept of economic equilibrium, as it is used in game theory. In game theory, several agents each choose a particular action, and depending on the choices of all of the agents, each receives a payoff (perhaps monetary, or simply in terms of happiness or satisfaction). Each agent wants to maximize their payoff. The objective is to find a “consistent” or equilibrium solution, in which all of the agents are choosing actions which maximize their payoff (keeping in mind that an agent cannot control another agent’s decision). A few examples are in order.

Table 1.2: Alice and Bob's game; Alice chooses the row and Bob the column.

		Bob	
		Cactus Café	Desert Drafthouse
Alice	Cactus Café	$(-1, -1)$	$(1, 1)$
	Desert Drafthouse	$(1, 1)$	$(-1, -1)$

Consider first a game with two players (call them Alice and Bob), who happen to live in a small town with only two bars (the Cactus Café and the Desert Drafthouse). Alice and Bob have recently broken off their relationship, so they each want to go out to a bar. If they attend different bars, both of them will be happy (signified by a payoff of $+1$), but if they attend the same bar an awkward situation will arise and they will regret having gone out at all (signified by a payoff of -1). Table 1.2 shows the four possible situations which can arise — each cell in the table lists Alice's payoff first, followed by Bob's. Two of these are boldfaced, indicating that they are equilibrium solutions: if Alice is at the Cactus Café and Bob at the Desert Drafthouse (or vice versa), they each receive a payoff of $+1$, which is the best they could hope to receive given what the other is doing. The states where they attend the same bar are not equilibria; either of them would be better off switching to the other bar. This is a game with two equilibria in which Alice and Bob always attend the same bar each week.¹

A second game involves the tale of Erica and Fred, two criminals who have engaged in a decade-long spree of major art thefts. They are finally apprehended by the police, but for a minor crime of shoplifting a candy bar from the grocery store. The police suspect the pair of the more serious crimes, but have no hard evidence. So, they place Erica and Fred in separate jail cells. They approach Erica, offering her a reduced sentence in exchange for testifying against Fred for the art thefts, and separately approach Fred, offering him a reduced sentence if he would testify against Erica. If they remain loyal to each other, they will be convicted only of shoplifting and will each spend a year in jail. If Erica testifies against Fred, but Fred stays silent, then Fred goes to jail for 15 years while Erica gets off free. (The same is true in reverse if Fred testifies against Erica.) If they both testify against each other, they will both be convicted of the major art theft, but will have a slightly reduced jail term of 14 years for being cooperative. This game is diagrammed in Table 1.3, where the "payoff" is the negative of the number of years spent in jail, negative because more years in jail represents a worse outcome. Surprisingly, the only equilibrium solution is for both of them to testify against each other. From Erica's perspective, she is better off testifying against Fred *no matter what Fred will do*. If he is

¹There is also a third equilibrium in which they each randomly choose a bar each weekend, but equilibria involving randomization are outside the scope of this book.

Table 1.3: Erica and Fred's game.

		Fred	
		Testify	Remain silent
Erica	Testify	$(-14, -14)$	$(0, -15)$
	Remain silent	$(-15, 0)$	$(-1, -1)$

Table 1.4: Ginger and Harold's game.

		Harold	
		Heads	Tails
Ginger	Heads	$(+1, -1)$	$(-1, +1)$
	Tails	$(-1, +1)$	$(+1, -1)$

going to testify against her, she can reduce her sentence from 15 years to 14 by testifying against Fred. If he is going to stay silent, she can reduce her sentence from one year to zero by testifying against him. Fred's logic is exactly the same. This seemingly-paradoxical result, known as the *prisoner's dilemma*, shows that agents maximizing their own payoff can actually end up in a very bad situation when you look at their combined payoffs!

A third game, far less dramatic than the first two, involves Ginger and Harold, who are retirees passing the time by playing a simple game. Each of them has a penny, and on a count of three each of them chooses to reveal either the head or the tail of their penny. If the pennies show the same (both heads or both tails), Ginger keeps them both. If one penny shows heads and the other shows tails, Harold keeps them both. (Table 1.4). In this case, there is *no* equilibrium solution: if Ginger always shows heads, Harold will learn and always show tails; once Ginger realizes this, she will start showing tails, and so on *ad infinitum*.

You may be wondering how these games are relevant to transportation problems. In fact, the route choice decision can be seen as a game with a very large number of players. Some drivers may choose to avoid the freeway, anticipating a certain level of congestion and trying to second-guess what others are doing — but surely other drivers are engaging in the exact same process.² Each of these three games has bearing on the traffic assignment problem. The game with Alice and Bob shows that some games have more than one equilibrium solution (an issue of equilibrium *uniqueness*), although the two equilibrium solutions are

²To borrow from Yogi Berra, nobody takes the freeway during rush hour anymore — it's too congested.

the same if you just look at the total number of people at each bar and not which individuals go at each. What does it mean for transportation planning if a model can give several seemingly valid predictions? The game with Erica and Fred shows that agents individually doing what is best for themselves may lead to an outcome which is quite bad overall, an issue of equilibrium *efficiency*. As we will see later on, in transportation systems this opens the door for seemingly helpful projects (like capacity expansion on congested roads) to actually make things worse. The game with Ginger and Harold is a case where there is no equilibrium at all (an issue of equilibrium *existence*). If this could happen in a transportation planning model, then perhaps equilibrium is the wrong concept to use altogether. These questions of uniqueness, efficiency, and existence are important, and will appear throughout the book.

The three example games described above can be analyzed directly, by enumerating all the possible outcomes. However, transportation systems involve thousands or even millions of different “players” and an analysis by enumeration is hopeless. The good news is that the number of players is so great that little is lost in assuming that the players can be treated as a continuum.³ This allows us to work with smooth functions, greatly simplifying the process of finding equilibria. The remainder of this chapter introduces the basic traffic assignment problem in terms of the equilibrium concept and with a few motivating examples, but still in generally qualitative terms and restricted to small networks. The following two chapters provide us with the mathematical vocabulary and network tools needed to formulate and solve equilibrium on realistic, large-scale systems.

1.4 Traffic Assignment

There are many possible measures of effectiveness for evaluating the impacts of a roadway transportation project or policy. However, many of these can be calculated if one can predict the number of drivers on each roadway segment. These are called *link flows*. Predicting link flows allows a city or state government to evaluate different options.

If link flows are the output of a planning model, the main input is demographic data. That is, *given certain information about a population (number of people, income, amount of employment, etc.), we want to predict how many trips they will make, and how they will choose to travel*. Census records form an invaluable resource for this, often supplemented with travel surveys. Commonly, a medium-to-large random sample of the population is offered some money in exchange for keeping detailed diaries indicating all of the trips made within the next several weeks, including the time of day, reason for traveling, and other

³This is analogous to solving structural design problems by assuming the usual stress-strain relationships, which assume a continuous material. In reality, a beam or column is composed of many distinct atoms, not a homogeneous material — but surely it is unnecessary to model each atom separately. The continuum assumption works almost as well and is much, much easier to work with.

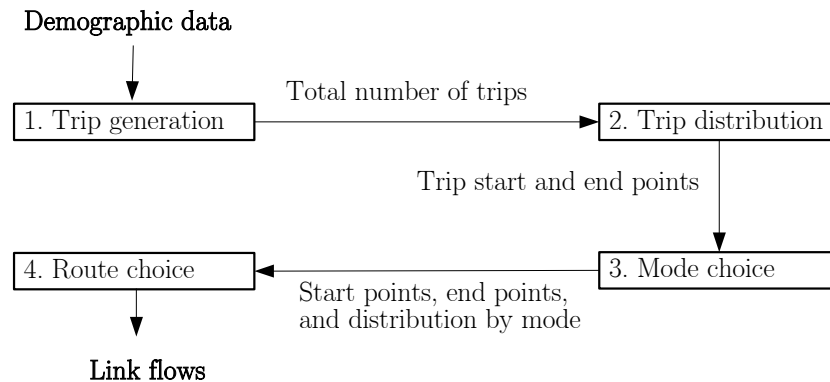


Figure 1.5: Schematic of the four-step process.

details.

To get link flows from demographic data, most regions use the so-called *four-step model* (Figure 1.5). The first step is *trip generation*: based on demographic data, how many trips will people make? The second is *trip distribution*: once we know the total number of trips people make, what are the specific locations people will travel to? The third is *mode choice*: once we know the trip locations, will people choose to drive, take the bus, or use another mode? The fourth and final step is *route choice*, also known as *traffic assignment*: once we know the modes people will take to their trip destinations, what routes will they choose? Thus, at the end of the four steps, the transition from demographic data to link flows has been accomplished.⁴

Demographics are not uniform in a city; some areas are wealthier than others, some areas are residential while others are commercial, some parts are more crowded while other parts have a lower population density. For this reason, planners divide a city into multiple *zones*, and assume certain parameters within each zone are uniform. Clearly this is only an approximation to reality, and the larger the number of zones, the more accurate the approximation. (At the extreme, each household would be its own zone and the uniformity assumption becomes irrelevant.) On the other hand, the more zones, the longer it takes to run each model, and at some point computational resources become limiting. Typical networks used for large metropolitan areas have a few thousand zones. Zones are often related to census tracts, to make it easy to get demographic information from census results.

The focus of this book is the last of the four steps, traffic assignment. In the beginning, we assume that the first three steps have been completed, and we know the number of drivers traveling between each origin zone and destination zone. From this, we want to know how many drivers are going to use each road-

⁴In more sophisticated models, the four steps may be repeated again, to ensure that the end results are consistent with the input data. There are also newer and arguably better alternatives to the four-step model.

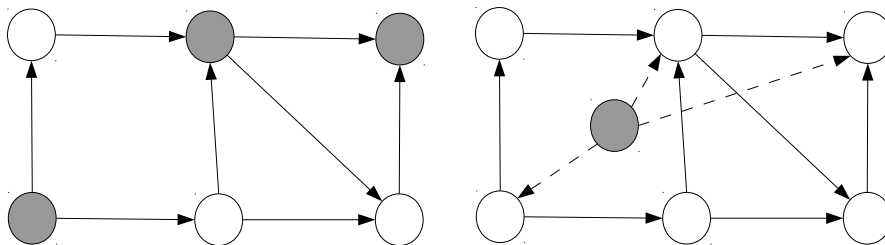


Figure 1.6: Centroids (shaded) coinciding with existing infrastructure, and artificial centroids. Dashed links on the right represent centroid connectors.

way segment, from which we can estimate congestion, emissions, toll revenue, or other measures of interest.

We've already discussed several of the pieces of information we need in order to describe traffic assignment models precisely, including zones and travel demand. The final piece of the puzzle is a representation of the transportation infrastructure itself: the transportation network described more in the next chapter.

It is usually convenient to use a node to represent each zone; such nodes are called *centroids*, and all trips are assumed to begin and end at centroids. The set of centroids is thus a subset of the set of nodes, defined in the next chapter. Centroids may coincide with physical nodes in the network. Centroids may also represent artificial nodes which do not correspond to any one physical point, and are connected to the physical infrastructure with links called *centroid connectors* (dashed lines in Figure 1.6).

1.5 Static Traffic Assignment

Figure 1.3 is the template for all traffic assignment models, be they static or dynamic: the choices of travelers lead to congestion patterns in the network (as predicted by a traffic flow model), and these patterns in turn influence the choices travelers make. The difference between static and dynamic traffic assignment lies in the traffic flow models used. Historically, static assignment models were the first to be developed, and research into dynamic models arose from the need to improve earlier, static models. Dynamic traffic assignment thus has a large number of parallels with static assignment; but where they differ, this difference is often intentional and important. Understanding these distinctions is key to knowing when dynamic models are appropriate to use.

1.5.1 Overview

In static assignment, the traffic flow model is based on *link performance functions*, which map the flow on each link to the travel time on that link. Mathe-

matically, if the notation (i, j) is used to refer to a roadway link connecting two nodes i and j , then x_{ij} is the flow on link (i, j) and the function $t_{ij}(x_{ij})$ gives the travel time on link (i, j) as a function of the flow on (i, j) . These functions $t_{ij}(\cdot)$ are typically assumed to be nonnegative, nondecreasing, and convex, reflecting the idea that as more vehicles attempt to drive on a link, the greater the congestion and the higher the travel times will be. A variety of link performance functions exist, but the most popular is the Bureau of Public Roads (BPR) function, which takes the form

$$t_{ij}(x_{ij}) = t_{ij}^0 \left(1 + \alpha \left(\frac{x_{ij}}{u_{ij}} \right)^\beta \right) \quad (1.1)$$

where t_{ij}^0 and u_{ij} are the free-flow time and capacity of link (i, j) , respectively, and α and β are shape parameters which can be calibrated to data. It is common to use $\alpha = 0.15$ and $\beta = 4$.

With such functions, the more travelers choose a path, the higher its travel time will be. Since travelers seek to minimize their travel time, travelers will not choose a path with high travel time unless there is no other option available. Indeed, if travelers only choose paths to minimize travel time, and if they have perfect knowledge of network conditions, then the network state can be described by the *principle of user equilibrium*: all used paths between the same origin and destination have equal and minimal travel times, for if this were not the case travelers would switch from slower routes to faster ones, which would tend to equalize their travel times.

It is not difficult to show that this *user equilibrium* state is not socially optimal, and that other configurations of traffic flow can reduce the average travel time (or even the travel time for all drivers) compared to the user equilibrium state. In other words, individual drivers seeking to minimize their own travel times will not always minimize travel times throughout the network, and this latter *system optimal* state can be contrasted with the user equilibrium one.

The prime advantage of using link performance functions like that in equation (1.1) is that the user equilibrium and system optimum states can be found with relative ease, even in realistic networks involving tens of thousands of links. Part II of the book discusses this in detail, showing how the static assignment problem can be formulated using the mathematical tools of optimization, fixed point, and variational inequality problems. These three representations of the equilibrium problem can be linked to powerful mathematical results which assure the existence and uniqueness of user equilibrium solutions under mild conditions. Efficient algorithms allow these states to be identified in a matter of minutes on large-scale networks.

For these reasons, static traffic assignment has been widely used in transportation planning practice for decades, and remains a powerful tool that can be used for performing alternatives analysis and generating forecasts of network conditions.

1.5.2 Critique

There are also a number of serious critiques of static assignment models, focused primarily on the link performance functions. By definition, static models do not monitor how network conditions (either demand or congestion) change over time, and implicitly assume a steady-state condition. This is clearly not the case. There are additional, subtler and more fundamental problems with link performance functions. This section describes a few of these problems.

First, not all vehicles on a link experience the same travel time. Even if the demand for travel on a link exceeds the capacity, the first vehicles to travel on that link will not experience much delay at all, while vehicles which arrive later may experience a very high delay. Together with the principle of user equilibrium, this means that the paths chosen by travelers will also depend on when they are departing. Route choices during periods of high congestion will be different from route choices made while this congestion is forming or dissipating. Furthermore, the travel time faced by a driver on a link depends crucially on the vehicles in front of them, and very little on the vehicles behind them. (A driver must adjust their speed to avoid colliding with vehicles downstream; a driver has no obligation to change their speed based on vehicles behind them.) This asymmetry is known as the *anisotropic* property of traffic flow, and it is violated by link performance functions — an increase in flow on the link is assumed to increase the travel time of all vehicles, and directly using link performance functions in a dynamic model would lead to undesirable phenomena, such as vehicles entering a link immediately raising the travel time for all other vehicles on the link, even those at the downstream end.

Second, the use of the word “flow” in static assignment is problematic. In traffic engineering, *flow* is defined as the (time) rate at which vehicles pass a fixed point on the roadway, and *capacity* is the greatest possible value for flow. By definition, flow cannot exceed capacity. However, the BPR function (1.1) imposes no such restriction, and it is common to see “flow-to-capacity” ratios much greater than one in static assignment.⁵ Instead, the x_{ij} values in static assignment are better thought of as *demand* rather than actual flow, since there is no harm in assuming that the demand for service exceeds the capacity, but it is impossible for the flow itself to exceed capacity. And for purposes of calibration, demand is much harder to observe than actual flow. These issues do not have clean resolutions.

Third, and related to the previous issue, link performance functions suggest that lower-capacity links have higher travel times under the same demand. But consider what happens at a freeway bottleneck, such as the lane drop shown in Figure 1.7. Congestion actually forms *upstream* of a bottleneck, and downstream of the lane drop there is no reason for vehicles to flow at a reduced speed. In

⁵There are several ways to add such a restriction, but these are less than satisfactory. A link performance function which tends to $+\infty$ as capacity is reached introduces numerical issues in solving for equilibrium. Explicitly adding link capacity constraints to the traffic assignment problem may make the problem infeasible, during peak periods there may be no way to assign all vehicles to the network without (temporarily) exceeding capacity.

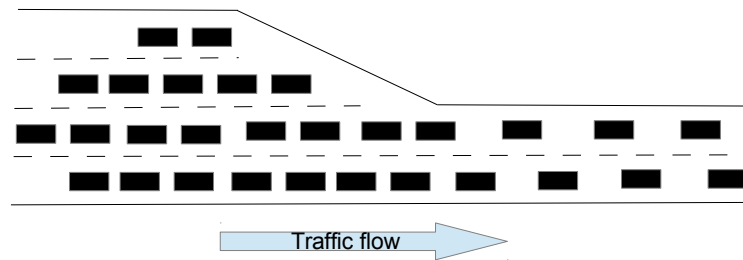


Figure 1.7: Congestion arises upstream of a bottleneck, not on the link with reduced capacity.

reality, it is upstream links that suffer when the demand for traveling on a link exceeds the capacity, not the bottleneck link itself.

Fourth, in congested urban systems it is very common for queues to fill the entire length of a link, causing congestion to spread to upstream links. This is observed on freeways (congested offramp queues) and in arterials (gridlock in central business districts) and is a major contributor to severe delay. In addition to the capacity, which is a maximum flow rate, real roadways also have a *jam density*, a maximum spatial concentration of vehicles. If a link is at jam density, no more vehicles can enter, which will create queues on upstream links. If these queues continue to grow, they will spread even further upstream to other links. Capturing this phenomenon can greatly enhance the realism of traffic models.

For all of these reasons, dynamic traffic assignment models shift to an entirely different traffic flow model. Rather than assuming simple, well-behaved link performance functions for each link we turn to traffic flow theory, to find more realistic ways to link traffic flow to congestion. Some early research in dynamic traffic assignment attempted to retain the use of link performance functions — for instance, modeling changes in demand by running a sequence of static assignment models over shorter time periods, with methods for linking together trips spanning multiple time periods. While this addresses the obvious shortcoming of static models, that they cannot accommodate changes in demand or model changes in congestion over time, it does nothing to address the more serious and fundamental problems with link performance functions described above. For this reason, this approach has largely been abandoned in favor of entirely different traffic flow models.

1.6 Dynamic Traffic Assignment

Dynamic traffic assignment arose from efforts to resolve the problems with static assignment noted in the previous section. While it is being used more and more in practice, it has not completely supplanted static traffic assignment. This is partially due to institutional inertia, but is also due to a few drawbacks associated with more realistic traffic flow models. Both these advantages and

drawbacks are discussed in this section.

1.6.1 Overview

The biggest difference between static and dynamic traffic assignment is in the traffic flow models used. A number of different traffic flow models are available, and a number of them are discussed in the following chapter. At a minimum, a traffic flow model for dynamic traffic assignment must be able to track changes in congestion at a fairly fine resolution, on the order of a few seconds. To do this, the locations of vehicles must be known at the same resolution. Most of them also address some or all of the shortcomings of link performance functions identified above.

The behavior of drivers is similar to that in static assignment in that drivers choose paths with minimum travel time. However, since congestion (and therefore travel time) changes over time, the minimum-time paths also change with time. Therefore, the principle of user equilibrium is replaced with a principle of *dynamic user equilibrium*: All paths used by travelers departing the same origin, for the same destination, *at the same time*, have equal and minimal travel times. Travelers departing at different times may experience different travel times; all the travelers departing at the same time will experience the same travel time at equilibrium, regardless of the path they choose. By virtue of representing demand changes over time, dynamic traffic assignment can also incorporate departure time choice of drivers, as well as route choice. Some dynamic traffic assignment models include both of these choices, while others focus only on route or departure time choice. Which choices are appropriate to model depends on which is more significant for a particular application, as well as the data and computational resources available. Most chapters of this book focus only on route choice, and as a general rule we will assume that departure times are fixed. In a few places we show how departure time choice can be added in.

It is important to specify that this equilibrium is based on the travel times the drivers actually experience, not the “instantaneous” travel times at the moment they depart. That is, we do not simply assume that drivers will pick the fastest route based on current conditions (as would be provided by most advanced traveler information services), but that they will anticipate changes in travel times which will occur during their journey. This suggests that drivers are familiar enough with the network that they know how congestion changes with time. This distinction is important — dynamic traffic assignment equilibrates on experienced travel times, not instantaneous travel times.

This means that it is impossible to find the dynamic user equilibrium in one step. Experienced travel times cannot be calculated at the moment of departure, but only after the vehicle has arrived at the destination. Therefore, dynamic traffic assignment is an iterative process, where route choices are updated at each iteration until an (approximate) dynamic user equilibrium solution has been found. This iterative process virtually always involves three steps, shown in Figure 1.8:

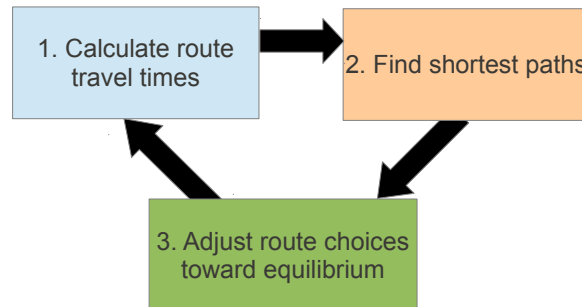


Figure 1.8: Three-step iterative process for dynamic traffic assignment.

Network loading: This is the process of using a traffic flow model to calculate the (time-dependent) travel times on each link, taking the routes and departure times of all drivers as inputs. In static assignment, this step was quite simple, involving nothing more than evaluating the link performance functions for each network link. In dynamic traffic assignment, this involves the use of a more sophisticated traffic flow model, or even the use of a traffic simulator. Network loading is discussed in Chapter 9.

Path finding: Once network loading is complete, the travel time of each link, at each point in time, is obtained. From this, we find the shortest path from each origin to each destination, *at each departure time*. Since we need experienced travel times, our shortest path finding must take into account that the travel time on each link depends upon the time a vehicle enters. This requires *time-dependent shortest path* algorithms, which are discussed in Chapter 10.

Route updating: Once time-dependent shortest paths have been found for all origins, destinations, and departure times, vehicles can be shifted from their current paths onto these new, shortest paths. As in static assignment, this step requires care, because shifting vehicles will change the path travel times as well. A few options for this are discussed in Chapter 11, along with other issues characterizing dynamic equilibrium. Unfortunately, and in contrast with static assignment, dynamic user equilibrium need not always exist, and when it exists it need not be unique.

1.6.2 Critique

The primary advantage of dynamic traffic assignment, and a significant one, is that the underlying traffic flow models are much more realistic. Link performance functions used in static assignment are fundamentally incapable of representing many important traffic phenomena. However, dynamic traffic assignment is not without its drawbacks as well.

Dynamic assignment requires considerably more computational time and memory than static assignment. As computers advance, this drawback is becoming less severe; but regardless of the computational resources available, one can run a large number of static traffic assignments in the time required for a single dynamic traffic assignment run. It may be advantageous to examine a large number of scenarios with static assignment, rather than a single run with dynamic traffic assignment, particularly if there is a lot of uncertainty in the input data.

Dynamic assignment models also require more input data for calibration. In addition to the usual link parameters such as capacity and free-flow time, dynamic traffic assignment models require a time-dependent origin-destination matrix, often known as a *demand profile*. Estimating even a static origin-destination matrix is difficult; estimating a dynamic demand profile can be even harder.

Furthermore, in addition to simply requiring more input data, dynamic traffic assignment also requires more accurate input data. Dynamic traffic assignment tends to be much more sensitive to the input data. Unfortunately, these models are more sensitive precisely because they are more realistic — features such as queue spillback mean that even a single erroneous capacity value can have ramifications throughout the entire network, not just on the link with the wrong value. If there is great uncertainty in the inputs (for instance, when attempting to predict demand decades into the future), then using a dynamic traffic assignment model may actually be further away from the truth than a static model, despite its more “realistic” congestion model.

Separately, dynamic traffic assignment is a relatively young field relative to static assignment, and there is no consensus on a single formulation. There are a large number of software packages (and an even larger number of academic models) which make differing assumptions and can lead to distinct results. By contrast, the optimization, variational inequality, and fixed point formulations in static assignment are completely standard, and therefore these models are more transparent. This book attempts to provide a high-level perspective, along with detailed discussions of a few of the most common modeling choices.

Finally, dynamic traffic assignment generally lacks neat, exact mathematical properties. In many dynamic traffic assignment models, one can create examples where no dynamic user equilibrium exists, or where this equilibrium is not unique. Proving convergence of iterative schemes is also more difficult.

All of these drawbacks must be traded off against the increased realism of dynamic traffic assignment models. Both static and dynamic traffic assignment models have their place as distinct tools for transportation engineering, and you should learn to identify circumstances where one or the other is more appropriate. As a general rule of thumb, dynamic models are most appropriate when the input data are known with high certainty (as in present-day traffic studies), and when queue lengths or other detailed congestion measures are required. Static models, by contrast, perform best when there is considerable uncertainty in the input data, or when it is more important to run a large number of scenarios than to obtain detailed congestion information.

1.7 Historical Notes and Further Reading

The first uses of networks were in solving mathematical puzzles like whether it is possible to walk through a city traversing each Bridge exactly once (the Königsberg bridge problem of Euler, 1736), or to visit each square on a chessboard exactly once using a knight (Vandermonde, 1774). The chemist Arthur Cayley also studied network-like structures to count particular types of hydrocarbon compounds (Cayley, Cayley). These investigations led to the development of the mathematical field of graph theory. There is no essential difference between what we call *networks* in this book, and what mathematicians more commonly call *graphs*. The “network” terminology is more common in engineering and optimization, so that is what we will use here. Our primary focus will be optimization problems defined over networks, and Ahuja et al. (1993) give a good treatment of applications, formulations, and algorithms for such problems. If you are interested in the broader field of graph theory, Diestel (2016) is a good reference.

As alluded to in Section 1.2, the network structure is very flexible and can represent more than just the “obvious” cases of physical transportation infrastructure with roads as links and intersections as nodes. Networks have been used to represent systems as broad as ecosystems, social structures, communication systems, waterways, power systems, organizational structures, and so on. Even within the transportation domain, phenomena such as elastic demand, destination choice, and mode choice can be accommodated within a network structure by adding links and nodes, and assigning them costs, in a special way (Sheffi, 1985, Chapter 9).

There is a long history of using equilibrium models in the field of economics. We are using the word *equilibrium* in the game theoretic sense, and the oligopoly pricing and production models of Cournot (1838), Bertrand (1883), and Edgeworth (1897) are the earliest ancestors of modern game theory. The modern formulations of game theory were given by von Neumann and Morgenstern (1944) and Nash (1950). Fudenberg and Tirole (1991), Gibbons (1992), Rasmussen (2006), and Ritzberger (2002) provide overviews of this field and its historical development. An early example of this type of analysis in the transportation field is Pigou (1920), who gave an example showing that the equilibrium, “user optimal” solution need not be efficient, in terms of maximizing total utility across travelers. (His example is described later in the book, in Section 4.3.) Among traffic engineers, Wardrop (1952) is credited with introducing two principles of route choice that correspond to the user optimal and system optimal states.

Traffic assignment is essentially solving an economic game for equilibrium on a network representing transportation infrastructure. As discussed briefly in Section 1.4, traffic assignment is part of the larger transportation planning process. Transportation planning emerged in the aftermath of World War II, due to the connected needs of suburbanization, population growth, and major investments in freeways and other infrastructure — how should this new infrastructure be designed, how should alternatives be compared, and which projects

deserve funding? Comprehensive overviews of transportation planning can be found in Domencich and McFadden (1975) and Meyer and Miller (2000).

The “four-step model” is very commonly used in transportation planning, consisting of four steps in sequence: trip generation, trip distribution, mode choice, and traffic assignment (route choice). Prior to executing these steps, a planner divides the region into zones in which trips originate and terminate. Trip generation aims to determine the number of trips originating at zone r , called the number of *productions* P_r , and the number of trips terminating at each zone s , called the number of *attractions* A_s . Productions and attractions are estimated based on land use, demographic data, and increasingly by observing cellular phone trajectories and other location data. The *Trip Generation Manual*, published by the Institute of Transportation Engineers (2021), is an example of a professional reference compiling trip generation data. Other approaches to trip generation are described in Kassoff and Deutschman (1969) and Chapter 4 of Ortúzar and Willumsen (2011).

Trip distribution converts zonal productions and attractions into interzonal trips N_{rs} , giving the number of trips which start at zone r and end at zone s . To match productions we must have $\sum_s N_{rs} = P_r$ for each origin r , and to match attractions we must have $\sum_r N_{rs} = A_s$ for each destination s , but these “consistency” constraints are not enough to specify the values N_{rs} . The gravity model provides one way to find these values, with the formula

$$N_{rs} = \alpha_r P_r \beta_s O_s \phi_{rs}, \quad (1.2)$$

where α_r and β_s are proportionality constants for each origin and destination, and ϕ_{rs} is a *friction function* indicating how onerous travel is between zones r and s . The constants α_r and β_s are set to satisfy the consistency constraints. Different forms of ϕ_{rs} are available, but the idea is that zones which are further away should have fewer trips between them, so ϕ_{rs} should shrink with distance, travel time, and/or accessibility. With a little bit of imagination, one can view equation (1.2) as analogous to Newton’s (1687) celebrated model of gravitation: factor out $\alpha_r \beta_s$ as the “gravitational constant,” interpret P_r and O_s as the masses of two objects, and take ϕ_{rs} to be the reciprocal of the squared distance between zones r and s , and *voilà*, we have Newton’s formula. This interpretation is a bit precious, and there is no real reason to believe that travelers behave in a way consistent with physical gravity. However, by selecting a negative exponential form of the friction function $\phi_{rs} = \exp(-\gamma \kappa_{rs})$, where κ_{rs} expresses the generalized cost of travel between zones r and s , and γ is a constant expressing the importance of this generalized cost in destination choice, we obtain a model with more solid foundations. The gravity model with negative exponential friction can be interpreted using the language of utility maximization (McFadden, 1974a), entropy maximization (Wilson, 1970), and maximum likelihood (Wilson et al., 1981), all of which are more plausible explanations of behavior than Newtonian gravity.

Mode choice is commonly performed using a logit model from discrete choice, following McFadden (1974b). Assume that a traveler must make a choice among

n modes (such as driving, taking transit, or riding a bicycle). Further assume that each traveler derives a utility associated from each mode choice. Some of these factors are known to the planner (such as travel time and monetary cost by each mode, and demographic data such as income level and vehicle ownership), whereas others are not (such as personal tastes). If we denote the former by V_m for each mode m , and the latter by a random variable ϵ — note here that we are assuming ϵ is independent and identically distributed for each mode — then one can show that the probability that a given traveler selects mode m is

$$P_m = \exp(V_m) \bigg/ \sum_{m'=1}^n \exp(V_{m'}) \quad (1.3)$$

if the random variables ϵ follow a Gumbel distribution.⁶ These probabilities are used to split the total interzonal trips N_{rs} calculated during trip distribution: the number of trips from zone r to s using mode m is given by $N_{rs}P_m$, and it is these values that are used as the origin-destination matrix for traffic assignment in the remainder of the book. The logit model is also discussed more in Section 8.3.

Although the four-step model has a long history of use, and remains the most common planning paradigm in practice, many researchers have suggested alternative techniques that can represent more sophisticated travel behavior. For instance, considering travelers who make several stops as part of a trip (*trip chaining*) in the four-step model is awkward. Activity-based modeling is one alternative to the traditional practice; see Bhat and Koppelman (1999) as a starting point for this growing field.

Static traffic assignment, as described in this book, was first formulated by Beckmann et al. (1956), who also introduced mathematical representations which will be described further in Chapter 5. A great deal of subsequent research further developed and extended the basic static traffic assignment model. The books by Sheffi (1985), Patriksson (1994), and Bell and Iida (1997) discuss many of these further developments.

Dynamic traffic assignment is relatively newer. The first dynamic traffic assignment models were developed by Merchant and Nemhauser (1978a,b). These models received increasing attention starting in the late 1990s, as more powerful computers became available, and Peeta and Ziliaskopoulos (2001) give a review of these advances. Another perspective, tailored towards practicing engineers and planners, was provided in a primer (Chiu et al., 2010).

1.8 Exercises

1. [10] If all models are wrong, how can some of them be useful?

⁶The Gumbel distribution has a similar shape to the normal distribution, but with fatter tails. The standard Gumbel distribution has cumulative distribution function $\exp(-\exp(-x))$. The one used in equation (1.3) has additional scaling and shifting parameters that are calibrated to data.

2. [10] All of the links in Figure 1.1 have a “mirror” connecting the same nodes, but in the opposite direction. When might mirror links not exist?
3. [23] A network is called *planar* if it can be drawn in two dimensions without links crossing each other. (The left network in Figure 1.2 is planar, but not the network on the right.) Do we expect to see planar graphs in transportation network modeling? Does it depend on the mode of transportation? What other factors might influence whether a transportation network is planar?
4. [35] Table 1.1 shows how networks can represent five types of transportation infrastructure. List at least five more systems (not necessarily in transportation) that can be modeled by networks, along with what nodes and links would represent.
5. [45] What factors might determine how large of a geographic area is modeled in a transportation network (e.g., corridor, city, region, state, national)? Illustrate your answer by showing how they would apply to the hypothetical projects or policies at the start of Section 1.1.
6. [45] What factors might determine the level of detail in a transportation network (e.g., freeways, major arterials, minor arterials, neighborhood streets)? Illustrate your answer by showing how they would apply to the hypothetical projects or policies at the start of Section 1.1.
7. [21] Provide an intuitive explanation of the prisoner’s dilemma, as described in the Erica-Fred game of Table 1.3. Why does it happen? Name other real-world situations which exhibit a similar phenomenon.
8. [20] For each of the following games, list all equilibria (or state that none exist). In which of these games do equilibria exist; in which are the equilibria unique; and in which are there some equilibria which are inefficient? These games are all played by two players A and B: A chooses the row and B chooses the column, and each cell lists the payoffs to A and B, in that order.

(a)	<i>L</i>	<i>R</i>	(b)	<i>L</i>	<i>R</i>
<i>U</i>	(8, 13)	(5, 14)	<i>U</i>	(12, 12)	(2, 10)
<i>D</i>	(10, 10)	(7, 12)	<i>D</i>	(10, 2)	(4, 5)

(c)	<i>L</i>	<i>R</i>	(d)	<i>L</i>	<i>R</i>
<i>U</i>	(8, 6)	(10, 8)	<i>U</i>	(4, 5)	(5, 6)
<i>D</i>	(2, 3)	(8, 4)	<i>D</i>	(3, 4)	(6, 3)

9. [1] Explain the difference between the demand for travel on a link, and the flow on a link.

10. [42] Explain why a less realistic model less sensitive to correct input data may be preferred to a more realistic model more sensitive to correct inputs, and in what circumstances. Give specific examples.

Chapter 2

Network Representations and Algorithms

This chapter introduces networks as they are used in the transportation field. Section 2.1 introduces the mathematical terminology and notation used to describe network elements. Section 2.2 discusses two special kinds of networks which are used frequently in network analysis, acyclic networks and trees. Section 2.3 then describes several ways of representing a network in a way computers can use when solving network problems. This third section can be skipped if you do not plan to do any computer programming. Section 2.4 describes the shortest path problem, a classic network algorithm which plays a central role in traffic assignment.

2.1 Terminology

Because of its relative youth, there are a variety of notational conventions in the transportation networks community. A common notation is adopted in the book to present the methods and topics in a consistent manner, but you should be aware that other authors may use slightly different conventions and definitions. Table 2.1 gives an example of some terms which are often used synonymously (or nearly synonymously) with ours. These differences are mainly a matter of style, not substance, but when reading other books or papers you should pay careful attention to the exact wording of their definitions.

The fundamental construct we will utilize is the *network*. A network is a collection of *nodes*, and a collection of *links* which connect the nodes. A network is denoted $G = (N, A)$, where N is the set of nodes and A is the set of links. Figure 2.1(a) shows a simple network, with four nodes in the set $N = \{1, 2, 3, 4\}$ and five links in the set $A = \{(1, 2), (1, 3), (2, 3), (2, 4), (3, 4)\}$. Notice that the notation for each link contains the two nodes connected by the link: the upstream node is called the *tail* of the link, and the downstream node the *head*. We will often refer to the total number of nodes in a network as n

Table 2.1: A thesaurus of network terminology

Terminology in this book	Alternative terms
Network	Graph
Node	Vertex
Link	Arc, edge, line
Tree	Arborescence

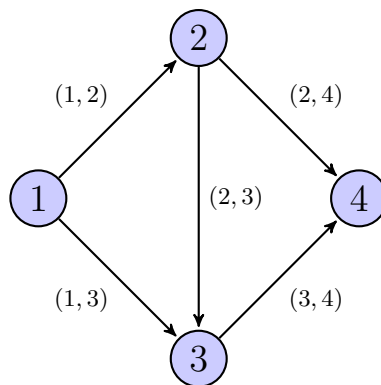


Figure 2.1: Example network notation

and the total number of links as m . This book is solely concerned with the case where n and m are finite — networks with infinitely many nodes and links are sometimes studied in theoretical mathematics, but rarely in transportation applications.

All of the networks in this book are *directed*. In a directed network, a link can only be traversed in one direction, specified by the ordering of the nodes. Therefore, $(1,2)$ and $(2,1)$ represent different links: they connect the same nodes, but represent travel in opposite directions. Unless stated otherwise, we assume that there are no “self-loops” (i,i) from a node to itself, and no parallel links, so the notation (i,j) is unambiguous. This is not usually a restrictive assumption, since we can introduce artificial nodes to split up self-loops or parallel links, as shown in Figure 2.2. The upper left panel in this figure shows a network with a self-loop $(2,2)$. By introducing a fourth node in the bottom left, we have divided the self-loop into two links $(2,4)$ and $(4,2)$. In the upper right panel of the figure, there are two networks connecting nodes 2 and 3, so the notation $(2,3)$ does not tell us which of these two links we are referring to. By introducing a fourth node in the bottom right, we now have three links: $(2,3)$, $(2,4)$, and $(4,3)$, which collectively represent both of the ways to travel between nodes 2 and 3 in the original network, but without parallel links. These new nodes are *artificial* in the sense that they do not represent physical transportation infrastructure, but are inserted for modeling convenience. Artificial nodes (and

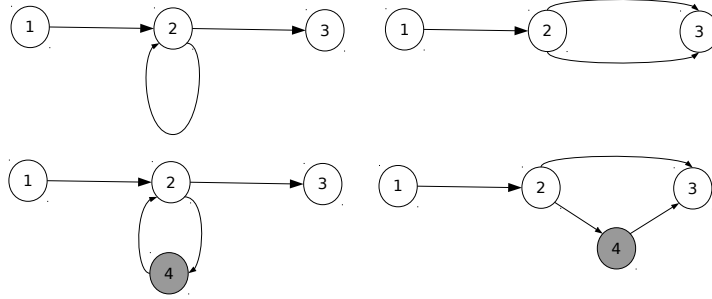


Figure 2.2: Artificial links can be introduced to simulate self-loops and parallel links.

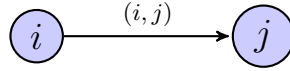


Figure 2.3: Generic two-node network

artificial links) play an important role in simplifying certain network problems, as discussed throughout the book.

To introduce some more terminology, Figure 2.3 shows a two-node network with nodes i and j connected by the link (i, j) . In this figure, we say that link (i, j) is *incident* to both i and j because it is connected to both. We would further say that (i, j) is an *outgoing* link of i and an *incoming* link of j . If $(i, j) \in A$, then we say that node j is *adjacent* to node i . In this example, j is adjacent to i , but i is not adjacent to j . Adjacency can also be applied to links; two links are adjacent if the head of the first link is the tail of the second link. In Figure 2.1, both links $(2, 3)$ and $(2, 4)$ are adjacent to $(1, 2)$.

The *forward star* of a node i is the set of all outgoing links, denoted $\Gamma(i)$; the *reverse star* is the set of all incoming links, denoted $\Gamma^{-1}(i)$. In Figure 2.1, $\Gamma(2) = \{(2, 3), (2, 4)\}$ and $\Gamma(3) = \{(3, 4)\}$, while $\Gamma^{-1}(2) = \{(1, 2)\}$ and $\Gamma^{-1}(3) = \{(1, 3), (2, 3)\}$.

The *degree* of a node is the total number of links incident to that node. The node degree can be separated into the *indegree* and *outdegree* of a node. The indegree is the total number of incoming links at a node, while the outdegree is the number of outgoing links at a node. The degree is then the sum of the indegree and outdegree. Referring back to Figure 2.1, node 2 has an indegree of 1, an outdegree of 2 and a degree of 3.

A *path* π is a sequence of adjacent links connecting two nodes i_0 and i_k . We can either write π as an ordered set of links

$$\{(i_0, i_1), (i_1, i_2), (i_2, i_3), \dots, (i_{k-1}, i_k)\},$$

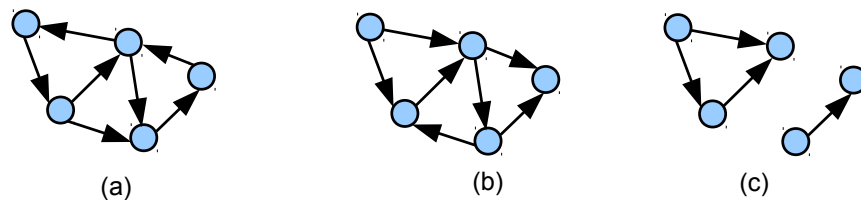


Figure 2.4: Networks which are (a) strongly connected; (b) connected, but not strongly connected; (c) disconnected.

or more compactly, by the nodes passed on the way with the notation

$$[i_0, i_1, i_2, i_3, \dots, i_{k-1}, i_k].$$

A path is a *cycle* if the starting and ending nodes are the same ($i_0 = i_k$). Paths which contain a cycle are called *cyclic*; paths which do not have a cycle as a component are called *acyclic*. Cyclic paths are uncommon in transportation problems, so unless stated otherwise, we only consider acyclic paths. Let Π^{rs} denote the set of all acyclic paths connecting nodes r and s , and let Π denote the set of all acyclic paths in the entire network, that is, $\Pi = \cup_{(r,s) \in N^2} \Pi^{rs}$. A network is *connected* if there is at least one path connecting any two nodes in the network, assuming that the links can be traversed in either direction (that is, ignoring the direction of the link); otherwise it is *disconnected*. A network is *strongly connected* if there is at least one path connecting any two nodes in the network, obeying the given directions of the links. Figure 2.4 shows networks which are strongly connected; connected, but not strongly connected; and disconnected.

2.2 Acyclic Networks and Trees

Networks which do not have any cycles are called *acyclic networks*. Acyclic networks are extremely important, because the lack of cycles can greatly simplify analysis. Even when networks actually have cycles in reality, many efficient transportation network algorithms temporarily divide the network into a set of acyclic subnetworks.

A defining characteristic of acyclic networks is that a *topological order* can be established. A topological order is a labeling of each node with a number between 1 and n (n is the number of nodes), such that every link connects a node of lower topological order to a node of higher topological order. Every path, therefore, consists of a sequence of nodes in increasing topological order. As an example, in Figure 2.1, the nodes are labeled in a topological order: each path ($[1, 3, 4]$, $[1, 2, 4]$, and $[1, 2, 3, 4]$) traverses nodes in increasing numerical order. (The phrase “a” topological order is used because it may not be unique, and there may be several ways to label the nodes so that links always connect

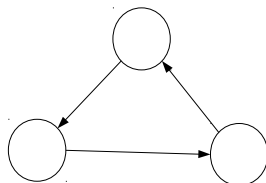


Figure 2.5: Cyclic network with no topological order.

lower-numbered nodes to higher-numbered ones.) In general networks, a topological order may not exist (see Figure 2.5, where a cycle makes it impossible to label nodes so that the numbers increase when traversing any link). The following theorem shows that the existence of a topological order is a *defining* characteristic of acyclic networks.

Theorem 2.1. *A topological order exists on a network if and only if it is acyclic.*

Proof. (\Rightarrow) Assume a topological order exists on G . Then every path contains a sequence of nodes whose topological order is strictly increasing (if this were not so, then there is a link connecting a higher topological order node to a lower one, contradicting the definition of topological order). Therefore, no path can repeat the same node more than once, and the network is acyclic.

(\Leftarrow) Assume the network G is acyclic. Then we can prove existence of a topological order by construction. Let $o(i)$ be the topological order of node i . We will describe a procedure to build such a topological order one step at a time. At any point in time, a “marked” node is one which has a topological order assigned, and an “unmarked” node is one which does not yet have its topological order. Because G is acyclic, there is at least one node r with no incoming links. (You will be asked to prove this statement as an exercise.) Let $o(r) = 1$. Again, because G is acyclic, there is at least one unmarked node with no incoming links from an unmarked node; mark this node with the topological order 2. This process can be repeated until all nodes are marked; if at any point every unmarked node has an incoming link from another unmarked node, then a cycle exists, contradicting the assumption of acyclicity. Otherwise, the topological order will have been constructed. The topological order so constructed is valid because when a node is assigned an order, the only incoming links are from nodes which have already been marked, and thus have a lower topological order. \square

A *tree* is a special type of acyclic network, which also shows up frequently in transportation network algorithms. A tree is defined as a network in which there is a unique node r (called the *root*) which has the property that exactly one path exists between r and every other node in the network.¹ Figure 2.6 shows a tree. Other useful properties of trees are:

¹In graph theory, this is usually called an *arborescence*, and trees are defined for undirected graphs. However, the transportation community generally uses the term “tree” in both cases, a convention followed in this book.

- A tree has at least two nodes with degree one.
- In a tree, there is at most one path between any two nodes.
- A tree would become disconnected if any of its links were deleted.
- Every node i in a tree (except the root) has a unique incoming link; the tail node of that unique link is called the *parent* of node i . Similarly, the head nodes of the links in the forward star of i are its *children*.

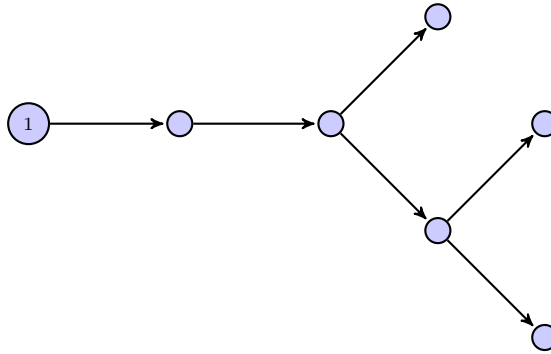


Figure 2.6: A tree

2.3 Data Structures

In practice, the methods and techniques in this book are carried out by computer programs. Thus, it is important not only to have a convenient way to theoretically represent a network (the network structure) but a way to store data so that it can be easily accessed and utilized by computer programs. To demonstrate, Figure 2.7 presents a network similar to those earlier, with additional information provided. In this case, the extra information is the travel time of the link from i to j , denoted t_{ij} .

The first data structure is the node-link incidence matrix, shown in Figure 2.8. The columns are indexed by the network links, and the rows by the network nodes. A “1” in the matrix indicates an outgoing link from that node, while a “−1” indicates an incoming link. A “0” indicates that the link is not incident to that particular node. This is a fairly inefficient data structure, especially for large networks, as in nearly any case there are going to be a large number of zeros in every row. Note that the forward and reverse stars of node i can respectively be identified with the columns in the matrix which have a 1 or −1 in the i -th row.

The second data structure, a node-node adjacency matrix is a simpler representation of a network but also suffers from inefficiencies. Each row represents a node i and each column a node j . A “1” indicates the existence of a link (i, j) .

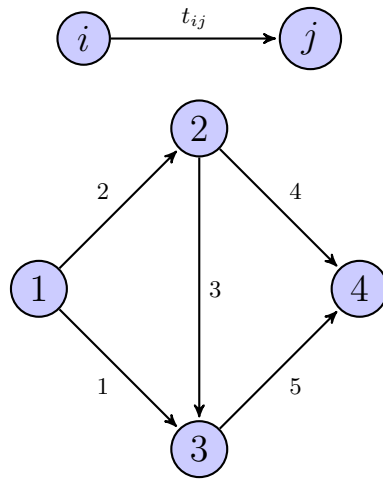


Figure 2.7: Graph used for network representation examples

$$\begin{array}{c}
 \begin{array}{ccccc}
 & (1, 2) & (1, 3) & (2, 3) & (2, 4) & (3, 4) \\
 1 & \left[\begin{array}{ccccc}
 1 & 1 & 0 & 0 & 0 \\
 -1 & 0 & 1 & 1 & 0 \\
 0 & -1 & -1 & 0 & 1 \\
 0 & 0 & 0 & -1 & -1
 \end{array} \right] \\
 2 \\
 3 \\
 4
 \end{array}
 \end{array}$$

Figure 2.8: Node-link incidence matrix for the network in Figure 2.7.

$$\begin{array}{c}
 \begin{array}{cccc}
 & 1 & 2 & 3 & 4 \\
 1 & \left[\begin{array}{cccc}
 0 & 1 & 1 & 0 \\
 0 & 0 & 1 & 1 \\
 0 & 0 & 0 & 1 \\
 0 & 0 & 0 & 0
 \end{array} \right] \\
 2 \\
 3 \\
 4
 \end{array}
 \end{array}$$

Figure 2.9: Node-node adjacency matrix for the network in Figure 2.7.

In this structure, a non-zero entry provides both existence and direction information. An example of a node-node adjacency matrix is given in Figure 2.9. In very dense networks, with many more links than nodes ($m \gg n$), node-node adjacency matrices can be efficient.

A shortcoming of the previous data structures is that they do not contain information beyond the existence and direction of the links in a network. In transportation applications we are typically working with networks for which we want to store and use more information about each element. For example, we may want to store information about the travel time, number of lanes (capacity), speed limit, signalization, etc. for each link. Adjacency lists give us an opportunity to store this information efficiently in a manner that is compatible with programming languages and their built-in data structures.

Storing a network as an array is more efficient in terms of space, and is easier to work with in some programming languages. Storage is not wasted on a large number of zeros, as with the adjacency matrices. A “forward star” representation is presented below. In the forward star representation, links are sorted in order of their tail node, as shown in Table 2.2. A second array, called `point`, is used to indicate where the forward star of each node begins. If the network has n nodes and m links, the `point` array has $n + 1$ entries, and `point(n + 1)` is always equal to $m + 1$; the reason why is discussed below. As shown in Table 2.2, `point(2) = 3` because the link 3 is the first link adjacent to node 2; `point(3) = 5` because link 5 is the first link adjacent to node 3, and so forth. Three conventions associated with the star representation are:

1. The number of entries in the `point` array is one more than the number of nodes.
2. We automatically set `point(n+1) = m + 1`.
3. If there are no outgoing links, `point(i) = point(i+1)`

With these conventions, we can say that the forward star for any node i consists of all links whose IDs are greater than or equal to `point(i)`, and strictly less than `point(i+1)`. This representation is most useful when we frequently need to loop over all of the links leaving a particular node, which can be accomplished by programming a “for” loop between `point(i)` and `point(i+1) - 1`, and referring to the link arrays with these indices. We can make this statement universally, because we defined the `point` array to have one more entry than the number of nodes. If `point` only had n entries, then referring to `point(n+1)`

Table 2.2: Forward star representation of a network.

Node	Point	Link	tail	head	cost	capacity
1	1	1	1	3	1	...
2	3	2	1	2	2	...
3	5	3	2	3	3	...
4	6	4	2	4	4	...
5	6	5	3	4	6	...

would be meaningless in the above statements. It is also possible to store a network in a “reverse star” representation along similar lines, which is most useful when we frequently need to loop over all the links entering a particular node. The exercises explore this further, along with an array-based method for easily iterating over both the forward and reverse stars of a node.

The purpose of this brief discussion of data structures is not to provide a comprehensive treatment of the topic. Rather, it is intended to present some basic ideas related to network storage and data representation. It should also prompt you to think about data structures carefully when working with networks. Transportation network problems tend to be large in size and complicated in nature. The difference between computer code that takes minutes to run, as opposed to hours, is often the way data is stored and passed.

2.4 Shortest Paths

As a first network algorithm, we’ll discuss how to find the shortest (least cost) path between any two nodes in a network in an efficient and easily-automated way. In shortest path algorithms, we are given a network $G = (N, A)$; each link $(i, j) \in A$ has a *fixed* cost c_{ij} . The word “cost” does not necessarily mean a monetary cost, and refers to whatever total quantity we are trying to minimize. In traffic applications, we often use the link travel times t_{ij} as the cost, to find the least travel-time path. If there are tolls in a network, the cost of each link may be the sum of the toll on that link, and the travel time on that link multiplied by a “value of time” factor converting time into monetary units. Costs may reflect still other factors, and some of these are explored in the exercises. Because we can solve all of these problems in the same way, we may as well use a single name for all of these quantities we are trying to minimize, and “cost” has become the standard term.

The word “fixed” in the last paragraph is emphasized because in many transportation problems the cost is dependent on the total flow and vehicle route choices, and it is not reasonable to assume fixed costs. However, even in such network problems the most practical solution methods involve solving several shortest path problems as part of an iterative framework, updating paths as travel times change. The context for a shortest path algorithm is to find the least travel-time path between two nodes, at the current travel times — with ev-

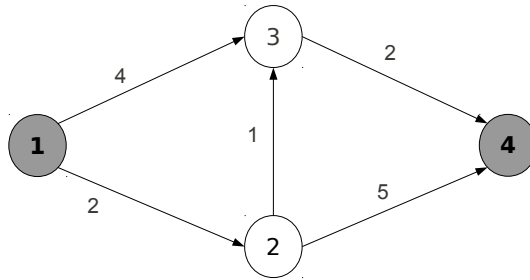


Figure 2.10: Example network for shortest paths (link costs indicated).

everybody's route choices held fixed. By separating the process of identifying the shortest path from the process of shifting path flows toward the shortest path, we simplify the problem and end up with something which is relatively easy to solve. This problem can also be phrased in the language of optimization, as shown in Appendix B.5, by identifying an objective function, decision variables, and constraints. Here, we develop specialized algorithms for the shortest path problem which are more efficient and more direct than general optimization techniques.

Although the shortest path problem most obviously fits into transportation networks, many other applications also exist in construction management, geometric design, operations research, and many other areas. For instance, the fastest way to solve a Rubik's Cube from a given position can be solved using a shortest path algorithm, as can the fewest number of links needed to connect an actor to Kevin Bacon when playing Six Degrees of Separation.

A curious fact of shortest path algorithms is that finding the shortest path from a single origin to *every other* node is only slightly harder than finding the shortest path from that origin to a single destination. This also happens to be the reason why we can find shortest paths without having to list all of the zillions of possible paths from an origin to a destination, and adding up their paths. This common reason is *Bellman's Principle*, which states that *any segment of a shortest path must itself be a shortest path between its endpoints*. For instance, consider the network in Figure 2.10, where the costs c_{ij} are printed next to each link. The shortest path from node 1 to node 4 is $[1, 2, 3, 4]$. Bellman's Principle requires that $[1, 2, 3]$ also be a shortest path from nodes 1 to 3, and that $[2, 3, 4]$ be a shortest path from nodes 2 to 4. It further requires that $[1, 2]$ be a shortest path from node 1 to node 2, $[2, 3]$ be a shortest path from node 2 to node 3, and $[3, 4]$ be a shortest path from node 3 to node 4. You should verify that this is true with the given link costs.

To see why this must be the case, assume that Bellman's Principle was violated. If the cost on link $(1, 3)$ was reduced to 2, then $[1, 2, 3]$ is no longer a shortest path from node 1 to node 3 (that path has a cost of 3, while the single-

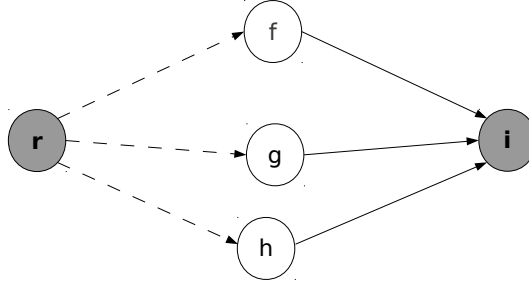


Figure 2.11: Illustration of Bellman's Principle (dashed lines indicated shortest paths between two nodes).

link path $[1, 3]$ has cost 2). Bellman's Principle then implies that $[1, 2, 3, 4]$ is no longer the shortest path between nodes 1 and 4. Why? The first part of the path can be replaced by $[1, 3]$ (the new shortest path between 1 and 3), reducing the cost of the path from 1 to 4: $[1, 3, 4]$ now has a cost of 4. In general, if a segment of a path does *not* form the shortest path between two nodes, we can replace it with the shortest path, and thus reduce the cost of the entire path. Thus, the shortest path must satisfy Bellman's Principle for all of its segments.

The implication of this is that *we can construct shortest paths one node at a time*, proceeding inductively. Let's say we want to find the shortest path from node r to a node i , and furthermore let's assume that we've already found the shortest paths from r to every node which is directly upstream of i (nodes f , g , and h in Figure 2.11). The shortest path from r to i must pass through either f , g , or h ; and according to Bellman's Principle, the shortest path from r to i must be either (a) the shortest path from r to f , plus link (f, i) ; the shortest path from r to g , plus link (g, i) ; or the shortest path from r to h , plus link (h, i) . This is efficient because, rather than considering all of the possible paths from r to i , we only have to consider three, which can be easily compared. Furthermore, we can re-use the information we found when finding shortest paths to f , g , and h , and don't have to duplicate the same work when finding the shortest path to i . This idea doesn't give a complete algorithm yet — how did we find the shortest paths to f , g , and h , for instance? — but gives the flavor of the shortest path algorithms presented next.

Bellman's Principle also gives us a compact way of expressing all of the shortest paths from an origin to every other node in the network: for each node, simply indicate the *last* node in the shortest path from that origin. This is called the backnode vector \mathbf{q}^r , where each component q_i^r is the node immediately preceding i in the shortest path from r to i . If $i = r$, then q_i^r is not well-defined (i is the origin itself; what is the shortest path from the origin to itself, and if we can define it, what node immediately precedes the origin?) so we say $q_r^r \equiv -1$ by definition. For the network in Figure 2.10 (with the original costs),

we thus have $q_1^1 = -1$, $q_2^1 = 1$, $q_3^1 = 2$, and $q_4^1 = 3$, or, in vector notation, $\mathbf{q}^1 = [-1 \ 1 \ 2 \ 3]$.

The backnode vector can be used as follows: say we want to look up the shortest path from node 1 to node 4. Starting at the destination, the backnode of 4 is 3, which means “the shortest path to node 4 is the shortest path to node 3, plus the link (3,4).” To find the shortest path to node 3, consult its backnode: “the shortest path to node 3 is the shortest path to node 2, plus the link (2,3).” For the shortest path to node 2, its backnode says: “the shortest path to node 2 is from node 1.” This is the origin, so we’ve found the start of the path, and can reconstruct the original path to node 4: [1, 2, 3, 4]. More briefly, we can use the backnodes to trace the shortest path back to an origin, by starting from the destination, and reading back one node at a time.

We will also define L_i^r to be the total cost on the shortest path from origin r to node i (the letter L is used because these values are often referred to as node labels), with $L_r^r \equiv 0$, so in this example we would have $L_1^1 = 0$, $L_2^1 = 2$, $L_3^1 = 3$, and $L_4^1 = 5$.

This chapter presents four shortest path algorithms. The first, in Section 2.4.1, only applies when the network is acyclic but is extremely fast and simple. Sections 2.4.2 and 2.4.3 next present the two most common approaches for solving shortest paths in networks with cycles: *label setting* and *label correcting*. Dijkstra’s algorithm and the Bellman-Ford algorithm are the quintessential label setting and label correcting algorithms, respectively, and they are used to show the difference between these approaches. These three algorithms actually find the shortest path from the origin r to *every* other node, not just the destination s ; this exploits Bellman’s Principle, because the shortest path from r to s must also contain the shortest path from r to every node in that path. Further, both of them contain a set of labels L_i^r associated with each node, representing the shortest path from r to node i . With a label setting approach, each node’s label is determined once and only once. On the other hand, with a label correcting approach, each node’s label can be updated multiple times.

The fourth algorithm, A^* is presented in Section 2.4.4. This algorithm can be faster than Dijkstra’s algorithm if we are only interested in the shortest path from a single origin to a single destination.

2.4.1 Shortest paths on acyclic networks

Recall from Section 2.2 that an *acyclic* network is one in which no cyclic paths exist; it is impossible to visit any node more than once on any path. Conversely, a *cyclic* network is one where a cycle does exist, and where (in theory) one could repeatedly drive around in a circle forever. Transportation networks are usually cyclic, and in fact any network where you can reach every node from every other node must be cyclic. (Why?) However, we’ll take a short diversion into acyclic networks for the purposes of finding shortest paths, for two reasons: (1) finding the shortest path on an acyclic network is much simpler and faster, and makes for a good first illustration; and (2) more advanced solution methods take advantage of the fact that people do *not* use cyclic paths (because of the shortest

path assumption), so we can only look at an acyclic portion of the network and thereby use the much faster shortest path algorithm for acyclic networks.

Once we have a topological order on a network, it becomes very easy to find the shortest path from any node r to any other node s (clearly r has lower topological order than s), using the following algorithm which is based directly on Bellman's Principle:

1. Initialize by setting $L_i^r = \infty$ and $q_i^r = -1$ for all nodes $i \in N$ to indicate that we have not found any shortest paths yet. Then set $L_r^r = 0$, because the distance from the origin to itself is zero.
2. Let i be the node topologically following r .
3. We can find the shortest path from r to i using Bellman's Principle, using these formulas:

$$L_i^r = \min_{(h,i) \in \Gamma^{-1}(i)} \{L_h^r + c_{hi}\} \quad (2.1)$$

$$q_i^r \in \arg \min_{(h,i) \in \Gamma^{-1}(i)} \{L_h^r + c_{hi}\} \quad (2.2)$$

(Essentially, these formulas have us search all of the links arriving at node i for the approach with minimum cost.)

4. Does $i = s$? If so, stop, and we have found the shortest path from r to s , which has cost L_s^r . Otherwise, let i be the next node topologically, and return to step 3.

Bellman's Principle lets us find all of the shortest paths in one pass over the nodes (in topological order), because there are no cycles which could make us loop back.

Here's how to apply this algorithm to the network in Figure 2.10, with node 1 as origin and node 4 as destination.

Step 1: Initialize: $\mathbf{L}^1 \leftarrow [0 \ \infty \ \infty \ \infty]$ and $\mathbf{q}^1 \leftarrow [-1 \ -1 \ -1 \ -1]$.

Step 2: We set $j \leftarrow 2$.

Step 3: The only approach to node 2 is (1,2), so $\Gamma^{-1}(2)$ is just $\{(1,2)\}$ and the minimization is trivial: $L_2^1 \leftarrow L_1^1 + c_{12} = 0 + 2 = 2$ and $q_2^1 \leftarrow 1$.

Step 4: $2 \neq 4$, so we advance j to the next node topologically: $j \leftarrow 3$ and return to step 3.

Step 3: There are two approaches to node 3, (1,3) and (2,3). So $L_3^1 \leftarrow \min\{L_1^1 + c_{13}, L_2^1 + c_{23}\} = \min\{0 + 4, 2 + 1\} = 3$, which corresponds to approach (2,3), so $q_3^1 \leftarrow 2$.

Step 4: $3 \neq 4$, so we advance j to the next node topologically: $j \leftarrow 4$ and return to step 3.

Step 3: There are two approaches to node 4, (2, 4) and (3, 4). So $L_4^1 \leftarrow \min\{L_2^1 + c_{24}, L_3^1 + c_{34}\} = \min\{2 + 5, 3 + 2\} = 5$, which corresponds to approach (3, 4), so $q_4^1 \leftarrow 3$.

Step 4: We have reached the destination (node 4) and terminate.

A simple induction proof shows that this algorithm must give the correct shortest paths to each node (except for those topologically before r , since such paths do not exist). By “give the correct shortest paths,” we mean that the labels \mathbf{L} give the lowest cost possible to each node when traveling from r , and that the backnodes \mathbf{q} yield shortest paths when traced back to r . Assume that the nodes are numbered in topological order. Clearly L_r^r and q_r^r are set to the correct values in the first step (the shortest path from r to itself is trivial), and are never changed again because the network proceeds in increasing topological order. Now assume that the algorithm has found correct L and q values for all nodes between r and k . In the next step, it updates labels for node $k + 1$. Let $(i, k + 1)$ be the last link in a shortest path from r to $k + 1$. By Bellman’s principle, the first part of this path must be a shortest path from r to i . Since i is topologically between r and k , by the induction hypothesis the L^r and q^r labels are correct for all nodes immediately upstream of k . Therefore, $L_i^r + c_{i,k+1} \leq L_j^r + c_{j,k+1}$ for all immediately upstream nodes j , and the algorithm makes the correct choices in step 3 for node $k + 1$.

2.4.2 Shortest paths on cyclic networks: label correcting

When there are cycles in the network, this previous approach can’t be applied, because there is no clear sequence in which to examine nodes and apply Bellman’s Principle. However, we can generalize the approach. Rather than scanning nodes in a rigid order, we can fan out from the origin, keeping in mind that we may have to scan a node multiple times in case of a cycle. To keep track of the nodes we need to examine, we define a scan eligible list SEL , a set of nodes that we still need to examine before we have found all of the shortest paths. This is a *label correcting* approach, because nodes may be scanned multiple times, and the labels updated.

1. Initialize every label L_i^r to ∞ , except for the origin, where $L_r^r \leftarrow 0$, and the backnode vector $\mathbf{q}^r \leftarrow -\mathbf{1}$.
2. Initialize the scan eligible list to contain all nodes immediately downstream of the origin, that is, $SEL \leftarrow \{i : (r, i) \in \Gamma(r)\}$.
3. Choose a node $i \in SEL$ and remove it from that list.
4. Scan node i by applying the same equations as in the previous algorithm:

$$L_i^r = \min_{(h,i) \in \Gamma^{-1}(i)} \{L_h^r + c_{hi}\} \quad (2.3)$$

$$q_i^r = \arg \min_{(h,i) \in \Gamma^{-1}(i)} \{L_h^r + c_{hi}\} \quad (2.4)$$

5. If the previous step changed the value of L_i^r , then add all nodes immediately downstream of i to the scan eligible list, that is, $SEL \leftarrow SEL \cup \{j : (i, j) \in \Gamma(i)\}$.
6. If SEL is empty, then terminate. Otherwise, return to step 3.

Repeating the same example, this algorithm works as follows:

- Step 1:** We set $\mathbf{L}^1 \leftarrow [0 \ \infty \ \infty \ \infty]$ and $\mathbf{q}^1 \leftarrow [-1 \ -1 \ -1 \ -1]$.
- Step 2:** We set $SEL \leftarrow \{2, 3\}$.
- Step 3:** Choose node 3 and remove it from SEL , so $i \leftarrow 3$, and $SEL = \{2\}$.
- Step 4:** The two possible approaches are $(1, 3)$ and $(2, 3)$, so $L_3^1 \leftarrow \min\{L_1^1 + c_{13}, L_2^1 + c_{23}\} = \min\{0 + 4, \infty + 1\} = 4$, which corresponds to approach $(1, 3)$, so $q_3^1 \leftarrow 1$. Note that, unlike in the case of the acyclic network, we are not claiming that this is the shortest path to node 3. We are simply claiming that this is the shortest path we have found thus far.
- Step 5:** The previous step reduced L_3^1 from ∞ to 4, so we add the downstream node 4 to SEL , so $SEL = \{2, 4\}$.
- Step 6:** SEL is nonempty, so we return to step 3.
- Step 3:** Choose node 4 and remove it from SEL , so $i \leftarrow 4$, and $SEL = \{2\}$.
- Step 4:** There are two approaches to node 4, $(2, 4)$ and $(3, 4)$. So $L_4^1 \leftarrow \min\{L_2^1 + c_{24}, L_3^1 + c_{34}\} = \min\{\infty + 5, 4 + 2\} = 6$, which corresponds to approach $(3, 4)$, so $q_4^1 \leftarrow 3$.
- Step 5:** The previous step reduced L_4^1 from ∞ to 6, but there are no downstream nodes to add, so $SEL = \{2\}$.
- Step 6:** SEL is nonempty, so we return to step 3.
- Step 3:** Choose node 2 and remove it from SEL , so $i \leftarrow 2$, and $SEL = \emptyset$.
- Step 4:** The only approach to node 2 is $(1, 2)$, so $\{(h, i) \in \Gamma^{-1}(i)\}$ is just $\{(1, 2)\}$ and the minimization is trivial: $L_2^1 \leftarrow L_1^1 + c_{12} = 0 + 2 = 2$ and $q_2^1 \leftarrow 1$.
- Step 5:** The previous step reduced L_2^1 from ∞ to 2, so we add the downstream nodes 3 and 4 to SEL , so $SEL = \{3, 4\}$.
- Step 6:** SEL is nonempty, so we return to step 3.
- Step 3:** Choose node 3 and remove it from SEL , so $i \leftarrow 3$, and $SEL = \{4\}$. Note that this is the *second* time we are scanning node 3. We have to do this, because we found a path to node 2, and there is a possibility that this could lead to a shorter path to node 3 as well.

Step 4: The two possible approaches are (1, 3) and (2, 3), so $L_3^1 \leftarrow \min\{L_1^1 + c_{13}, L_2^1 + c_{23}\} = \min\{0 + 4, 2 + 1\} = 3$, which corresponds to approach (2,3), so $q_3^1 \leftarrow 2$. Note that we have changed the labels again, showing that we have indeed found a better path through node 2.

Step 5: The previous step reduced L_3^1 from 4 to 3, so we add the downstream node 4 to SEL , so $SEL = \{4\}$.

Step 6: SEL is nonempty, so we return to step 3.

Step 3: Choose node 4 and remove it from SEL , so $i \leftarrow 4$, and $SEL = \emptyset$.

Step 4: There are two approaches to node 4, (2, 4) and (3, 4). So $L_4^1 \leftarrow \min\{L_2^1 + c_{24}, L_3^1 + c_{34}\} = \min\{2 + 5, 3 + 2\} = 5$, which corresponds to approach (3, 4), so $q_4^1 \leftarrow 3$.

Step 5: The previous step reduced L_4^1 from 6 to 5, but there are no downstream nodes to add, so SEL remains empty.

Step 6: SEL is empty, so we terminate.

As you can see, this method required more steps than the algorithm for acyclic networks (because there is a possibility of revisiting nodes), but it does not rely on having a topological order and can work in any network. One can show that the algorithm converges no matter how you choose the node from SEL , but it is easier to prove if you choose a systematic rule. If you operate SEL in a “first-in, first-out” manner, always choosing a node which has been in the queue for the greatest number of iterations, it is possible to show that after m iterations, you have certainly found the shortest paths from r which consist of only a single link. (You’ve probably found quite a few more shortest paths, but even in the worst case you’ll have found at least these.) After $2m$ iterations, you will have certainly found the shortest paths from r which consist of one or two links only, and so on. So, after mn iterations, you will have found all of the shortest paths, since a shortest path cannot use more than n links. The exercises ask you to fill in the details of this proof sketch.

2.4.3 Shortest paths on general networks: label setting

In the label-correcting method, we accounted for cycles by allowing nodes to be scanned more than once, relieving us of having to scan nodes in the “right” order when there is no topological order to guide us. A label-setting method works in cyclic networks, yet only has to scan each node once. This requires a bit more care to scan nodes in the proper order, since there is no opportunity to revisit nodes to correct their labels later on. So, while label-setting methods only require one scan per node, determining which node to scan requires more effort at each iteration.

The quintessential label setting algorithm was developed by Edsger Dijkstra. Dijkstra’s algorithm finds the shortest path from the origin r to every other node,

when every link has a nonnegative cost. Furthermore, at each step it finds the shortest path to at least one additional node. It uses the concept of *finalized* nodes, that is, nodes to which the shortest path has already been found. It uses the same \mathbf{L} and \mathbf{q} labels as before. The primary distinction between label setting and label correcting is that label setting requires fewer iterations than label correcting, but each iteration takes a greater amount of effort. Which one is superior depends on the network topology, the specific implementation of each algorithm, and the skill of the programmer.

Dijkstra's algorithm can be stated as follows:

1. Initialize every label L_i^r to ∞ , except for the origin, where $L_r^r \leftarrow 0$.
2. Initialize the set of finalized nodes $F = \emptyset$ and the backnode vector $\mathbf{q}^r \leftarrow -\mathbf{1}$.
3. Find an unfinalized node i (i.e., not in F) for which L_i^r is minimal.
4. Finalize node i by adding it to set F ; if all nodes are finalized ($F = N$) then terminate.
5. Update labels for links leaving node i : for each $(i, j) \in A$, update $L_j = \min\{L_j^r, L_i^r + c_{ij}\}$.
6. Update backnodes: for each $(i, j) \in A$, if L_j^r was reduced in the previous step set $q_j^r \leftarrow i$.
7. If all nodes are finalized, stop. Otherwise, return to step 3

This algorithm is considered *label setting*, because once a node's label is updated, the node is finalized and never revisited again. We consider how this algorithm can be applied to the same example used for the other algorithm demonstrations.

Initialization. We initialize $\mathbf{L}^1 = [0 \ \infty \ \infty \ \infty]$, $F = \emptyset$, and $\mathbf{q}^1 = [-1 \ -1 \ -1 \ -1]$.

Iteration 1. The unfinalized node with least L value is the origin, so we set $i = 1$ and finalize it: $F = \{1\}$. We update the downstream labels: $L_2^1 = \min\{\infty, 0 + 2\} = 2$ and $L_3^1 = 4$. Both labels were reduced, so we update the backnodes: $q_2^1 = q_3^1 = 1$.

Iteration 2. Of the unfinalized nodes, $i = 2$ has the lowest label, so we finalize the node: $F = \{1, 2\}$. We update the downstream labels: $L_3^1 = \min\{4, 2 + 1\} = 3$ and $L_4^1 = 7$. Both labels were reduced, so we update the backnodes: $q_3^1 = q_4^1 = 2$.

Iteration 3. Of the unfinalized nodes, $i = 3$ has the lowest label, so we finalize it: $F = \{1, 2, 3\}$. We update the downstream label $L_4^1 = \min\{7, 3 + 2\} = 5$ and backnode $q_4^1 = 3$.

Iteration 4. Node 4 is the only unfinalized node, so $i = 4$. We finalize it, and since $F = N$ we are done.

The correctness of Dijkstra’s algorithm can be proved by an induction argument, on the size of the set of finalized nodes $|F|$. Specifically, we show that at every step of the algorithm the backnodes to every finalized node represent a shortest path. The algorithm always finalizes the origin r first, so when $|F| = 1$ the only finalized node is the origin, and the shortest path to this node is trivial.

Now assume that when $|F| = k$, the algorithm has correctly found shortest paths to every finalized node, and consider what happens when the $k + 1$ -th node is added to this set. This occurs in step 4 of the algorithm. Call this newly finalized node j , and say that at this point the backnode is $q_j^r = i$. By contradiction, assume that there is a shorter path from r to j than the one through node i . Notice that the entries of the backnode vector which are not -1 all point to finalized nodes, because step 6 is the only step which changes this vector, and only changes entries of this vector to a newly finalized node. Furthermore, from the induction hypothesis, the backnode vectors from all of these previously-finalized nodes describe shortest paths.

So, if there is a shorter path to j than the one through i , it must differ in the last link, that is, this shorter path must end with a link (h, j) where $h \neq i$. Either h is a finalized node, or it is not. If h were finalized, then by the induction hypothesis L_h^r is the correct shortest path cost to r , and since it leads to a shorter path to j we must have $L_h^r + c_{hj} < L_i^r + c_{ij}$. But if this were the case, then steps 5 and 6 would have set q_j^r to h when node h was finalized, so q_j^r could not equal i , a contradiction. Or, if h were not finalized, then $L_h^r > L_j^r$, since in step 3 we always finalize a node with the smallest L value. Since all links have nonnegative costs, we cannot have $L_h^r + c_{hj} < L_j^r$, which is what we need if h led to a shorter path to j . Since both cases lead to contradictions, we conclude that a correct shortest path to node j has been found, and by induction, this argument extends to every node.

2.4.4 Shortest paths from one origin to one destination

The previous sections gave algorithms to find the shortest path from one origin to all destinations. Slight modifications to these algorithms can find the shortest path from all origins to one destination (see Exercise 47). As discussed above, Bellman’s principle lets us re-use information from one shortest path when finding another, and as a result even finding a shortest path from one origin to one destination can provide many other shortest paths “for free.” In traffic assignment, where we are modeling the flows of many drivers, it often makes sense to use an algorithm that finds many shortest paths simultaneously.

However, there are times when we are only concerned with a single origin and destination, and do not care about the “free” shortest paths to other destinations we get. Examples are in-vehicle routing systems, or if the number of origin-destination pairs with positive demand is small compared to the overall size of the matrix (i.e., the OD matrix is sparse) so there are relatively few paths we need to find. In such cases we can use a more focused algorithm to find a single shortest path in less time than it takes to find shortest paths to all destinations.

The A^* algorithm is a simple modification to label-setting, which in some

cases can dramatically reduce the running time, in exchange for limiting the scope to one origin r and one destination s . This algorithm requires an additional value g_i^s for each node, representing an estimate of the cost of the shortest path from i to s . This estimate (often called the “heuristic”) should be a *lower bound* on the actual cost of the shortest path from i to s . Some examples of how these estimates are chosen are discussed below.

Once the estimates g_i^s are chosen, the label-setting algorithm from Section 2.4.3 proceeds as before with a small modification in Step 3: rather than choosing a node i in E which minimizes L_i^r , we choose a node which minimizes $L_i^r + g_i^s$. Everything else proceeds exactly as before. The intuition is that we are taking into account our estimates of which nodes are closer to the destination. The algorithm in Section 2.4.3 fans out in all directions from the origin (by simply looking at L_i^r), rather than directing the search towards a particular destination.

It can be shown that this algorithm will always yield the correct shortest path from r to s as long as the g_i^s are lower bounds on actual shortest path costs from i to s . If this is not the case, A^* is not guaranteed to find the shortest path from r to s . Some care must be taken in how these estimates are found. Two extreme examples are:

- Choose $g_i^s = 0$ for all i . This is certainly a valid lower bound on the shortest path costs (recall that label-setting methods assume nonnegative link costs), so A^* will find the shortest path from r to s . However, zero is a very poor estimate of the actual shortest path costs. With this choice of g_i^s , A^* will run exactly the same as Dijkstra’s algorithm, and there is no time savings.
- Choose g_i^s to be the *actual* shortest path cost from i to s . This is the tightest possible “lower bound,” and will make A^* run extremely quickly — in fact, it will only scan nodes along the shortest path, the best possible performance that can be achieved. However, coming up with these estimates is just as hard as solving the original problem! So in the end we aren’t saving any effort; what we gain from A^* is more than lost by the extra effort we need to compute g_i^s in the first place.

So, there is a tradeoff between choosing tight bounds (the closer g_i^s to the true costs, the faster A^* will be) while not spending too long in computing the estimates (which might swamp the savings in A^* itself). Luckily, in transportation networks, there are several good bounds available which can be computed fairly quickly. For instance:

- The Euclidean (“as the crow flies”) distance between i and s , divided by the fastest travel speed in the network, is a lower bound on the travel time between i and s .
- Replace every link with a lower bound on its cost (say, free-flow travel time) and find shortest paths between all nodes and all destinations (repeatedly using one of the previous algorithms from this chapter). This

takes more time, but only needs to be done once and can be done as a preprocessing step. As we will see in Chapter 6, solving traffic assignment requires many shortest path computations. The extra time spent finding these costs once might result in total time savings over many iterations.

You may find it instructive to think about other ways we can estimate g_i^s values, and how they might be used in transportation settings.

2.5 Historical Notes and Further Reading

(These sections are incomplete in this beta version of the text, and will be substantially expanded in the complete first edition.)

Networks arise frequently in transportation and optimization, particularly network flow problems such as shortest path, maximum flow, and minimum cost flow. The text by (Ahuja et al., 1993) discusses a number of ways to formulate and solve such problems, along with a breadth of applications. This text also provides more detail on data structures used for working with networks in computer programs. Bellman’s principle was stated in Bellman (1957).

The first label-correcting shortest path algorithm was developed by Ford (1956). Subsequent experience has shown that the practical performance of this method depends significantly on the order in which nodes are retrieved from the scan list; good choices are “first-in, first-out,” (FIFO) in which the node selected from *SEL* is always one of the *oldest* there, and a double-ended queue implementation which provides an exception to FIFO in that when nodes are return to the scan list after being scanned earlier they are moved to the front of the queue. These choices were first described in Bellman (1958) and Pape (1974).

The quintessential label-setting shortest path method is that of Dijkstra (1959). The A^* method is essentially an extension to Dijkstra’s algorithm, and was developed by Hart et al. (1968). For more thorough reviews of shortest path algorithms, the reader is referred to the survey paper by Deo and Pang (1984), and to Ahuja et al. (1993).

There are many variants of the shortest path program, for instance, adding side constraints on the amount of some other resource consumed (such as money spent on tolls or battery charge from an electric vehicle). Ziegelmann (2001) provides a review of approaches for this problem.

Another common variant is aimed at addressing reliability and uncertainty in link costs, by having the link costs be drawn from some probability distribution rather than known exactly in advance. If one aims to minimize the expected travel time, it is enough to replace each link cost with its expected value. However, travelers may also care about reliability specifically. This can be modeled by adding variance or standard deviation to the objective function (Xing and Zhou, 2011; Shahabi et al., 2013; Khani and Boyles, 2015; Zhang et al., 2016), adding a constraint on variance (Sivakumar and Batta, 1994), adopting a robust optimization perspective (Yu and Yang, 1998; Montemanni and Gambardella,

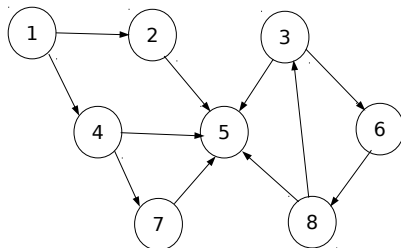


Figure 2.12: Network for Exercises 1–3

2004; Shahabi et al., 2015), applying multiobjective optimization (Sen et al., 2001), introducing a nonlinear utility function to represent arrival time preferences (Loui, 1983; Eiger et al., 1985; Murthy and Sarkar, 1996; Boyles, 2006; Gao, 2005), or changing the objective entirely to maximize the probability of on-time arrival (Fan et al., 2005; Nie and Wu, 2009).

Yet another way to address reliability and real-time information provision is to allow the traveler's path to change *en route* based on information learned while traveling; this leads to a class of *online* shortest path formulations. For examples of such formulations, see Andreatta and Romeo (1988), Psaraftis and Tsitsiklis (1993), Polychronopoulos and Tsitsiklis (1996), Miller-Hooks (2001), Waller and Ziliaskopoulos (2002), Provan (2003), Gao (2005), Boyles (2006), Boyles and Rambha (2016), and Boyles (2009).

2.6 Exercises

1. [9] In the network in Figure 2.12, list the indegree, outdegree, degree, forward star, and reverse star of each node.
2. [3] In the network in Figure 2.12, list all of the paths between nodes 1 and 5.
3. [4] State whether the network in Figure 2.12 is or is not (a) cyclic; (b) a tree; (c) connected; (d) strongly connected.
4. [15] For each of the following, either draw a network with the stated properties, or explain why no such network can exist: (a) connected, but not strongly connected; (b) strongly connected, but not connected; (c) cyclic, but not strongly connected.
5. [25] If m and n are the number of links and nodes in a network, show that $m < n^2$.
6. [25] If a network is connected, show that $m \geq n - 1$.
7. [25] If a network is strongly connected, show that $m \geq n$.

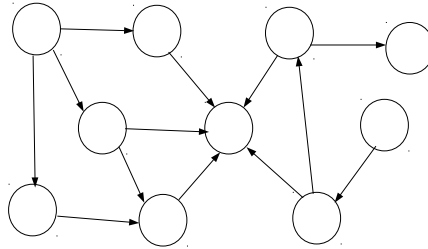


Figure 2.13: Network for Exercise 11

8. [10] Show that $\sum_{i \in N} |\Gamma(i)| = \sum_{i \in N} |\Gamma^{-1}(i)| = m$.
9. [1] Why must transportation infrastructure networks be cyclic?
10. [15] Although we do not usually expect drivers to use cyclic paths, there are some exceptions. Name one.
11. [15] Find a topological order for the network in Figure 2.13.
12. [11] Is the topological order for an acyclic network unique? Either explain why it is, or provide a counterexample showing it is not.
13. [42] Show that any acyclic network has at least one node with no incoming links. This was the “missing step” in the proof of Theorem 2.1. (Since this is part of the proof, you can’t use the result of this theorem in your answer.)
14. [36] Let i and j be any two nodes in an acyclic network. Give a procedure for calculating the number of paths between i and j which involves at most one calculation per node in the network.
15. [57] Give a procedure for determining whether a given network is strongly connected. Try to make your method require as few steps as possible.
16. [58] Give a procedure for determining whether a given network is connected. Try to make your method require as few steps as possible.
17. [30] Show that there is at most one path between *any* two nodes in a tree.
18. [30] Show that any node in a tree (excluding its root) has exactly one link in its reverse star (and thus one parent).
19. [30] Show that any tree is an acyclic network.
20. [30] Show that removing any link from a tree produces a disconnected network.
21. [50] Show that any tree has at least two nodes of degree one.

22. [55] Consider a rectangular grid network of one-way streets, with r rows of nodes and c columns of nodes. All links are directed northbound and eastbound. How many paths exist between the lower-left node (southwest) and the upper-right (northeast) node?
23. [14] Write down the node-node adjacency matrix of the network in Figure 2.12.
24. [16] Consider the network defined by this node-node adjacency matrix:

$$\begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

- (a) Draw the network.
- (b) How many links does the network have?
- (c) How many links enter node 1? How many links leave node 1?
25. [17] Is the network represented by the following node-node adjacency matrix strongly connected?

$$\begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

26. [10] If the nodes in an acyclic network are numbered in a topological order, show that the node-node adjacency matrix is upper triangular.
27. [37] Let \mathbf{A} be the node-node adjacency matrix for a network. What is the interpretation of the matrix product \mathbf{A}^2 ?
28. [42] Let \mathbf{A} be the node-node adjacency matrix for an acyclic network. First show that $\sum_{n=1}^{\infty} \mathbf{A}^n$ exists, and give an interpretation of this sum.
29. [65] A *unimodular* matrix is a square matrix whose elements are integers and whose determinant is either $+1$ or -1 . A matrix is *totally unimodular* if every nonsingular square submatrix is unimodular. (Note that a totally unimodular matrix need not be square). Show that every node-link incidence matrix is totally unimodular.

30. [10] One disadvantage of the forward star representation is that it is time-consuming to identify the reverse star of a node — one must search through the entire array to find every link with a particular head node. Describe a “reverse star” data structure using arrays, where the reverse star can be easily identified.
31. [52] By combining the forward star representation from the text and the reverse star representation from the previous exercise, we can quickly identify both the forward and reverse stars of every node. However, a naive implementation will have two different sets of arrays, one sorted according to the forward star representation, and the other sorted according to the reverse star representation. This duplication wastes space, especially if there are many attributes associated with each link (travel time, cost, capacity, etc.) Identify a way to easily identify the forward and reverse stars of every node, with only *one* set of arrays of link data, by adding an appropriate attribute to each link.
32. [58] In the language of your choice, write computer code to do the following:
 - (a) Produce the node-node adjacency matrix of a network when given the node-link incidence matrix.
 - (b) Produce the node-link incidence matrix of a network when given the node-node adjacency matrix.
 - (c) Produce the node-node adjacency matrix of a network when given the forward star representation of the network.
 - (d) Produce the forward star representation of a network when given the node-node adjacency matrix.
33. [13] After solving a shortest path problem from node 3 to every other node, I obtain the backnode vector shown in Table 2.3. Write the shortest paths (a) from node 3 to node 5; (b) from node 3 to node 7; (c) from node 4 to node 8.
34. [26] Find the shortest path from node 1 to every other node in the network shown in Figure 2.14. Report the final labels and backnodes (L and q values) for all nodes.
35. [25] The network in Figure 2.15 has a link with a negative cost. Show that the label-correcting algorithm still produces the correct shortest paths in this network, while the label-setting algorithm does not.
36. [57] Prove or disprove the following statement: “Any network with negative costs can be transformed into a network with nonnegative costs by adding a large enough constant to every link’s cost. We can then use the label-setting algorithm on this new network. Therefore, the label-setting algorithm can find the shortest paths on any network, even if it

Table 2.3: Backnode vector for Exercise 33.

Node	Backnode
1	4
2	6
3	-1
4	7
5	4
6	5
7	3
8	10
9	5
10	6

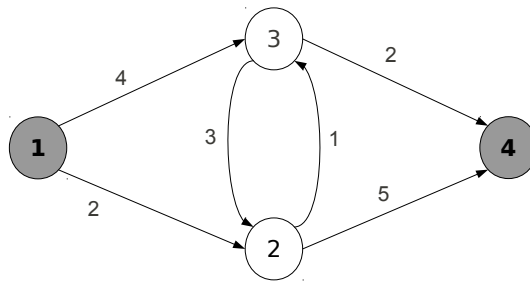


Figure 2.14: Network for Exercises 34 and 41.

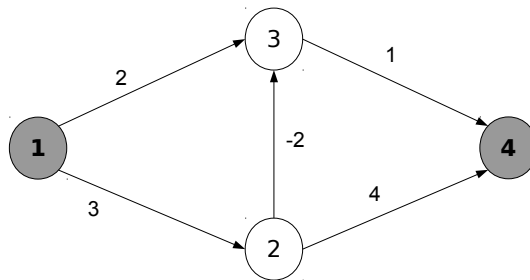


Figure 2.15: Network for Exercise 35.

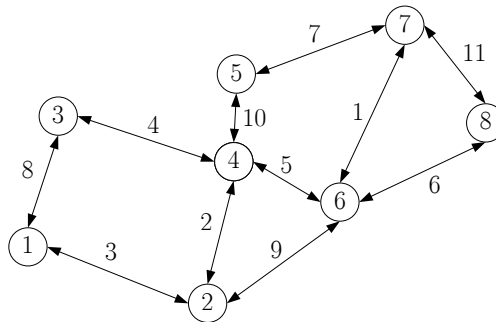


Figure 2.16: Network for Exercise 37, link labels are costs.

has negative-cost links.” (Proving this statement means showing that it is true in any network; to disprove it, it is sufficient to find a single counterexample.)

37. [37]. Find the shortest paths on the network in Figure 2.16, using both the label-correcting and label-setting algorithms. Sketch the *shortest path tree* (that is, the network with only the links implied by the backnode vector) produced by each algorithm. Which algorithm required fewer iterations?
38. [33]. You and your friends are camping at Yellowstone Park, when you suddenly realize you have to be at the Denver airport in ten hours to catch a flight. You don’t have Internet access, but you do have an atlas showing the distance and travel time between selected cities (Figure 2.17). Assuming these travel times are accurate, can you make it to Denver in time for your flight?
39. [43]. Consider the following variation of the shortest path problem: instead of the costs c_{ij} being fixed, instead assume that they are random, and the cost of link (i, j) takes the values $\{c_{ij}^1, c_{ij}^2, \dots, c_{ij}^s\}$ with probabilities $\{p_{ij}^1, p_{ij}^2, \dots, p_{ij}^s\}$ independent of the cost of any other link. How can you adapt the shortest path algorithms in this chapter to find the path with the least *expected* cost?
40. [53]. Assume that each link (i, j) in the network fails with probability p_{ij} , and that link failures are independent. (Failure can represent some kind of damage or disruption, the probability of being detected in a military routing problem, etc.) A path fails if any link in that path fails. Explain how you can find the most reliable path (that is, the one with the least failure probability) with a shortest path algorithm.
41. [23] Instead of trying to find the *shortest* path between two nodes, let’s try to find the *longest* path between two nodes. (As we will see later,

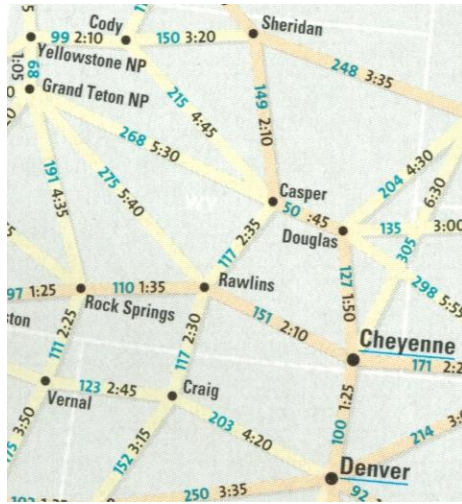


Figure 2.17: Atlas page for Exercise 38.

there are actually cases when this is useful.)

- (a) Modify the algorithm presented in Section 2.4.1 to find the longest path in an acyclic network.
 - (b) If you modify the label-correcting or label-setting algorithms for general networks in a similar way, will they find the longest paths successfully? Try them on the network in Figure 2.14.
42. [25]. As a modification to the node-node adjacency matrix you might use to represent the network in a computer program (cf. Section 2.3), you could store the cost of the links in this matrix, with ∞ where no link exists (as opposed to '1' where links exist and '0' where they do not). Find the shortest path between nodes 7 and 4 using the modified adjacency matrix below, where the entry in row i and column j is the cost c_{ij} if this link exists, and ∞ if it does not.

$$\begin{bmatrix} \infty & \infty & \infty & 5 & 3 & \infty & \infty \\ 4 & \infty & 7 & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & 6 & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & 2 & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty & 4 & \infty \\ 3 & 2 & \infty & \infty & \infty & \infty & \infty \\ \infty & 1 & \infty & \infty & \infty & 4 & \infty \end{bmatrix}$$

43. [22]. In the game “Six Degrees of Kevin Bacon,” players are given the name of an actor or actress, and try to connect them to Kevin Bacon in as few steps as possible, winning if they can make the connection in six

steps or less. Two actors or actresses are “connected” if they were in the same film together. For example, Alfred Hitchcock is connected to Kevin Bacon in three steps: Hitchcock was in *Show Business at War* with Orson Welles, who was in *A Safe Place* with Jack Nicholson, who was in *A Few Good Men* with Kevin Bacon. Beyoncé Knowles is connected to Kevin Bacon in two steps, since she was in *Austin Powers: Goldmember*, where Tom Cruise had a cameo, and Cruise was in *A Few Good Men* with Bacon. Assuming that you have total, encyclopedic knowledge of celebrities and films, show how you can solve “Six Degrees of Kevin Bacon” as a shortest path problem. Specify what nodes, links, costs, origins, and destinations represent in the network you construct.

44. [14] In a network with no negative-cost cycles, show that $-nC$ is a lower bound on the shortest path cost between any two nodes in a network.
45. [57] Show that if the label-correcting algorithm is performed, and that at each iteration you choose a node in *SEL* which has been in the list the longest, at the end of kn iterations the cost and backnode labels correctly reflect all shortest paths from r which are no more than k links long. (Hint: try an induction proof.)
46. [44] Assume that the label-correcting algorithm is terminated once the label for some node i falls below $-mC$. Show that the following the current backnode labels from i will lead to a negative-cost cycle.
47. [32] Modify three of the shortest path algorithms in this chapter so that they find shortest paths from all origins to one destination, rather than one origin to all destinations:
 - (a) The acyclic shortest path algorithm from Section 2.4.1.
 - (b) The label-correcting algorithm from Section 2.4.2.
 - (c) The label-setting algorithm from Section 2.4.3.
48. [73] It is known that the label correcting algorithm will find the correct shortest paths as long as the initial labels \mathbf{L} correspond to the distance of *some* path from the origin (they do not necessarily need to be initialized to $+\infty$). Assume that we are given a vector of backnode labels \mathbf{q} which represents some tree (not necessarily the shortest paths) rooted at the origin r . Develop a one-to-all shortest path algorithm that uses this vector to run more efficiently. In particular, if the given backnode labels \mathbf{q} *do* correspond to a shortest path tree, your algorithm should recognize this fact and terminate in a number of steps linear in the number of network links.

Chapter 3

Mathematical Techniques for Equilibrium

This chapter provides a survey of mathematical techniques used in network analysis. There is no attempt to be comprehensive; many indeed, books have been written about each of the sections in this chapter. Rather, the intent is to cover topics which are used frequently in transportation network problems.

Several of the appendices may be useful at this point. Appendix A reviews definitions and facts related to vectors, matrices, sets, and functions which are needed, including the topics of convex functions, convex sets, and multivariable calculus (the gradient vector, and the Jacobian and Hessian matrices will play a particularly important role). If these topics are new to you, you may wish to consult more extended treatments of these topics in other books or references. Appendix B introduces basic concepts of optimization, focusing on what will ultimately be relevant for static and dynamic traffic assignment. Optimization is a much richer and deeper field than what is used in traffic assignment, and Appendices C and D go into further detail here — the material from these latter two appendices is not strictly necessary for this book, but we believe it to be of interest to many readers nevertheless.

This chapter focuses on mathematical material which is both specialized to the traffic assignment problem, and which is likely to be new to anticipated readers of the book. This will involve discussion of three main techniques: the *fixed point problem*, in Section 3.1; the *variational inequality*, in Section 3.2, and *convex optimization*, in Section 3.3. Chapter 1 characterized equilibrium as a “consistent” state in which no driver can improve his or her satisfaction by unilaterally changing routes. Under the assumption of continuous flow variables, we can use calculus to greatly simplify the problem. All three of the mathematical techniques discussed in this chapter formalizes this equilibrium principle in different ways.

Each approach has its own advantages from the standpoint of this book. The fixed point formulation is perhaps the most intuitive and generally-applicable

definition, but does not give much indication as to how one might actually find this equilibrium. The variational inequality formulation lends itself to physical intuition and can also accommodate a number of variations on the equilibrium problem. The convex optimization approach provides an intuitive interpretation of solution methods, provides an elegant proof of equilibrium uniqueness in link flows, and powers the best-known solution algorithms, but the connection between the equilibrium concept and optimization requires more mathematical explanation and is less obvious at first glance.

3.1 Fixed-Point Problems

A **fixed point** of a function f is a value x such that $f(x) = x$, that is, the value x is unchanged by f . As an example, the function $f_1(x) = 2x - 1$ has only one fixed point at $x = 1$, because $f_1(1) = 2 \times 1 - 1 = 1$, the function $f_2(x) = x^2$ has two fixed points at 0 and 1 ($0^2 = 0$ and $1^2 = 1$), while the function $f_3(x) = x^2 + 1$ has no fixed points at all. A helpful visual illustration is that the fixed points of a function are the points of intersection between the function's graph and the 45-degree line $y = x$ (Figure 3.1).

Equilibrium solutions can often be formulated as fixed points of a suitable function. For instance, in the traffic assignment problem, a route choice model and a congestion model are mutually interdependent: drivers choose routes to avoid congestion, but congestion is determined by the routes drivers choose. (Figure 1.3). An equilibrium solution is consistent in the sense that drivers are satisfied with the travel times calculated by the paths they chose. Feeding the route choices into the congestion model, then feeding the resulting travel times into the route choice model, one can obtain the original route choices back. This is reminiscent of fixed point problems: when you evaluate a function at its fixed-point, after performing whatever calculations the function requires you obtain the fixed-point again. Fixed points thus arise naturally when dealing with these kinds of “circular” dependencies.

As a first example, consider the problem of trying to estimate the number of bus riders in a dense urban area. The bus system is subject to congestion; when there are x riders, the average delay to customers is given by the function $t = T(x)$ which is assumed continuous and increasing. (Assuming that the fleet of buses and timetables are fixed, more riders mean longer boarding and offloading times, more frequent stops, and the possibility of full buses skipping waiting passengers.) However, the number of bus riders depends on the congestion in the system — as the buses become more crowded, some riders may switch to alternate modes of transportation or combine trips, so we can write $x = X(t)$ for some function X which is continuous and decreasing. For a concrete example, assume that the system is designed such that $T(x) = 10 + x$, and that the ridership function is given by $X(t) = [20 - t]^+$, when x and t are measured in appropriate units (say, thousands of passengers and minutes).¹

¹The notation $[\cdot]^+$ is used to mean the *positive part* of the term in brackets, that is, $[y]^+ = \max\{y, 0\}$. If the term in brackets is negative, it is replaced by zero; otherwise it is

The goal is to find the ridership x ; but $x = X(t)$ and $t = T(x)$, which means that we need to find some value of x such that $x = X(T(x))$. This is a fixed point problem! Here the function f is the composition of X and T :

$$\begin{aligned} x &= X(T(x)) \\ &= [20 - T(x)]^+ \\ &= [20 - (10 + x)]^+ \\ &= [10 - x]^+ \end{aligned}$$

Assuming that $10 - x$ is nonnegative, we can replace the right-hand side of the last equation by $10 - x$. Solving the resulting equation $x = 10 - x$, we obtain $x = 5$ as the fixed point. Checking, when there are 5 riders the average travel time will be 15 minutes (based on T); and when the travel time is 15 minutes, there will indeed be 5 riders (based on X). Finally, if $x = 5$, our assumption that $10 - x$ was nonnegative was true so this solution is valid.

For this example, it is relatively easy to find the fixed point by substituting the definitions of the functions and performing some algebra. However, in more complex problems it will be difficult or impossible to calculate the fixed point directly. Despite this, fixed points are important because of so-called “fixed-point theorems” which guarantee the existence of a fixed point under certain conditions of the function f . A fixed point theorem by Brouwer is provided below, and another by Kakutani is provided in Section 5.1.1. These fixed point theorems are *non-constructive* because they give us no indication of how to find a fixed point, they simply guarantee that at least one exists. Brouwer’s theorem can be stated as:

Theorem 3.1. (Brouwer). *Let f be a continuous function from the set K to itself, where K is convex and compact. Then there is at least one point $x \in K$ such that $f(x) = x$.*

The exercises ask you to show that each of the conditions in Brouwer’s theorem is necessary; you might find it helpful to visualize these conditions geometrically similar to Figure 3.1.

Notice the stipulation that f be a function “from the set K to itself;” this means that the range of its function must be contained in its domain. Intuitively, this means that any “output” of the function f must also be a valid “input” to that same function. In other words, iteration is possible: starting from any value x in its domain, you can apply the function over and over again to produce a sequence of values $x, f(x), f(f(x)), \dots$. If this condition does not hold, then Brouwer’s theorem does not guarantee anything about a fixed point.

Let us apply Brouwer’s theorem to the transit ridership example. Both X and T are continuous functions, so their composition $X \circ T$ is continuous as well. (Alternately, by substituting one function into the other we obtain $X(T(x)) = [10 - x]^+$, which is evidently continuous.) What are the domain

unchanged.

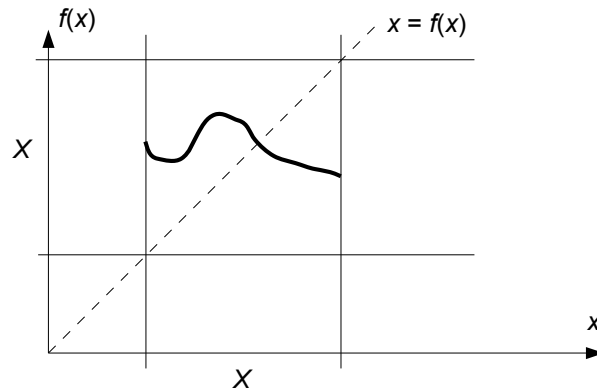


Figure 3.1: Visualizing fixed-point problems; the intersection of $f(x)$ and the 45-degree line is a point where $x = f(x)$.

and range of $X(T(x))$? Since $X(T(x))$ is the positive part of $20 - T(x)$, then $x = X(T(x)) \geq 0$. Further note that because $x \geq 0$, $T(x) \geq 10$, so $x = X(T(x)) \leq 10$. That is, we have shown that x must lie between 0 and 10, so the function $X(T(x)) = [10 - x]^+$ can be defined from the set $[0, 10]$ to itself. This set is convex and compact, so Brouwer's theorem would have told us that at least one fixed point must exist, even if we didn't know how to find it.

3.2 Variational Inequalities

Fixed point problems often lend themselves to elegant theorems like Brouwer's, which prove that a fixed point must exist. However, such problems often lack easy solution methods. The variational inequality can be more convenient to work with in this respect. Variational inequalities can be motivated with a physical analogy. Imagine an object, initially stationary, which is confined to move within some frictionless container (Figure 3.2) and cannot leave. This object is acted on by a force whose magnitude and direction can be different at each point. If the object is in the interior of the container, the object will begin to move in the same direction as the force at that point. If the object starts at the edge of the container, it may not be able to move in the same direction as the force, but it might slide along the side of the object. The problem is to determine where in the container (if anywhere) the object can be placed so that it will not move under the action of the force field.

Figure 3.3 shows some examples. In this figure, the direction of the force is drawn with black arrows, shown only at the points under consideration for clarity. At three of the points (A, B, and C) the object will move: at A in the direction of the force, and at B and C sliding along the edge of the container in the general direction of the force. At the other two points (D and E), the

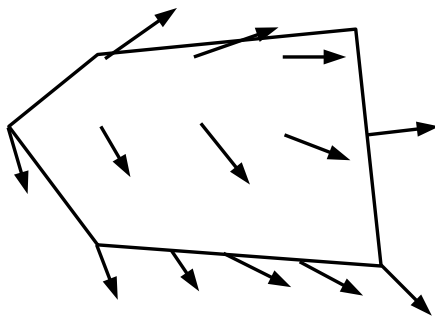


Figure 3.2: A container with a force field.

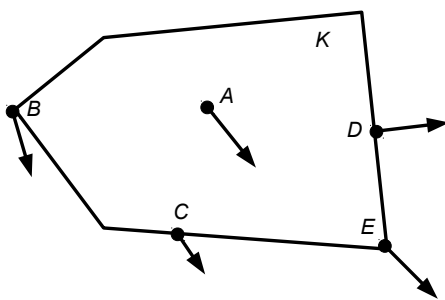


Figure 3.3: Two solutions, and three non-solutions, to a variational inequality.

object will not move under the action of the force, being effectively resisted by the container wall.

How can we think about such problems in general? A little thought should convince you that (1) if the object is on the boundary of the container, but not at a corner point, it will be unmoved if and only if the force is perpendicular to the boundary (point D in Figure 3.3), and (2) if the object is at a corner of the container, it will be unmoved if and only if the force makes a right or obtuse angle with all of the boundary directions (point E). These two cases can be combined together: a point on the boundary is an equilibrium if and only if the force makes a right or obtuse angle with all boundary directions. In fact, if the force makes such an angle with all boundary directions, it will do so with any other direction pointing into the feasible set (Figure 3.4). So, we see that *a point is unmoved by the force if and only if the direction of the force at that point makes a right or obtuse angle with any possible direction the object could move in.*

The mathematical definition of a variational inequality is little more than translating the above physical problem into algebraic terminology. The “container” is replaced by a set K of n -dimensional vectors, which for our purposes

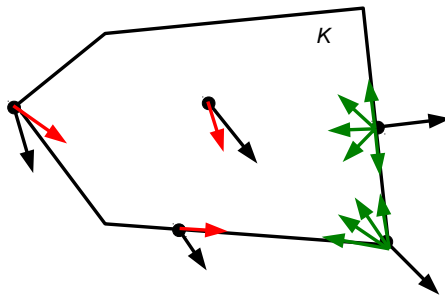


Figure 3.4: Stable points (in green) make an obtuse or right angle with all feasible directions; unstable points (in red) make an acute angle with some feasible directions.

can be assumed compact and convex (as in all of the figures so far). The “force field” is replaced by a vector-valued function $\mathbf{F} : \mathbb{R}^n \rightarrow \mathbb{R}^n$ which depends on n variables and produces an n -dimensional vector as a result. Recalling vector operations, specifically equation (A.16), saying that two vectors make a right or obtuse angle is equivalent to saying that their dot product is negative. A “solution” to the variational inequality is a point which is unmoved by the force field. (In the above example, we want to say that D and E are solutions to the variational inequality problem created by the container shape and force field, while A, B, and C are not.) Therefore, rewriting the condition in the previous paragraph with this mathematical notation, we have the following definition:

Definition 3.1. *Given a convex set $K \subseteq \mathbb{R}^n$ and a function $\mathbf{F} : K \rightarrow \mathbb{R}^n$, we say that the vector $\hat{\mathbf{x}} \in K$ solves the VI(K, \mathbf{F}) if, for all $\mathbf{x} \in K$, we have*

$$\mathbf{F}(\hat{\mathbf{x}}) \cdot (\mathbf{x} - \hat{\mathbf{x}}) \leq 0. \quad (3.1)$$

In other words, as \mathbf{x} ranges over all possible points in K , $\mathbf{x} - \hat{\mathbf{x}}$ represents all possible directions the object can move from $\hat{\mathbf{x}}$. The point $\hat{\mathbf{x}}$ solves the variational inequality precisely when the dot product of the force and all possible directions is negative.

There is a relationship between fixed point problems and variational inequalities, which can be motivated again by the physical analogy of a force acting within a container. The solutions to the variational inequality are quite literally “fixed points” in the sense that an object placed there will not move. Consider some point \mathbf{x} under the action of the force \mathbf{F} . Assume furthermore that this force is constant and does not change magnitude or direction as this point moves. Then, since K is convex, the trajectory of the object can be identified with the curve $\text{proj}_K(\mathbf{x} + \alpha\mathbf{F}(\mathbf{x}))$ where α ranges over all positive numbers and proj_K means projection onto the set K as defined in Section A.3. See Figure 3.5 for a few examples. If a point is a solution to VI(K, \mathbf{F}), then the corresponding “trajectory” will simply be the same point no matter what α is. So, for the sake

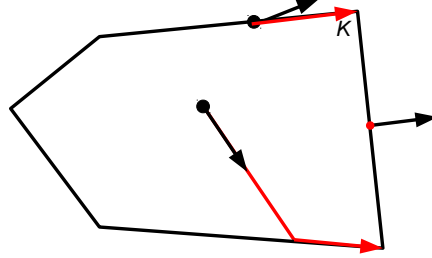


Figure 3.5: Trajectories of different points (red) assuming the force (black) is constant.

of convenience we arbitrarily choose $\alpha = 1$ and look at the location of the point $\text{proj}_K(\mathbf{x} + \mathbf{F}(\mathbf{x}))$. If this is the same as the initial point \mathbf{x} , then \mathbf{x} is a solution to the variational inequality. So, finally, if we let

$$\mathbf{f}(\mathbf{x}) = \text{proj}_K(\mathbf{x} + \mathbf{F}(\mathbf{x})), \quad (3.2)$$

then the fixed points of \mathbf{f} coincide exactly with solutions to $\text{VI}(K, \mathbf{F})$.

In many cases of practical interest, \mathbf{F} will be a continuous function. Furthermore, it can be shown that the projection mapping onto a convex set (such as K) is a well-defined (i.e., single-valued), continuous function. Then by Proposition A.6, the function \mathbf{f} defined by equation (3.2) is a continuous function. So, if the set K is compact in addition to being convex, then Brouwer's theorem shows that the variational inequality must have at least one solution:

Theorem 3.2. *If $K \subseteq \mathbb{R}^n$ is a compact, convex set and $\mathbf{F} : \mathbb{R}^n \rightarrow \mathbb{R}^n$ is a continuous function, then the variational inequality $\text{VI}(K, \mathbf{F})$ has at least one solution.*

You should convince yourself that all of these conditions are necessary: if the container K is not bounded or not closed, or if the force field \mathbf{F} is not continuous, then it is possible that an object placed at any point in the container will move under the action of the force field.

3.3 Convex Optimization

As we will see in the coming chapters, fixed points and variational inequalities are relatively intuitive ways to represent the idea of an equilibrium in transportation systems. A fixed point captures the idea that “at equilibrium, nobody can make a better choice than the one they are currently making; therefore the state of the system is the same from one day to the next.” The transit example in Section 3.1 illustrates how such a fixed point problem might arise. The physical

analogy with variational inequalities suggests how travelers might change their behavior from one day to the next; the “container” represents the set of possible travel choices, and the “force field” represents travelers’ desires to move to lower-cost options. An equilibrium solution is one where there is no way to move in such an improving direction.

Both of these methods have disadvantages. Fixed point theorems are “non-constructive,” which means they often lack methods guaranteed to find a fixed point, even if one exists. Brouwer’s theorem gives us conditions under which a fixed point exists, but tells us nothing about how to find it. Sometimes applying f repeatedly from a starting point will converge to a fixed point, but not always. The force field analogy in variational inequalities suggests a natural algorithm (pick a starting point, and see where the force carries you), but again this is not always guaranteed to work. It is also possible to have multiple solutions to a fixed point or variational inequality problem. From the standpoint of transportation planning this is inconvenient — how can you consistently rank alternatives if you have several different predictions for what might happen under each alternative?

Convex optimization is a more powerful tool in that we have uniqueness guarantees on solutions, and efficient algorithms that provably converge to an optimal point. The downside is that it is not immediately obvious how a user equilibrium problem can be formulated in terms of optimization. Chapter 5 takes up this task; this section presents what you need to know about convex optimization for this to make sense. If you have never encountered optimization problems before, please read Appendix B before proceeding further, to get familiar with the terminology and notation used in presenting optimization problems. This subsection will focus on *convex optimization*, a specific kind of optimization problem which is both easier to solve, and well-suited for solving transportation network problems. If you are interested in other applications, Appendix C discusses methods and properties of other kinds of optimization problems.

The reason to restrict attention to convex optimization is that some optimization problems are much easier to solve than others. For instance, the function in Figure 3.6 has many local minima and is unbounded below as $x \rightarrow -\infty$, both of which can cause serious problems if we’re trying to minimize this function. Usually, the best that a software program can do is find a local minimum. If it finds one of the local minima for this function, it may not know if there is a better one somewhere else (or if there is, how to find it). Or if it starts seeking x values which are negative, we could run into the unbounded part of this function.

On the other hand, some functions are very easy to minimize. The function in Figure 3.7 only has one minimum point, is not unbounded below, and there are many algorithms which can find that minimum point efficiently.

What distinguishes these is a property called *convexity*, which is defined in Appendix A. If the feasible region is a convex set, and if the objective function is a convex function, then it is much easier to find the optimal solution. Checking convexity of the objective function is not usually too difficult. To check

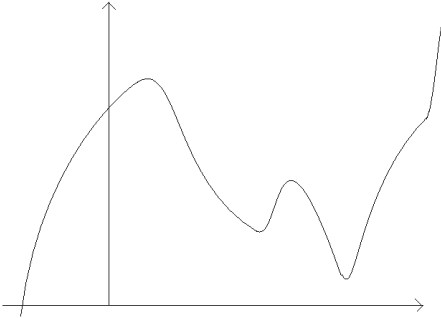


Figure 3.6: A function which is not convex.

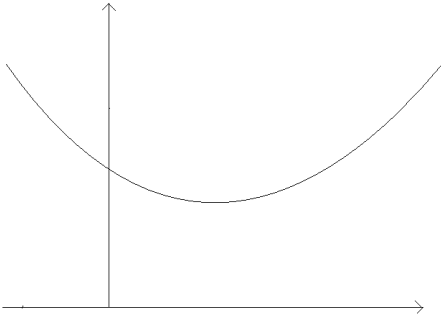


Figure 3.7: A convex function.

convexity of the feasible region, the following result is often useful.

Theorem 3.3. *Consider an optimization program whose constraints all take the form $g_i(\mathbf{x}) \leq 0$ or $h_j(\mathbf{x}) = 0$, where $i = 1, 2, \dots, k$ indexes the inequality constraints and $j = 1, 2, \dots, \ell$ indexes the equality constraints. If each function $g_1(\mathbf{x}), \dots, g_k(\mathbf{x})$ is convex, and if each function $h_1(\mathbf{x}), \dots, h_\ell(\mathbf{x})$ is linear, then the feasible region $X = \{\mathbf{x} \in \mathbb{R}^n : g_i(\mathbf{x}) \leq 0, h_j(\mathbf{x}) = 0, i \in \{1, \dots, k\}, j \in \{1, \dots, \ell\}\}$ is a convex set.*

Proof. Let $Y_i = \{\mathbf{x} \in \mathbb{R}^n : g_i(\mathbf{x}) \leq 0\}$ represent the values of \mathbf{x} which satisfy the i -th inequality constraint, and let $Z_j = \{\mathbf{x} \in \mathbb{R}^n : h_j(\mathbf{x}) = 0\}$ be the values of \mathbf{x} which satisfy the j -th equality constraint. From Proposition A.14, all of the sets Y_i are convex. From Example A.2, all of the sets Z_j are convex. The feasible region X is the set of vectors \mathbf{x} which satisfy *all* of the inequality and equality constraints, that is, the intersection of all of the sets Y_i and Z_j . By Proposition A.5, therefore, X is convex. \square

This is a very common situation, where the functions representing the inequality constraints are convex, and the functions representing equality constraints are linear. From this theorem, this means that the feasible region must be convex.

This subsection collects a few useful results justifying why we're spending some time on convexity: minimizing a convex function over a feasible region that is a convex set is extremely advantageous. Every local minimum is a global minimum, every stationary point is a local minimum, and the set of global minima is a convex set. Furthermore, if the function is strictly convex, the global minimum is unique. In other words, unlike in elementary calculus, you don't have to perform any "second derivative tests" on solutions to ensure they are truly minima, or distinguish between local and global minima. The set of global minima being a convex set is useful because it means that all solutions are "connected" or "adjacent" in some sense — there are no far-flung optimal solutions.

First, a few definitions; in everything that follows, we are trying to minimize a function $f(\mathbf{x})$ over a feasible region X . (That is, X is the set of all \mathbf{x} which satisfy all of the constraints.)

Proposition 3.1. *If f is a convex function and X is a convex set, then every local minimum of f is also a global minimum.*

Proof. By contradiction, assume that \mathbf{x}_1 is a local minimum of f , but not a global minimum. Then there is some $\mathbf{x}_2 \in X$ such that $f(\mathbf{x}_2) < f(\mathbf{x}_1)$. Because f is a convex function, for all $\lambda \in (0, 1]$ we have

$$f((1 - \lambda)\mathbf{x}_1 + \lambda\mathbf{x}_2) \leq (1 - \lambda)f(\mathbf{x}_1) + \lambda f(\mathbf{x}_2) = f(\mathbf{x}_1) + \lambda(f(\mathbf{x}_2) - f(\mathbf{x}_1))$$

and furthermore all points $(1 - \lambda)\mathbf{x}_1 + \lambda\mathbf{x}_2$ are feasible since X is a convex set and \mathbf{x}_1 and \mathbf{x}_2 are feasible. Since $f(\mathbf{x}_2) < f(\mathbf{x}_1)$, this means that

$$f((1 - \lambda)\mathbf{x}_1 + \lambda\mathbf{x}_2) < f(\mathbf{x}_1)$$

even as $\lambda \rightarrow 0$, contradicting the assumption that \mathbf{x}_1 is a local minimum. \square

Proposition 3.2. *If f is a convex function and X is a convex set, then the set of global minima is convex.*

Proof. Let \hat{X} be the set of global minima of f over the feasible region X . Choose any two global optima $\hat{\mathbf{x}}_1 \in \hat{X}$ and $\hat{\mathbf{x}}_2 \in \hat{X}$, and any $\lambda \in [0, 1]$.

Since $\hat{\mathbf{x}}_1$ and $\hat{\mathbf{x}}_2$ are global minima, $f(\hat{\mathbf{x}}_1) = f(\hat{\mathbf{x}}_2)$; let \hat{f} denote this common value. Because X is a convex set, the point $(1 - \lambda)\hat{\mathbf{x}}_1 + \lambda\hat{\mathbf{x}}_2$ is also feasible. Because f is a convex function,

$$\begin{aligned} f((1 - \lambda)\hat{\mathbf{x}}_1 + \lambda\hat{\mathbf{x}}_2) &\leq (1 - \lambda)f(\hat{\mathbf{x}}_1) + \lambda f(\hat{\mathbf{x}}_2) \\ &= (1 - \lambda)\hat{f} + \lambda\hat{f} \\ &= \hat{f} \end{aligned}$$

Therefore $f((1 - \lambda)\hat{\mathbf{x}}_1 + \lambda\hat{\mathbf{x}}_2) \leq \hat{f}$. But at the same time, $f((1 - \lambda)\hat{\mathbf{x}}_1 + \lambda\hat{\mathbf{x}}_2) \geq \hat{f}$ because \hat{f} is the global minimum value of f . So we must have $f((1 - \lambda)\hat{\mathbf{x}}_1 + \lambda\hat{\mathbf{x}}_2) = \hat{f}$ which means that this point is also a global minimum and $(1 - \lambda)\hat{\mathbf{x}}_1 + \lambda\hat{\mathbf{x}}_2 \in \hat{X}$ as well, proving its convexity. \square

Proposition 3.3. (*Uniqueness.*) *If f is a strictly convex function and X is a convex set, the set \hat{X} contains at most one element.*

Proof. By contradiction, assume that \hat{X} contains two distinct elements $\hat{\mathbf{x}}_1$ and $\hat{\mathbf{x}}_2$. Repeating the proof of the previous proposition, because f is strictly convex, the first inequality becomes strict and we must have $f((1 - \lambda)\hat{\mathbf{x}}_1 + \lambda\hat{\mathbf{x}}_2) < \hat{f}$. This contradicts the assumption that $\hat{\mathbf{x}}_1$ and $\hat{\mathbf{x}}_2$ are global minima. \square

We next discuss techniques which are common in transportation network analysis. We begin with simple problems: optimization problems with a single decision variable, and optimization problems with no constraints, before moving to more general problem classes. In this section, we'll assume that both the objective function and feasible region are convex and differentiable, which greatly simplifies matters.

3.3.1 One-dimensional problems

To start off, consider a simple minimization problem in one variable with no constraints:

$$\min_x f(x)$$

Because f is convex, any local minimum is also a global minimum. So, all we need is to know when we've reached a local minimum. Thus, we're looking for a set of *optimality conditions*, that is, a set of equations and/or inequalities

in x which are true if and only if x is optimal. For the unconstrained case, this is easy: we know from basic calculus that \hat{x} is a local minimum if

$$f'(\hat{x}) = 0$$

(We don't have to check whether \hat{x} is a local minimum or a local maximum because f is convex.)

Example 3.1. Find the value of x which minimizes $f(x) = x^2 - 3x + 5$.

Solution. $f'(x) = 2x - 3$, which vanishes if $x = 3/2$. Therefore $x = 3/2$ minimizes $f(x)$. ■

It's a little bit more complicated if we add constraints to the picture. For instance, consider the function in Figure 3.7 (which could very well be the function from Example 3.1), but with the added constraint $x \geq 0$. In this case, nothing is different, and the optimum still occurs where $f'(x)$ vanishes, that is, at $x = 3/2$. But what if the constraint was $x \geq 2$? In this case, $x = 3/2$ is infeasible, and $f'(x)$ is always strictly positive in the entire feasible region. This means that f is strictly increasing over the entire feasible region, so the minimum value is obtained at the smallest possible value of x , that is, $x = 2$. So we see that sometimes the local minimum of a constrained optimization problem can be at a point where $f'(x)$ is nonzero.

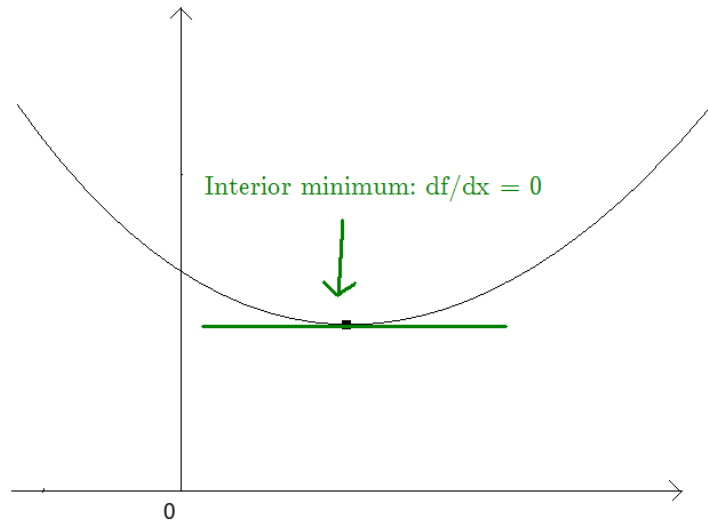
To simplify things a little bit, assume that the constraint is of the form $x \geq 0$, that is, we are trying to solve

$$\begin{array}{ll} \min_x & f(x) \\ \text{s.t.} & x \geq 0 \end{array}$$

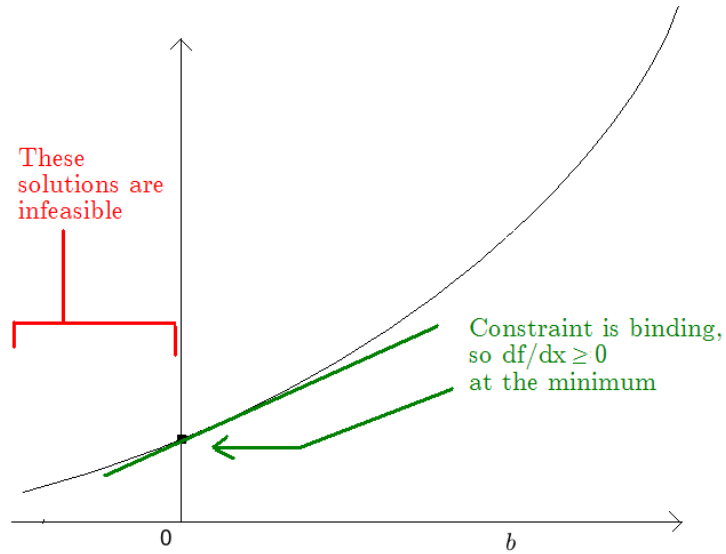
As Figure 3.8 shows, there are only two possibilities. In the first case, the minimum occurs when x is strictly positive. We can call this an *interior* minimum, or we can say that the constraint $x \geq 0$ is *nonbinding* at this point. In this case, clearly $f'(x)$ must equal zero: otherwise, we could move slightly in one direction or the other, and reduce f further. The other alternative is that the minimum occurs for $x = 0$, as in Figure 3.8(b). For this to be a minimum, we need $f'(0) \geq 0$ — if $f'(0) < 0$, f is decreasing at $x = 0$, so we could move to a slightly positive x , and thereby reduce f .

Let's try to draw some general conclusions. For the interior case of Figure 3.8(a), we needed $x \geq 0$ for feasibility, and $f'(x) = 0$ for optimality. For the boundary case of Figure 3.8(b), we had $x = 0$ exactly, and $f'(x) \geq 0$. So we see that in both cases, $x \geq 0$ and $f'(x) \geq 0$, and furthermore that at least one of these has to be exactly equal to zero. To express the fact that either x or $f'(x)$ must be zero, we can write $xf'(x) = 0$. So a solution \hat{x} solves the minimization problem if and only if

$$\begin{array}{rcl} \hat{x} & \geq & 0 \\ f'(\hat{x}) & \geq & 0 \\ \hat{x}f'(\hat{x}) & = & 0. \end{array}$$



(a) Convex function where the constraint is not binding at the minimum.



(b) Convex function where the constraint is binding at the minimum.

Figure 3.8: Two possibilities for minimizing a convex function with a constraint.

Whenever we can find \hat{x} that satisfies these three conditions, we know it is optimal. These are often called *first-order conditions* because they are related to the first derivative of f . The condition $\hat{x}f(\hat{x}) = 0$ is an example of a *complementarity constraint* because it forces either \hat{x} or $f(\hat{x})$ to be zero.

3.3.2 Bisection method

The bisection method allows us to solve one-dimensional problems over bounded feasible regions. Consider the one-dimensional optimization problem

$$\begin{aligned} \min_x \quad & f(x) \\ \text{s.t.} \quad & a \leq x \leq b \end{aligned}$$

where f is continuously differentiable and convex. The bisection method works by constantly narrowing down the region where the optimal solution lies. After the k -th iteration, the bisection method will tell you that the optimum solution lies in the interval $[a_k, b_k]$, with this interval shrinking over time (that is, $b_k - a_k < b_{k-1} - a_{k-1}$). A natural termination criterion is to stop when the interval is sufficiently small, that is, when $b_k - a_k < \epsilon$, where ϵ is the precision you want for the final solution.

The idea is that the sign of the derivative of the midpoint tells you where the optimum is. If the derivative is negative at a_k , but positive at b_k , the optimum occurs somewhere in-between, at a point where it is zero. So, if the derivative is positive at the midpoint, we know that the zero point has to happen somewhere to its left; if negative, somewhere to its right. If we happen to get lucky, the derivative at the midpoint will be exactly zero, and you can stop — but this is really rare. Figure 3.9 illustrates how bisection works. (The exercises ask you to show that bisection converges to the optimum solution even if the derivative initially has the same sign at both endpoints; the case where f' is positive at a_k but negative at b_k is impossible for a convex function.)

Here's how the algorithm works.

Step 0: Initialize. Set the iteration counter $k = 0$, $a_0 = a$, $b_0 = b$.

Step 1: Evaluate midpoint. Identify the midpoint $c_k = (a_k + b_k)/2$ and calculate the derivative of f there, $d_k = f'(c_k)$

Step 2: Bisect. If $d_k > 0$, set $a_{k+1} = a_k$, $b_{k+1} = c_k$. Otherwise, set $a_{k+1} = c_k$, $b_{k+1} = b_k$.

Step 3: Iterate. Increase the counter k by 1 and check the termination criterion. If $b_k - a_k < \epsilon$, then terminate; otherwise, return to step 1.

Example 3.2. Use the bisection algorithm to find the minimum of $f(x) = (x - 1)^2 + e^x$ on the interval $x \in [0, 2]$, to within a tolerance of $\epsilon = 0.01$.

Solution. You may find it useful to follow along in Table 3.1. We start off with $k = 0$, $a_0 = 0$, and $b_0 = 2$. We calculate the derivative at the midpoint:

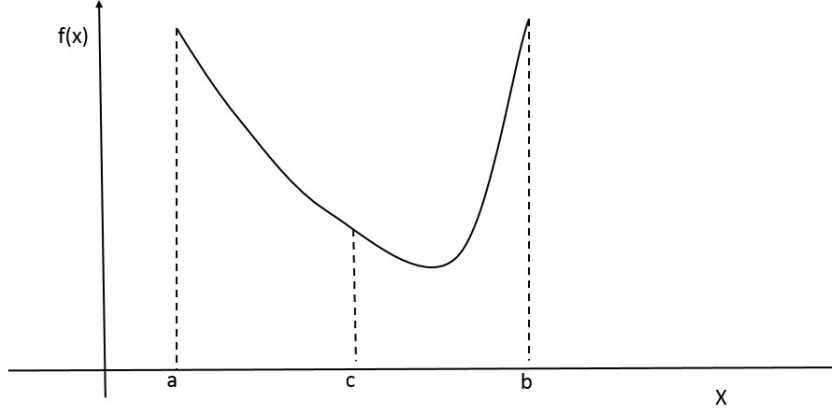


Figure 3.9: Bisection method.

Table 3.1: Demonstration of the bisection algorithm with $f(x) = (x - 1)^2 + e^x$, $x \in [0, 2]$.

k	a_k	b_k	$c_k = (a_k + b_k)/2$	d_k
0	0	2	1	$2.71 > 0$
1	0	1	$1/2$	$0.649 > 0$
2	0	$1/2$	$1/4$	$-0.216 < 0$
3	$1/4$	$1/2$	$3/8$	$0.205 > 0$
4	$1/4$	$3/8$	$5/16$	$-0.00816 < 0$
5	$5/16$	$3/8$	$11/32$	$0.0978 > 0$
6	$5/16$	$11/32$	$21/64$	$0.0446 > 0$
7	$5/16$	$21/64$	$41/128$	$0.0182 > 0$
8	$5/16$	$41/128$	$81/256$	

$f'(x) = 2(x - 1) + e^x$, so $f'(1) = 2.71$, which is positive. Since f' is positive at $x = 1$, the minimum must occur to the *left* of this point, that is, somewhere in the interval $[0, 1]$. We set a_1 and b_1 equal to these new values, and repeat. The new midpoint is $1/2$, and $f'(1/2) = 0.649$ is again positive, so the minimum must occur to the left of this point, in the interval $[0, 1/2]$. The midpoint of 0 and $1/2$ is $1/4$, and $f'(1/4) = -0.216$ is negative, so the minimum is to the *right* of the midpoint. Thus, the new interval is $[1/4, 1/2]$, and we repeat as shown in Table 3.1.

During the eighth iteration, the width of the interval is $41/128 - 5/16 = 0.00781$, which is less than the tolerance of 0.01; therefore we stop, and return our best guess of the optimum as the midpoint of this interval: $\hat{x} \approx 81/256 = 0.31640625$. The true minimum point occurs at $\hat{x} = 0.3149230578\dots$; if we had chosen a smaller tolerance ϵ , the algorithm would have narrowed the interval further, with both ends converging towards this point. ■

There are additional methods for solving one-dimensional convex optimization problems like these. Appendix C also describes the “golden section” method, which is useful when the objective function is not differentiable, and a method based on Newton’s method, which is useful when the objective is twice differentiable.

3.3.3 Multidimensional problems with nonnegativity constraints

Most interesting optimization problems have more than one decision variable. However, we will keep the assumption that the only constraint on the decision variables is nonnegativity. Using the vector \mathbf{x} to refer to all of the decision variables, we solve the problem

$$\begin{array}{ll} \min_{\mathbf{x}} & f(\mathbf{x}) \\ \text{s.t.} & \mathbf{x} \geq \mathbf{0} \end{array}$$

Using the same logic as before, we can show that $\hat{\mathbf{x}}$ solves this problem if and only if the following conditions are satisfied for *every* decision variable x_i :

$$\begin{array}{rcl} \hat{x}_i & \geq & 0 \\ \frac{\partial f(\hat{\mathbf{x}})}{\partial x_i} & \geq & 0 \\ \hat{x}_i \frac{\partial f(\hat{\mathbf{x}})}{\partial x_i} & = & 0 \end{array}$$

You should convince yourself that if these conditions are not met for each decision variable, then $\hat{\mathbf{x}}$ cannot be optimal: if the first condition is violated, the solution is infeasible; if the second is violated, the objective function can be reduced by increasing x_i ; if the first two are satisfied but the third is violated, then $\hat{x}_i > 0$ and $\frac{\partial f(\hat{\mathbf{x}})}{\partial x_i} > 0$, and the objective function can be reduced by decreasing x_i .

This can be compactly written in vector form as

$$\mathbf{0} \leq \hat{\mathbf{x}} \perp \nabla f(\hat{\mathbf{x}}) \geq \mathbf{0} \tag{3.3}$$

where the \perp symbol indicates orthogonality, i.e. that the dot product of $\hat{\mathbf{x}}$ and $\nabla f(\hat{\mathbf{x}})$ is zero.

Unfortunately, the bisection algorithm does not work nearly as well in higher dimensions. It is difficult to formulate an extension that always works, and those that do are inefficient. We’ll approach solution methods for higher-dimensional problems somewhat indirectly, tackling a few other topics first: addressing constraints other than nonnegativity, and a few highlights of linear optimization.

3.3.4 Constrained problems

Constraints can take a very large number of forms. For the purposes of this book, we can restrict attention to only two types of constraints: *linear equality* constraints and *nonnegativity* constraints. The previous two sections showed you how to deal with nonnegativity constraints; this section discusses linear equality constraints. A linear equality constraint is of the form

$$\sum_i a_i x_i = b, \quad (3.4)$$

where the x_i are decision variables, and the a_i and b are constants.

We can handle these using the technique of Lagrange multipliers. This technique is demonstrated in the following example for the case of a single linear equality constraint.

$$\begin{aligned} \min_{x_1, x_2} \quad & x_1^2 + x_2^2 \\ \text{s.t.} \quad & x_1 + x_2 = 5 \end{aligned}$$

(It is a useful exercise to verify that $x_1^2 + x_2^2$ is a strictly convex function, and that $\{x_1, x_2 : x_1 + x_2 = 5\}$ is a convex set.)

The main idea behind Lagrange multipliers is that unconstrained problems are easier than constrained problems. The technique is an ingenious way of nominally removing a constraint while still ensuring that it holds at optimality. The equality constraint is “brought into the objective function” by multiplying the difference between the right- and left-hand sides by a new decision variable κ (called the Lagrange multiplier), adding the original objective function. This creates the Lagrangian function

$$\mathcal{L}(x_1, x_2, \kappa) = x_1^2 + x_2^2 + \kappa(5 - x_1 - x_2) \quad (3.5)$$

It is possible to show that *the optimal solutions of the original optimization problem correspond to stationary points of the Lagrangian function*, that is, to values of x_1 , x_2 , and κ such that $\nabla \mathcal{L}(x_1, x_2, \kappa) = \mathbf{0}$. To find this stationary point, take partial derivatives with respect to each variable and set them all equal to zero:

$$\frac{\partial \mathcal{L}}{\partial x_1} = 2x_1 - \kappa = 0 \quad (3.6)$$

$$\frac{\partial \mathcal{L}}{\partial x_2} = 2x_2 - \kappa = 0 \quad (3.7)$$

$$\frac{\partial \mathcal{L}}{\partial \kappa} = 5 - x_1 - x_2 = 0 \quad (3.8)$$

Notice that the third optimality condition (3.8) *is simply the original constraint*, so this stationary point must be feasible. Equations (3.6) and (3.7) respectively tell us that $x_1 = \kappa/2$ and $x_2 = \kappa/2$; substituting these expressions into (3.8) gives $\kappa = 5$, and therefore the optimal solution occurs for $x_1 = x_2 = 5/2$.

This technique generalizes perfectly well to the case of multiple linear equality constraints. Consider the general optimization problem

$$\begin{aligned} \min_{x_1, \dots, x_n} \quad & f(x_1, \dots, x_n) \\ \text{s.t.} \quad & \sum_{i=1}^n a_{1i}x_i = b_1 \\ & \sum_{i=1}^n a_{2i}x_i = b_2 \\ & \vdots \\ & \sum_{i=1}^n a_{mi}x_i = b_m \end{aligned}$$

where f is convex. The corresponding Lagrangian is

$$\begin{aligned} \mathcal{L}(x_1, \dots, x_n, \kappa_1, \dots, \kappa_m) = & f(x_1, \dots, x_n) + \kappa_1 \left(b_1 - \sum_{j=1}^n a_{1j}x_j \right) + \\ & \kappa_2 \left(b_2 - \sum_{j=1}^n a_{2j}x_j \right) + \dots + \kappa_m \left(b_m - \sum_{j=1}^n a_{mj}x_j \right) \end{aligned}$$

For an optimization problem that has *both* linear equality constraints and nonnegativity constraints, we form the optimality conditions by combining the Lagrange multiplier technique with the complementarity technique from the previous section. Thinking back to Section 3.3.3, in the same way that we replaced the condition $f'(\hat{x}) = 0$ for the unconstrained case with the three conditions $\hat{x} \geq 0$, $f'(\hat{x}) \geq 0$, and $x f'(\hat{x}) = 0$ when the nonnegativity constraint was added, we'll adapt the Lagrangian optimality conditions. If the optimization problem has the form

$$\begin{aligned} \min_{x_1, \dots, x_n} \quad & f(x_1, \dots, x_n) \\ \text{s.t.} \quad & \sum_{i=1}^n a_{1i}x_i = b_1 \\ & \sum_{i=1}^n a_{2i}x_i = b_2 \\ & \vdots \\ & \sum_{i=1}^n a_{mi}x_i = b_m \\ & x_1, \dots, x_n \geq 0 \end{aligned}$$

where $m \leq n$, then the Lagrangian is

$$\begin{aligned} \mathcal{L}(x_1, \dots, x_n, \kappa_1, \dots, \kappa_m) = & f(x_1, \dots, x_n) + \kappa_1 \left(b_1 - \sum_{j=1}^n a_{1j}x_j \right) + \\ & \kappa_2 \left(b_2 - \sum_{j=1}^n a_{2j}x_j \right) + \dots + \kappa_m \left(b_m - \sum_{j=1}^n a_{mj}x_j \right) \end{aligned}$$

and the optimality conditions are

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial x_i} &\geq 0 && \forall i \in \{1, \dots, n\} \\ \frac{\partial \mathcal{L}}{\partial \kappa_j} &= 0 && \forall j \in \{1, \dots, m\} \\ x_i &\geq 0 && \forall i \in \{1, \dots, n\} \\ x_i \frac{\partial \mathcal{L}}{\partial x_i} &= 0 && \forall i \in \{1, \dots, n\} \end{aligned}$$

Be sure you understand what each of these formulas implies. Each decision variable must be nonnegative, the partial derivative of \mathcal{L} with respect to this variable must be nonnegative, and their product must equal zero (for the same reasons as discussed in Section 3.3.3). For the Lagrange multipliers $(\kappa_1, \dots, \kappa_m)$, the corresponding partial derivative of \mathcal{L} must be zero. Notice how this is a combination of the two techniques.

For small optimization problems, we can write down each of these conditions and solve for the optimal solution, as above. However, for large-scale problems this process can be very inconvenient. Later chapters in the book explain methods which work better for large problems. As a final note, the full theory of Lagrange multipliers is more involved than what is discussed here. However, it suffices for the case of a convex objective function and linear equality constraints. Optimality conditions for some other cases are given in Appendix C.

3.4 Historical Notes and Further Reading

(These sections are incomplete in this beta version of the text, and will be substantially expanded in the complete first edition.)

There are a great number of resources on single and multivariable calculus, and on linear algebra, that readers can consult if they require additional background than what is provided in this chapter on these subjects.

For the mathematical concepts introduced in this chapter, the main result on fixed point problems (Brouwer's theorem) was presented in Brouwer (1910). Variational inequalities are described at length in the book by Facchinei and Pang (2003). For extended treatments of convex optimization, see textbooks of Rockafellar (1997) and Bertsekas (2003).

Mathematical optimization has a long history, including important contributions by Fermat, Newton, the Bernoullis, Lagrange, and Gauss. These earlier methods are largely based in calculus and analytical in nature (with Newton's method being a notable exception). With the advent of computers in the early 20th century, and the logistics demands imposed by World War II, the field of optimization took on additional foci centered on computation and solution of large-scale problems. The seminal work of Dantzig (1963) in linear programming substantially expanded both optimization theory and the range of applications where optimization was used.

Optimization problems are often classified based on the structure of the objective function, decision variables, and constraints. Contemporary treatments can be found in linear programming (Bertsimas and Tsitsiklis, 1997), nonlinear programming (Bazaraa et al., 2006; Bertsekas, 2016), integer programming (Wolsey, 1998), stochastic optimization (Birge and Louveaux, 1997), and network optimization (Ahuja et al., 1993).

Simulated annealing was proposed by Kirkpatrick et al. (1983), building on an earlier algorithm of Metropolis et al. (1953). Genetic algorithms for optimization were first proposed by Rechenberg (1973) and Schwefel (1977). Other metaheuristics not described in the text include GRASP (Feo and Resende, 1995), tabu search (Glover, 1989, 1990), ant swarms (Dorigo and Stützle, 2004), and bee swarms (Pham et al., 2005).

3.5 Exercises

1. [31] For each of the following functions, find all of its fixed points or state that none exist.

(a) $f(x) = x^2$, where $X = \mathbb{R}$

(b) $f(x) = 1 - x^2$, where $X = [0, 1]$

(c) $f(x) = e^x$, where $X = \mathbb{R}$.

(d) $f(x_1, x_2) = \begin{bmatrix} -x_1 \\ x_2 \end{bmatrix}$ where $X = \{(x_1, x_2) : -1 \leq x_1 \leq 1, 0 \leq x_2 \leq 1\}$.

(e) $f(x, y) = \begin{bmatrix} -y \\ x \end{bmatrix}$ where $X = \{(x, y) : x^2 + y^2 \leq 1\}$ is the unit disc.

2. [54] Brouwer's theorem guarantees the existence of a fixed point for the function $f : X \rightarrow X$ if f is continuous and X is closed, bounded, and convex. Show that each of these four conditions is necessary by creating examples of a function f and set X which satisfy only three of those conditions but do not have a fixed point. Come up with such examples with each of the four conditions missing. (The notation $f : X \rightarrow X$ means that the range of the function must be contained in its domain; every "output" from f is also a valid "input.") Hint: you will probably find it easiest to work with simple functions and sets whenever possible, e.g. something like $X = [0, 1]$. Visualizing fixed points as intersections with the diagonal line through the origin may help you as well.

3. [45] Find all of the solutions of each of the following variational inequalities $\text{VI}(K, \mathbf{F})$.

(a) $F(x) = x + 1$, $K = [0, 1]$

(b) $F(x) = \frac{x}{2}$, $K = [-1, 1]$

(c) $F(x, y) = \begin{bmatrix} 0 \\ -y \end{bmatrix}$, $K = \{(x, y) : x^2 + y^2 \leq 1\}$

4. [58] Theorem 3.2 guarantees the existence of a solution to the variational inequality $\text{VI}(K, \mathbf{F})$ if K is closed, bounded, and convex, and if \mathbf{F} is continuous. Show that each of these three conditions is necessary by creating counterexamples of functions \mathbf{F} and sets K which satisfy only three of these conditions, but for which $\text{VI}(K, \mathbf{F})$ has no solution. It may be helpful to include sketches.

Part II

Static Traffic Assignment

Chapter 4

Introduction to Static Assignment

This chapter introduces the static traffic assignment problem, laying the groundwork for more extended discussions in later chapters in this part. It defines the main concepts and notation to be used throughout this part of the book (Section 4.1) and explains how the equilibrium principle applies in static traffic assignment (Section 4.2). In this chapter we present the equilibrium idea in the simplest possible way. Chapter 5 will formulate these ideas in terms of the mathematical concepts introduced in Chapter 3, an approach which will scale better for problems of a more realistic scale. Section 4.2 also gives a basic way to solve for equilibrium using trial-and-error. Chapter ?? will present approaches better-suited for real world problems.

As additional motivation, Section 4.3 presents three “paradoxes” where the equilibrium solution behaves in counterintuitive ways. Adding additional capacity to a network, or retiming signals with the aim of reducing delay, may actually *increase* delay if proper care is not taken to anticipate how users of the system will react to these changes. Finally, Section 4.4 discusses what data is needed to construct a static traffic assignment model.

4.1 Notation and Concepts

As discussed in Chapter 1, there are many possible measures of effectiveness for evaluating the impacts of a transportation project or policy. Recall that the traffic assignment problem assumes that the number of drivers traveling between each origin zone and destination zone is given, and we want to find the number of drivers using each roadway segment. These link flow values provide information on congestion, emissions, toll revenue, or other measures of interest. We are also given the underlying network, which has a set of links A representing the roadway infrastructure, and a set of nodes N representing junctions, and a set of centroids Z where trips are allowed to begin and end. Every centroid is

represented by a node, so Z is a subset of N .

Let d^{rs} denote the number of drivers whose trips start at centroid r and end at centroid s . It is often convenient to write these values in matrix form, the *origin-destination matrix* (*OD matrix*, for short) \mathbf{D} where the number of trips between r and s is given by the value in the r -th row and s -th column. Together, an origin r and destination s form an *OD pair* (r, s) .

Associated with each link is its *flow*, denoted x_{ij} , representing the total number of vehicles wanting to use link (i, j) during the analysis period. The flow is also known as the *volume* or *demand*. For reasons described below, “demand” is actually the most accurate term, but “flow” and “volume” are the most common for reasons of tradition, and “flow” is used in this book. The travel time on link (i, j) is expressed as t_{ij} , and to represent congestion, we let this travel time be a function of x_{ij} and write $t_{ij}(x_{ij})$. Because of congestion effects, t_{ij} is typically increasing and convex, that is, its first two derivatives are typically positive. The function used to relate flow to travel time is called a *link performance function*. The most common used in practice is the Bureau of Public Roads (BPR) function, named after the agency which developed it:

$$t_{ij}(x_{ij}) = t_{ij}^0 \left(1 + \alpha \left(\frac{x_{ij}}{u_{ij}} \right)^\beta \right), \quad (4.1)$$

where t_{ij}^0 is the “free-flow” travel time (the travel time with no congestion), u_{ij} is the practical capacity (typically the value of flow which results in a level of service of C or D), and α and β are shape parameters which can be calibrated to data. The values $\alpha = 0.15$ and $\beta = 4$ are commonly used if no calibration is done.

Notice that this function is well-defined for any value of x_{ij} , even when flows exceed the stated “capacity” u_{ij} . In the basic traffic assignment problem, there are no explicit upper bounds enforced on link flows. The interpretation of a “flow” greater than the capacity is actually that the *demand* for travel on the link exceeds the capacity, and queues will form. The delay induced by these queues is then reflected in the link performance function. Alternately, one can choose a link performance function which asymptotically grows to infinity as $x_{ij} \rightarrow u_{ij}$, implicitly enforcing the capacity constraint. However, this approach can introduce numerical issues in solution methods. More discussion on this issue follows in Section 4.1.1, but the short answer is that properly addressing capacity constraints in traffic assignment requires a *dynamic* traffic assignment model, which is the subject of Part III.

From the modeler’s perspective, the goal of traffic assignment is to determine the link flows in a network. But from the standpoint of the travelers themselves, it is easier to think of them each choosing a path connecting their origin to their destination. Let d^{rs} denote the number of vehicles which will be departing origin r to destination s , where r and s are both centroids. Using h^π to represent the number of vehicles who choose path π , a *feasible assignment* is defined as a vector of path flows \mathbf{h} satisfying the following conditions:

1. $h^\pi \geq 0$ for all paths $\pi \in \Pi$. That is, path flows must be nonnegative.

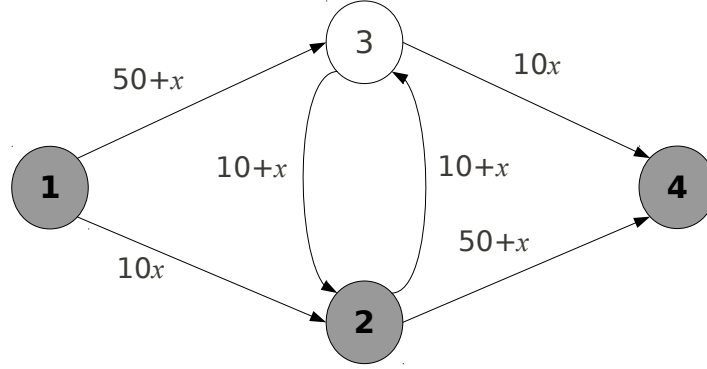


Figure 4.1: Example network for demonstration, with link performance functions shown.

2. $\sum_{\pi \in \Pi^{rs}} h^\pi = d^{rs}$ for all OD pairs (r, s) . Together with the nonnegativity condition, this requires that every vehicle traveling from r to s is assigned to exactly one of the paths connecting these zones.

Let H denote the set of feasible assignments.

Link flows x_{ij} and path flows h^π are closely linked, and we can obtain the former from the latter. Let δ_{ij}^π denote the number of times link (i, j) is used by path π , so $\delta_{ij}^\pi = 0$ if path π does not use link (i, j) , and $\delta_{ij}^\pi = 1$ if it does. With this notation, we have

$$x_{ij} = \sum_{r \in Z} \sum_{s \in Z} \sum_{\pi \in \Pi^{rs}} \delta_{ij}^\pi h^\pi. \quad (4.2)$$

This can be more compactly written using matrix notation as

$$\mathbf{x} = \mathbf{\Delta} \mathbf{h}, \quad (4.3)$$

where \mathbf{x} and \mathbf{h} are the vectors of link and path flows, respectively, and $\mathbf{\Delta}$ is the *link-path adjacency matrix*. The number of rows in this matrix is equal to the number of links, and the number of columns is equal to the number of paths in the network, and the value in the row corresponding to link (i, j) and the column corresponding to path π is δ_{ij}^π .

Given a feasible assignment, the corresponding link flows can be obtained by using equation (4.2) or (4.3). The set of *feasible link assignments* is the set X of vectors \mathbf{x} which satisfy (4.3) for some feasible assignment $\mathbf{h} \in H$.

Similarly, the path travel times c^π are directly related to the link travel times t_{ij} : the travel time of a path is simply the sum of the travel times of the links

comprising that path. By the same logic, we have

$$c^\pi = \sum_{(i,j) \in A} \delta_{ij}^\pi t_{ij} \quad (4.4)$$

or, in matrix notation,

$$\mathbf{c} = \mathbf{\Delta}^T \mathbf{t}. \quad (4.5)$$

A small example illustrates these ideas. Consider the network in Figure 4.1 with four nodes and six links. The centroid nodes are shaded, and the link performance functions are as indicated. For travelers from node 2 to node 4, there are two paths: $\{(2,3), (3,4)\}$ and $\{(2,4)\}$. Using the compact notation for paths, we could also write these as $[2,3,4]$ and $[2,4]$. Therefore, the set of acyclic paths between these nodes is $\Pi^{24} = \{[2,3,4], [2,4]\}$. You should verify for yourself that

$$\Pi^{14} = \{[1,2,4], [1,2,3,4], [1,3,4], [1,3,2,4]\}.$$

It will be useful to have specific indices for each path, i.e. $\Pi^{24} = \{\pi_1, \pi_2\}$ so $\pi_1 = [2,3,4]$ and $\pi_2 = [2,4]$, and similarly $\Pi^{14} = \{\pi_3, \pi_4, \pi_5, \pi_6\}$ with $\pi_3 = [1,2,4]$, $\pi_4 = [1,2,3,4]$, $\pi_5 = [1,3,4]$, and $\pi_6 = [1,3,2,4]$.

Let's say that the demand for travel from centroid 1 to centroid 4 is 40 vehicles, and that the demand from 2 to 4 is 60 vehicles. Then $d^{14} = 40$ and $d^{24} = 60$, and we must have

$$\sum_{\pi \in \Pi^{14}} h^\pi = h^3 + h^4 + h^5 + h^6 = d^{14} = 40 \quad (4.6)$$

and

$$\sum_{\pi \in \Pi^{24}} h^\pi = h^1 + h^2 = d^{24} = 60. \quad (4.7)$$

Let's assume that the vehicles from each OD pair are divided evenly among all of the available paths, so $h^\pi = 10$ for each $\pi \in \Pi^{14}$ and $h^\pi = 30$ for each $\pi \in \Pi^{24}$. We can now use equation (4.2) to calculate the link flows. For instance the flow on link (1,2) is

$$\begin{aligned} & \sum_{r \in Z} \left\{ \sum_{s \in Z} \left\{ \sum_{\pi \in \Pi^{rs}} \delta_{1,2}^\pi h^\pi \right\} \right\} = \\ & = \left\{ \left\{ \delta_{(1,2)}^3 h^3 + \delta_{(1,2)}^4 h^4 + \delta_{(1,2)}^5 h^5 + \delta_{(1,2)}^6 h^6 \right\} \right\} + \left\{ \left\{ \delta_{(1,2)}^1 h^1 + \delta_{(1,2)}^2 h^2 \right\} \right\} = \\ & = \{ \{ 1 \times 10 + 1 \times 10 + 0 \times 10 + 0 \times 10 \} \} + \{ \{ 0 \times 30 + 0 \times 30 \} \} = 20. \quad (4.8) \end{aligned}$$

where the braces show how the summations “nest.” Remember, this is just a fancy way of picking the paths which use link (1,2), and adding their flows. The equation is for use in computer implementations or for large networks; when solving by hand, it's perfectly fine to just identify the paths using a particular

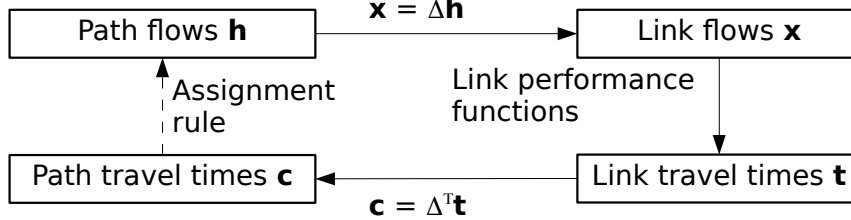


Figure 4.2: Traffic assignment as an iterative process.

link by inspection — in this case, only paths $\pi_1^{1,4}$ and $\pi_2^{1,4}$ use link (1,2). Repeating similar calculations, you should verify that $x_{13} = 20$, $x_{23} = 40$, $x_{32} = 10$, $x_{24} = 50$, and $x_{34} = 50$.

From these link flows we can get the link travel times by substituting the flows into the link performance functions, that is, $t_{12} = 10x_{12} = 200$, $t_{13} = 50 + x_{13} = 70$, $t_{23} = 10 + x_{23} = 50$, $t_{32} = 10 + x_{32} = 20$, $t_{24} = 50 + x_{24} = 100$, and $t_{34} = 10x_{34} = 500$. Finally, the path travel times can be obtained by either adding the travel times of their constituent links, or by applying equation (4.4). You should verify that $c^1 = 550$, $c^2 = 100$, $c^3 = 300$, $c^4 = 750$, $c^5 = 570$, and $c^6 = 190$.

The role of traffic assignment is to choose one path flow vector $\hat{\mathbf{h}}$ for purposes of forecasting and ranking alternatives, out of all of the feasible assignments in the network. An *assignment rule* is a principle used to determine this path flow vector. The most common assignment rule in practice is that the path flow vector should place all vehicles on a path with minimum travel time between their origins and destinations, although other rules are possible as well and will be discussed later in the book.

4.1.1 Commentary

The equations and concepts mentioned in the previous subsection can be related as follows. Given a vector of path flows \mathbf{h} , we can obtain the vector of link flows \mathbf{x} from equation (4.3); from these, we can obtain the vector of link travel times \mathbf{t} by substituting each link's flow into its link performance function; from this, we can obtain the vector of path travel times \mathbf{c} from equation (4.5). This process is shown schematically in Figure 4.2.

The one component which does not have a simple representation is how to obtain path flows from path travel times using an assignment rule, “completing the loop” with the dashed line in the figure. This is actually the most complicated step, and answering it will require most of the remainder of the chapter. The main difficulty is that introducing some rule for relating path travel times to path flows creates a circular dependency: the path flows would depend on

the path travel times, which depend on the link travel times, which depend on the link flows, which depend on the path flows, which depend on the path travel times and so on *ad infinitum*. We need to find a *consistent* solution to this process, which in this case means a set of path flows which remain unchanged when we go around the circuit: the path flows must be consistent with the travel times we obtain from those same path flows. Furthermore, the assignment rule must reflect the gaming behavior described in Section 1.3.

At this point, it is worthwhile to discuss the assumptions that go into the traffic assignment problem as stated here. The first assumption concerns the time scale over which the network modeling is occurring. This part of the book is focused entirely on what is called *static* network modeling, in which we assume that the network is close to a “steady state” during whatever length of time we choose to model (whether a peak hour, a multi-hour peak period, or a 24-hour model), and the link flows and capacities are measured with respect to the entire time period. That is, the capacity is the capacity *over the entire analysis period*, so if a facility has a capacity of 2200 veh/hr and we are modeling a three-hour peak period, we would use a capacity of 6600. Likewise, the link flows are the total flow over the three hours.

How long the analysis period should be is a matter of some balancing. Obviously, the longer the analysis period, the less accurate the steady state assumption is likely to be. In the extreme case of a 24-hour model, it is usually a (very big) stretch to assume that the congestion level will be the same over all 24 hours. On the other hand, choosing too short an analysis period can be problematic as well. In particular, if we are modeling a congested city or a large metropolitan area, trips from one end of the network to the other can easily take an hour or two, and it is good practice to have the analysis period be at least as long as most of the trips people are taking.

Properly resolving the issue of the “steady state” assumption requires moving to a *dynamic* traffic assignment (DTA) model. DTA models have the potential to more accurately model traffic, but are harder to calibrate, are more sensitive to having correct input data, and require more computer time. Further, DTA models end up requiring very different formulations and approaches. In short, while useful in some circumstances, they are not universally better than static models, and in any case they are surprisingly dissimilar. As a result, a full discussion of DTA models is deferred to the final part of this volume.

Another assumption we make is that *link and path flows can take any non-negative real value*. In particular, they are not required to be whole numbers, and there is no issue with saying that the flow on a link is 12.5 vehicles or that the number of vehicles choosing a path is, say, $\sqrt{2}$. This is often called the *continuum assumption*, because it treats vehicles as an infinitely-divisible fluid, rather than as a discrete number of “packets” which cannot be split. The reason for this assumption is largely computational — without the continuum assumption, traffic assignment problems become extremely difficult to solve, even on small networks. Further, from a practical perspective most links of concern have volumes in the hundreds or thousands of vehicles per hour, where the difference between fractional and integer values is negligible. Some also justify the con-

tinuum assumption by interpreting link and path flows to describe a long-term average of the flows, which may fluctuate to some degree from day to day.

4.2 Principle of User Equilibrium

The previous section introduced the main ideas and components of static traffic assignment, but did not provide much explanation of assignment rules other than claiming that a reasonable assignment rule results in all used routes connecting an origin and destination to have equal and minimal travel time, and that the “gaming” equilibrium idea described in Section 1.3 is relevant to route choice. This section explains this assignment rule in more detail.

Assignment rules are more complex than the other steps in traffic assignment shown in Figure 4.2, for two reasons. The first is that, being behavioral, it does not follow as mechanically as the other three steps, and has a more complicated answer. The other reason, alluding to the discussion in the previous section, is that adding this link creates a cycle of dependency. Linking path flows to path travel times, we now have four quantities which all depend on each other, and untangling this cycle requires a little bit of thought.

To handle this, let’s consider a simpler situation first. Rather than trying to find path flows (which require stating *every* driver’s route choice), let’s stick to a single driver. Why do they pick the route that they do? The number of potential paths in a network between two even slightly distant points in a network is enormous; in a typical city there are literally millions of possible routes one could theoretically take between an origin and destination. The vast majority of these are ridiculous, say, involving extraneous trips to the outskirts of the city and then back to the destination even though the origin is nearby.

Why are such paths ridiculous on their face? Table 4.1 gives a list of potential criteria which a desirable route would have, given in roughly decreasing order (at least according to my tastes for a trip to work or school). The most important times of day to properly model are the morning and evening peak periods, when most of the trips made are work-related commutes. For these types of trips, most people will choose the most direct route to their destination, where “direct” means some combination of travel time, cost, and distance (which are usually correlated). For simplicity, we’ll use “low travel time” as our starting point, specified as Assumption 4.1.

Assumption 4.1. (Shortest path assumption.) *Each driver wants to choose the path between their origin and their destination with the least travel time.*

Notice that this principle does not include the impact a driver has on other drivers in the system, and a pithy characterization of Assumption 4.1 is that “people are greedy.” If this seems unnecessarily pejorative, Section 4.3 shows a few ways that this principle can lead to suboptimal flow distributions on networks. Again, I emphasize that we adopt this assumption because we are modeling urban, peak period travel which is predominantly composed of work trips. If we were modeling, say, traffic flows around a national park during

Table 4.1: Potential criteria in choosing a route.

Low travel time
Reliable travel time
Low out-of-pocket cost (tolls, fuel, etc.)
Short distance
Bias toward (or away from) freeways
Low accident risk
Few potholes
Scenic view

summer weekends, quality of scenery may be considerably more important than being able to drive at free-flow speed, and a different assumption would be needed.

Further, the basic model developed in the first few weeks could really function just as well with cost or some other criterion, as long as it is *separable* and *additive* (that is, you can get the total value of the criterion by adding up its value on each link — the travel time of a route is the sum of the travel times on its component links, the total monetary cost of a path is the sum of the monetary costs of each link, but the total scenic quality of a path may not be the sum of the scenic quality of each link), and can be expressed as a function of the link flows \mathbf{x} . So even though we will be speaking primarily of travel times, it is quite possible, and sometimes appropriate, to use other measures as well. A second assumption follows from modeling commute trips:

Assumption 4.2. *Drivers have perfect knowledge of link travel times.*

In reality, drivers' knowledge is not perfect — but commutes are typically habitual trips, so it is not unreasonable to assume that drivers are experienced and well-informed about congestion levels at different places in the network, at least along routes they might plausibly choose. We will later relax this assumption, but for now we'll take it as it greatly simplifies matters and is not too far from the truth for commutes. (Again, in a national park or other place with a lot of tourist traffic, this would be a poor assumption.)

Now, with a handle on individual behavior, we can try to scale up this assumption to large groups of travelers. What will be the resulting state if there are a large number of travelers who all want to take the fastest route to their destinations? For example, if you have to choose between two routes, one of which takes ten minutes and the other fifteen, you would always opt for the first one. If you are the only one traveling, this is all well and good. The situation becomes more complicated if there are others traveling. If there are ten thousand people making the same choice, and all ten thousand pick the first route, congestion will form and the travel time will rapidly increase. According to Assumption 4.2, drivers would become aware of this, and some people would switch from the first route to the second route, because the first would no longer be faster. This process would continue: as long as the first route is slower, people

would switch away to the second route. If too many people switch to the second route, so the first becomes faster again, people would switch back.

With a little thought, it becomes clear that if there is *any* difference in the travel times between the two routes, people will switch from the slower route to the faster one. Note that Assumption 4.1 does not make any allowance for a driver being satisfied with a path which is a minute slower than the fastest path, or indeed a second slower, or even a nanosecond slower. Relaxing Assumption 4.1 to say that people may be indifferent as long as the travel time is “close enough” to the fastest path leads to an interesting, but more complicated line of research based on the concept of “bounded rationality,” which is discussed later, in Section 5.3.3. It is much simpler to assume that Assumption 4.1 holds strictly, in which case there are only three possible stable states:

1. Route 1 is faster, even when all of the travelers are using it.
2. Route 2 is faster, even when all of the travelers are using it.
3. Most commonly, neither route dominates the others. In this case, people use both Routes 1 and 2, and their travel times are *exactly equal*.

Because the third case is most common, this basic route choice model is called *user equilibrium*: the two routes are in equilibrium with each other. Why must the travel times be equal? If the first route was faster than the second, people would switch from the second to the first. This would decrease the travel time on the second route, and increase the travel time on the first, and people would continue switching until they were equal. The reverse is true as well: if the second route were faster, people would switch from the first route to the second, decreasing the travel time on the first route and increasing the travel time on the second. The *only* outcome where nobody has any reason to change their decision, is if the travel times are equal on both routes.

This is important enough to state again formally:

Corollary 4.1. (Principle of user equilibrium.) *Every used route connecting an origin and destination has equal and minimal travel time.*

Unused routes may of course have a higher travel time, and used routes connecting different origins and destinations may have different travel times, but any two used routes connecting the same origin and destination must have exactly the same travel times. Notice that we call this principle a corollary rather than an assumption: the real assumptions are the shortest path and full information assumptions. If you believe these are true, the principle of user equilibrium follows immediately and does not require you to assume anything more than you already have. The next section describes how to solve for equilibrium, along with a small example.

4.2.1 A trial-and-error solution method

We can develop a simple method for solving for path flows using the principle of user equilibrium, using nothing but the definition itself. For now, assume there

is a single OD pair (r, s) . The method is as follows:

1. Select a set of paths $\hat{\Pi}^{rs}$ which you think will be used by travelers from this OD pair.
2. Write equations for the travel times of each path in $\hat{\Pi}^{rs}$ as a function of the path flows.
3. Solve the system of equations enforcing equal travel times on all of these paths, together with the requirement that the total path flows must equal the total demand d^{rs} .
4. Verify that this set of paths is correct; if not, refine $\hat{\Pi}^{rs}$ and return to step 2.

The first step serves to reduce a large potential set of paths to a smaller set of reasonable paths $\hat{\Pi}^{rs}$, which you believe to be the set of paths which will be used by travelers from r to s . Feel free to use engineering judgment here; if this is not the right set of paths, we'll discover this in step 4 and can adjust the set accordingly. The second step involves applying equations (4.4) and (4.2). Write the equations for link flows as a function of the flow on paths in $\hat{\Pi}^{rs}$ (assuming all other paths have zero flow). Substitute these expressions for link flows into the link performance functions to get an expression for link travel times; then write the equations for path travel times as a function of link travel times.

If $\hat{\Pi}^{rs}$ truly is the set of used paths, all the travel times on its component paths will be equal. So, we solve a system of equations requiring just that. If there are n paths, there are only $n - 1$ independent equations specifying equal travel times. (If there are three paths, we have one equation stating paths one and two have equal travel time, and a second one stating paths two and three have equal travel time. An equation stating that paths one and three have equal travel time is redundant, because it is implied by the other two, and so it doesn't help us solve anything.) To solve for the n unknowns (the flow on each path), we need one more equation: the requirement that the sum of the path flows must equal the total flow from r to s (the "no vehicle left behind" equation requiring every vehicle to be assigned to a path). Solving this system of equations gives us path flows which provide equal travel times.

The last step is to verify that the set of paths is correct. What does this mean? There are two ways that $\hat{\Pi}^{rs}$ could be "incorrect": either it contains paths that it shouldn't, or it omits a path that should be included. In the former case, you will end up with an infeasible solution (e.g., a negative or imaginary flow on one path), and should eliminate the paths with infeasible flows, and go back to the second step with a new set $\hat{\Pi}^{rs}$. In the latter case, you have a feasible solution and the travel times of paths in $\hat{\Pi}^{rs}$, but they are not minimal: you have missed a path which has a faster travel time than any of the ones in $\hat{\Pi}^{rs}$, so you need to include this path in $\hat{\Pi}^{rs}$ and again return to the second step.

Let's take a concrete example: Figure 4.3 shows 7000 travelers traveling from zone 1 to zone 2 during one hour, and choosing between the two routes

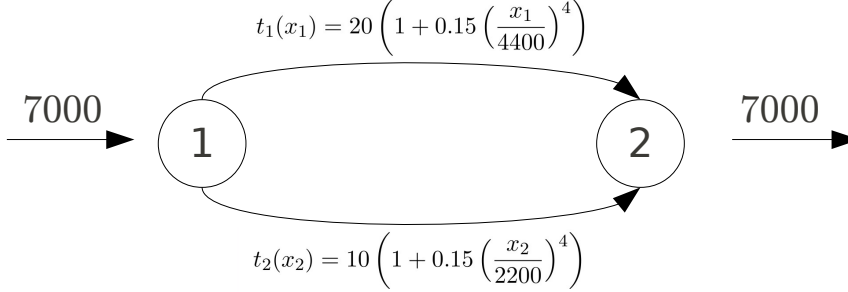


Figure 4.3: Small example using the two links.

mentioned above: route 1, with free-flow time 20 minutes and capacity 4400 veh/hr, and route 2, with free-flow time 10 minutes and capacity 2200 veh/hr. That means we have

$$t_1(x_1) = 20 \left(1 + 0.15 \left(\frac{x_1}{4400} \right)^4 \right), \quad (4.9)$$

$$t_2(x_2) = 10 \left(1 + 0.15 \left(\frac{x_2}{2200} \right)^4 \right). \quad (4.10)$$

This example is small enough that there is no real distinction between paths and links (because each path consists of a single link), so link flows are simply path flows ($x_1 = h_1$ and $x_2 = h_2$), and path travel times are simply link travel times ($c_1 = t_1$ and $c_2 = t_2$). As a starting assumption, we assume that both paths are used, so $\Pi^{12} = \{\pi_1, \pi_2\}$. We then need to choose the path flows h_1 and h_2 so that $t_1(h_1) = t_2(h_2)$ (equilibrium) and $h_1 + h_2 = 7000$ (no vehicle left behind). Substituting $h_2 = 7000 - h_1$ into the second delay function, the equilibrium equation becomes

$$20 \left(1 + 0.15 \left(\frac{h_1}{4400} \right)^4 \right) = 10 \left(1 + 0.15 \left(\frac{7000 - h_1}{2200} \right)^4 \right). \quad (4.11)$$

Using a numerical equation solver, we find that equilibrium occurs for $h_1 = 3376$, so $h_2 = 7000 - 3376 = 3624$, and $t_1(x_1) = t_2(x_2) = 21.0$ minutes. Alternately, we can use a graphical approach. Figure 4.4 plots the travel time on *both* routes as a function of the flow on route 1 (because if we know the flow on route 1, we also know the flow on route 2). The point where they intersect is the equilibrium: $h_1 = 3376$, $t_1 = t_2 = 21.0$.

In the last step, we verify that the solution is reasonable (no paths have negative flow) and complete (there are no paths we missed which have a shorter

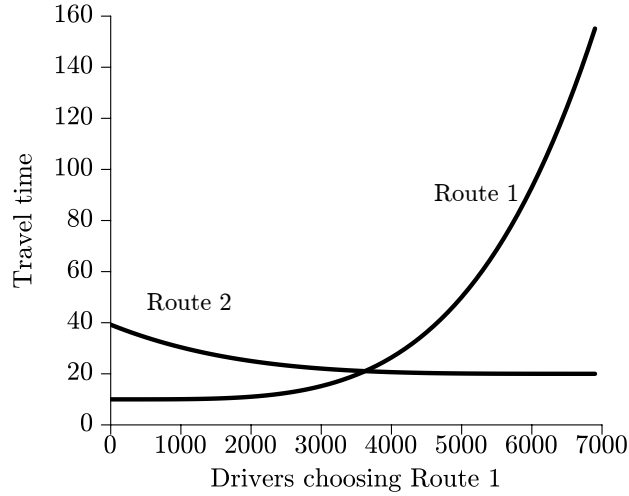


Figure 4.4: The equilibrium point lies at the intersection of the delay functions.

travel time). Both conditions are satisfied, so we have found the equilibrium flow and stop.

Let's modify the problem slightly, so the travel time on link 1 is now

$$t_1(x_1) = 50 \left(1 + 0.15 \left(\frac{x_1}{4400} \right)^4 \right).$$

In this case, solving

$$50 \left(1 + 0.15 \left(\frac{h_1}{4400} \right)^4 \right) = 10 \left(1 + 0.15 \left(\frac{7000 - h_1}{2200} \right)^4 \right)$$

leads to a nonsensical solution: none of the answers involve real numbers without an imaginary part. The physical interpretation of this is that *there is no way to assign 7000 vehicles to these two paths so they have equal travel times*. Looking at a plot (Figure 4.5), we see that this happens because path 2 dominates path 1: even with all 7000 vehicles on path 2, it has a smaller travel time.

So, the solution to the modified problem is: $h_1 = 0$, $h_2 = 7000$, $t_1 = 50$, and $t_2 = 39.2$. This still satisfies the principle of user equilibrium: path 1 is not *used*, so it is fine for its travel time to not equal that of path 2.

To find this solution in the trial-and-error method, we would start by removing *both* paths (since imaginary path flows are not meaningful), and then adding in the shortest path at free-flow (since there must be at least one used path). When there is only one path in $\hat{\Pi}^{rs}$, the equilibrium state is easy to find, because the “system of equations” reduces trivially to placing all demand onto the single used path.

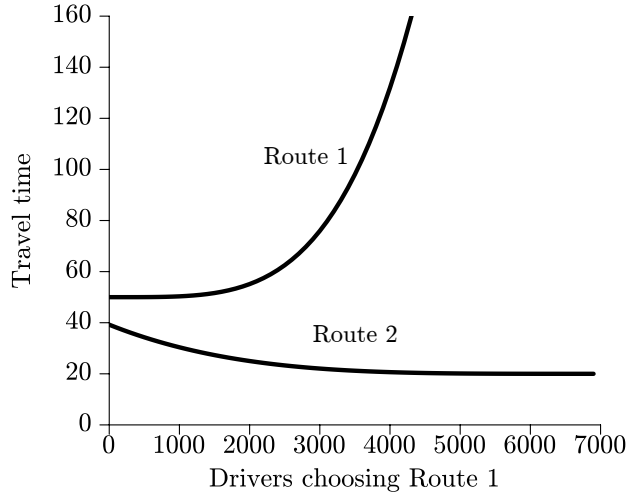


Figure 4.5: There is no intersection point: path 1 is dominated by path 2.

In more complicated networks, writing all of the equilibrium equations and solving them simultaneously is much too difficult. Instead, a more systematic approach is taken, and will be described in Chapter 6.

4.3 Three Motivating Examples

It is worth asking whether user equilibrium is the best possible condition. “Best” is an ambiguous term, but can be related towards our general goals as transportation engineers. User equilibrium probably does not lead to the flow pattern with, say, the best emissions profile, simply because there’s no reason to believe that the principle of user equilibrium has any connection whatsoever to emissions — it is based on people trying to choose fastest paths. But maybe it is related to congestion in some way, and you might find it plausible that each individual driver attempting to choose the best path for himself or herself would minimize congestion system-wide. This section presents three examples which should shatter this innocent-seeming idea. In this section, we will not concern ourselves with how a feasible assignment satisfying this assignment rule is found (the trial-and-error method from the previous section would suffice), but will focus instead on how equilibrium can be used to evaluate the performance of potential alternatives.

In the first example, consider the network shown in Figure 4.6 where the demand between nodes 1 and 2 is $d^{12} = 30$ vehicles. Using h^\uparrow and h^\downarrow to represent the flows on the top and bottom paths in this network, the set of feasible assignments are the two-dimensional vectors $\mathbf{h} = [h^\uparrow \quad h^\downarrow]$ which satisfy the conditions $h^\uparrow \geq 0$, $h^\downarrow \geq 0$, and $h^\uparrow + h^\downarrow = 30$. The figure shows the link

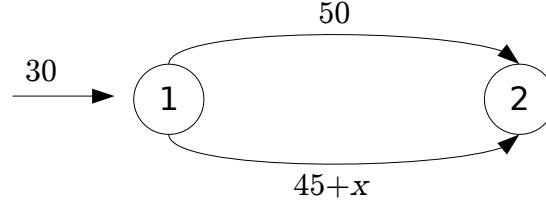


Figure 4.6: A two-link network.

performance functions for each of the two links in the network. Notice that the travel time on the top link is constant irrespective of the flow, while the travel time on the bottom link increases with its flow. However, at low values of flow, the bottom link is faster than the top one. This corresponds to a scenario where one link is shorter, but more subject to congestion, while the other link is longer but free of congestion.

With our stated assignment rule, the solution to the traffic assignment problem is $h^\uparrow = 25$, $h^\downarrow = 5$, because this will result in link flows of 25 and 5 on the top and bottom links, respectively, giving travel times of 50 on both paths. In this state, all vehicles in the network experience a travel time of 50.

Now, suppose that it is possible to improve one of the links in the network. The intuitive choice is to improve the link which is subject to congestion, say, changing its link performance function from $45 + x^\downarrow$ to $40 + \frac{1}{2}x^\downarrow$, as shown in Figure 4.7. (Can you see why this is called an improvement?) However, in this case the path flow solution which corresponds to the assignment rule is $h^\uparrow = 10$, $h^\downarrow = 20$, because this results in equal travel times on both the top and bottom paths. These travel times are still 50! Even though the bottom link was improved, the effect was completely offset by vehicles switching paths away from the top link and onto the bottom link.

In other words, improving a link (even the only congested link in a network) does not necessarily reduce travel times, because drivers can change their behavior in response to changes on the network. (If we could somehow force travelers to stay on the same paths they were using before, then certainly some vehicles would experience a lower travel time after the improvement is made.) This is called the *Knight-Pigou-Downs* paradox.

The second example was developed by Dietrich Braess, and shows a case where building a new roadway link can actually worsen travel times for all travelers in the network, after the new equilibrium is established. Assume we have the network shown in Figure 4.8a, with the link performance functions next to each link. The user equilibrium solution can be found by symmetry: since the top and bottom paths are exactly identical, the demand of six vehicles

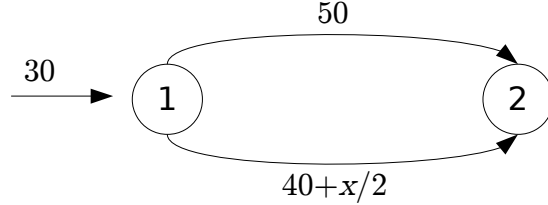


Figure 4.7: The two-link network with an improvement on the bottom link.

will evenly split between them. A flow of three vehicles on the two paths leads to flow of three vehicles on each of the four links; substituting into the link performance functions gives travel times of 53 on (1,3) and (2,4) and 30 on (1,2) and (3,4), so the travel time of both paths is 83, and the principle of user equilibrium is satisfied.

Now, let's modify the network by adding a new link from node 2 to node 3, with link performance function $10 + x_{23}$ (Figure 4.8b). We have added a new path to the network; let's label these as follows. Path 1 is the top route [1, 3, 4]; path 2 is the middle route [1, 2, 3, 4], and path 3 is the bottom route [1, 2, 4]. Paths 1 and 3 each have a demand of three vehicles and a travel time of 83. Path 2, on the other hand, has a flow of zero vehicles and a travel time of 70, so the principle of user equilibrium is violated: the travel time on the used paths is equal, but not minimal.

Assumption 4.1 suggests that someone will switch their path to take advantage of this lower travel time; let's say someone from path 1 switches to path 2, so we have $h_1 = 2$, $h_2 = 1$, and $h_3 = 3$. From this we can predict new link flows: $x_{12} = 4$, $x_{13} = 2$, $x_{23} = 1$, $x_{24} = 3$, $x_{34} = 3$. Substituting into link performance functions gives new link travel times: $t_{12} = 40$, $t_{13} = 52$, $t_{23} = 11$, $t_{24} = 53$, $t_{34} = 30$, and finally we can recover the new path travel times: $c_1 = 82$, $c_2 = 81$, and $c_3 = 93$.

This is still not an equilibrium; perhaps someone from path 3 will switch to path 2 in an effort to save 12 minutes of time. So now $h_1 = h_2 = h_3 = 2$, the link flows are $x_{12} = x_{34} = 4$ and $x_{13} = x_{23} = x_{24} = 2$, the travel times are $t_{12} = t_{34} = 40$, $t_{13} = t_{24} = 52$ and $t_{23} = 12$, and the path travel times are $c_1 = c_2 = c_3 = 92$. We have found the new equilibrium, and it is unconditionally worse than the old one. Before adding the new link, each driver had a travel time of 83; now every driver has a travel time of 92. Nobody is better off. Everyone is worse off. This is the famous *Braess paradox*.

The third example, adapted from M. J. Smith, shows how the traffic assignment problem can provide more insight into other traffic engineering problems such as signal timing. Consider the network shown in Figure 4.9. Drivers choose

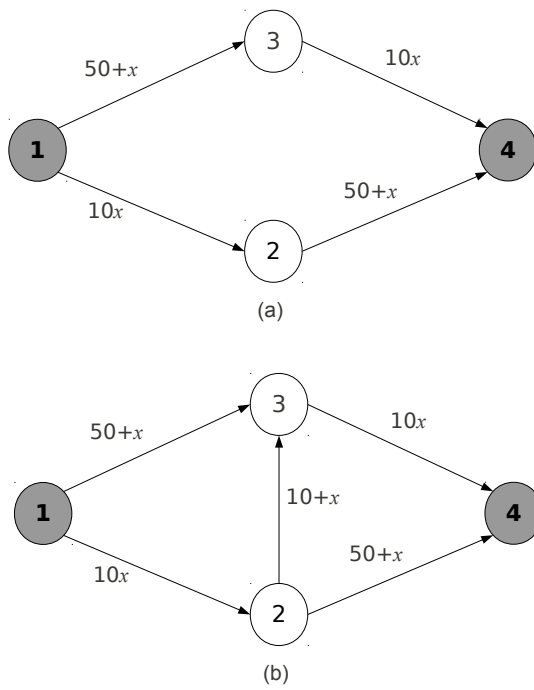


Figure 4.8: Braess network, before and after construction of a new link.

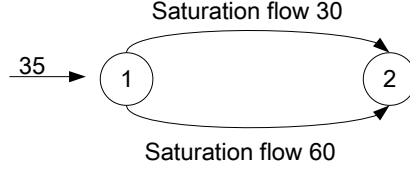


Figure 4.9: Network for Smith's paradox.

one of two links, which join at a signalized intersection at the destination. Assume that each of these links has the same free-flow time ($t_0^\uparrow = t_0^\downarrow = 1$ min), but that the saturation flow of the bottom link is twice that of the top link ($s^\uparrow = 30$ veh/min and $s^\downarrow = 60$ veh/min). Vehicles enter the network at a demand level of $d = 35$ veh/min. The signal has a cycle length of $C = 1$ min and, for simplicity's sake, assume that there is no lost time so that the green times allocated to the top and bottom links equal the cycle length: $G^\uparrow + G^\downarrow = 1$.

In traditional traffic signal analysis, the *capacity* u of an approach is the saturation flow scaled by the proportion of green time given to that approach, so $u_i = s_i(G_i/C)$ where i can refer to either the top or bottom link. The *degree of saturation* X for an approach is the ratio of the link flow to the capacity, so $X_i = x_i/u_i = (x_i C)/(s_i G_i)$. With these quantities, the total travel time on each link (free-flow time plus signal delay) can be written as

$$t_i = 1 + \frac{9}{20} \left[\frac{C(1 - G_i/C)^2}{1 - x_i/s_i} + \frac{X_i^2}{x_i(1 - X_i)} \right]. \quad (4.12)$$

Assume that initially, the signal is timed such that $G^\uparrow = 40$ sec and $G^\downarrow = 20$ sec. Then the formulas for delay on the top and bottom links are functions of their flows alone, obtained from (4.12) by substituting the corresponding green times into the expressions for X_i , and the equilibrium solution can be obtained by solving the equations $t^\uparrow = t^\downarrow$ and $x^\uparrow + x^\downarrow = 35$ simultaneously. With these green times, the equilibrium solution occurs when $x^\uparrow = 23.6$ veh/min and $x^\downarrow = 11.4$ veh/min, and the reader can confirm that the travel times on the two links are $t^\uparrow = t^\downarrow = 2.11$ min.

So far, so good. Now assume that it has been a long time since the signal was last retimed, and a traffic engineer decides to check on the signal and potentially change the timing. A traditional rule in traffic signal timing is that the green time given to an approach should be proportional to the degree of saturation. However, with the given solution, the degrees of saturation are $X^\uparrow = 0.982$ and $X^\downarrow = 0.953$, which are unequal — the top link is given slightly less green time than the equisaturation rule suggests, and the bottom link slightly more. Therefore, the engineer changes the green times to equalize these degrees of saturation, which occurs if $G^\uparrow = 48.3$ sec and $G^\downarrow = 11.7$ sec, a smallish adjustment.

If drivers could be counted on to remain on their current routes, all would be well. However, changing the signal timing changes the delay formulas (4.12) on the two links. Under the assumption that drivers always seek the shortest path, the equilibrium solution will change as drivers swap from the (now longer) path to the (now shorter) path. Re-equating the travel time formulas, the flow rates on the top and bottom links are now $x^\uparrow = 23.8$ veh/min and $x^\downarrow = 11.2$ veh/min, with equal delays $t^\uparrow = t^\downarrow = 2.26$ min on each link. Delay has actually increased, because the signal re-timing (aimed at reducing delay) did not account for the changes in driver behavior after the fact.

A bit surprised by this result, our diligent traffic engineer notes that the degrees of saturation are still unequal, with $X^\uparrow = 0.984$ and $X^\downarrow = 0.959$, actually further apart than before the first adjustment. Undeterred, the engineer changes the signal timings again to $G^\uparrow = 48.5$ s and $G^\downarrow = 11.5$ s, which results in equal degrees of saturation under the new flows. But by changing the green times, the delay equations have changed, and so drivers re-adjust to move toward the shorter path, leading to $x^\uparrow = 23.9$ veh/min and $x^\downarrow = 11.1$ veh/min, and new delays of 2.43 minutes on each approach, even higher than before!

You can probably guess what happens from here, but Table 4.2 tells the rest of the story. As our valiant engineer stubbornly retimes the signals in a vain attempt to maintain equisaturation, flows always shift in response. Furthermore, the delays grow faster and faster, asymptotically growing to infinity as more and more adjustments are made. The moral of the story? *Changing the network will change the paths that drivers take*, and “optimizing” the network without accounting for how these paths will change is naive at best, and counterproductive at worst.

Table 4.2: Evolution of the network as signals are iteratively retimed.

Iteration	G^\uparrow (s)	G^\downarrow	x^\uparrow (v/min)	x^\downarrow	t^\uparrow (min)	t^\downarrow	X^\uparrow	X^\downarrow
0	48	12	23.6	11.4	2.11	2.11	0.982	0.953
1	48.3	11.7	23.8	11.2	2.26	2.26	0.984	0.959
2	48.5	11.5	23.9	11.1	2.43	2.43	0.986	0.965
3	48.7	11.3	24.1	10.9	2.63	2.63	0.988	0.969
4	48.9	11.1	24.2	10.8	2.86	2.86	0.99	0.974
5	49.1	10.9	24.3	10.7	3.12	3.12	0.991	0.977
10	49.5	10.5	24.7	10.3	5.11	5.11	0.996	0.989
20	49.88	10.12	24.91	10.09	16.58	16.58	0.9988	0.9971
50	49.998	10.002	24.998	10.002	855.92	855.93	0.99998	0.99995
∞	50	10	25	10	∞	∞	1	1

4.4 Constructing a Static Traffic Assignment Network

Solving the traffic assignment problem on a network requires several pieces of data. This section briefly describes some of the issues involved in collecting this data, and in calibrating such a model. At a high level, three types of data are needed: (i) the network topology, (ii) the link performance functions, and (iii) the demand (OD) matrix. Each of these has different considerations, and is discussed in turn.

Before discussing these specific data types, it is crucial to remember that *traffic assignment models are used to predict future, hypothetical conditions*. As sensing technologies become cheaper and more ubiquitous, it is becoming easier and easier to know the *current* and *historical* states of a traffic network, but that is not the domain of traffic assignment. Rather, we are using these models to *predict* or *anticipate* how changes in the network (demand or supply) will affect congestion and other metrics of interest, and thereby provide policy and resource allocation recommendations. Knowing the current and historical patterns on the network is important in building such a model, but they are not sufficient to make these decisions on their own.

Selecting a network topology means deciding what the nodes and links are in your network. This involves questions of geographic scope and of detail. In terms of geographic scope, how large of an area do you need to model? The networks used by planning organizations commonly involve an entire metropolitan area: a major city, its surrounding suburbs, and some of the adjoining rural areas. Such a network can be re-used for many applications in this metropolitan area. Networks can also be purpose-built for a specific application, and in this case they often cover a smaller area. It is important for the geographic scale to be large enough to include the main route alternatives being considered in your application. If there is no route choice, the traffic assignment model will not tell you much of anything meaningful.

In terms of detail, you need to decide which links and nodes are important enough to model. One option is to include *all* of the roadway links within the geographic scope you have chosen. This is the simplest choice, and there are tools designed to automatically generate networks this way, using open data sources. However, in most cities the great majority of these links are neighborhood streets and other local roads. These links are usually not of interest: they carry little traffic volume and are rarely congested. Including all of these will often increase the size of your network by an order of magnitude, with little added value. On the other hand, highways, freeways, and major arterials clearly need to be included. Minor arterials are often included as well.

Many of the node locations will be determined once the links are chosen, at the points where they intersect. Additional nodes may be created to serve as demand centroids, a notion first introduced in Section 1.4. Considerations related to centroids are discussed later in this section, along with the OD matrix. If centroids are used, you will also need to add *centroid connector* links joining

each centroid to the links representing the physical transportation infrastructure. Care should be taken to place centroid connectors in a way that represents where travelers starting or ending trips in the neighborhood represented by the centroid will actually get on or off of the main roadway network. This is an underappreciated point of network construction: the choice of connectors (how many and where) can play a surprisingly large role in assignment results. This is even more true in dynamic traffic assignment, as will be discussed more in Part III.

Selecting link performance functions means choosing a function $t_{ij}(x_{ij})$ for each link (i, j) , giving its travel time in terms of the number of vehicles trying to use that link. (As stated in Section 1.5, although the word “flow” is commonly used to describe x_{ij} in a static model, it is better to think of it as a “demand” which can possibly exceed the capacity of the link.) The unit of x_{ij} is in vehicles over the entire modeling horizon. If your study is for one peak hour, then x_{ij} is in vehicles per hour; if you are modeling a three-hour peak period, then x_{ij} is in vehicles per three hours; and so forth. This is important to keep in mind: many delay formulas, such as signal delay formulas in the Highway Capacity Manual or other professional references, make assumptions about the unit of flow, and failing to adapt these formulas to the units of x_{ij} will cause major errors.

The gold standard is to estimate these functions on each link based on observed conditions on that specific link, over a long enough time period to estimate how demand relates to travel time on that link. There are several complications in this process. First, networks for a large metropolitan area commonly include tens of thousands of links, and you may not have data on all of them. Freeways and major arterials are often equipped with permanent sensors recording flow, speed, and other traffic variables, but other links may not be. Second, congested traffic behaves in a complex way. Sensor data during *uncongested* times usually shows a clear trend between flow and volume, but during congested times there is considerable scatter in the data. This is because of phenomena like stop-and-go oscillations and hysteresis which are visible in traffic flow observations, but not simple to model, especially not with a link performance function (which is crude relative to state-of-the-art traffic flow theories). Finally, traffic sensors can easily record speed (from which travel time can be inferred) and volume, but measuring *demand* is quite a bit harder. Observed volumes can never exceed the roadway capacity, by definition. But demand — what the x_{ij} variables really represent — certainly can.

For all of these reasons, the most common practice is to adopt a small number of functional forms that can be applied to specific links. One example that we’ve already seen is the Bureau of Public Roads function

$$t_{ij} = t_{ij}^0 \left(1 + \alpha \left(\frac{x_{ij}}{u_{ij}} \right)^\beta \right), \quad (4.13)$$

where t_{ij}^0 and u_{ij} are the free-flow travel time and capacity on link (i, j) , and α and β are shape parameters. The advantage of this function is that it can be applied with relatively little data on each link: free-flow travel time can

be estimated as the link length divided by its speed limit, and capacity can be estimated by multiplying the number of lanes by a standard value (such as 2100 vehicles per hour per lane for a freeway). Professional references contain more sophisticated ways to estimate free-flow speed and capacity if you have information such as lane length, interchange density, and so on. When available, sensor data can also provide estimates of t_{ij}^0 and u_{ij} .

The shape parameters are commonly chosen as $\alpha = 0.15$ and $\beta = 4$, although some researchers have suggested other values may be better choices. In particular, the traditional values of 0.15 and 4 were estimated with u_{ij} being the “practical capacity” of the link, which is an obsolete term used in a different way than “capacity” is used in contemporary transportation. To modern transportation professionals, *capacity* refers to the maximum flow rate that can be sustained on a link, roughly corresponding to a level of service E. *Practical capacity*, by contrast, describes a state when speeds are about 15% lower than free-flow, roughly corresponding to a level of service C. Practical capacity is about 80 percent of the true capacity of the link. It is a mistake to use $\alpha = 0.15$ and $\beta = 4$ without also using “practical capacity” (by reducing capacity by 20%). **This mistake is surprisingly common in practice, and will lead to a systematic underestimation of congestion.** If you are using the true capacity for u_{ij} , Horowitz (1991) recommends $\alpha = 0.83$ and $\beta = 5.5$ for freeways, and Dowling et al. (1997) recommend $\alpha = 0.05$ and $\beta = 10$ for arterials, and $\alpha = 0.20$ and $\beta = 10$ for freeways.

Equation (4.13) is not the only functional form for a link performance functions. Other popular choices are the conical function

$$t_{ij} = t_{ij}^0 \left[2 + \sqrt{\beta^2 \left(1 - \frac{x_{ij}}{u_{ij}}\right)^2 + \gamma^2} - \beta \left(1 - \frac{x_{ij}}{u_{ij}}\right) - \gamma \right], \quad (4.14)$$

where β is the same shape parameter used in equation (4.13) but γ is fixed at $(2\beta - 1)/(2\beta - 2)$, and the Akcelik function

$$t_{ij} = t_{ij}^0 + \frac{1}{4}T \left[\left(\frac{x_{ij}}{u_{ij}} - 1 \right) + \sqrt{\left(\frac{x_{ij}}{u_{ij}} - 1 \right)^2 + \frac{8Jx_{ij}}{Tu_{ij}^2}} \right], \quad (4.15)$$

with T the length of the analysis period and J a parameter representing queueing dynamics on the link¹. See Section 4.5 for additional references on different link performance functions.

The OD matrix is the most challenging input to prepare, for several reasons. Unlike the network topology and physical characteristics of the roadway, the demand pattern is not directly observable. Sensors on links can report volumes, speeds, and so forth, but cannot report vehicle origins or destinations. There are emerging techniques for observing demand patterns directly from cell phone, GPS, or Bluetooth traces, but there are complications — for instance, trip starts

¹? recommends $J = 0.1$ for freeways; for arterials the situation is more complex and depends on the type and spatial density of junction controls along the link.

and ends are often “fuzzed” for privacy reasons, but this is exactly the data we need for an OD matrix. Such data is also not fully representative of the traveling population. There are many *travel demand models* that estimate an OD matrix from publically-available demographic data such as census records, often calibrated using travel surveys. These methods are beyond the scope of this book, but Section 4.5 provides some references for interested readers.

Many have hoped for methods that can directly estimate an OD matrix from observed traffic volumes on links. The main problem with this is *overfitting*. The OD matrix grows with the square of network size (an entry for every pair of zones), whereas the number of links that can provide observations grows linearly. This means that estimating an OD matrix from link volumes is greatly underdetermined in practical networks, and in general there are infinitely many OD matrices which can match a given set of link volumes. In other words, simply matching link volumes is not hard — rather, the problem is that it’s too easy. Only one of these infinite number of OD matrices is actually the correct one which will respond in the right way when the network changes (remember, the goal of transportation network models is to evaluate counterfactual scenarios for guiding infrastructure decisions and policy). Therefore, *estimating an OD matrix solely from observed link flows cannot be recommended*. You are likely to find an OD matrix which fits the observed data very well, but responds in a very different way when anything changes.

A better strategy is to use observed link flows or other field observations to improve on the OD matrix produced by another travel demand model. One way to do this is discussed in Chapter 7. The idea is to use a travel demand model to produce a “seed” OD matrix. We then solve for equilibrium using this matrix, and compare the predicted link flows to field observations. The seed OD matrix can then be adjusted in a way that improves the fit with observed link flows, while not moving too far way from the seed point. This retains much of the travel behavior which is embedded in travel demand models, and results in a better OD matrix to use for scenario comparisons.

4.5 Historical Notes and Further Reading

(These sections are incomplete in this beta version of the text, and will be substantially expanded in the complete first edition.)

Beckmann et al. (1956) were the first to pose the static traffic assignment assignment problem in its full generality, although similar concepts were expressed earlier in Pigou (1920) and Wardrop (1952).

The first motivating example is actually that of Pigou’s in the previous reference; it is referred to as the “Knight-Pigou-Downs” paradox as the same effect was discovered by Knight (1924) and Downs (1962). The second example was first described by Braess (1969), and an up-to-date list of research and popular works further exploring this paradox is maintained at <https://homepage.ruhr-uni-bochum.de/Dietrich.Braess/#paradox>. The third example is adapted from Smith (1979b).

4.6 Exercises

1. [15] Expand the diagram of Figure 1.4 to include a “government model” which reflects public policy and regulation regarding tolls. What would this “government” agent influence, and how would it be influenced?
2. [23] How realistic do you think link performance functions are? Name at least two assumptions they make about traffic congestion, and comment on how reasonable you think they are.
3. [23] How realistic do you think the principle of user equilibrium is? Name at least two assumptions it makes (either about travelers or congestion), and comment on how reasonable you think they are.
4. [12] Develop analogies to the equilibrium principle to represent the following phenomena: (a) better restaurants tend to be more crowded; (b) homes in better school districts tend to be more expensive; (c) technologies to improve road safety (such as antilock brakes) can increase reckless driving. Specify what assumptions you are making in these analogies.
5. [34] (*All-or-nothing solutions are corner points.*) The point \mathbf{x} is a *corner point* of the convex set X if it is impossible to write $\mathbf{x} = \lambda\mathbf{y} + (1 - \lambda)\mathbf{z}$ for $\mathbf{y} \in X$, $\mathbf{z} \in X$, and $\lambda \in (0, 1)$. Show that if H is the set of feasible path flows, \mathbf{h} is a corner point if and only if it is an *all-or-nothing solution* (a solution where all of the flow from each OD pair is on a single path). Repeat for the set of feasible link flows.
6. [14] There are three routes available to travelers between a single OD pair. For each case below, you are given the travel time t on each route, as well as the number of travelers x using each route. Indicate whether each case satisfies the principle of user equilibrium. If this principle is violated, explain why.
 - (a) $t_1 = 45$, $t_2 = 60$, and $t_3 = 30$ while $x_1 = 0$, $x_2 = 50$, and $x_3 = 0$.
 - (b) $t_1 = 45$, $t_2 = 30$, and $t_3 = 30$ while $x_1 = 0$, $x_2 = 50$, and $x_3 = 25$.
 - (c) $t_1 = 45$, $t_2 = 30$, and $t_3 = 30$ while $x_1 = 0$, $x_2 = 50$, and $x_3 = 0$.
 - (d) $t_1 = 15$, $t_2 = 30$, and $t_3 = 45$ while $x_1 = 30$, $x_2 = 0$, and $x_3 = 0$.
 - (e) $t_1 = 15$, $t_2 = 30$, and $t_3 = 45$ while $x_1 = 30$, $x_2 = 20$, and $x_3 = 10$.
 - (f) $t_1 = 15$, $t_2 = 30$, and $t_3 = 30$ while $x_1 = 0$, $x_2 = 50$, and $x_3 = 30$.
7. [26] The network in Figure 4.10 has 8 nodes, 12 links, and 4 zones. The travel demand is $d^{13} = d^{14} = 100$, $d^{23} = 150$, and $d^{24} = 50$. The dashed links have a constant travel time of 10 minutes regardless of the flow on those links; the solid links have the link performance function $15 + x/20$ where x is the flow on that link.

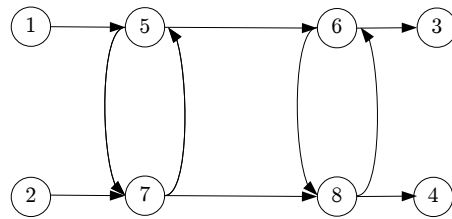


Figure 4.10: Network for Exercise 7.

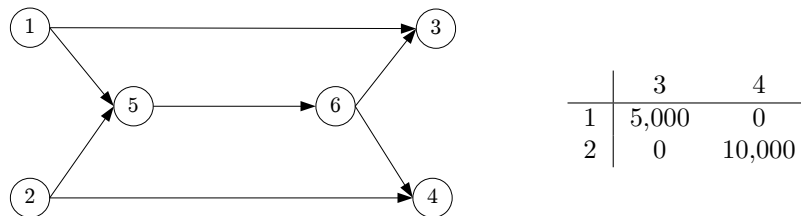


Figure 4.11: Network and OD matrix for Exercise 8.

- (a) For each of the four OD pairs with positive demand, list all acyclic paths connecting that OD pair. In total, how many such paths are in the network?
 - (b) Assume that the demand for each OD pair is divided evenly among all of the acyclic paths you found in part (a) for that OD pair. What are the resulting link flow vector \mathbf{x} , travel time vector \mathbf{t} , and path travel time vector \mathbf{C} ?
 - (c) Does that solution satisfy the principle of user equilibrium?
 - (d) What is the total system travel time?
8. [32] Consider the network and OD matrix shown in Figure 4.11. The travel time on *every* link is $10 + x/100$, where x is the flow on that link. Find the link flows and link travel times which satisfy the principle of user equilibrium.
 9. [36] Find the equilibrium path flows, path travel times, link flows, and link travel times on the Braess network (Figure 4.8b) when the travel demand from node 1 to node 4 is (a) 2; (b) 7; and (c) 10.
 10. [52] In the Knight-Pigou-Downs network, the simple and seemingly reasonable heuristic to improve the congested link (i.e., changing its link performance function to lower the free-flow time) did not help. Can you identify an equally simple heuristic that would lead you to improve the other link? (Ideally, such a heuristic would be applicable in many kinds of networks, not just one with an uncongestible link.)

11. [20] Give nontechnical explanations of why uniqueness, efficiency, and existence of equilibrium solutions have practical implications, not just theoretical ones. Concrete examples may be helpful.
12. [77] In the trial-and-error method, identify several different strategies for choosing the initial set of paths. Test these strategies on networks of various size and complexity. What conclusions can you draw about the effectiveness of these different strategies, and the amount of effort they involve?
13. [55] The trial-and-error method generally involves the solution of a system of nonlinear equations. *Newton's method* for solving a system of n nonlinear equations is to move all quantities to one side of the equation, expressing the resulting equations in the form $\mathbf{F}(\mathbf{x}) = \mathbf{0}$ where \mathbf{F} is a vector-valued function mapping the n -dimensional vector of unknowns \mathbf{x} to another n -dimensional vector giving the value of each equation. An initial guess is made for \mathbf{x} , which is then updated using the rule $\mathbf{x} \leftarrow \mathbf{x} - (J\mathbf{F}(\mathbf{x}))^{-1}\mathbf{F}(\mathbf{x})$, where $(J\mathbf{F}(\mathbf{x}))^{-1}$ is the inverse of the Jacobian matrix of \mathbf{F} , evaluated at \mathbf{x} . This process continues until (hopefully) \mathbf{x} converges to a solution. A *quasi-Newton method* approximates the Jacobian with a diagonal matrix, which is equal to the Jacobian along the diagonal and zero elsewhere, which is much faster to calculate and convenient to work with. Extend your experiments from Exercise 12 to see whether Newton or quasi-Newton methods work better.
14. [84] What conditions on the link performance functions are needed for Newton's method (defined in the previous exercise) to converge to a solution of the system of equations? What about the quasi-Newton method? Assume that the network consists of n parallel links connecting a single origin to a single destination. Can you guarantee that the solution to the system of equations only involves real numbers, and not complex numbers? *Hint: it may be useful to redefine the link performance functions for negative x values. This should not have any effect on the ultimate equilibrium solution, since negative link flows are infeasible, but cleverly redefining the link performance functions in this region may help show convergence.*
15. [96] Repeat the previous exercise, but for a general network with any number of links and nodes, and where paths may overlap.

Chapter 5

The Traffic Assignment Problem

This chapter formalizes the user equilibrium traffic assignment problem defined in Chapter 4. Using the mathematical language from Chapter 3, we are prepared to model and solve equilibrium problems even on networks of realistic size, with tens of thousands of links and nodes. Section 5.1 begins with fixed point, variational inequality, and optimization formulations of the user equilibrium problem. Section 5.2 then introduces important existence and uniqueness properties of the user equilibrium assignment, as was first introduced with the small two-player games of Section 1.3. Section 5.3 names several alternatives to the user equilibrium rule for assigning traffic to a network, including the system optimal principle, and the idea of bounded rationality. This chapter concludes with Section 5.4, exploring the inefficiency of the user equilibrium rule and unraveling the mystery of the Braess paradox.

5.1 Mathematical Formulations

The previous section introduced a trial-and-error method for solving the user equilibrium problem. To solve equilibrium on larger networks, we need something more sophisticated than trial-and-error, but doing so requires a more convenient representation of the principle of user equilibrium in mathematical notation, using the tools defined in the preceding chapters. This section does so by first formulating the equilibrium problem as the solution to a variational inequality; as the solution to a fixed point problem based on the variational inequality; and as the solution to a convex optimization problem. The reason for presenting all three of these formulations (rather than just one) is that each is useful in different ways. Just as true fluency in a language requires being able to explain the same concept in different ways, fluency in traffic assignment problems requires familiarity with all of these ways of understanding the principle of user equilibrium. (It is also possible to formulate the principle of user

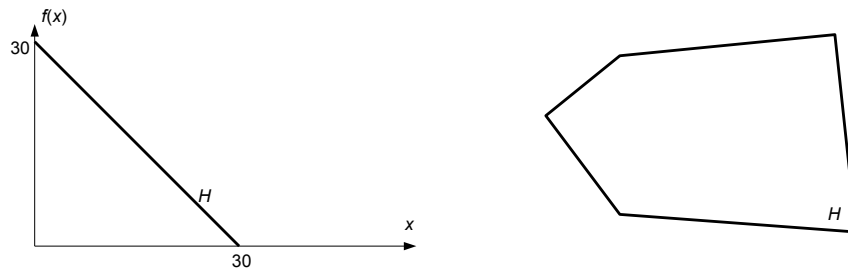


Figure 5.1: Feasible path-flow sets H for the simple two-link network (left) and a schematic representing H for more complex problems (right).

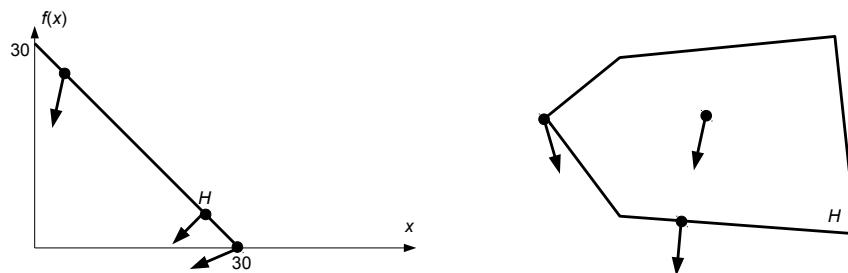


Figure 5.2: “Force” vectors $-\mathbf{c}$ for the simple two-link network (left) and a schematic representing H for more complex problems (right).

equilibrium as a fixed point problem without referring to a variational inequality through the use of multifunctions, as described in the optional subsection 5.1.1.)

The variational inequality formulation proceeds as follows. For each feasible path flow vector, we can associate a direction representing travelers’ desire for lower travel time paths. Figure 5.1 shows the feasible set for the two-link equilibrium problem on the left, and a conceptual schematic of the feasible set for the general equilibrium problem on the right. Notice that in all cases the feasible set is convex and has no “gaps.” For each feasible vector \mathbf{h} , we can calculate the path travel time vector \mathbf{c} associated with these path flows. Associate with each point a vector pointing in the *opposite* direction as \mathbf{c} , that is, each point \mathbf{h} is associated with the direction $-\mathbf{c}(\mathbf{h})$. You can think about this as a force acting on the point \mathbf{h} . The interpretation is that the path flow vector is “pulled” in the direction of decreasing travel times, and we will shortly show that an equilibrium is a point which is unmoved by this force. Of course, this intuitive interpretation must be established mathematically as well, and we will show that a vector \mathbf{h} satisfies the principle of user equilibrium if, and only if, the force criterion is satisfied.

A few points are shown in Figure 5.2 as examples. Recall that a non-corner point is unmoved by such a force only if the force is perpendicular to the bound-

ary of the feasible set at that point. At a corner point, the force must make a right or obtuse angle with all boundary directions. So we can characterize stable path flow vectors $\hat{\mathbf{h}}$ as points where the force $-\mathbf{c}(\hat{\mathbf{h}})$ makes a right or obtuse angle with any feasible direction $\mathbf{h} - \hat{\mathbf{h}}$, where \mathbf{h} is any other feasible path flow vector. Recalling vector operations, saying that two vectors make a right or obtuse angle with each other is equivalent to saying that their dot product is nonpositive. Thus, $\hat{\mathbf{h}}$ is a stable point if it satisfies

$$-\mathbf{c}(\hat{\mathbf{h}}) \cdot (\mathbf{h} - \hat{\mathbf{h}}) \leq 0 \quad \forall \mathbf{h} \in H \quad (5.1)$$

or, equivalently,

$$\mathbf{c}(\hat{\mathbf{h}}) \cdot (\hat{\mathbf{h}} - \mathbf{h}) \leq 0 \quad \forall \mathbf{h} \in H. \quad (5.2)$$

This is a variational inequality (VI) in the form shown in Section 3.2. We now prove that solutions of this VI correspond to equilibria, as shown by the next result:

Theorem 5.1. *A path flow vector $\hat{\mathbf{h}}$ solves the variational inequality (5.2) if and only if it satisfies the principle of user equilibrium.*

Proof. The theorem can equivalently be written “a path flow vector $\hat{\mathbf{h}}$ does not solve (5.2) if and only if it does not satisfy the principle of user equilibrium,” which is easier to prove. Assume $\hat{\mathbf{h}}$ does not solve (5.2). Then there exists some $\mathbf{h} \in H$ such that $\mathbf{c}(\hat{\mathbf{h}}) \cdot (\hat{\mathbf{h}} - \mathbf{h}) > 0$, or equivalently $\mathbf{c}(\hat{\mathbf{h}}) \cdot \hat{\mathbf{h}} > \mathbf{c}(\hat{\mathbf{h}}) \cdot \mathbf{h}$. Now, $\mathbf{c}(\hat{\mathbf{h}}) \cdot \hat{\mathbf{h}}$ is the total system travel time when the path flows are $\hat{\mathbf{h}}$, and $\mathbf{c}(\hat{\mathbf{h}}) \cdot \mathbf{h}$ is the total system travel time if the travel times were held constant at $\mathbf{c}(\hat{\mathbf{h}})$ even when the flows changed to \mathbf{h} . For the latter to be strictly less than the former, switching from $\hat{\mathbf{h}}$ to \mathbf{h} must have reduced at least one vehicle’s travel time even though the path travel times did not change. This can only happen if that vehicle was not on a minimum travel time path to begin with, meaning that $\hat{\mathbf{h}}$ does not satisfy user equilibrium.

Conversely, assume that $\hat{\mathbf{h}}$ is not a user equilibrium. Then there is some OD pair (r, s) and path π such that $\hat{h}_\pi > 0$ even though $c_\pi(\hat{\mathbf{h}}) > \min_{\pi' \in \Pi^{rs}} c_{\pi'}(\hat{\mathbf{h}})$. Let $\hat{\pi}$ be a minimum travel time path for this OD pair. Create a new path flow vector \mathbf{h} which is the same as $\hat{\mathbf{h}}$ except that \hat{h}_π is reduced by some small positive amount ϵ and $\hat{h}_{\hat{\pi}}$ is increased by ϵ . As long as $0 < \epsilon < \hat{h}_\pi$ the new point \mathbf{h} remains feasible. By definition all the components of $\hat{\mathbf{h}} - \mathbf{h}$ are equal to zero, except the component for π is ϵ and the component for $\hat{\pi}$ is $-\epsilon$. So $\mathbf{c}(\hat{\mathbf{h}}) \cdot (\hat{\mathbf{h}} - \mathbf{h}) = \epsilon(c_\pi(\hat{\mathbf{h}}) - c_{\hat{\pi}}(\hat{\mathbf{h}})) > 0$, so $\hat{\mathbf{h}}$ does not solve (5.2). \square

This variational inequality formulation leads directly to a fixed point formulation, as was discussed in Section 3.2. Theorem 5.1 shows that the equilibrium path flow vectors are exactly the solutions of the VI (5.2), that is, the stable points with respect to the fictitious force $-\mathbf{c}$. Therefore, an equilibrium vector $\hat{\mathbf{h}}$ is a fixed point of the function $\text{proj}_H(\mathbf{h} - \mathbf{c}(\mathbf{h}))$, that is, for all equilibrium solutions $\hat{\mathbf{h}}$ we have

$$\hat{\mathbf{h}} = \text{proj}_H(\hat{\mathbf{h}} - \mathbf{c}(\hat{\mathbf{h}})). \quad (5.3)$$

Finally, user equilibrium can also be shown as the solution to a convex optimization problem. The main decision variables are the path flows \mathbf{h} , and we need to choose one from the set of feasible assignments H ; the trick is identifying an appropriate function $f(\mathbf{h})$ in terms of the path flows, which is minimized when the path flows satisfy the principle of user equilibrium. So, the constraint set of our optimization problem consists of the equations defining H :

$$\sum_{\pi \in \Pi^{rs}} h^\pi = d^{rs} \quad \forall (r, s) \in Z^2 \quad (5.4)$$

$$h^\pi \geq 0 \quad \forall \pi \in \Pi \quad (5.5)$$

Different assignment rules will lead to different objective functions. It turns out that the objective function corresponding to the principle of user equilibrium has a somewhat unintuitive form. Therefore, we will derive this function in the same way that it was originally derived, by working backwards. *Rather than writing down the optimization problem, and then deriving the optimality conditions, we start by writing down the optimality conditions we want, then determining what kind of optimization problem has that form.*

Writing the (unknown!) objective function as $f(\mathbf{h})$, using the procedures described in Chapter 3 we can Lagrangianize the constraints (5.4) introducing multipliers κ^{rs} for each OD pair, providing the following optimality conditions:

$$\frac{\partial f}{\partial h^\pi} - \kappa^{rs} \geq 0 \quad \forall (r, s) \in Z^2, \pi \in \Pi^{rs} \quad (5.6)$$

$$h^\pi \left(\frac{\partial f}{\partial h^\pi} - \kappa^{rs} \right) = 0 \quad \forall (r, s) \in Z^2, \pi \in \Pi^{rs} \quad (5.7)$$

$$\sum_{\pi \in \Pi^{rs}} h^\pi = d^{rs} \quad \forall (r, s) \in Z^2 \quad (5.8)$$

$$h^\pi \geq 0 \quad \forall \pi \in \Pi \quad (5.9)$$

The last two of these are simply (5.4) and (5.5), requiring that the solution be feasible. The condition (5.7) is the most interesting of these, requiring that the product of each path's flow and another term involving f must be zero. For this product to be zero, either h^π must be zero, or $\frac{\partial f}{\partial h^\pi} = \kappa^{rs}$ must be true; and by (5.6), for all paths $\frac{\partial f}{\partial h^\pi} \geq \kappa^{rs}$. That is, in a solution to this problem, whenever h^π is positive we have $\frac{\partial f}{\partial h^\pi} = \kappa^{rs}$; if a path is unused, then $\frac{\partial f}{\partial h^\pi} \geq \kappa^{rs}$ — in other words, at optimality all used paths connecting OD pair (r, s) must have equal and minimal $\frac{\partial f}{\partial h^\pi}$, which is equal to κ^{rs} . According to the principle of user equilibrium, all used paths have equal and minimal travel time... so if we can choose a function $f(\mathbf{h})$ such that $\frac{\partial f}{\partial h^\pi} = c^\pi$, we are done!

So, the objective function must involve some integral of the travel times. A first guess might look something like

$$f(\mathbf{h}) = \sum_{\pi \in \Pi} \int_?^? c^\pi dh^\pi, \quad (5.10)$$

where the bounds of integration and other details are yet to be determined. The trouble is that c^π is not a function of h^π alone: the travel time on a path depends on the travel times on other paths as well, so the partial derivative of this function with respect to h^π will not simply be c^π , but contain other terms as well. However, these interactions are not arbitrary, but instead occur where paths overlap, that is, where they share common links. In fact, if we try writing a similar function to our guess but in terms of *link flows*, instead of *path flows*, we will be done.

To be more precise, let $\mathbf{x}(\mathbf{h})$ be the link flows as a function of the path flows \mathbf{h} , as determined by equation (4.2). Then the function

$$f(\mathbf{h}) = \sum_{(i,j) \in A} \int_0^{x_{ij}(\mathbf{h})} t_{ij}(x) dx \quad (5.11)$$

satisfies our purposes. To show this, calculate the partial derivative of f with respect to the flow on an arbitrary path π , using the fundamental theorem of calculus and the chain rule:

$$\frac{\partial f}{\partial h^\pi} = \sum_{(i,j) \in A} t_{ij}(x_{ij}(\mathbf{h})) \frac{\partial x_{ij}}{\partial h^\pi} = \sum_{(i,j) \in A} \delta_{ij}^\pi t_{ij}(x_{ij}(\mathbf{h})) = c^\pi \quad (5.12)$$

where the last two equalities respectively follow from differentiating (4.2) and from (4.4).

Finally, we can clean up the notation a bit by simply introducing the link flows \mathbf{x} as a new set of decision variables, adding equations (4.2) as constraints to ensure they are consistent with the path flows. This gives the following optimization problem, first formulated by Martin Beckmann:

$$\min_{\mathbf{x}, \mathbf{h}} \sum_{(i,j) \in A} \int_0^{x_{ij}} t_{ij}(x) dx \quad (5.13)$$

$$\text{s.t. } x_{ij} = \sum_{\pi \in \Pi} h^\pi \delta_{ij}^\pi \quad \forall (i, j) \in A \quad (5.14)$$

$$\sum_{\pi \in \Pi^{rs}} h^\pi = d^{rs} \quad \forall (r, s) \in Z^2 \quad (5.15)$$

$$h^\pi \geq 0 \quad \forall \pi \in \Pi \quad (5.16)$$

Section 5.2 shows that the objective function is convex, and the feasible region is a convex set, so this problem will be relatively easy to solve.

5.1.1 Multifunctions and application to network equilibrium (*)

(This section is optional and can be skipped. However if you are curious about the “other road” to using fixed points to show equilibrium existence, read this section to learn about multifunctions and Kakutani’s theorem.)

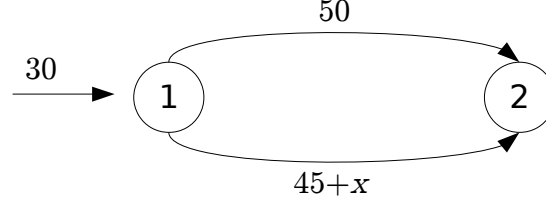


Figure 5.3: A simple two-link network for demonstration.

In the transit ridership example of Section 3.1, we could formulate the solution to the problem as a fixed point problem directly, without first going through a variational inequality. The traffic assignment problem is slightly more complex, because at equilibrium all used routes will have equal travel time. So, if we try to apply the same technique as in the transit ridership problem, we will run into a difficulty with “breaking ties.” To see this concretely, consider the two-route network in Figure 5.3, and let \mathbf{h} and \mathbf{c} be the vectors of route flows and route travel times. As Figure 1.3 suggests, we can try to define two functions: $\mathbf{H}(\mathbf{c})$ gives the vector of path flows representing route choices *if the travel times were fixed at \mathbf{c}* , and $\mathbf{C}(\mathbf{h})$ gives the vector of path travel times when the path flows are \mathbf{h} . The function \mathbf{C} is clearly defined: simply calculate the cost on each route using the link performance functions. However, if both paths are used at equilibrium, then $c_1 = c_2$, which means that *any* path flow vector would be a valid choice for $\mathbf{H}(\mathbf{c})$. Be sure you understand this difficulty: because \mathbf{H} assumes that the travel times are fixed, if they are equal any path flow vector is consistent with our route choice assumptions. If we relax the assumption that the link performance functions are fixed, then \mathbf{H} is no simpler than solving the equilibrium problem in the first place.

To resolve this, we introduce the concept of a *multifunction*.¹ Recall that a regular function from X to Y associates each value in its domain X with exactly one value in the set Y . A multifunction, on the other hand, associates each value in the domain X with some subset of Y . If F is such a multifunction the notation $F : X \rightrightarrows Y$ can be used. For example, consider the multifunction $R(\mathbf{c})$. If $c_1 < c_2$ (and these travel times were fixed), then the only consistent path flows are to have everybody on path 1. Likewise, if $c_1 > c_2$, then everyone would have to be on path 2. Finally, if $c_1 = c_2$ then people could split in any proportion while satisfying the rule that drivers are using least-time paths. That

¹Also known by many other names, including *correspondence*, *point-to-set map*, *set-valued map*, or *set-valued function*.

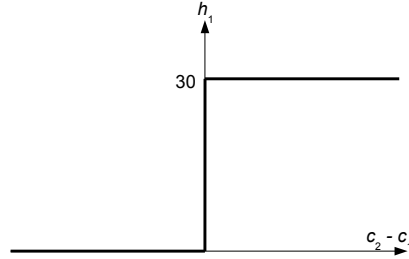


Figure 5.4: Multifunction corresponding to the two-link network; notice multiple values when $c_1 = c_2$.

is,

$$R(c_1, c_2) = \begin{cases} \left\{ \left[\begin{array}{cc} 30 & 0 \\ 0 & 30 \end{array} \right] \right\} & c_1 < c_2 \\ \left\{ \left[\begin{array}{cc} h & 30 - h \end{array} \right] : h \in [0, 30] \right\} & c_1 = c_2 \\ \left\{ \left[\begin{array}{cc} 0 & 30 \end{array} \right] \right\} & c_1 > c_2 \end{cases} \quad (5.17)$$

The notation in equation (5.17) is chosen very carefully to reflect the fact that H is a multifunction, which means that its values are *sets*, not a specific number. In the first two cases, the set only consists of a single element, so this distinction may seem a bit pedantic; but in the latter case, the set contains a whole range of possible values. In this way, multifunctions generalize the concept of a function by allowing R to take multiple “output” values for a single input. This is graphically represented in Figure 5.4, using the fact that the function can be parameterized in terms of a single variable $c_2 - c_1$.

Fixed-point problems can be formulated for multifunctions as well as for functions. If F is an arbitrary multifunction defined on the set K , then a fixed point of F is a point x such that $x \in F(x)$. Note the use of set inclusion \in rather than equality $=$ because F can associate multiple values with x . Just as Brouwer’s theorem guarantees existence of fixed points for functions, Kakutani’s theorem guarantees existence of fixed points for multifunctions under general conditions:

Theorem 5.2. (Kakutani). *Let $F : K \rightrightarrows K$ be a multifunction (from the set K to itself), where K is convex and compact. If F has a closed graph and $F(x)$ is nonempty and convex for all $x \in K$, then there is at least one point $x \in K$ such that $x \in F(x)$.*

The new terminology here is a *closed graph*; we say that the multifunction F has a closed graph if the set $\{(\mathbf{x}, \mathbf{y}) : \mathbf{x} \in X, \mathbf{y} \in F(\mathbf{x})\}$ is closed. Again, each of these conditions is necessary; you might find it helpful to visualize these conditions geometrically similar to Figure 3.1.

As a result of Kakutani’s theorem, we know that an equilibrium solution must always exist in the traffic assignment problem. Let H denote the set of

all feasible path flow vectors, that is, vectors \mathbf{h} such that $\sum_{\pi \in \Pi^{rs}} h^\pi = d^{rs}$ for all OD pairs (r, s) and $h^\pi \geq 0$ for all $\pi \in \Pi$. Let the function $\mathbf{C}(\mathbf{h})$ represent the path travel times when the path flows are \mathbf{h} .² Let the multifunction $R(\mathbf{c})$ represent the set of path flows which could possibly occur if all travelers chose least cost paths given travel times \mathbf{c} . Mathematically

$$R(\mathbf{c}) = \left\{ \mathbf{h} \in H : h^\pi > 0 \text{ only if } c^\pi = \min_{\pi' \in \Pi^{rs}} c^{\pi'} \text{ for all } (r, s) \in Z^2, \pi \in \Pi^{rs} \right\} \quad (5.18)$$

This is the generalization of equation (5.17) when there are multiple OD pairs and multiple paths connecting each OD pair — be sure that you can see how (5.17) is a special case of (5.18) when there is just one OD pair connected by two paths.

Theorem 5.3. *If the link performance functions t_{ij} are continuous, then at least one equilibrium solution exists to the traffic assignment problem.*

Proof. Let the multifunction F be the composition of the multifunction R and the function \mathbf{C} defined above, so $F(\mathbf{h}) = R(\mathbf{C}(\mathbf{h}))$ is the set of path flow choices which are consistent with drivers choosing fastest paths when the travel times correspond to path flows \mathbf{h} . An equilibrium path flow vector is a fixed point of F : if $\mathbf{h} \in F(\mathbf{h})$ then the path flow choices and travel times are consistent with each other. The set H of feasible \mathbf{h} is convex and compact by Lemma 5.1 in the next section. Examining the definition of R in (5.18), we see that for any vector \mathbf{c} , $R(\mathbf{c})$ is nonempty (since there is at least one path of minimum cost), and compact by the same argument used in the proof of Lemma 5.1. Finally, the graph of F is closed, as you are asked to show in the exercises. \square

5.2 Properties of User Equilibrium

This section uses the mathematical formulations from the previous section to explore basic properties of user equilibrium solutions. In Section 5.2.1 we give fairly general conditions under which there is one (and only one) user equilibrium link flow solution in a network. Although this section is mostly mathematical, these properties are actually of great practical importance. We ultimately want to use traffic assignment as a tool to help evaluate and rank transportation projects, using the principle of user equilibrium to predict network conditions. If it is possible that there is no feasible assignment which satisfies the principle of user equilibrium, then we would be at a loss as to how projects should be ranked, and we would need to find another assignment rule. Or, if there could be multiple feasible assignments which satisfy the principle of user equilibrium, again it is unclear how projects should be ranked.

Interestingly, while the conditions for a unique *link flow* user equilibrium are fairly mild, there will almost certainly be multiple *path flow* solutions which satisfy the principle of equilibrium — infinitely many, in fact. As explained in

²If you look back to Chapter 4, you will see that $\mathbf{C}(\mathbf{h}) = \mathbf{\Delta}^T \mathbf{t}(\mathbf{\Delta}\mathbf{h})$ in matrix notation.

Section 5.2.2, this suggests that the principle of user equilibrium is not strong enough to identify path flows, simply link flows. If path flows are needed to evaluate a project, an alternative approach is needed, and this section explains the concepts of maximum entropy and proportionality which provide this alternative.

Lastly, when solving equilibrium problems on large networks both the link flow and path flow representations have significant limitations — algorithms only using link flows tend to be slow, while algorithms only using path flows can require a large amount of memory and tend to return low-entropy path flow solutions. A compromise is to use the link flows, but distinguish the flow on each link by its origin or destination. While explored more in Chapter 6, Section 5.2.3 lays the groundwork by showing how flow can be decomposed this way, and that at equilibrium the links with positive flow from each origin or destination form an acyclic network.

5.2.1 Existence and link flow uniqueness

We would like to use the mathematical formulations from the previous section — variational inequality, fixed point, and convex optimization — to identify conditions under which one (and only one) user equilibrium solution exists. This section relies heavily on mathematical formulations and results proved previously; interested readers may refer to those sections for more detail, and others can still read the proofs here to obtain the general idea. Recall the following results from Chapter 3:

- **Brouwer’s Theorem 3.1:** Let X be a compact and convex set, and let $f : X \rightarrow X$ be a continuous function. Then there is at least one fixed point $x \in X$.
- **Proposition 3.3:** Let X be a convex set, and let $f : X \rightarrow \mathbb{R}$ be a strictly convex function. Then there is a unique global minimum \hat{x} of f on X .

Since all of these results rely on some properties of H and X , we establish them first:

Lemma 5.1. *The set of feasible path assignments H and the set of feasible link assignments X are both compact and convex.*

Proof. To show that H is compact, we must show that it is closed and bounded. The set H is defined by a combination of linear weak inequalities ($h^\pi \geq 0$) and linear equalities ($\sum h^\pi = d^{rs}$), so together Propositions A.3b and A.4a–b show that it is closed. For boundedness, consider any $\mathbf{h} \in H$, and OD pair (r, s) , and any path $\pi \in \Pi^{rs}$. We must have $h^\pi \leq d^{rs}$, since each h^π is nonnegative and $\sum_{\pi \in \Pi^{rs}} h^\pi = d^{rs}$. Therefore, if D is the largest entry in the OD matrix, we have $h^\pi \leq D$ for any path π . Therefore $|\mathbf{h}| \leq \sqrt{|\Pi|}D$, so H is contained in the ball $B_{\sqrt{|\Pi|}D}(\mathbf{0})$ and H is bounded.

Finally, for convexity, consider any $\mathbf{h}_1 \in H$, any $\mathbf{h}_2 \in H$, any $\lambda \in [0, 1]$, and the resulting vector $\mathbf{h} = \lambda\mathbf{h}_1 + (1 - \lambda)\mathbf{h}_2$. For any path π , $h_1^\pi \geq 0$ and $h_2^\pi \geq 0$,

so $h^\pi = \lambda h_1^\pi + (1 - \lambda)h_2^\pi \geq 0$ as well. Furthermore, for any OD pair (r, s) we have

$$\begin{aligned} \sum_{(r,s) \in \Pi^{rs}} h^\pi &= \sum_{(r,s) \in \Pi^{rs}} [\lambda h_1^\pi + (1 - \lambda)h_2^\pi] \\ &= \lambda \sum_{(r,s) \in \Pi^{rs}} h_1^\pi + (1 - \lambda) \sum_{(r,s) \in \Pi^{rs}} h_2^\pi \\ &= \lambda d^{rs} + (1 - \lambda)d^{rs} \\ &= d^{rs} \end{aligned}$$

so $\mathbf{h} \in H$ as well.

Every feasible link assignment $\mathbf{x} \in X$ is obtained from a linear transformation of some $\mathbf{h} \in H$ by (5.14) so X is also closed, bounded, and convex. \square

To show existence of solutions, we use Brouwer's Theorem based on the formulation of user equilibrium as a fixed point of the function $f(\mathbf{h}) = \text{proj}_H(\mathbf{h} - \mathbf{c}(\mathbf{h}))$.

Proposition 5.1. *If the link performance functions t_{ij} are continuous for each link $(i, j) \in A$, then there is a feasible assignment satisfying the principle of user equilibrium.*

Proof. Taking H as the set of feasible assignments, the range of the function $f(\mathbf{h}) = \text{proj}_H(\mathbf{h} - \mathbf{c}(\mathbf{h}))$ clearly lies in H because of the projection. By Lemma 5.1, H is compact and convex, so if we show that f is continuous, then Brouwer's Theorem guarantees existence of a fixed point. The discussion in Section 5.1 showed that each fixed point is a user equilibrium solution, which will be enough to prove the result.

We use the result that the composition of continuous functions is continuous (Proposition A.6). The function f is the composition of two other functions (call them f_1 and f_2), where $f_1(\mathbf{h}) = \text{proj}_H(\mathbf{h})$ and $f_2(\mathbf{h}) = \mathbf{h} - \mathbf{c}(\mathbf{h})$. By Proposition A.7, f_1 is continuous because H is a convex set. Furthermore, f_2 is continuous if \mathbf{c} is a continuous function of \mathbf{h} , which is true by hypothesis. Therefore, the conditions of Brouwer's Theorem are satisfied, and f has at least one fixed point (which satisfies the principle of user equilibrium). \square

The easiest way to show uniqueness of solutions is to make use of the convex optimization formulation, minimizing the function $f(\mathbf{x}) = \sum_{ij} \int_0^{x_{ij}} t_{ij}(x) dx$ over the set X .

Proposition 5.2. *If the link performance functions t_{ij} are differentiable, and $t'_{ij}(x) > 0$ for all x and links (i, j) , there is exactly one feasible link assignment satisfying the principle of user equilibrium.*

Proof. The previous section showed that user equilibrium link flow solutions correspond to minimum points of f on X . If we can show that f is strictly convex, then there can only be one minimum point. (Differentiability implies

continuity, so we can assume that at least one minimum point exists by Proposition 5.1). Since f is a function of multiple variables (each link's flow), to show that f is convex we can write its Hessian matrix of second partial derivatives. The first partial derivatives take the form $\frac{\partial f}{\partial x_{ij}} = t_{ij}(x_{ij})$. So, the diagonal entries of the Hessian take the form $\frac{\partial^2 f}{\partial x_{ij}^2} = t'_{ij}(x_{ij})$, while the off-diagonal entries take the form $\frac{\partial^2 f}{\partial x_{ij} \partial x_{k\ell}} = 0$. Since the Hessian is a diagonal matrix, and its diagonal entries are strictly positive by assumption, f is a strictly convex function. Therefore there is only one feasible link assignment satisfying the principle of user equilibrium. \square

It is also possible to prove the same result under the slightly weaker condition that the link performance functions are continuous and strictly increasing; this is undertaken in the exercises.

To summarize the above discussion, if the link performance functions are continuous, then at least one user equilibrium solution exists; if in addition the link performance functions are strictly increasing, then *exactly* one user equilibrium solution exists. For many problems representing automobile traffic, these conditions seem fairly reasonable, if not universally so: adding one more vehicle to the road is likely to increase the delay slightly, but not dramatically.

5.2.2 Path flow nonuniqueness, entropy, and proportionality

The previous subsection gave a relatively mild condition for the equilibrium solution to be unique in terms of link flows. However, a more subtle point is that the equilibrium solution need not be unique in terms of path flows. Consider the example shown in Figure 5.5. With the paths as numbered in the figure, the reader can verify that both $\mathbf{h}_1 = [20 \ 10 \ 20 \ 10]$ and $\mathbf{h}_2 = [30 \ 0 \ 10 \ 20]$ produce the equilibrium link flows, and therefore satisfy the principle of user equilibrium (all paths having equal travel time). Furthermore, any weighted average of \mathbf{h}_1 and \mathbf{h}_2 also produces the same link flows, so there are an infinite number of feasible path flow solutions which satisfy the principle of user equilibrium.

At first glance this may appear to contradict the proof of Proposition 5.2, which showed that the function f was strictly convex. Although f is strictly convex *as a function of \mathbf{x}* , it is *not* strictly convex as a function of \mathbf{h} . This is easy to see mathematically — if there are two distinct path flow solutions \mathbf{h}_1 and \mathbf{h}_2 which satisfy the principle of user equilibrium, then $\lambda\mathbf{h}_1 + (1 - \lambda)\mathbf{h}_2$ is an equilibrium as well for $\lambda \in (0, 1)$. Since all three are equilibria, $f(\mathbf{h}_1) = f(\mathbf{h}_2) = f(\lambda\mathbf{h}_1 + (1 - \lambda)\mathbf{h}_2)$ even though strict convexity would require $f(\lambda\mathbf{h}_1 + (1 - \lambda)\mathbf{h}_2) < \lambda f(\mathbf{h}_1) + (1 - \lambda)f(\mathbf{h}_2) = f(\mathbf{h}_1)$. From a more intuitive standpoint, in typical networks there are many more paths than links; therefore, it is very likely that multiple path flow solutions correspond to the same link flow solution. Since path travel times only depend on the values of the link flows (because link flows determine link travel times, which are added up to get

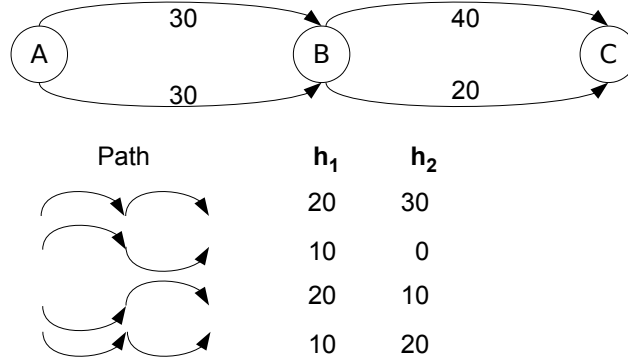


Figure 5.5: Nonuniqueness of equilibrium in terms of path flows

path travel times), and since the principle of user equilibrium is defined in terms of path travel times, it is reasonable to expect multiple path flow solutions to satisfy the equilibrium principle.

The existence of multiple path flow equilibria (despite a unique link flow equilibrium) is not simply a technical curiosity. Rather, it plays an important role in using network models to evaluate and rank alternatives. Thus far, we've been content to ask ourselves what the equilibrium link flows are, but in practice these numbers are usually used to generate other, more interesting measures of effectiveness such as the total system travel time and total vehicle-miles traveled. If a neighborhood group is worried about increased traffic, link flows can support this type of analysis. If we are concerned about safety, we can use link flows as one input in estimating crash frequency, and so forth.

Other important metrics, however, require more information than just the total number of vehicles on a link. Instead, we must know the entire *paths* used by drivers. Examples include

Select link analysis: In transportation planning parlance, a “select link analysis” goes beyond asking “how many vehicles are on a link,” to “exactly which vehicles are using this link?” This is used to produce nice visualizations, as well as to identify which origins and destinations are using a particular link. This is needed to identify which neighborhoods and regions are affected by transportation improvements, both positively and negatively.

Equity and environmental justice: As a special case of the above, planners are focusing more and more on the notion of equity. Most transportation projects involve both “winners” and “losers.” For instance, constructing a new freeway through an existing neighborhood may improve regional mobility, but cause significant harm to those whose neighborhood is disrupted in the process. Historically, disadvantaged populations tend

to bear the brunt of such projects. To know who is benefiting or suffering from a project, we need to know the origins and destinations of the travelers on different links.

Emissions: Vehicles emit more pollutants toward the start of trips, when the engine is cold, than later on once the catalytic converter has warmed up. Therefore, to use the output of a network model to predict emissions from vehicles on a link, ideally we need to distinguish between vehicles which have just started their trips from vehicles which have already traveled some distance. This requires knowing path flows.

Pavement loading: Heavy vehicles cause much more damage to pavement than lighter vehicles; a commonly-used relation in pavement engineering estimates that pavement damage is proportional to the *fourth power* of vehicle weight. If a network model is being used to forecast pavement deterioration, it is therefore important to know which links are being used by heavy vehicles. Again, this requires more than just the total flow on a link; we must know what specific type of vehicle it is (which often depends heavily on the origin-destination pair).

In fact, software vendors often discuss such abilities as key features of their applications. In this light, the fact that equilibrium path flows are nonunique should trouble you somewhat. If there are multiple path flow equilibria (and therefore multiple values for emissions forecasts, equity analyses, etc.), which one is the “right” one? Simply picking one possible path flow solution and hoping that it’s the right one is not particularly rigorous, and inappropriate for serious planning applications.

So, the principle of user equilibrium is only strong enough to determine link flows, not path flows, even though having a path flow solution would be very helpful in analyzing engineering alternatives. This has motivated the search for the “most likely path flow” solution which satisfies equilibrium. Even though user equilibrium is not a strong enough concept to determine path flows uniquely, we can still ask “of all the path flow vectors which satisfy the principle of user equilibrium, which do we think is most likely to occur?” This section describes the concepts most commonly used to this end. To my knowledge, there has been little to no research directly validating this principle, but nobody has come up with a better, generally agreed-upon solution to this issue.

A physical analogy is presented to motivate the idea. Figure 5.6 shows the locations of n gas molecules within a box. In the left panel, all n molecules happen to be located in the top half of the box, while in the right panel $n/2$ molecules are located in the top half and $n/2$ in the bottom half. Both of these situations are physically possible (that is, they satisfy the laws of mechanics, etc.) although you would certainly think that the scenario on the right is more likely or plausible than the scenario on the left. In exactly the same way, although there are many path flow solutions which satisfy the “law” of user equilibrium, we are not forced to conclude that all such solutions are equally likely to occur.

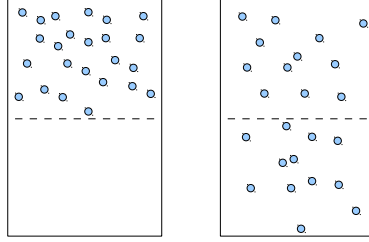


Figure 5.6: Molecules of an ideal gas in a box. Which situation is more likely?

Why does the scenario on the left seem so unlikely? If the molecules form an ideal gas, the location of each molecule is independent of the location of every other molecule. Therefore, the probability that any given molecule is in the top half of the box is $1/2$, and the probability that *all* n molecules are as in the left figure is

$$p^L = \left(\frac{1}{2}\right)^n. \quad (5.19)$$

To find the probability that the distribution of molecules is as in the right figure (p^R), we use the binomial distribution:

$$p^R = \binom{n}{n/2} \left(\frac{1}{2}\right)^{n/2} \left(\frac{1}{2}\right)^{n/2} = \frac{n!}{(n/2)!(n/2)!} \left(\frac{1}{2}\right)^n \quad (5.20)$$

Since $n! \gg [(n/2)!]^2$, we have $p^R \gg p^L$, that is, the situation on the right is much, much likelier than that on the left even though both are physically possible.

We want to use the same principle for path flows. The principle of user equilibrium assumes that each user is identical (in the sense that they choose routes only according to minimal travel time) and chooses routes independent of other drivers (except insofar as other drivers' choices affect travel times, drivers do not coordinate and strategize about their route choices with each other). Therefore, we can apply the same logic as with the gas molecules. For now, assume there is a single OD pair, with integer demand d , and let $\hat{\Pi} = \{\pi_1, \dots, \pi_k\}$ denote the set of minimum travel-time paths at the equilibrium solution. For simplicity assume that these paths have constant travel times independent of flow. Since drivers only care about travel time, each path is essentially identical, and the probability any particular driver chooses any particular path is $1/k$. The probability that the path flow vector \mathbf{h} takes a particular value $[h_1 \ \dots \ h_k]$ is then given by the multinomial distribution:

$$p = \frac{d!}{h_1!h_2! \dots h_k!} \left(\frac{1}{k}\right)^d \quad (5.21)$$

and the most likely path flow vector is the one which maximizes this product. Since $(1/k)^d$ is a constant, the most likely path flow vector simply maximizes

$$p = \frac{d!}{h_1!h_2!\cdots h_k!} \quad (5.22)$$

Now we introduce a common trick: since the logarithm function is strictly increasing, the path flows which maximize p also maximize $\log p$, which is

$$\log p = \log d! - \sum_{\pi=1}^k \log h_{\pi}! \quad (5.23)$$

To simplify further, we use Stirling's approximation, which states that $n! \approx n^n e^{-n} \sqrt{2\pi n}$, or equivalently $\log n! \approx n \log n - n + (1/2) \log(2\pi n)$. This approximation is asymptotically exact in the sense that the ratio between $n!$ and $n^n e^{-n} \sqrt{2\pi n}$ approaches 1 as n grows large. Further, when n is large, n is much larger than $\log n$, so the last term can be safely ignored and $\log n! \approx n \log n - n$. Substituting into (5.23) we obtain

$$\log p \approx (d \log d - d) - \sum_{\pi} (h_{\pi} \log h_{\pi} - h_{\pi}) \quad (5.24)$$

Since $d = \sum_{\pi} h_{\pi}$, we can manipulate (5.24) to obtain

$$\log p \approx - \sum_{\pi} h_{\pi} \log(h_{\pi}/d) \quad (5.25)$$

and the path flows h_1, \dots, h_k maximizing this quantity (subject to the constraint $d = \sum_{\pi} h_{\pi}$ and nonnegativity) approximately maximize the probability that this particular path flow vector will occur in the field if travelers are choosing routes independent of each other.

To move towards the traffic assignment problem we're used to, we need to make the following changes:

1. In the traffic assignment problem, the demand d and path flows h_{π} do not need to be integers, but can instead take on real values. The corresponding interpretation is that the demand is "divided" into smaller and smaller units, each of which is assumed to act independently of every other unit. (This is how user equilibrium works with continuous flows.) This doesn't cause any problems, and in fact helps us out: as we take the limit towards infinite divisibility, Stirling's approximation becomes exact — so we can replace the \approx in our formulas with exact equality.
2. There are multiple origins and destinations. This doesn't change the basic idea, the formulas just become incrementally more complex as we sum equations of the form (5.25) for each OD pair.
3. Travel times are not constant, but are instead flow dependent. Again, this doesn't change any basic ideas; we just have to add a constraint

stating that the path flows correspond to an equilibrium solution. Since the equilibrium link flows are unique, we simply have to state that the path flows correspond to the equilibrium link flows $\hat{\mathbf{x}}$.

Putting all this together, we seek the path flows \mathbf{h} which solve the optimization problem

$$\max_{\mathbf{h}} \quad - \sum_{(r,s) \in Z^2} \sum_{\pi \in \hat{\Pi}^{rs}} h_{\pi} \log(h_{\pi}/d^{rs}) \quad (5.26)$$

$$\text{s.t.} \quad \sum_{\pi \in \Pi} \delta_{ij}^{\pi} h_{\pi} = \hat{x}_{ij} \quad \forall (i, j) \in A \quad (5.27)$$

$$\sum_{\pi \in \hat{\Pi}^{rs}} h_{\pi} = d^{rs} \quad \forall (r, s) \in Z^2 \quad (5.28)$$

$$h_{\pi} \geq 0 \quad \forall \pi \in \Pi \quad (5.29)$$

The objective function (5.26) is called the *entropy* of a particular path flow solution, and the constraints (5.28), (5.27), and (5.29) respectively require that the path flows are consistent with the OD matrix, equilibrium link flows, and nonnegativity. The set $\hat{\Pi}^{rs}$ is the set of paths which are used by OD pair (r, s) .

The word *entropy* is meant to be suggestive. The connection with the thermodynamic concept of entropy may be apparent from the physical analogy at the start of this section.³ In physics, entropy can be interpreted as a measure of disorder in a system. Both the left and right panels in Figure 5.6 are “allowable” in the sense that they obey the laws of physics. However, the scenario in the right has much less structure and much higher entropy, and is therefore more likely to occur.

It is not trivial to solve the optimization problem (5.26)–(5.29). It turns out that entropy maximization implies a much simpler condition, proportionality among pairs of alternate segments. Any two paths with a common origin and destination imply one or more pairs of alternate segments where the paths diverge, separated by links common to both paths. In Figure 5.5, there are two pairs of alternate segments: the top and bottom links between nodes A and B, and the top and bottom links between B and C. It turns out that path flows \mathbf{h}_1 are the solution to the entropy maximization problem.

You might notice some regularity or patterns in this solution. For instance, looking only at the top and bottom links between A and B, at equilibrium these links have equal flow (30 vehicles each). Paths 1 and 3 are the same except for between nodes A and B, and these paths also have equal flow (20 vehicles each). The same is true for paths 2 and 4 (10 vehicles each). Or, more subtly, between B and C, the ratio of flows between the top and bottom link is 2:1, the same as the ratio of flows on paths 1 and 2 (which only differ between B and C), and the ratio of flows on paths 3 and 4. This is no coincidence; in fact, we can show that *entropy maximization implies proportionality*.

³Actually, the term here is more directly drawn from the fascinating field of information theory, which is unfortunately beyond the scope of this text.

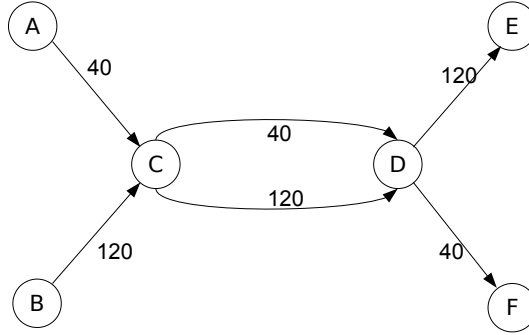


Figure 5.7: Multiple OD pair example of proportionality.

Before proving this fact, we show that it holds even for different OD pairs. The network in Figure 5.7 has 40 vehicles traveling from origin A to destination F, and 120 vehicles from B to E, and the equilibrium link flows are shown in the figure. Each OD pair has two paths available to it, one using the top link between C and D, and the other using the bottom link between C and D. Using an upward-pointing arrow to denote the first type of path, and a downward-pointing arrow to describe the second, the four paths are h_{AF}^{\uparrow} , h_{AF}^{\downarrow} , h_{BE}^{\uparrow} , and h_{BE}^{\downarrow} . Solving the optimization problem (5.26)–(5.29), we find the most likely path flows are $h_{AF}^{\uparrow} = 10$, $h_{AF}^{\downarrow} = 30$, $h_{BE}^{\uparrow} = 30$, and $h_{BE}^{\downarrow} = 90$. The equilibrium flows for the top and bottom links between C and D have a ratio of 1:3; you can see that this ratio also holds between h_{AF}^{\uparrow} and h_{AF}^{\downarrow} , as well as between h_{BE}^{\uparrow} and h_{BE}^{\downarrow} .

In particular, the obvious-looking solution $h_{AF}^{\uparrow} = 40$, $h_{AF}^{\downarrow} = 0$, $h_{BE}^{\uparrow} = 0$, $h_{BE}^{\downarrow} = 120$ has extremely low entropy, because it implies that for some reason all travelers from one OD pair are taking one path, and all travelers from the other are taking the other path, even though both paths are perceived identically by all travelers and even though travelers are making choices independent of each other. This is exceptionally unlikely.

We now derive the proportionality condition by defining it a slightly more precise way:

Theorem 5.4. *Let π_1 and π_2 be any two paths connecting the same OD pair. If the path flows \mathbf{h} solve the entropy-maximizing problem (5.26)–(5.29), then the ratio of flows h_1/h_2 for paths π_1 and π_2 is identical regardless of the OD pair these paths connect, and only depends on the pairs of alternate segments distinguishing these two paths.*

The proof is a bit lengthy, and is deferred to the end of the section. But even though the derivation is somewhat involved, the proportionality condition itself is fairly intuitive: when confronted with the same set of choices, travelers from different OD pairs should behave in the same way. The proportionality

condition implies nothing more than that the share of travelers choosing one alternative over another is the same across OD pairs. Proportionality is also a relatively easy condition to track and enforce.

It would be especially nice if proportionality can be shown fully equivalent to entropy maximization. Theorem 5.4 shows that entropy maximization implies proportionality, but is the converse true? Unfortunately, the result is no, and examples can be created which satisfy proportionality without maximizing entropy. Therefore, proportionality is a weaker condition than entropy maximization. The good news is that proportionality “gets us most of the way there.” In the Chicago regional network, there are over 93 million equal travel-time paths at equilibrium; after accounting for the equilibrium and “no vehicle left behind” constraints (5.27) and (5.28), there are still over 90 million degrees of freedom in how the path flows are chosen. Accounting for proportionality reduces the number of degrees of freedom to 91 (a reduction of 99.9999%!) So, in practical terms, enforcing proportionality seems essentially equivalent to maximizing entropy.

5.2.3 Aggregation by origins or destinations

Thus far, we have looked at flow representations in one of two ways: path flows \mathbf{h} or link flows \mathbf{x} . In a way, these can be thought of as two extremes. The path flow solution contains the most information, showing the exact path chosen by every traveler in the network. The price paid for this level of detail is the amount of memory needed. The number of paths in a typical network is very, very large and storing the entire vector of path flows is impractical. The link flows, on the other hand, are much more compact and provide enough information for many common measures of effectiveness (such as vehicle-hours or vehicle-miles traveled, and volumes or delays on individual links). The link flow vector can be thought of as an “aggregated” form of the path flow vector, where we lump together all the travelers using the same link (regardless of which specific path, they are using, or which specific OD pair they are from). Naturally, some information is lost in this process of aggregation. So, there is a tradeoff: path flow solutions contain a great deal of information, at the expense of requiring a large amount of memory; link flow solutions are more compact but contain less information.

So, it is natural to ask if there is an intermediate “Goldilocks” flow representation which contains more detail than link flow solutions, without invoking the combinatorial explosion in storage space associated with a path flow solution. One such representation involves aggregating all travelers departing the same origin together.⁴ On a link (i, j) , let x_{ij}^r be the flow on this link who left origin r , that is,

$$x_{ij}^r = \sum_{s \in Z} \sum_{\pi \in \Pi^{rs}} \delta_{ij}^{\pi} h^{\pi} \quad (5.30)$$

⁴Everything in this section would apply equally well to an aggregation by destination; the presentation from here on will be in terms of origins only to avoid repetition.

and let \mathbf{x}^r be the vector of link flows associated with origin r .

Many of the most efficient algorithms for solving the traffic assignment problem use such a representation, often called an *origin-based* or *bush-based* representation. The term “origin-based” describes how the aggregation occurs. The term “bush-based” will make more sense once we get to Chapter 6. Let $\hat{\mathbf{h}}$ be a feasible assignment satisfying the principle of user equilibrium, and let \mathbf{x}^r be the link flows associated with origin r at this solution.

It turns out that, at equilibrium, the links with positive x_{ij}^r values form an acyclic subnetwork. As a result, they have a topological order, which allows network calculations to be done much more rapidly than in general networks. This property is exploited heavily by bush-based traffic assignment algorithms, which are discussed in Section 6.4.

The following result shows that this must be true when each link’s travel time is strictly positive. With slightly more effort, a similar result can be shown even when there are zero-time links.

Proposition 5.3. *Let \mathbf{x}^r be the link flows associated with origin r at some feasible assignment satisfying the principle of user equilibrium. If $t_{ij} > 0$ for all links (i, j) , then the subset of arcs $A^r = \{(i, j) \in A : x_{ij}^r > 0\}$ with positive flow from origin r contains no cycle.*

Proof. By contradiction, assume that there is a cycle $[i_1, i_2, \dots, i_k, i_1]$, where $(i_1, i_2), (i_2, i_3), \dots, (i_k, i_1)$ are all in A^r . Let L_i^r represent the travel time on the shortest path from origin r to node i . Consider any link $(i, j) \in A^r$; this implies that $x_{ij}^r > 0$ and that some used path starting from origin r includes (i, j) . Because the principle of user equilibrium holds, this path must be a shortest path to some destination. Clearly $L_i^r + t_{ij} \geq L_j^r$ by the definition of the labels \mathbf{L} ; but if $L_i^r + t_{ij} > L_j^r$, then there is a shorter path to node j than any of them passing through node i , and such a path cannot be used at equilibrium. Therefore we must have $L_i^r + t_{ij} = L_j^r$ for all links in A^r ; since all travel times are strictly positive this implies $L_i^r < L_j^r$ for all $(i, j) \in A^r$. So, for the cycle under consideration, we have $L_1 < L_2 < \dots < L_k < L_1$, which is impossible. Therefore the links in A^r cannot contain a cycle. \square

Proof of Theorem 5.4:

Begin by forming the Lagrangian for the entropy-maximizing problem:

$$\begin{aligned} \mathcal{L}(\mathbf{h}, \boldsymbol{\beta}, \boldsymbol{\gamma}) = & - \sum_{(r,s) \in Z^2} \sum_{\pi \in \hat{\Pi}^{rs}} h_\pi \log \left(\frac{h_\pi}{d^{rs}} \right) + \sum_{(i,j) \in A} \beta_{ij} \left(\hat{x}_{ij} - \sum_{\pi \in \Pi} \delta_{ij}^\pi h_\pi \right) \\ & + \sum_{(r,s) \in Z^2} \gamma_{rs} \left(d^{rs} - \sum_{\pi \in \hat{\Pi}^{rs}} h_\pi \right) \end{aligned} \quad (5.31)$$

using β and γ to denote the Lagrange multipliers. Note that the nonnegativity constraint can be effectively disregarded since the objective function is only

defined for strictly positive \mathbf{h} . Therefore at the optimum solution the partial derivative of \mathcal{L} with respect to any path flow must vanish:

$$\frac{\partial \mathcal{L}}{\partial h_\pi} = -1 - \log \left(\frac{h_\pi}{d^{rs}} \right) - \sum_{(i,j) \in A} \delta_{ij}^\pi \beta_{ij} - \gamma_{rs} = 0 \quad (5.32)$$

where (r, s) is the OD pair connected by path π . Solving for h^π we obtain

$$h^\pi = d^{rs} \exp \left(-1 - \sum_{(i,j) \in A} \delta_{ij}^\pi \beta_{ij} - \gamma_{rs} \right) \quad (5.33)$$

Likewise we have

$$\frac{\partial \mathcal{L}}{\partial \gamma_{rs}} = d^{rs} - \sum_{\pi' \in \hat{\Pi}^{rs}} h_{\pi'} = 0 \quad (5.34)$$

so

$$d^{rs} = \sum_{\pi' \in \hat{\Pi}^{rs}} h_{\pi'} = d^{rs} \exp(-1 - \gamma_{rs}) \sum_{\pi \in \hat{\Pi}^{rs}} \exp \left(- \sum_{(i,j) \in A} \delta_{ij}^\pi \beta_{ij} \right) \quad (5.35)$$

substituting the result from (5.33). Therefore we can solve for γ_{rs} :

$$\gamma_{rs} = -1 + \log \left(\sum_{\pi' \in \hat{\Pi}^{rs}} \exp \left(- \sum_{(i,j) \in A} \delta_{ij}^{\pi'} \beta_{ij} \right) \right) \quad (5.36)$$

Finally, substituting (5.36) into (5.33) and simplifying, we obtain

$$h^\pi = \frac{d^{rs}}{\sum_{\pi' \in \hat{\Pi}^{rs}} \exp \left(- \sum_{(i,j) \in A} \delta_{ij}^{\pi'} \beta_{ij} \right)} \exp \left(- \sum_{(i,j) \in A} \delta_{ij}^\pi \beta_{ij} \right) \quad (5.37)$$

Noting that the fraction in (5.37) only depends on the OD pair (r, s) , we can simply write

$$h^\pi = K_{rs} \exp \left(- \sum_{(i,j) \in A} \delta_{ij}^\pi \beta_{ij} \right) \quad (5.38)$$

where K_{rs} is a constant associated with OD pair (r, s) .

Let A_1 be the set of links not in either π_1 or π_2 , A_2 the set of links common to both paths, A_3 the links in π_1 but not π_2 , and A_4 the links in π_2 but not π_1 .

Then the ratio h_1/h_2 can be written

$$\frac{h_1}{h_2} = \frac{K_{rs} \exp\left(-\sum_{(i,j) \in A} \delta_{ij}^{\pi_1} \beta_{ij}\right)}{K_{rs} \exp\left(-\sum_{(i,j) \in A} \delta_{ij}^{\pi_2} \beta_{ij}\right)} \quad (5.39)$$

$$= \frac{K_{rs} \exp\left(\begin{array}{l} -\sum_{(i,j) \in A_1} \delta_{ij}^{\pi_1} \beta_{ij} - \sum_{(i,j) \in A_2} \delta_{ij}^{\pi_1} \beta_{ij} \\ -\sum_{(i,j) \in A_3} \delta_{ij}^{\pi_1} \beta_{ij} - \sum_{(i,j) \in A_4} \delta_{ij}^{\pi_1} \beta_{ij} \end{array}\right)}{K_{rs} \exp\left(\begin{array}{l} -\sum_{(i,j) \in A_1} \delta_{ij}^{\pi_2} \beta_{ij} - \sum_{(i,j) \in A_2} \delta_{ij}^{\pi_2} \beta_{ij} \\ -\sum_{(i,j) \in A_3} \delta_{ij}^{\pi_2} \beta_{ij} - \sum_{(i,j) \in A_4} \delta_{ij}^{\pi_2} \beta_{ij} \end{array}\right)} \quad (5.40)$$

$$= \frac{K_{rs} \exp\left(-\sum_{(i,j) \in A_2} \delta_{ij}^{\pi_1} \beta_{ij} - \sum_{(i,j) \in A_3} \delta_{ij}^{\pi_1} \beta_{ij}\right)}{K_{rs} \exp\left(-\sum_{(i,j) \in A_2} \delta_{ij}^{\pi_2} \beta_{ij} - \sum_{(i,j) \in A_4} \delta_{ij}^{\pi_2} \beta_{ij}\right)} \quad (5.41)$$

$$= \frac{K_{rs} \exp\left(-\sum_{(i,j) \in A_2} \delta_{ij}^{\pi_1} \beta_{ij}\right) \exp\left(-\sum_{(i,j) \in A_3} \delta_{ij}^{\pi_1} \beta_{ij}\right)}{K_{rs} \exp\left(-\sum_{(i,j) \in A_2} \delta_{ij}^{\pi_2} \beta_{ij}\right) \exp\left(-\sum_{(i,j) \in A_4} \delta_{ij}^{\pi_2} \beta_{ij}\right)} \quad (5.42)$$

$$= \frac{\exp\left(-\sum_{(i,j) \in A_3} \delta_{ij}^{\pi_1} \beta_{ij}\right)}{\exp\left(-\sum_{(i,j) \in A_4} \delta_{ij}^{\pi_1} \beta_{ij}\right)} \quad (5.43)$$

where the steps of the derivation respectively involve substituting (5.37), expanding A into $A_1 \cup A_2 \cup A_3 \cup A_4$, using the definitions of the A_i sets to identify sums where $\delta_{ij}^{\pi_j} = 0$, splitting exponential terms, and canceling common factors.

Thus in the end we have

$$\frac{h_1}{h_2} = \frac{\exp\left(-\sum_{(i,j) \in A_3} \delta_{ij}^{\pi_1} \beta_{ij}\right)}{\exp\left(-\sum_{(i,j) \in A_4} \delta_{ij}^{\pi_1} \beta_{ij}\right)} \quad (5.44)$$

regardless of the OD pair h_1 and h_2 connect, and this ratio only depends on A_3 and A_4 , that is, the pairs of alternate segments distinguishing π_1 and π_2 .

5.3 Alternative Assignment Rules

The previous section described the most important assignment rule, the principle of user equilibrium. However, it is not the only possible assignment rule, and it is not hard to see that other factors may enter route choice beyond travel time, or that other assumptions of the user equilibrium model are not perfectly realistic. Over time researchers have introduced a number of alternative assignment rules that can be used. This text by necessity can only cover a few of these alternative assignment rules, and three representative rules were chosen: assignment with perception errors, system optimal assignment, and bounded rationality.

Assignment with perception errors relaxes the assumption that drivers have perfect knowledge of travel times. The interpretation is that every driver *perceives* a particular travel time on each path (which may or may not equal its actual travel time), and chooses a path that they believe to be shortest based on these perceptions. The net effect is that travelers are distributed across paths with different travel times, but are concentrated more on paths with lower travel times. (If a path is very much longer than the shortest one, it is unlikely that someone's perception would be so far off as to mistakenly think it is shortest.)

The mathematical tool most commonly used to model perception errors is the theory of *discrete choice* from economics. The resulting model is termed *stochastic user equilibrium*, and this important model is the focus of Section 8.3.

The other two alternative assignment rules are discussed next.

5.3.1 System optimal assignment

Imagine for a moment that route choice was taken out of the hands of individual travelers, and instead placed in the hands of a dictator who could assign each traveler a path that they would be required to follow. Suppose also that this dictator was benevolent, and wanted to act in a way to minimize average travel delay in the network. The resulting assignment rule results in the *system optimal* state.

While this scenario is a bit fanciful, the system optimal state is important for several reasons. First, it provides a theoretical lower bound on the delay in a network, a benchmark for the best performance that could conceivably be achieved that more realistic assignment rules can be compared to. Second, there are ways to actually achieve the system optimal state even without taking route choice out of the hands of travelers, if one has the ability to freely charge tolls or provide incentives on network links. Third, there are some network problems where a single agent can exert control over the routes chosen by travelers, as in certain logistics problems where a dispatcher can assign specific routes to vehicles.

At first, it may not be obvious that this assignment rule is meaningfully different from the user equilibrium assignment rule. After all, in user equilibrium each traveler is individually choosing routes, while in system optimum a single agent is choosing routes for everyone, but both are doing so to minimize travel times. To see why they might be different, consider the network used for the Knight-Pigou-Downs paradox in Section 1.5: a two-link network (Figure 4.6) with a constant travel time of 50 minutes on the top link, a link performance function $45 + x^\downarrow$ on the bottom link, and a total demand of 30 vehicles. The user equilibrium solution was $x^\uparrow = 25$, $x^\downarrow = 5$, with equal travel times of 50 minutes on both top and bottom routes.

So, in the user equilibrium state the average travel time is 50 minutes for all

vehicles. Is it possible to do better? We can write the average travel time as

$$\begin{aligned} \frac{1}{50} (50x^\uparrow + (45 + x^\downarrow)x^\downarrow) &= \frac{1}{50} (50(30 - x^\downarrow) + (45 + x^\downarrow)x^\downarrow) \\ &= \frac{1}{50} ((x^\downarrow)^2 - 5x^\downarrow + 1500) . \end{aligned} \quad (5.45)$$

This is a quadratic function which obtains its minimum at $x^\downarrow = 2.5$. At the solution $x^\uparrow = 27.5$, $x^\downarrow = 2.5$, the travel times are unequal ($t^\uparrow = 50$, $t^\downarrow = 47.5$), but the average travel time of 49.8 minutes is slightly less than the average travel time of 50 minutes at the user equilibrium solution.

Therefore, travelers individually choosing routes to minimize their own travel times may create more delay than would be obtained with central control. This reveals an important, but subtle point about user equilibrium assignment, a point important enough that Section 5.4 is devoted entirely to explaining this issue. So, we will not belabor the point here, but it will be instructive to start thinking about why the user equilibrium and system optimal states need not coincide.

To be more precise mathematically, the system optimal assignment rule chooses the feasible assignment minimizing the average travel time. Since the total number of travelers is a constant, we can just as well minimize the *total system travel time*, defined as

$$TSTT = \sum_{\pi \in \Pi} h^\pi c^\pi = \sum_{(i,j) \in A} x_{ij} t_{ij} \quad (5.46)$$

where it is not hard to show that calculating $TSTT$ through path flows and link flows produces the same value.

5.3.2 Perception errors

An alternative assignment rule tries to relax the assumption that travelers choose the path with exactly the shortest travel time. After all, this assumption implicitly requires *drivers to have perfect knowledge of the travel times on all routes in the network*. In reality, we know this is not true: do you know the travel times on literally *all* routes between an origin and destination? And can you accurately distinguish between a route with a travel time of 16 minutes, and one with a travel time of 15 minutes and 59 seconds? And perhaps travelers take other factors into account anyway, travel time may be just one of several criteria used to choose routes.

Both of these factors can be modeled using discrete choice concepts. In discrete choice models, an individual chooses one option from a set of alternatives to maximize his or her utility, consisting of both an observed portion and an unobserved portion. Rather than requiring that each driver follow the true shortest path between each origin and destination, we assume that drivers follow the path they *believe* to be shortest, but allow for some perception error between their belief and the actual travel times. An alternative, mathematically

equivalent, interpretation (explained below) is that drivers do in fact perceive travel times accurately, but care about factors other than travel time. This leads to another important traffic assignment model commonly called *stochastic user equilibrium* (SUE). We discuss this model briefly here, and at length in Section 8.3.

Consider a traveler leaving origin r for destination s . They must choose one of the paths π connecting r to s , that is, they must make a choice from the set $\Pi_{r,s}$. The most straightforward way to generalize the principle of user equilibrium to account for perception errors is to set the observed utility equal to the negative of path travel time, so

$$U^\pi = -c^\pi + \epsilon^\pi \quad (5.47)$$

with the negative sign indicating that *maximizing utility* for drivers means *minimizing travel time*. Assuming that the ϵ^π are independent, identically distributed Gumbel random variables, we can use the logit formula (8.37) to express the probability that path π is chosen:

$$p^\pi = \frac{\exp(-\theta c^\pi)}{\sum_{\pi' \in \Pi^{r,s}} \exp(\theta C_{\pi'})} \quad (5.48)$$

The comments in the previous section apply to the interpretation of this formula. As θ approaches 0, drivers' perception errors are large relative to the path travel times, and each path is chosen with nearly equal probability. (The errors are so large, the choice is essentially random.) As θ grows large, perception errors are small relative to path travel times, and the path with lowest travel time is chosen with higher and higher probability. At any level of θ , there is a strictly positive probability that each path will be taken.

For concreteness, the route choice discussion so far corresponds to the interpretation where the unobserved utility represents perception errors in the utility. The other interpretation would mean that ϵ^π represents factors other than travel time which affect route choice (such as comfort, quality of scenery, etc.). Either of these interpretations is mathematically consistent with the discussion here.

The fact that the denominator of (8.39) includes a summation over *all* paths connecting r to s is problematic, both theoretically and practically. From a theoretical standpoint, it implies that drivers are considering literally every path between the origin and destination, even when this path is very circuitous or illogical (driving all around town when going to a store a half mile away). Presumably no driver would ever choose such a path, no matter how ill-informed they are about travel conditions or how poorly they can estimate travel times — yet the logit formula (8.39) suggests that some travelers will indeed choose such paths. From a practical standpoint, evaluating the formula (8.39) first requires *enumerating* all of these paths. Since the number of paths grows exponentially with the network size, any approach which requires an explicit listing of all the paths in a network will not scale to realistic-sized problems.

To address this fact, we can restrict the choice set somewhat. Rather than using all paths $\Pi^{r,s}$ connecting r to s , we can restrict path choices to a subset

of *reasonable paths* (denoted $\hat{\Pi}^{rs}$). There are different ways to identify sets of reasonable paths, but they should address both the theoretical and practical difficulties in the previous paragraph. That is, the paths in $\hat{\Pi}^{rs}$ correspond to a plausible-sounding behavioral principle (why might travelers only consider paths in this set) and lead to a formula which is efficiently computable even in large networks. There may be some tension between these ideas, in that there may be very efficient formulas which do not correspond to realistic choices, or that the most realistic models of reasonable paths may not lead to an efficient formula. Section 8.3 discusses these in more detail.

So, equation (8.39) leads us an expression for path flows in terms of travel times, which is the assignment rule filling the place of the question mark in Figure 4.2. To be explicit, the formula for path flows as a function of path travel times is

$$h^\pi = d^{rs} \frac{\exp(-\theta c^\pi)}{\sum_{\pi' \in \Pi^{rs}} \exp(\theta C_{\pi'})} \quad (5.49)$$

where (r, s) is the OD pair corresponding to path π . The complete traffic assignment problem with this assignment rule can then be expressed as follows: find a feasible path flow vector \mathbf{h}^* such that $\mathbf{h}^* = \mathbf{H}(\mathbf{C}(\mathbf{h}^*))$. This is a standard fixed-point problem. Clearly \mathbf{H} and \mathbf{c} are continuous functions if the link performance functions are continuous, and the feasible path set is compact and convex, so Brouwer's theorem immediately gives existence of a solution to the SUE problem.

Notice that this was much easier than showing existence of an equilibrium solution to the original traffic assignment problem! For that problem, there was no equivalent of (8.73). Travelers were all using shortest paths, but if there were two or more shortest paths there was no rule for how those ties should be broken. As a result, we had to reformulate the problem as a variational inequality and introduce an auxiliary function based on movement of a point under a force. With this assignment rule, there is no need for such machinations, and we can write down the fixed point problem immediately.

That said, it *is* also possible to formulate the assignment problem with this rule as the solution to a variational inequality, and as the solution to a convex minimization problem. However, the objective function is rather complicated and computationally expensive to evaluate, and thus it is less useful for identifying an equilibrium point. The objective function is

$$z(\mathbf{x}) = - \sum_{(r,s) \in Z^2} d^{rs} E \left[\min_{\pi \in \Pi^{rs}} U^\pi \right] + \sum_{(i,j) \in A} x_{ij} t_{ij} - \sum_{(i,j) \in A} \int_0^{x_{ij}} t_{ij}(x) dx \quad (5.50)$$

where the expectation is taken with respect to the “unobserved” random variables ϵ , and U^π and t_{ij} in the first two terms are understood to be functions of x_{ij} . It can be shown that this function is convex, and therefore that the solution is unique. Unfortunately, this function is less useful for actually providing a solution. The first term in (8.112) involves an expectation over all paths connecting an OD pair, and as a result requires enumerating all paths, which is impractical in large networks.

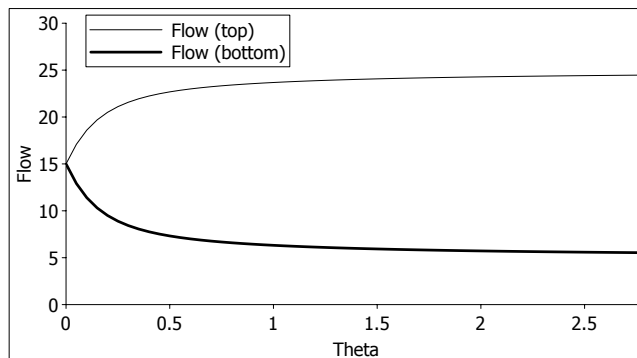


Figure 5.8: Flows with different perception errors.

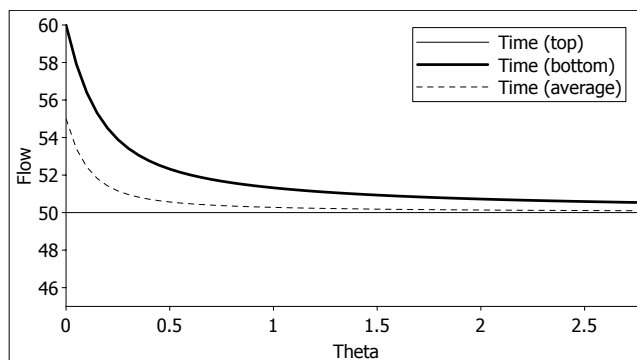


Figure 5.9: Travel times with different perception errors.

To demonstrate what this assignment rule looks like, refer again to the network in Figure 5.3. Figures 5.8 and 5.9 show how the flows and travel times in the network vary with θ . A few observations worth noting: first, the travel times on the two links are *not* equal, due to the presence of perception errors. However, as drivers' perceptions become more accurate (θ increasing), the travel times on the two paths become closer, and the solution asymptotically approaches the user equilibrium solution.

5.3.3 Bounded rationality (*)

(This optional section discusses an assignment rule which relaxes the assumption that all travelers are on the shortest path.)

An alternative assignment rule is based on another variation of shortest path-seeking behavior. Under *bounded rationality*, drivers will choose paths that are within some threshold of the true shortest path travel time. The behavioral rationale is that a driver will switch paths from a longer path to a shorter one, but only if the new path is sufficiently faster than their current choice. Perhaps

a driver is unlikely to switch paths to save, say, a tenth of a second in travel time: the burden of learning a new path outweighs any travel time savings which are possible.

There is some evidence from behavioral economics that humans often make choices in a boundedly rational way. Specifically in network assignment, bounded rationality can explain observed reactions to changes in a network in a way that the principle of user equilibrium cannot. Of course, these advantages must be balanced against other concerns: there may be many distinct bounded rational assignments, and efficient methods for finding bounded rational assignments in large networks have not yet been developed.

The boundedly rational user equilibrium (BRUE) problem can be stated as follows. Let ϵ represent some tolerance value in terms of travel time, and let κ^{rs} be the travel time on the shortest path connecting origin r to destination s .

Definition 5.1. (Principle of boundedly rational user equilibrium [BRUE].) *Every used path has travel time no more than ϵ in excess of the travel time on the shortest path connecting these nodes. That is, the feasible assignment $\mathbf{h} \in H$ is a BRUE if, for each $(r, s) \in Z^2$ and $\pi \in \Pi^{rs}$, $h^\pi > 0$ implies $c^\pi \leq \kappa^{rs} + \epsilon$.*

As an example, consider once again the network in Figure 5.3, but now assume we are seeking a BRUE where the tolerance is $\epsilon = 2$ minutes. That is, travelers are willing to use any path as long as it is within one minute of being the shortest path. The user equilibrium solution $x^\uparrow = 25$, $x^\downarrow = 5$ is clearly a BRUE as well. However, there are others: if $x^\uparrow = 26$, $x^\downarrow = 4$, then the travel times on the top and bottom paths are 50 and 49 minutes, respectively. Even though the majority of travelers (those on the top link) are not on the shortest path, the travel time on their path is close enough to the shortest path that the situation is acceptable, and this solution is BRUE as well. However, the flows $x^\uparrow = 28$, $x^\downarrow = 2$ are not BRUE: the travel times are 50 and 47 minutes, and the difference between the (used) top path and the shortest path (3 minutes) exceeds the tolerance value (2 minutes). You should convince yourself that $x^\uparrow \in [23, 27]$ and $x^\downarrow = 30 - x^\uparrow$ characterize all of the BRUE solutions in this simple network.

To come up with a mathematical formulation of BRUE assignment, it is helpful to introduce an auxiliary variable ρ^π indicating the amount by which the travel time on path π falls short of the maximum acceptable travel time on a path. Specifically, define

$$\rho^\pi = [\kappa^{rs} + \epsilon - c^\pi]^+ \quad \forall (r, s) \in Z^2, \pi \in \Pi^{rs} \quad (5.51)$$

So, in the example of the previous paragraph, if $\epsilon = 1$, $x^\uparrow = 26$ and $x^\downarrow = 4$, then $t^\uparrow = 50$, $t^\downarrow = 49$, and thus $\rho^\uparrow = [49 + 2 - 50]^+ = 1$ and $\rho^\downarrow = [49 + 2 - 49]^+ = 2$. If the path π is unacceptable ($c^\pi > \kappa^{rs} + \epsilon$), then $\rho^\pi = 0$ by (5.51). We can then define the *effective travel time* \hat{c}^π as

$$\hat{c}^\pi = c^\pi + \rho^\pi. \quad (5.52)$$

These new quantities are helpful because the principle of BRUE can now be formulated in a more familiar way:

Proposition 5.4. *A feasible assignment $\mathbf{h} \in H$ is a BRUE if and only if every used path has equal and minimal effective travel time, that is, for each $(r, s) \in Z^2$ and $\pi \in \Pi^{rs}$, $h^\pi > 0$ implies $\hat{c}^\pi = \min_{\pi' \in \Pi^{rs}} \hat{c}^{\pi'}$.*

Proof. Let $\mathbf{h} \in H$ satisfy the principle of BRUE, and consider any OD pair (r, s) and path $\pi \in \Pi^{rs}$ with $h^\pi > 0$. Since \mathbf{h} is BRUE, $c^\pi \leq \kappa^{rs} + \epsilon$, so $\rho^\pi = \kappa^{rs} + \epsilon - c^\pi$ and $\hat{c}^\pi = c^\pi + \kappa^{rs} + \epsilon - c^\pi = \kappa^{rs} + \epsilon$. But for any $\pi' \in \Pi^{rs}$, $\hat{c}^{\pi'} = c^{\pi'} + \rho^{\pi'} \geq c^{\pi'} + \kappa^{rs} + \epsilon - c^{\pi'} = \kappa^{rs} + \epsilon$, so $\hat{c}^\pi = \min_{\pi' \in \Pi^{rs}} \hat{c}^{\pi'}$.

Contrariwise, let $\mathbf{h} \in H$ be such that $h^\pi > 0$ implies $\hat{c}^\pi = \min_{\pi' \in \Pi^{rs}} \hat{c}^{\pi'}$, and consider any OD pair (r, s) . Let $\hat{\pi}$ be a shortest path connecting r to s , so $\hat{c}^{\hat{\pi}} = \kappa^{rs} + [\kappa^{rs} + \epsilon - \kappa^{rs}]^+ = \kappa^{rs} + \epsilon$ and this must be minimal among all effective travel times on paths connecting r to s . For any path $\pi \in \Pi^{rs}$ with $h^\pi > 0$, we thus have $\hat{c}^\pi = \kappa^{rs} + \epsilon$, so $c^\pi = \hat{c}^\pi - \rho^\pi = \kappa^{rs} + \epsilon - \rho^\pi \leq \kappa^{rs} + \epsilon$ since $\rho^\pi \geq 0$. Thus \mathbf{h} satisfies the principle of BRUE as well. \square

Proposition 5.4 may give you hope that we can write a convex optimization problem whose solutions correspond to BRUE. Unfortunately, this is not the case, for reasons that will be shown shortly. However, we can write a variational inequality where both the path flows \mathbf{h} and the auxiliary variables $\boldsymbol{\rho}$ are decision variables. In what follows, let $\kappa^{rs}(\mathbf{h}) = \min_{\pi \in \Pi^{rs}} c^\pi(\mathbf{h})$ and, with slight abuse of notation, let κ^π refer to the κ^{rs} value corresponding to the path π , and $\boldsymbol{\kappa}$ be the vector of κ^π values.

Proposition 5.5. *A vector of path flows $\hat{\mathbf{h}} \in H$ is a BRUE if there exists a vector of nonnegative auxiliary costs $\hat{\boldsymbol{\rho}} \in \mathbb{R}_+^{|\Pi|}$ such that*

$$\begin{bmatrix} \hat{\mathbf{c}}(\hat{\mathbf{h}}, \hat{\boldsymbol{\rho}}) \\ \mathbf{c}(\hat{\mathbf{h}}) + \hat{\boldsymbol{\rho}} - \boldsymbol{\kappa}(\hat{\mathbf{h}}) - \epsilon \end{bmatrix} \cdot \begin{bmatrix} \hat{\mathbf{h}} - \mathbf{h} \\ \hat{\boldsymbol{\rho}} - \boldsymbol{\rho} \end{bmatrix} \leq 0 \quad (5.53)$$

for all $\mathbf{h} \in H$, $\boldsymbol{\rho} \in \mathbb{R}_+^{|\Pi|}$.

Proof. Assume that $(\hat{\mathbf{h}}, \hat{\boldsymbol{\rho}})$ solve the variational inequality (5.53). The first component of the VI $\hat{\mathbf{c}}(\hat{\mathbf{h}}, \hat{\boldsymbol{\rho}}) \cdot (\hat{\mathbf{h}} - \mathbf{h}) \leq 0$ shows that all used paths have equal and minimal effective cost, and by Proposition 5.4 satisfy the principle of BRUE, assuming that $\boldsymbol{\rho}$ is consistent with (5.51). To show this, consider the second component of the VI: $(\boldsymbol{\kappa}(\hat{\mathbf{h}}) + \epsilon - \hat{\boldsymbol{\rho}}) \cdot \hat{\boldsymbol{\rho}} - \boldsymbol{\rho} \leq 0$, or equivalently

$$\sum_{\pi \in \Pi} (c^\pi(\hat{\mathbf{h}}) + \hat{\rho}_\pi - \kappa^\pi(\hat{\mathbf{h}}) - \epsilon) \hat{\rho}_\pi \leq \sum_{\pi \in \Pi} (c^\pi(\hat{\mathbf{h}}) + \hat{\rho}_\pi - \kappa^\pi(\hat{\mathbf{h}}) - \epsilon) \rho_\pi \quad (5.54)$$

for all $\boldsymbol{\rho} \in \mathbb{R}_+^{|\Pi|}$. If $\hat{\rho}_\pi > 0$ for any path, inequality (5.54) can be true only if $\rho^\pi = \kappa^\pi(\hat{\mathbf{h}}) + \epsilon - c^\pi(\hat{\mathbf{h}})$; or, if $\hat{\rho}_\pi = 0$, then inequality (5.54) can be true only if $\rho^\pi = 0 \leq \kappa^\pi(\hat{\mathbf{h}}) + \epsilon - c^\pi(\hat{\mathbf{h}})$. In either case (5.51) is satisfied.

In the reverse direction, assume that $\hat{\mathbf{h}}$ is BRUE. Choose $\hat{\boldsymbol{\rho}} \in \mathbb{R}_+^{|\Pi|}$ according to (5.51). By Proposition 5.4 surely $\hat{\mathbf{c}}(\hat{\mathbf{h}}, \hat{\boldsymbol{\rho}}) \cdot (\hat{\mathbf{h}} - \mathbf{h}) \leq 0$ for all $\mathbf{h} \in H$. For any path where $\rho^\pi = \kappa^\pi(\hat{\mathbf{h}}) + \epsilon - c^\pi(\hat{\mathbf{h}})$, clearly $\sum_{\pi \in \Pi} (c^\pi(\hat{\mathbf{h}}) + \hat{\rho}_\pi - \kappa^\pi(\hat{\mathbf{h}}) -$

$\epsilon)(\hat{\rho}_\pi - \rho^\pi) = 0$; and for any path where $\rho^\pi = 0$, $\kappa^\pi(\hat{\mathbf{h}}) + \epsilon - c^\pi(\hat{\mathbf{h}}) \geq 0$ and $\sum_{\pi \in \Pi} (c^\pi(\hat{\mathbf{h}}) + \hat{\rho}_\pi - \kappa^\pi(\hat{\mathbf{h}}) - \epsilon)(\hat{\rho}_\pi - \rho^\pi) \geq 0$. In either case the second component of the VI is satisfied as well. \square

This BRUE formulation has been helpful in explaining observed changes in network flows after a disruption. For instance, assume that a link is removed from the network due to a disaster of some sort, and that flows adjust towards a new equilibrium in the network without the affected link. When the link is restored, flows will adjust again. If the principle of user equilibrium is true, the flows will move back to exactly the same values as before. However, in practice there has been some “stickiness” observed, and not all drivers will return to the same routes they were initially on. The BRUE framework provides a logical explanation for this: when the network is disrupted, certain drivers were forced to choose new paths. When the network is restored, they will only switch back to their original paths if the travel time savings are sufficiently large. Otherwise, they will remain on their new paths.

However, an implication of this finding is that the BRUE solution need not be unique, even in link flows. (This was also seen in the small example at the start of this subsection.) In general, the set of BRUE solutions is not convex either — which immediately shows that there is no convex optimization problem whose solutions correspond to BRUE assignments, since the sets of minima to convex optimization problems are always convex. However, there is always at least one BRUE solution by Proposition 5.1, since any path flows satisfying the principle of user equilibrium also satisfy BRUE.

Still, the lack of uniqueness means that BRUE should be applied carefully. While it is certainly more realistic than the principle of user equilibrium (since user equilibrium can always be obtained as a special case with $\epsilon = 0$), it is considerably more difficult to use BRUE to rank projects (which BRUE assignment should be used?) and at present it is not clear what value of ϵ best represents travel choices. Whether the added realism offsets these disadvantages is application and network dependent.

5.4 User Equilibrium and System Optimal

The relationship between the user equilibrium and system optimal assignment rules is worth studying further, for a few reasons. First, the fact that these rules lead to distinct solutions means that travelers cannot be relied upon to independently behave in a way that minimizes congestion⁵, even though each driver is minimizing his or her own travel time. This means that there is a role for transportation professionals to reduce congestion through planning and operational control. Exploring why this happens leads to a discussion of economic externalities, which are illustrated in the Braess paradox discussed in Section 4.3.

⁵Perhaps not a very surprising observation to those used to driving in crowded cities.

Second, the mathematical structure of the system optimal assignment problem is actually very similar to that of the user equilibrium problem. In fact, if one has a “black box” procedure for solving user equilibrium assignment, one can just as easily solve system optimal assignment using that same procedure, making some easy modifications to the network. Likewise, if one has a procedure for solving system optimal assignment, one can just as easily solve user equilibrium by modifying the network. So, when we discuss algorithms for solving the user equilibrium traffic assignment problem in Chapter 6, they all apply equally well to the system optimal problem. This relationship is explored in Section 5.4.2.

Finally, there is a more recent and interesting set of results which attempts to quantify how far apart the user equilibrium and system optimal assignments could possibly be, in terms of total system travel time. At first glance it seems that these assignment rules would tend to give relatively similar total travel times, since both are based on travel time minimization (one individual and the other collective). However, the Braess paradox shows that these can diverge in surprising ways. Nevertheless, in many cases one can in fact bound how much inefficiency is caused by allowing individual drivers to pick their own routes, a set of results called the *price of anarchy* and discussed in Section 5.4.3.

5.4.1 Externalities

Recall the three examples in Section 4.3, in which equilibrium traffic assignment exhibited counterintuitive results. Perhaps the most striking of these was the Braess paradox, in which adding a new roadway link actually worsened travel times for all travelers. This is counterintuitive, because if the travelers simply agreed to stay off of the newly-built road, their travel times would have been no worse than before. This section describes exactly why this happens. But before delving into the roots of the Braess paradox, let’s discuss some of the implications:

User equilibrium does *not* minimize congestion. In principle, everyone could have stayed on their original paths and retained the lower travel time of 83; the only catch was that this was no longer an equilibrium solution with the new link, and the equilibrium solution with the link is worse. Therefore, there may be traffic assignments with lower travel times (for everybody!) than the user equilibrium one.

The “invisible hand” may not work in traffic networks. In economics, Adam Smith’s celebrated “invisible hand” suggests that each individual acting in his or her own self-interest also tends to maximize the interests of society as well. The Braess paradox shows that this is not necessarily true in transportation networks. The two drivers switched paths out of self-interest, to reduce their own travel times; but the end effect made things worse for everyone, including the drivers who switched paths.

There may be room to “improve” route choices. A corollary of the previous point, when the invisible hand does not function well, a case can be made for regulation or another coordination mechanism to make everyone better off. In this case, this mechanism might take the form of certain transportation policies. What might they be?

If you have studied economics, you might already have some idea of why the Braess paradox occurs. The “invisible hand” requires certain things to be true if individual self-interest is to align with societal interest, and fails otherwise. A common case where the invisible hand fails is in the case of *externalities*: an externality is a cost or benefit resulting from a decision made by one person, and felt by another person who had no say in the first person’s decision. Examples of externalities include industrial pollution (we cannot rely on industries to self-regulate pollution, because all citizens near a factory suffer the effects of poor air quality, even though they had no say in the matter), driving without a seatbelt (if you get into an accident without a seatbelt, it is likely to be more serious, costing others who help pay your health care costs and delaying traffic longer while the ambulance arrives; those who help pay your costs and sit in traffic had no control over your decision to not wear a seatbelt), and education (educated people tend to commit less crime, which benefits all of society, even those who do not receive an education). The first two examples involve costs, and so are called negative externalities; the latter involves a benefit, and is called a positive externality.

Another example of an externality is seen in the prisoner’s dilemma (the Erica-Fred game of Section 1.3). When Erica chooses to testify against Fred, she does so because it reduces her jail time by one year. The fact that her choice also increases Fred’s jail time by fourteen years was irrelevant (such is the nature of greedy decision-making). When *both* Erica and Fred behaved in this way, the net effect was to dramatically increase the jail time they experienced, even though each of them made the choices they did in an attempt to minimize their jail time.

The relevance of this economics tangent is that *congestion* can be thought of as an externality. Let’s say I’m driving during the peak period, and I can choose an uncongested back road which is out of my way, or a congested freeway which is a direct route (and thus faster, even with the congestion). If I choose the freeway, I end up delaying everyone with the bad luck of being behind me. Perhaps not by much; maybe my presence increases each person’s commute by a few seconds. However, multiply these few seconds by a large number of drivers sitting in traffic, and we find that my presence has cost society as a whole a substantial amount of wasted time. This is an externality, because those other drivers had no say over my decision to save a minute or two to take the freeway. (This is why the user equilibrium assumption is said to be “greedy.” It assumes a driver will happily switch routes to save a minute even if it increases the total time people spend waiting in traffic by an hour.) These concepts are made more precise in the next subsection.

As a society, we tend to adopt regulation or other mechanisms for minimizing

the impact of negative externalities, such as emissions regulation or seatbelt laws (although you can probably think of many negative externalities which still exist). Transfer payments are another option: if a factory had to directly compensate each citizen for the harm they suffer as a result of pollution, the externality no longer exists because the factory now has to account for the costs it imposed on society, and will only continue to operate if its profits exceed the cost of the pollution it causes. What types of regulation or transfer payments might exist in the transportation world?

5.4.2 Mathematical equivalence

Recall that the user equilibrium traffic assignment was the solution to the following convex optimization problem:

$$\min_{\mathbf{x}, \mathbf{h}} \sum_{(i,j) \in A} \int_0^{x_{ij}} t_{ij}(x) dx \quad (5.55)$$

$$\text{s.t. } x_{ij} = \sum_{\pi \in \Pi} h^\pi \delta_{ij}^\pi \quad \forall (i, j) \in A \quad (5.56)$$

$$\sum_{\pi \in \Pi^{rs}} h^\pi = d^{rs} \quad \forall (r, s) \in Z^2 \quad (5.57)$$

$$h^\pi \geq 0 \quad \forall \pi \in \Pi \quad (5.58)$$

while the system optimal traffic assignment solves this convex optimization problem:

$$\min_{\mathbf{x}, \mathbf{h}} \sum_{(i,j) \in A} t_{ij}(x_{ij}) x_{ij} \quad (5.59)$$

$$\text{s.t. } x_{ij} = \sum_{\pi \in \Pi} h^\pi \delta_{ij}^\pi \quad \forall (i, j) \in A \quad (5.60)$$

$$\sum_{\pi \in \Pi^{rs}} h^\pi = d^{rs} \quad \forall (r, s) \in Z^2 \quad (5.61)$$

$$h^\pi \geq 0 \quad \forall \pi \in \Pi \quad (5.62)$$

Inspecting these, the only difference is in their objective functions, and even these are quite similar: both are sums of terms involving each link in the network. The problems share essentially the same structure, the only difference is the term corresponding to each link. For the user equilibrium problem, the term for link (i, j) is $\int_0^{x_{ij}} t_{ij}(x) dx$, while for the system optimum problem it is $t_{ij}(x_{ij})x_{ij}$.

Suppose for a moment that we had some algorithm which can solve the user equilibrium traffic assignment problem for any input network and OD matrix. (A number of these algorithms will be discussed in Chapter 6.) Now suppose that we had to solve the system optimal problem. If we were to replace the link performance functions $t_{ij}(x_{ij})$ with modified functions $\hat{t}_{ij}(x_{ij})$ defined by

$$\hat{t}_{ij}(x_{ij}) = t_{ij}(x_{ij}) + t'_{ij}(x_{ij})x_{ij} \quad (5.63)$$

where t'_{ij} is the derivative of the link performance functions, then

$$\int_0^{x_{ij}} \hat{t}_{ij}(x) dx = \int_0^{x_{ij}} (t_{ij}(x) + t'_{ij}(x)x) dx \quad (5.64)$$

Integrating the second term by parts, this simplifies to

$$\int_0^{x_{ij}} \hat{t}_{ij}(x) dx = t_{ij}(x_{ij})x_{ij} \quad (5.65)$$

which is the same term for link (i, j) used in the system optimal function.

That is, solving user equilibrium with the modified functions \hat{t}_{ij} produces the same link and path flow solution as solving system optimum with the original performance functions t_{ij} . The interpretation of the formula (5.63) is closely linked with the concept of externalities introduced in the previous subsection. Equation (5.63) consists of two parts: the actual travel time $t_{ij}(x_{ij})$, and an additional term $t'_{ij}(x_{ij})x_{ij}$. If an additional vehicle were to travel on link (i, j) , it would experience a travel time of $t_{ij}(x_{ij})$. At the margin, the travel time on link (i, j) would increase by approximately $t'_{ij}(x_{ij})$, and this marginal increase in the travel time would be felt by the other x_{ij} vehicles on the link. So, the second term in (5.63) expresses the additional, external increase in travel time caused by a vehicle using link (i, j) . For this reason, the modified functions \hat{t}_{ij} can be said to represent the *marginal cost* on a link.

Since the system optimal solution is a “user equilibrium” with respect to the modified costs \hat{t}_{ij} , we can formulate a “principle of system optimum” similar to the principle of user equilibrium:

Definition 5.2. (Principle of system optimum.) *Every used route connecting an origin and destination has equal and minimal marginal cost.*

So, system optimum can be seen as a special case of user equilibrium, with modified link performance functions. The converse is true as well. Suppose we had an algorithm which would solve the system optimal problem for any input network and OD matrix. If we replace the link performance functions $t_{ij}(x_{ij})$ with modified functions $\tilde{t}_{ij}(x_{ij})$ defined by

$$\tilde{t}_{ij}(x_{ij}) = \frac{\int_0^{x_{ij}} t_{ij}(x) dx}{x_{ij}} \quad (5.66)$$

when $x_{ij} > 0$ and $\tilde{t}_{ij}(x_{ij}) = 0$ otherwise, a similar argument shows that the objective function for the system optimal problem with link performance functions \tilde{t}_{ij} is identical to that for the user equilibrium problem with link performance functions t_{ij} .

As a result, despite very different physical interpretations, the user equilibrium and system optimal problems are essentially the same mathematically. If you can solve one, you can solve the other by making some simple changes to the link performance functions. So, there is no need to develop separate algorithms for the user equilibrium and system optimal problems.

5.4.3 Price of anarchy

The discussion of the user equilibrium and system optimal problems so far has gone in several directions. On the one hand, it seems like the problems are fairly similar: both involve routes chosen according to some travel time minimization rule, and they are mathematically equivalent. On the other hand, the real-world interpretations are very different (individual control vs. centralized control), and the Braess paradox shows that user equilibrium can behave in a far more counterintuitive manner than system optimal. This subsection concludes the discussion by showing that, while the user equilibrium solution may be different than the system optimal solution, and worse in terms of higher total system travel time, in many cases there is a limit to how much worse it can be.

In particular, if the link performance functions are linear (that is, of the form $t_{ij}(x_{ij}) = a_{ij} + b_{ij}x_{ij}$), the ratio between the total system travel time at user equilibrium, and the total system travel time at system optimum, can be no greater than $\frac{4}{3}$. This is often called the *price of anarchy*, since it represents the amount of additional delay which can potentially be caused by allowing individual drivers to choose their own routes, compared to a centrally-controlled solution. This bound holds no matter what the network structure or the level of demand, as long as the link performance functions are affine. This is an elegant and perhaps surprising result, which is actually not difficult to show using the variational inequality formulation of equilibrium.

Theorem 5.5. *Given a network with link performance functions of the form $t_{ij}(x_{ij}) = a_{ij} + b_{ij}x_{ij}$ with $a_{ij} \geq 0$ and $b_{ij} \geq 0$ for all links (i, j) , let feasible link flows $\hat{\mathbf{x}} \in X$ satisfy the principle of user equilibrium and $\hat{\mathbf{y}} \in X$ the system optimum principle. Denoting the total system travel times corresponding to these solutions as $TSTT(\hat{\mathbf{x}})$ and $TSTT(\hat{\mathbf{y}})$, we have*

$$\frac{TSTT(\hat{\mathbf{x}})}{TSTT(\hat{\mathbf{y}})} \leq \frac{4}{3} \quad (5.67)$$

Proof. Since $\hat{\mathbf{x}}$ satisfies the principle of user equilibrium, we have

$$\mathbf{t}(\hat{\mathbf{x}}) \cdot (\hat{\mathbf{x}} - \hat{\mathbf{y}}) \leq 0 \quad (5.68)$$

or, equivalently,

$$\mathbf{t}(\hat{\mathbf{x}}) \cdot \hat{\mathbf{x}} \leq \mathbf{t}(\hat{\mathbf{x}}) \cdot \hat{\mathbf{y}} \quad (5.69)$$

$$= (\mathbf{t}(\hat{\mathbf{x}}) - \mathbf{t}(\hat{\mathbf{y}})) \cdot \hat{\mathbf{y}} + \mathbf{t}(\hat{\mathbf{y}}) \cdot \hat{\mathbf{y}} \quad (5.70)$$

which is the same as

$$TSTT(\hat{\mathbf{x}}) \leq (\mathbf{t}(\hat{\mathbf{x}}) - \mathbf{t}(\hat{\mathbf{y}})) \cdot \hat{\mathbf{y}} + TSTT(\hat{\mathbf{y}}). \quad (5.71)$$

If we can show that

$$(\mathbf{t}(\hat{\mathbf{x}}) - \mathbf{t}(\hat{\mathbf{y}})) \cdot \hat{\mathbf{y}} \leq \frac{1}{4}TSTT(\hat{\mathbf{x}}) \quad (5.72)$$

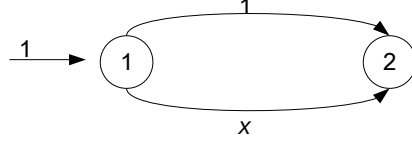


Figure 5.10: The price of anarchy bound is tight.

then we have established the result, by substitution into (5.71). We do this by showing an even stronger result, namely

$$(t_{ij}(\hat{x}_{ij}) - t_{ij}(\hat{y}_{ij}))\hat{y}_{ij} \leq \frac{1}{4}t_{ij}(\hat{x}_{ij})\hat{x}_{ij} \quad (5.73)$$

for all links (i, j) which is clearly sufficient.

If $\hat{x}_{ij} \leq \hat{y}_{ij}$, then the left-hand side of (5.73) is nonpositive (the link performance functions are nondecreasing since $b_{ij} \geq 0$), while the right-hand side is nonnegative (since $a_{ij} \geq 0$), so the inequality is certainly true. On the other hand, if $\hat{x}_{ij} > \hat{y}_{ij}$, then

$$(t_{ij}(\hat{x}_{ij}) - t_{ij}(\hat{y}_{ij}))\hat{y}_{ij} = b_{ij}(\hat{x}_{ij} - \hat{y}_{ij})\hat{y}_{ij}. \quad (5.74)$$

Now, the function $b_{ij}(\hat{x}_{ij} - z)z$ is a concave quadratic function in z , which obtains its maximum when $z = \hat{x}_{ij}/2$. Therefore

$$(t_{ij}(\hat{x}_{ij}) - t_{ij}(\hat{y}_{ij}))\hat{y}_{ij} \leq \frac{b_{ij}(\hat{x}_{ij})^2}{4} \leq \frac{1}{4}(a_{ij} + b_{ij}\hat{x}_{ij})\hat{x}_{ij} = t_{ij}(\hat{x}_{ij})\hat{x}_{ij}, \quad (5.75)$$

proving the result. \square

Furthermore, this bound is tight. Consider the more extreme version of the Knight-Pigou-Downs network shown in Figure 5.10 where the demand is 1 unit, the travel time on the upper link is 1 minute, and the travel time on the bottom link is $t^\downarrow = x^\downarrow$. The user equilibrium solution is $x^\uparrow = 0$, $x^\downarrow = 1$, when both links have equal travel times of 1 minute, and the total system travel time is 1. You can verify that the system optimal solution is $x^\uparrow = x^\downarrow = \frac{1}{2}$, when the total system travel time is $\frac{3}{4}$. Thus the ratio between the user equilibrium and system optimal total system travel times is $\frac{4}{3}$.

You may be wondering if a price of anarchy can be found when we relax the assumption that the link performance functions are affine. In many cases, yes; for instance, if the link performance functions are quadratic, then the price of anarchy is $\frac{3\sqrt{3}}{3\sqrt{3}-2}$, if cubic, then the price of anarchy is $\frac{4\sqrt[3]{4}}{4\sqrt[3]{4}-3}$, and so on. In all of these cases the modified Knight-Pigou-Downs network can show that this bound is tight. On the other hand, if the link performance functions have a

vertical asymptote (e.g., $t_{ij} = (u_{ij} - x_{ij})^{-1}$), then the ratio between the total system travel times at user equilibrium and system optimum may be made arbitrarily large.

5.5 Historical Notes and Further Reading

(These sections are incomplete in this beta version of the text, and will be substantially expanded in the complete first edition.)

The variational inequality formulation of the traffic assignment problem was presented by Smith (1979a), Dafermos (1980), and Smith (1983). The convex optimization formulation of the traffic assignment problem was first presented in Beckmann et al. (1956). The fixed-point formulation described in the optional section relies on a more general fixed point theorem due to Kakutani (1941). Although not described in this book, the traffic assignment problem can also be described as a nonlinear complementarity problem (Aashtiani and Magnanti, 1981).

Rossi et al. (1989) proposed the use of entropy to distinguish a “most likely” path flow solution among all of those satisfying the principle of user equilibrium. Bar-Gera and Boyce (1999) and Bar-Gera (2006) discuss how the more intuitive (but slightly weaker) proportionality condition can be derived from entropy maximization.

The bounded rationality formulations and results presented in this chapter are drawn from Mahmassani and Chen (1987), Lou et al. (2010), and Di et al. (2013). The first results on the price of anarchy in traffic assignment are due to Roughgarden (2002), and the specific proof given in this book is from Correa et al. (2004)

non-additive, RECURSE, and so on. Or maybe this should go in Chapter 9?

5.6 Exercises

1. [34] Consider a network with two parallel links connecting a single origin to a single destination; the link performance function on each link is $2 + x$ and the total demand is $d = 10$.
 - (a) Write the equations and inequalities defining the set of feasible path flows H , and draw a sketch.
 - (b) What are the vectors $-\mathbf{C}(\mathbf{h})$ for the following path flow vectors? (1) $\mathbf{h} = [0, 10]$ (2) $\mathbf{h} = [5, 5]$ (3) $\mathbf{h} = [10, 0]$ Draw these vectors at these three points on your sketch.
 - (c) For each of the vectors from part (b), identify the point $\text{proj}_H(\mathbf{h} - \mathbf{C}(\mathbf{h}))$ and include these on your sketch.
2. [45] Consider a network with two parallel links connecting a single origin to a single destination; the link performance function on the first link is

$10 + x$, and the link performance function on the second link is $x^2/20$. The total demand is $d = 30$.

- (a) Demonstrate that the solution $\hat{\mathbf{h}} = [20 \ 10]$ is not an equilibrium, and then provide a vector \mathbf{h} showing that the variational inequality (5.2) is not satisfied.
 - (b) Demonstrate that the solution $\hat{\mathbf{h}} = [10 \ 20]$ is an equilibrium, and prove that the variational inequality (5.2) is satisfied no matter what $\mathbf{h} \in H$ is.
3. [43] For the Braess network of Figure 4.8(b), write down the optimality conditions corresponding to Beckmann's formulation (5.13)–(5.16).
 4. [27] Which of the following multifunctions F are closed? Each of these multifunctions maps $[0, 1]$ to subsets of $[0, 1]$. Draw sketches of each of these multifunctions.
 - (a) $F(x) = \{y : 0 \leq y \leq x\}$
 - (b) $F(x) = \{y : 0 < y < 1\}$
 - (c) $F(x) = \{0\} \cup \{1 - x\} \cup \{1\}$
 - (d) $F(x) = \{x^2\}$
 5. [62] Show that if $f(x)$ is a continuous function with a compact domain, the single-valued “multifunction” $F(x) = \{f(x)\}$ is closed.
 6. [23] For each of the multifunctions in Exercise 4, identify all of the fixed points.
 7. [62] Specify the multifunction $R(\mathbf{c})$ for the Braess network (Figure 4.8b), and identify all of its fixed points.
 8. [42] Complete the proof of Theorem 5.3 by showing that the graph of F is indeed closed.
 9. [51] Create a simple network where one or more link performance functions are *not* continuous, and where no user equilibrium solution exists. (Don't worry about creating “realistic” functions for this problem.)
 10. [51] Create a simple network where one or more link performance functions are *not* strictly increasing, and where the user equilibrium link flows are not unique.
 11. [11] Show that the BPR link delay function (4.1) does not satisfy the condition $t'_{ij}(x_{ij}) > 0$ for all x_{ij} if $\beta > 1$.
 12. [46] In spite of Exercise 11, the BPR link delay functions are strictly increasing, and the resulting link flow equilibrium solution is still unique. Generalize the proof of Proposition 5.2 to handle the case when the link delay functions are differentiable and strictly increasing, by showing that

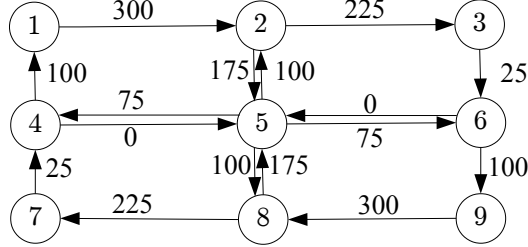


Figure 5.11: Network for Exercises 14 and 15. Each link has link performance function $10 + x_{ij}$

the Beckmann function is still strictly convex even if $t'_{ij}(x_{ij})$ is not always strictly positive.

13. [51] Show that the user-equilibrium and system-optimal link flows are the same if there is no congestion, that is, if $t_a(x_a) = \tau_a$ for some constant τ_a .
14. [10] Consider the network of Figure 5.11, ignoring the labels next to the link in the figure. The demand in this network is given by $d^{19} = 400$. Does the following path-flow solution satisfy the principle of user equilibrium? 100 vehicles on path $[1,2,3,6,9]$, 100 vehicles on path $[1,2,3,6,9]$, 100 vehicles on path $[1,4,5,6,9]$, and 100 vehicles on path $[1,4,5,2,3,6,9]$. You can answer this without any calculation.
15. [45] In the network in Figure 5.11, the demand is given by $d^{13} = d^{19} = d^{17} = d^{91} = d^{93} = d^{97} = 100$. The flow on each link is shown in the figure.
 - (a) Find a path flow vector \mathbf{h} which corresponds to these link flows.
 - (b) Show the resulting origin-based link flows \mathbf{x}^1 and \mathbf{x}^9 from the two origins in the network.
 - (c) Show that the links used by these two origins form an acyclic subnetwork by finding topological orders for each subnetwork.
 - (d) Determine whether these link flows satisfy the principle of user equilibrium.
16. [35] Consider the network in Figure 5.12, along with the given (equilibrium) link flows. There is only one OD pair, from node 1 to node 3. Identify three values of path flows which are consistent with these link flows, *in addition* to the most likely (entropy-maximizing) path flows.
17. [37] In the network shown in Figure 5.13, 320 vehicles travel from A to C, 640 vehicles travel from A to D, 160 vehicles travel from B to C, and 320 vehicles travel from B to D. The equilibrium link flows are shown.

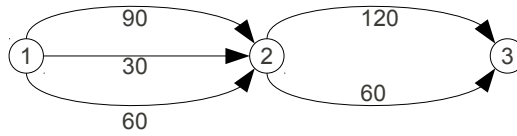


Figure 5.12: Network for Exercise 16.

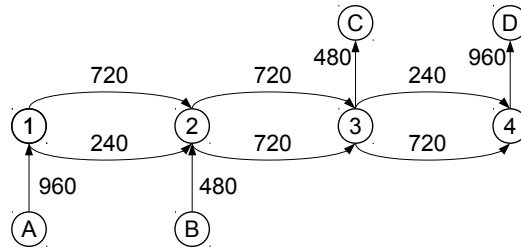


Figure 5.13: Network for Exercise 17.

- (a) Give a path flow solution which satisfies proportionality and produces the equilibrium link flows.
- (b) At the proportional solution, what fraction of flow on the top link connecting 3 and 4 is from origin A?
- (c) Is there a path flow solution which produces the equilibrium link flows, yet has *no* vehicles from origin A on the top link connecting 3 and 4? If so, list it. If not, explain why.
18. [38] Consider the network in Figure 5.14, where 2 vehicles travel from 1 to 4 (with a value of time of \$20/hr), and 4 vehicles travel from 2 to 4 (with a value of time of \$8/hr). The equilibrium volumes are 3 vehicles on Link 1, and 3 vehicles on Link 2.
- (a) Assuming that the vehicles in this network are discrete and cannot be split into fractions, identify every combination of path flows which give the equilibrium link volumes (there should be 20). Assuming each combination is equally likely, show that the proportional division of flows has the highest probability of being realized.
- (b) What is the average value of travel time on Link 1 at the most likely path flows? What are the upper and lower limits on the average value of travel time on this link?
19. [65] Derive the optimality conditions for the system-optimal assignment, and provide an interpretation of these conditions which intuitively relates them to the concept of system optimality.

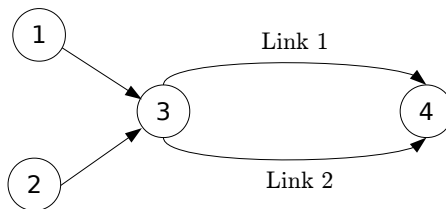


Figure 5.14: Network for Exercise 18.

20. [33] Find the system optimal assignment in the Braess network (Figure 4.8b), assuming a demand of 6 vehicles from node 1 to node 4.
21. [51] Consider a network where every link performance function is linear, of the form $t_{ij} = a_{ij}x_{ij}$. Show that the user equilibrium and system optimum solutions are the same.
22. [34] Calculate the ratio of the total system travel time between the user equilibrium and system optimal solutions in the Braess network (Figure 4.8b).
23. [46] Find the set of boundedly rational assignments in the Braess network (Figure 4.8b).

Chapter 6

Algorithms for Traffic Assignment

This chapter presents algorithms for solving the basic traffic assignment problem (TAP), which was defined in Chapter 5 as the solution $\hat{\mathbf{x}}$ to the variational inequality

$$\mathbf{t}(\hat{\mathbf{x}}) \cdot (\hat{\mathbf{x}} - \mathbf{x}) \leq 0 \quad \forall \mathbf{x} \in X, \quad (6.1)$$

which can also be expressed in terms of path flows $\hat{\mathbf{h}}$ as

$$\mathbf{c}(\hat{\mathbf{h}}) \cdot (\hat{\mathbf{h}} - \mathbf{h}) \leq 0 \quad \forall \mathbf{h} \in H, \quad (6.2)$$

or equivalently as the solution $\hat{\mathbf{x}}$ to the optimization problem

$$\min_{\mathbf{x}, \mathbf{h}} \sum_{(i,j) \in A} \int_0^{x_{ij}} t_{ij}(x) dx \quad (6.3)$$

$$\text{s.t. } x_{ij} = \sum_{\pi \in \Pi} h^\pi \delta_{ij}^\pi \quad \forall (i, j) \in A \quad (6.4)$$

$$\sum_{\pi \in \Pi^{rs}} h^\pi = d^{rs} \quad \forall (r, s) \in Z^2 \quad (6.5)$$

$$h^\pi \geq 0 \quad \forall \pi \in \Pi \quad (6.6)$$

While there are general algorithms for variational inequalities or nonlinear optimization problems, TAP involves tens of thousands or even millions of variables and constraints for practical problems. So, these general algorithms are outperformed by specialized algorithms which are designed to exploit some specific features of TAP. The most significant features to exploit are the network structure embedded in the constraints, and the fact that the objective function is separable by link.

This chapter presents four types of algorithms for TAP. The first three are aimed at finding an equilibrium solution (either link flows $\hat{\mathbf{x}}$ or path flows $\hat{\mathbf{h}}$). Broadly speaking, these algorithms can be divided into link-based, path-based,

and bush-based algorithms, according to the way the solution is represented. As discussed in Section 5.2.2, this is not trivial even though it plays a major role in certain types of applications, such as select link analysis.

The chapter focuses entirely on the basic TAP, and not the alternative assignment rules from Section 5.3. System optimal assignment can be transformed mathematically into a user equilibrium problem with modified cost functions, so all of these algorithms can be easily adapted for the system optimal problem, but most of these algorithms cannot be directly applied to the variations with bounded rationality or perception errors.

Section 6.1 provides an introduction, justifying the need for efficient algorithms for TAP and discussing issues such as convergence criteria which are relevant to all algorithms. The next three sections respectively present link-based, path-based, and bush-based algorithms for finding user equilibrium solutions.

6.1 Introduction to Assignment Algorithms

This introductory section touches on three topics: first, the advantages and disadvantages of link-based, path-based, and bush-based algorithms; second, the general framework for solving equilibrium problems; and third, the question of convergence criteria, which is pertinent to all of these.

6.1.1 Why do different algorithms matter?

It may not be clear why we need to present so many different algorithms for the same problem. Why not simply present the one “best” algorithm for solving TAP? Link-based, path-based, and bush-based algorithms all exhibit advantages and disadvantages relative to each other. In this introductory section, these algorithms are compared qualitatively. As you read through the following sections, which include details and specifications of each algorithm, keep these concepts in mind.

Link-based algorithms only keep track of the aggregate link flows \mathbf{x} , not the path flows \mathbf{h} which lead to these link flows. Since the number of links in a large network is much less than the number of paths, link-based algorithms are very economical in terms of computer memory. For this reason, link-based algorithms were the first to be used in practice decades ago, when computer memory was very expensive. Link-based algorithms also tend to be easier to parallelize. With modern desktop machines containing multiple cores, this parallelization can reduce computation time significantly. Finally, link-based algorithms tend to be easy to code and implement (even in a spreadsheet). A significant drawback is that they tend to be slow, particularly when high precision is demanded. The first few iterations of link-based algorithms make good progress, but they stall quickly, and final convergence to the equilibrium solution can be extremely slow.

Path-based algorithms, by contrast, keep track of the path flow vector \mathbf{h} , which can be used to generate the link flows \mathbf{x} whenever needed. Although this

representation requires more memory than a link-based solution, it also retains a great deal of information which is lost when one aggregates path flows to link flows. This information can be exploited to converge much faster than link-based algorithms, especially when a very precise solution is needed. Because the number of paths in a network is so large, much of the effort involved in coding path-based algorithms is associated with clever data structures and algorithm schemes which are aimed at minimizing the number of paths which need to be stored. This increases the complexity of the coding of these algorithms, even if the algorithmic concepts are not difficult.

Bush-based algorithms are the most recent developed, and aim to offer speed comparable to path-based algorithms, while requiring less memory (although still significantly more than link-based algorithms). Bush-based algorithms do this by selectively aggregating the path flows: as shown in Section 5.2.3, at equilibrium the set of paths used by all travelers from the same origin (or traveling to the same destination) forms an acyclic subgraph (a bush). Bush-based algorithms aim to identify these bushes for each origin and destination, exploiting the fact that calculations in acyclic networks are very fast. Bush-based algorithms also tend to return higher-entropy solutions than path-based algorithms, which is important when interpreting the path flow solution. The downside to these algorithms is that their design involves more complexity, and the success of these algorithms depends highly on implementation and programming skill.

6.1.2 Framework

It turns out that none of these solution methods get to the right answer immediately, or even after a finite number of steps. There is no “step one, step two, step three, and then we’re done” recipe for solving large-scale equilibrium problems. Instead, an iterative approach is used where we start with some feasible assignment (link or path), and move closer and closer to the equilibrium solution as you repeat a certain set of steps over and over, until you’re “close enough” to quit and call it good. One iterative algorithm you probably saw in calculus was Newton’s method for finding zeros of a function. In this method, one repeats the same step over and over until the function is sufficiently close to zero.

Broadly speaking, all equilibrium solution algorithms repeat the following three steps:

1. Find the shortest (least travel time) path between each origin and each destination.
2. Shift travelers from slower paths to faster ones.
3. Recalculate link flows and travel times after the shift, and return to step one unless we’re close enough to equilibrium.

The shortest path computation can be done quickly and efficiently even in large networks, as was described in Section 2.4. The third step is even more

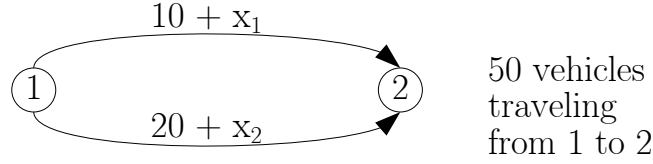


Figure 6.1: Two-link example for demonstration.

straightforward, and is nothing more than re-evaluating the link performance functions on each link with the new volumes. The second step requires the most care; the danger here is shifting either too few travelers onto faster paths, or shifting too many. If we shift too few, then it will take a long time to get to the equilibrium solution. On the other hand, systematically shifting too many can be even more dangerous, because it creates the possibility of “infinite cycling” and never finding the true equilibrium.

In the simple example in Figure 6.1, by inspection the equilibrium is for thirty travelers to choose the top route, and twenty to choose the bottom route, with an equal travel time of 40 minutes on both paths. Solving this example using the above process, initially (i.e., with nobody on the network) the fastest path is the top one (step one), so let’s assign all 50 travelers onto the top path (step two). Performing the third step, we recalculate the travel times as 60 minutes on the top link, and 20 on the bottom. This is not at all an equilibrium, so we go back to the first step, and see that the bottom path is now faster, so we have to shift some people from the top to the bottom. If we wanted, we could shift travelers one at a time, that is, assigning 49 to the top route and 1 to the bottom, seeing that we still haven’t found equilibrium, so trying 48 and 2, then 47 and 3, and so forth, until finally reaching the equilibrium with 30 and 20. Clearly this is not efficient, and is an example of shifting too few travelers at a time.

At the other extreme, let’s say we shift *everybody* onto the fastest path in the second step. That is, we go from assigning 50 to the top route and 0 to the bottom, to assigning 0 to the top and 50 to the bottom. Recalculating link travel times, the top route now has a travel time of 10 minutes, and the bottom a travel time of 70. This is even worse!¹ Repeating the process, we try to fix this by shifting everybody back (50 on top, 0 on bottom), but now we’re just back in the original situation. If we kept up this process, we’d keep bouncing back and forth between these solutions. This is clearly worse than shifting too few, because we never reach the equilibrium no matter how long we work! You might think it’s obvious to detect if something like this is happening. With this small example, it might be. Trying to train a computer to detect this, or trying to detect cycles with over millions OD pairs (as is common in practice), is much harder. The key step in all of the algorithms for finding equilibria is

¹By “worse” we mean farther from equilibrium.

determining how much flow to shift.

6.1.3 Convergence criteria

A general issue is how one chooses to stop the iterative process, that is, how one knows when a solution is “good enough” or close enough to equilibrium. This is called a *convergence criterion*. Many convergence criteria have been proposed over the years; perhaps the most common in practice is the *relative gap*, which is defined first. Unfortunately, the relative gap has been defined in several different ways, so it is important to be familiar with all of the definitions.

Remembering that the multiplier κ^{rs} represents the time spent on the fastest path between origin r and destination s , one definition of the relative gap γ_1 is

$$\gamma_1 = \frac{\sum_{(i,j) \in A} t_{ij} x_{ij}}{\sum_{(r,s) \in Z^2} \kappa^{rs} d^{rs}} - 1 = \frac{\mathbf{t} \cdot \mathbf{x}}{\boldsymbol{\kappa} \cdot \mathbf{d}} - 1. \quad (6.7)$$

The numerator of the fraction is the total system travel time (TSTT). The relative gap is always nonnegative, and it is equal to zero if and only if the flows \mathbf{x} satisfy the principle of user equilibrium. It is these properties which make the relative gap a useful convergence criterion: once it is close enough to zero, our solution is “close enough” to equilibrium. For most practical purposes, a relative gap of 10^{-4} – 10^{-6} is small enough.

A second definition of the relative gap γ_2 is based on the Beckmann function itself. Let f denote the Beckmann function, and \hat{f} its value at equilibrium (which is a global minimum). In many algorithms, given a current solution \mathbf{x} , it is not difficult to generate upper and lower bounds on \hat{f} based on the current solution, respectively denoted $\bar{f}(\mathbf{x})$ and $\underline{f}(\mathbf{x})$. A trivial upper bound is its value at the current solution: $\bar{f} = f(\mathbf{x})$, since clearly $\hat{f} \leq f(\mathbf{x})$ for any feasible link assignment \mathbf{x} . Sometimes, a corresponding lower bound can be identified as well. Assuming that these bounds can become tighter over time, and that in the limit both $\bar{f} \rightarrow \hat{f}$ and $\underline{f} \rightarrow \hat{f}$, the difference or gap $\bar{f}(\mathbf{x}) - \underline{f}(\mathbf{x})$ can be used as a convergence criterion. These values are typically normalized, leading to one definition of the relative gap:

$$\gamma_2 = \frac{\bar{f} - \underline{f}}{\underline{f}}, \quad (6.8)$$

or a slightly modified version

$$\gamma_3 = \frac{\bar{f} - \max \underline{f}}{\max \underline{f}}, \quad (6.9)$$

where $\max \underline{f}$ is the greatest lower bound found to date, in case the sequence of \underline{f} values is not monotone over iterations. A disadvantage of these definitions of the relative gap is that different algorithms calculate these upper and lower bounds differently. While they are suitable as termination criteria in an algorithm, it is

not possible (or at least not easy) to directly compare the relative gap calculated by one algorithm to that produced by another to assess which of two solutions is closer to equilibrium.

One drawback of the relative gap (in all of its forms) is that it is unitless and does not have an intuitive meaning. Furthermore, it can be somewhat confusing to have several slightly different definitions of the relative gap, even though they all have the same flavor. A more recently proposed metric is the *average excess cost*, defined as

$$AEC = \frac{\sum_{(i,j) \in A} t_{ij} x_{ij} - \sum_{(r,s) \in Z^2} \kappa^{rs} d^{rs}}{\sum_{(r,s) \in Z^2} d^{rs}} = \frac{\mathbf{t} \cdot \mathbf{x} - \boldsymbol{\kappa} \cdot \mathbf{d}}{\mathbf{d} \cdot \mathbf{1}}. \quad (6.10)$$

This quantity represents the average difference between the travel time on each traveler's *actual* path, and the travel time on the *shortest* path available to him or her. Unlike the relative gap, *AEC* has units of time, and is thus easier to interpret.

Another convergence measure with time units is the *maximum excess cost*, which relates directly to the principle of user equilibrium. The maximum excess cost is defined as the largest amount by which a used path's travel time exceeds the shortest path travel time available to that traveler:

$$MEC = \max_{(r,s) \in Z^2} \left\{ \max_{\pi \in \Pi^{rs}: h^\pi > 0} \{c^\pi - \kappa^{rs}\} \right\}. \quad (6.11)$$

This is often a few orders of magnitude higher than the average excess cost. One disadvantage of the maximum excess cost is that it is only applicable when the path flow solution is known. This is easy in path-based or bush-based algorithms. However, since many path-flow solutions correspond to the same link-flow solution (cf. Section 5.2.2), *MEC* is not well suited for link-based algorithms.

Finally, this section concludes with two convergence criteria which are inferior to those discussed thus far. The first is to simply use the Beckmann function itself; when it is sufficiently close to the global optimal value \hat{f} , terminate. A moment's thought should convince you that this criterion is not practical: there is no way to know the value of \hat{f} until the problem has already been solved. (Upper and lower bounds are possible to calculate, though, as with γ_2 .) A more subtle version is to terminate when the Beckmann function stops decreasing, or (in a more common form) to terminate the algorithm when the link or path flows stabilize from one iteration to the next. The trouble with these convergence criteria is that they cannot distinguish between a situation when the flows stabilize because they are close to the equilibrium solution, and when they stabilize because the algorithm "gets stuck" and cannot improve further due to a flaw in its design or a bug in the programming. For this reason, it is always preferable to base the termination criteria on the equilibrium principle itself.

6.2 Link-Based Algorithms

Link-based algorithms for traffic assignment are the simplest to understand and implement, and require the least amount of computer memory. Given a current set of link flows \mathbf{x} , a link-based algorithm attempts to move closer to equilibrium by performing two steps. First, a target point \mathbf{x}^* is identified; moving in the direction of \mathbf{x}^* from \mathbf{x} should lead towards an equilibrium solution. Then, a step of size $\lambda \in [0, 1]$ is taken in this direction, updating the link flows to $\lambda\mathbf{x}^* + (1 - \lambda)\mathbf{x}$. This is a specific way of implementing the first two steps of the framework specified in the previous section: λ represents the fraction of flow which is shifted from the paths at the current solution \mathbf{x} to the paths at the target solution \mathbf{x}^* . If $\lambda = 1$, all of the flow has shifted to the target solution; if $\lambda = 0$, no flow has shifted at all. Since the set of feasible link flow solutions X is convex, as long as \mathbf{x} and \mathbf{x}^* are feasible, we can be assured that the convex combination $\lambda\mathbf{x}^* + (1 - \lambda)\mathbf{x}$ is feasible as well.

Link-based algorithms differ in two primary ways: first, how the target \mathbf{x}^* is chosen; and second, how the step size λ is chosen. If λ is too small, convergence to equilibrium will be very slow, but if λ is too large, the solution may never converge at all — the example from Figure 6.1 in the previous section is an example of what can happen if $\lambda = 1$ for all iterations. This section presents three link-based algorithms, in increasing order of sophistication (but in increasing order of convergence speed.) The first is the method of successive averages (MSA), which is perhaps the simplest equilibrium algorithm. The second is the Frank-Wolfe algorithm, which can be thought of as a version of MSA with a more intelligent choice of step size λ . (Frank-Wolfe was the most common used in practice for several decades.) The third is the conjugate Frank-Wolfe algorithm, which can be thought of as a version of Frank-Wolfe with a more intelligent choice of target \mathbf{x}^* .

All of these algorithms use the following framework; the only difference is how \mathbf{x}^* and λ are calculated.

1. Generate an initial solution $\mathbf{x} \in X$.
2. Generate a target solution $\mathbf{x}^* \in X$.
3. Update the current solution: $\mathbf{x} \leftarrow \lambda\mathbf{x}^* + (1 - \lambda)\mathbf{x}$ for some $\lambda \in [0, 1]$.
4. Calculate the new link travel times.
5. If the convergence criterion is satisfied, stop; otherwise return to step 2.

6.2.1 Method of successive averages

Although the method of successive averages (MSA) is not competitive with other equilibrium solution algorithms, its simplicity and clarity in applying the three-step iterative process make it an ideal starting place. To specify MSA, we need to specify how the target solution \mathbf{x}^* is chosen, and how the step size λ is chosen.

The target \mathbf{x}^* is an all-or-nothing assignment. That is, *assuming that the current travel times are fixed*, identify the shortest path between each origin and destination, and load all of the demand for that OD pair onto that path. Thus, \mathbf{x}^* is the state which would occur if literally every driver was to switch paths onto what is currently the shortest path. Of course, if we were to switch everybody onto these paths, which would occur if we choose $\lambda = 1$, those paths would almost certainly not be “shortest” anymore. But \mathbf{x}^* can still be thought of as a target, or a direction in which travelers would feel pressure to move.

So, what should λ be? As discussed above, there are problems if you shift too few travelers, and potentially even bigger problems if you shift too many. MSA adopts a reasonable middle ground: initially, we shift a lot of travelers, but as the algorithm progresses, we shift fewer and fewer until we settle down on the average. The hope is that this avoids both the problems of shifting too few (at first, we’re taking big steps, so hopefully we get somewhere close to equilibrium quickly) and of shifting too many (eventually, we’ll only be moving small amounts of flow so there is no worry of infinite cycling).

Specifically, on the i -th iteration, MSA uses $\lambda = 1/(i + 1)$. So, the first time through, half of the travelers are shifted to the current shortest paths. The second time through, a third of the people shift to the current shortest paths (and two thirds stay on their current path). On the third iteration, a fourth of the people shift to new paths, and so on. (MSA can also be applied with different step size rules; see Exercise 11.)

At this point, it’s worth using the Beckmann formulation to show that the choice of an all-or-nothing assignment for \mathbf{x}^* has mathematical justification, in addition to the intuitive interpretation of shifting towards shortest path. Let \mathbf{x} be the current set of link flows, and \mathbf{x}^* an all-or-nothing assignment. As a result of a shift of size λ , the Beckmann function will be changed from $f(\mathbf{x})$ to $f(\mathbf{x}')$, and we want to show that it’s possible to choose λ in some way to guarantee $f(\mathbf{x}') \leq f(\mathbf{x})$. That is, we want to show that we can reduce the Beckmann function (and thus move closer to the equilibrium solution) by taking a (correctly-sized) step in the direction $\mathbf{x}^* - \mathbf{x}$. Define $f(\mathbf{x}(\lambda)) = f((1 - \lambda)\mathbf{x} + \lambda\mathbf{x}^*)$ to be the Beckmann function after taking a step of size λ . Using the multivariate chain rule, the derivative of $f(\lambda)$ is

$$\frac{df}{d\lambda} = \sum_{(i,j) \in A} \frac{\partial f}{\partial x_{ij}} \frac{dx_{ij}}{d\lambda} = \sum_{(i,j) \in A} t_{ij}((1 - \lambda)x_{ij} + \lambda x_{ij}^*)(x_{ij}^* - x_{ij}).$$

Evaluating this derivative at $\lambda = 0$ gives

$$\frac{df}{d\lambda}(0) = \sum_{(i,j) \in A} t_{ij}(x_{ij})(x_{ij}^* - x_{ij}).$$

Now, \mathbf{x}^* was specifically chosen to put all vehicles on the shortest paths at travel times $\mathbf{t}(\mathbf{x})$, and so $\sum_{(i,j) \in A} x_{ij}^* t_{ij}(x_{ij}) \leq \sum_{(i,j) \in A} x_{ij} t_{ij}(x_{ij})$, and therefore $\frac{df}{d\lambda}(0) \leq 0$. Furthermore, if we are not at the equilibrium solution already, $\sum_{(i,j) \in A} x_{ij}^* t_{ij}(x_{ij})$ is strictly less than $\sum_{(i,j) \in A} x_{ij} t_{ij}(x_{ij})$. This implies

$\frac{df}{d\lambda}(0) < 0$ or, equivalently, we can decrease the Beckmann function if we take a small enough step in the direction $\mathbf{x}^* - \mathbf{x}$, by shifting people from longer paths onto shorter ones.

Two examples of MSA are shown below. A proof of convergence is sketched in Exercise 12.

Small network example Here we solve the small example of Figure 6.1 by MSA, using the relative gap γ_1 to measure how close we are to equilibrium.

Initialization. Find the shortest paths: with no travelers on the network, the top link has a travel time of 10, and the bottom link has a travel time of 20. Therefore the top link is the shortest path, so $\mathbf{x}^* = [50 \ 0]$. We take this to be the initial solution $\mathbf{x} \leftarrow \mathbf{x}^* = [50 \ 0]$. Recalculating the travel times, we have $t_1 = 10 + x_1 = 60$ and $t_2 = 20 + x_2 = 20$ (or, in vector form, $\mathbf{t} = [60 \ 20]$).

Iteration 1. With the new travel times, the shortest path is now the bottom link, so $\kappa = 20$ and the relative gap is

$$\gamma_1 = \frac{\mathbf{t} \cdot \mathbf{x}}{\kappa \cdot \mathbf{d}} - 1 = \frac{50 \times 60 + 0 \times 20}{20 \times 50} - 1 = 2.$$

This is far too big, so we continue with the second iteration. If everyone were to take the new shortest path, the flows would be $\mathbf{x}^* = [0 \ 50]$. Because this is the first iteration, we shift 1/2 of the travelers onto this path, so $\mathbf{x} \leftarrow (1/2)\mathbf{x}^* + (1/2)\mathbf{x} = [0 \ 25] + [25 \ 0] = [25 \ 25]$. The new travel times are thus $\mathbf{t} = [35 \ 45]$.

Iteration 2. With the new travel times, the shortest path is now the top link, so $\kappa = 35$ and the relative gap is

$$\gamma_1 = \frac{\mathbf{t} \cdot \mathbf{x}}{\kappa \cdot \mathbf{d}} - 1 = \frac{25 \times 35 + 25 \times 45}{35 \times 50} - 1 = 0.143.$$

If everyone were to take the new shortest path, the flows would be $\mathbf{x}^* = [50 \ 0]$. Because this is the second iteration, we shift 1/3 of the travelers onto this path, so $\mathbf{x} \leftarrow (1/3)\mathbf{x}^* + (2/3)\mathbf{x} = [50/3 \ 0] + [50/3 \ 50/3] = [100/3 \ 50/3]$. The new travel times are thus $\mathbf{t} = [43.33 \ 36.67]$.

Iteration 3. With the new travel times, the shortest path is now the bottom link, so $\kappa = 36.67$ and the relative gap is $\gamma_1 = 0.121$. A bit better, but still too big, so we carry on. Here $\mathbf{x}^* = [0 \ 50]$, $\mathbf{x} \leftarrow (1/4)\mathbf{x}^* + (3/4)\mathbf{x} = [0 \ 50/4] + [25 \ 50/4] = [25 \ 25]$. The new travel times are $\mathbf{t} = [35 \ 45]$. Note that we have returned to the same solution found in Iteration 1. *Don't despair; this just means the last shift was too big. Next time we'll shift fewer vehicles (because λ is smaller with each iteration).*

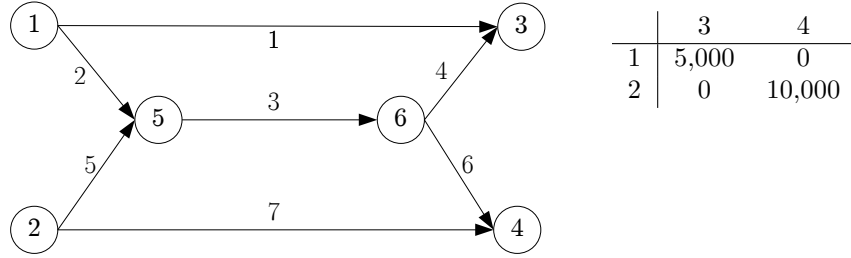


Figure 6.2: Larger example with two OD pairs. (Link numbers shown.)

Iteration 4. With the new travel times, the shortest path is now the top link, so $\kappa = 35$ and the relative gap is $\gamma_1 = 0.143$. The new target is $\mathbf{x}^* = [50 \ 0]$, $\mathbf{x} \leftarrow (1/5)\mathbf{x}^* + (4/5)\mathbf{x} = [30 \ 20]$. The new travel times are $\mathbf{t} = [40 \ 40]$. With the new travel times, the shortest path is the top link, so $\kappa = 40$ and the relative gap is $\gamma_1 = 0$, so we stop. *In fact, either path could have been chosen for the shortest path. Whenever there is a tie between shortest paths, you are free to choose among them.*

Larger network example Here we apply MSA to a slightly larger network with two OD pairs, shown in Figure 6.2, where each link has the link performance function $t(x) = 10 + x/100$.

There are four paths in this network; for OD pair (1,3) these are denoted $[1, 3]$ and $[1, 5, 6, 3]$ according to their link numbers, and for OD pair (2,4) these are $[2, 5, 6, 4]$ and $[2, 4]$. In this example, we'll calculate the average excess cost, rather than the relative gap.

Initialization. Find the shortest paths: with no travelers on the network, paths $[1, 3]$, $[1, 5, 6, 3]$, $[2, 5, 6, 4]$, and $[2, 4]$ respectively have travel times of 10, 30, 30, and 10. Therefore $[1, 3]$ is shortest for OD pair (1,3), and $[2, 4]$ is shortest for OD pair (2,4), so $\mathbf{x}^* = [5000 \ 0 \ 0 \ 0 \ 0 \ 0 \ 10000]$.² Since is the first iteration, we simply set

$$\mathbf{x} \leftarrow \mathbf{x}^* = [5000 \ 0 \ 0 \ 0 \ 0 \ 0 \ 10000] .$$

Recalculating the travel times, we have

$$\mathbf{t} = [60 \ 10 \ 10 \ 10 \ 10 \ 10 \ 110] .$$

Iteration 1. With the new travel times, the shortest path for (1,3) is now $[1, 5, 6, 3]$, with a travel time of 30, so $\kappa^{13} = 30$. Likewise, the new shortest

²For each OD pair, we add the total demand from the OD matrix onto each link in the shortest path.

path for (2,4) is [2, 5, 6, 4], so $\kappa^{24} = 30$ and the average excess cost is

$$\begin{aligned} AEC &= \frac{\mathbf{t} \cdot \mathbf{x} - \kappa \cdot \mathbf{d}}{\mathbf{d} \cdot \mathbf{1}} \\ &= \frac{5000 \times 60 + 10000 \times 110 - 30 \times 5000 - 30 \times 10000}{5000 + 10000} = 63.33. \end{aligned}$$

This is far too big and suggests that the average trip is 63 minutes slower than the shortest paths available! If everyone were to take the new shortest paths, the flows would be

$$\mathbf{x}^* = [0 \quad 5000 \quad 15000 \quad 5000 \quad 10000 \quad 10000 \quad 0].$$

(Be sure you understand how we calculated this.) Because this is iteration 1, we shift 1/2 of the travelers onto this path, so

$$\mathbf{x} \leftarrow (1/2)\mathbf{x}^* + (1/2)\mathbf{x} = [2500 \quad 2500 \quad 7500 \quad 2500 \quad 5000 \quad 5000 \quad 5000].$$

The new travel times are thus

$$\mathbf{t} = [35 \quad 35 \quad 85 \quad 35 \quad 60 \quad 60 \quad 60].$$

Iteration 2. With the new travel times, the shortest path for (1,3) is now [1, 3], with $\kappa^{13} = 35$. The new shortest path for (2,4) is [2, 4], so $\kappa^{24} = 60$ and the average excess cost is

$$\begin{aligned} AEC &= \\ &= \frac{\left(\begin{array}{l} 2500 \times 35 + 2500 \times 35 + 7500 \times 85 + 2500 \times 35 + 5000 \times 60 \\ + 5000 \times 60 + 5000 \times 60 - 35 \times 5000 - 60 \times 10000 \end{array} \right)}{15000} \\ &= 68.33. \end{aligned}$$

This is still big (and in fact worse), but we persistently continue with the second iteration. If everyone were to take the new shortest paths, the flows would be

$$\mathbf{x}^* = [5000 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 10000],$$

so

$$\mathbf{x} \leftarrow (1/3)\mathbf{x}^* + (2/3)\mathbf{x} = [3333 \quad 1667 \quad 5000 \quad 1667 \quad 3333 \quad 3333 \quad 6667].$$

The new travel times are

$$\mathbf{t} = [43.3 \quad 26.7 \quad 60 \quad 26.7 \quad 43.3 \quad 43.3 \quad 76.7].$$

Iteration 3. With the new travel times, the shortest path for (1,3) is still [1, 3], with $\kappa^{13} = 43.3$, and the shortest path for (2,4) is still [2, 4] with

$\kappa^{24} = 76.7$ and the average excess cost is $AEC = 23.6$. Continuing the fourth iteration, as before

$$\mathbf{x}^* = [5000 \ 0 \ 0 \ 0 \ 0 \ 0 \ 10000] ,$$

so

$$\mathbf{x} \leftarrow (1/4)\mathbf{x}^* + (3/4)\mathbf{x} = [3750 \ 1250 \ 3750 \ 1250 \ 2500 \ 2500 \ 7500] .$$

The new travel times are

$$\mathbf{t} = [47.5 \ 22.5 \ 47.5 \ 22.5 \ 35 \ 35 \ 85] .$$

Iteration 4. With the new travel times, the shortest path for (1,3) is still [1, 3], with $\kappa^{13} = 47.5$, and the shortest path for (2,4) is still [2, 4] with $\kappa^{24} = 85$ and the average excess cost is $AEC = 9.42$. Continuing the fifth iteration, as before

$$\mathbf{x}^* = [5000 \ 0 \ 0 \ 0 \ 0 \ 0 \ 10000] ,$$

so

$$\mathbf{x} \leftarrow (1/5)\mathbf{x}^* + (4/5)\mathbf{x} = [4000 \ 1000 \ 3000 \ 1000 \ 2000 \ 2000 \ 8000] .$$

The new travel times are

$$\mathbf{t} = [50 \ 20 \ 40 \ 20 \ 30 \ 30 \ 90] .$$

Iteration 5. With the new travel times, the shortest path for (1,3) is still [1, 3], with $\kappa^{13} = 50$, and the shortest path for (2,4) is still [2, 4] with $\kappa^{24} = 90$ and the average excess cost is $AEC = 3.07$. *Note that the shortest paths have stayed the same over the last three iterations. This means that we really could have shifted more flow than we actually did. The Frank-Wolfe algorithm, described in the next section, fixes this problem.* We have

$$\mathbf{x}^* = [5000 \ 0 \ 0 \ 0 \ 0 \ 0 \ 10000] ,$$

so

$$\mathbf{x} \leftarrow (1/6)\mathbf{x}^* + (5/6)\mathbf{x} = [4167 \ 833 \ 2500 \ 833 \ 1667 \ 1667 \ 8333] .$$

The new travel times are

$$\mathbf{t} = [51.7 \ 18.3 \ 35 \ 18.3 \ 26.7 \ 26.7 \ 93.3] .$$

Iteration 6. With the new travel times, the shortest path for (1,3) is still [1, 3], with $\kappa^{13} = 51.7$, but the shortest path for (2,4) is now [2, 5, 6, 4] with $\kappa^{24} = 88.3$. The average excess cost is $AEC = 3.80$. *Note that the OD pairs are no longer behaving “symmetrically,” the shortest path for*

(1,3) stayed the same, but the shortest path for (2,4) has changed. We have

$$\mathbf{x}^* = [5000 \ 0 \ 10000 \ 0 \ 10000 \ 10000 \ 0] .$$

so

$$\mathbf{x} \leftarrow (1/7)\mathbf{x}^* + (6/7)\mathbf{x} = [4286 \ 714 \ 3571 \ 714 \ 2857 \ 2857 \ 7142] .$$

The new travel times are

$$\mathbf{t} = [52.9 \ 17.1 \ 45.7 \ 17.1 \ 38.6 \ 38.6 \ 81.4] .$$

This process continues over and over until the average excess cost is sufficiently small. Even with such a small network, MSA requires a very long time to converge. An average excess cost of 1 is obtained after eleven iterations, 0.1 after sixty-three iterations, 0.01 after three hundred thirty-two, and the rate of convergence only slows down from there.

6.2.2 Frank-Wolfe

One of the biggest drawbacks with MSA is that it has a fixed step size. Iteration i moves exactly $1/(i+1)$ of the travelers onto the new shortest paths, no matter how close or far away we are from the equilibrium. Essentially, MSA decides its course of action before it even gets started, then sticks stubbornly to the plan of moving $1/(i+1)$ travelers each iteration. The Frank-Wolfe (FW) algorithm fixes this problem by using an *adaptive* step size. At each iteration, FW calculates exactly the right amount of flow to shift to get as close to equilibrium as possible.

We might try to do this by picking λ to minimize the relative gap or average excess cost, but this turns out to be harder to compute. Instead, we pick λ to solve a “restricted” VI where the feasible set is the line segment connecting \mathbf{x} and \mathbf{x}^* . It turns out that this is the same as choosing λ to minimize the Beckmann function (6.3) along this line segment. Both approaches for deriving the step size λ are discussed below.

Define X' to be the link flows lying on the line segment between \mathbf{x} and \mathbf{x}^* . That is, $X' = \{\mathbf{x}' : \mathbf{x} = \lambda\mathbf{x}^* + (1-\lambda)\mathbf{x} \text{ for some } \lambda \in [0, 1]\}$. The restricted VI is: find $\hat{\mathbf{x}}' \in X'$ such that $\mathbf{t}(\hat{\mathbf{x}}') \cdot (\hat{\mathbf{x}}' - \mathbf{x}') \leq 0$ for all $\mathbf{x}' \in X'$.

This VI is simple enough to be solved as a single equation. The set X has two endpoints (\mathbf{x} and \mathbf{x}^* , corresponding to $\lambda = 0$ and $\lambda = 1$, respectively). For now, assume that the solution $\hat{\mathbf{x}}'$ to the VI is not at one of these endpoints.³ In this case, the force vector $-\mathbf{t}(\hat{\mathbf{x}}')$ is perpendicular to the direction $\mathbf{x}^* - \mathbf{x}$. (Figure 6.3), so $-\mathbf{t}(\hat{\mathbf{x}}') \cdot (\mathbf{x}^* - \mathbf{x}) = 0$. Writing this equation out in terms of individual components, we need to solve

$$\sum_{ij} t_{ij}(\hat{x}'_{ij}) (x^*_{ij} - x_{ij}) = 0 \tag{6.12}$$

³Exercise 13 asks you to show that the solution methods provided below will still give the right answer even in these cases.

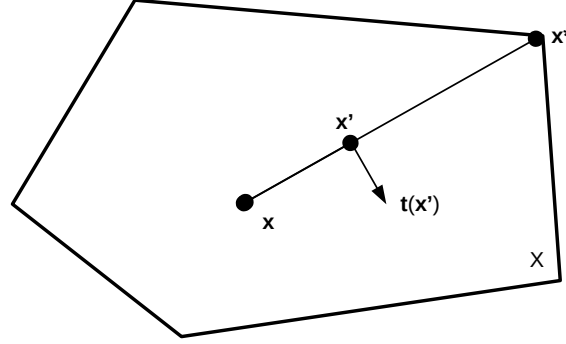


Figure 6.3: A solution to the restricted VI in the Frank-Wolfe method.

or equivalently

$$\sum_{ij} t_{ij}(\hat{x}'_{ij})x_{ij}^* = \sum_{ij} t_{ij}(\hat{x}'_{ij})x_{ij} \quad (6.13)$$

The same equation can be derived based on the Beckmann function. Recall the discussion above, where we wrote the function $f(\mathbf{x}(\lambda)) = f((1-\lambda)\mathbf{x} + \lambda\mathbf{x}^*)$ to be the value of the Beckmann function after taking a step of size λ , and furthermore found the derivative of $f(\mathbf{x}(\lambda))$ to be

$$\frac{df}{d\lambda} = \sum_{(i,j) \in A} t_{ij}((1-\lambda)x_{ij} + \lambda x_{ij}^*)(x_{ij}^* - x_{ij}). \quad (6.14)$$

It is not difficult to show that $f(\mathbf{x}(\lambda))$ is a convex function of λ , so we can find its minimum by setting the derivative equal to zero, which occurs if the condition

$$\sum_{(i,j) \in A} x_{ij}^* t_{ij}((1-\lambda)x_{ij} + \lambda x_{ij}^*) = \sum_{(i,j) \in A} x_{ij} t_{ij}((1-\lambda)x_{ij} + \lambda x_{ij}^*). \quad (6.15)$$

is satisfied, which is the same as (6.13).

Study these equations carefully: the coefficients x_{ij} and x_{ij}^* are constants and do not change with λ ; the only part of this condition which is affected by λ are the travel times. You can interpret this equation as trying to find a balance between \mathbf{x} and \mathbf{x}^* in the following sense: different values of λ correspond to shifting a different number of travelers from their current paths to shortest paths, which will result in different travel times on all the links. You want to pick λ so that, *after* you make the switch, both the old paths \mathbf{x} and the old shortest paths \mathbf{x}^* are equally attractive in terms of their travel times.

It is convenient to write equation (6.12) or (6.15) as a function in terms of λ as

$$\zeta'(\lambda) = \sum_{ij} t_{ij}(\lambda x_{ij}^* + (1 - \lambda)x_{ij}) (x_{ij}^* - x_{ij}) = 0, \quad (6.16)$$

which we need to solve for $\lambda \in [0, 1]$ and $\zeta(\lambda)$ is used as a shorthand for $f(\mathbf{x}(\lambda))$. Since the link performance functions are typically nonlinear, we cannot expect to be able to solve this equation analytically to get an explicit formula for λ . General techniques such as Newton's method or an equation solver can be used; but it's not too difficult to use an enlightened trial-and-error method such as a binary search or bisection. Details for all of these line search techniques were presented in Section 3.3.1, and will not be repeated here.

To summarize: in FW, \mathbf{x}^* is an all-or-nothing assignment (just as with MSA). The difference is that λ is chosen to solve (6.16). So, there is a little bit more work at each iteration (we have to solve an equation for λ instead of using a precomputed formula as in MSA), but the reward is much faster convergence to the equilibrium solution. Two examples of FW now follow; you are asked to provide a proof of correctness in Exercise 14 and 15.

6.2.3 Small network example

Here we solve the small example of Figure 6.1 by FW. Some steps are similar to MSA, and therefore omitted. Here, when we do the bisection method, we do five interval reductions, so we are within $1/2^5 = 1/32$ of the correct λ^* value. When solving by computer, you would usually perform more steps than this, because the bisection calculations are very fast.

Initialization. As before, we load everybody on the initial shortest path, so $\mathbf{x} = \mathbf{x}^* = [50 \ 0]$ and $\mathbf{t} = [60 \ 20]$.

Iteration 1. As before, the relative gap is $\gamma_1 = 2$. With the new shortest paths, $\mathbf{x}^* = [0 \ 50]$. Begin the bisection method.

Bisection Iteration 1. Initially $\lambda^* \in [0, 1]$. Calculate $\zeta'(1/2) = (0 - 50) \times (10 + 25) + (50 - 0) \times (20 + 25) = 500 > 0$ so we discard the upper half.

Bisection Iteration 2. Now we know $\lambda^* \in [0, 1/2]$. Calculate

$$\zeta'(1/4) = (0 - 50) \times (10 + 37.5) + (50 - 0) \times (20 + 12.5) = -750 < 0$$

so we discard the lower half.

Bisection Iteration 3. Now we know $\lambda^* \in [1/4, 1/2]$. Calculate

$$\zeta'(3/8) = (0 - 50) \times (10 + 31.25) + (50 - 0) \times (20 + 18.75) = -125 < 0$$

so we discard the lower half.

Bisection Iteration 4. Now we know $\lambda^* \in [3/8, 1/2]$. Calculate

$$\zeta'(7/16) = (0 - 50) \times (10 + 28.125) + (50 - 0) \times (20 + 21.875) = 187.5$$

so we discard the upper half.

Bisection Iteration 5. Now we know $\lambda^* \in [3/8, 7/16]$. Calculate

$$\zeta'(13/32) = (0 - 50) \times (10 + 29.6875) + (50 - 0) \times (20 + 20.3125) = 31.25$$

so we discard the upper half.

From here we take the midpoint of the last interval $[3/8, 13/32]$ to estimate $\lambda^* \approx 25/64 = 0.390625$, so $\mathbf{x} = (25/64)\mathbf{x}^* + (39/64)\mathbf{x} = [30.47 \quad 19.53]$ and $\mathbf{t} = [40.47 \quad 39.53]$.

Iteration 2. The relative gap is calculated as $\gamma_1 = 0.014$. (This is an order of magnitude smaller than the relative gap MSA found by this point.) The shortest paths are still $\mathbf{x}^* = [0 \quad 50]$, and we begin bisection.

Bisection Iteration 1. Initially $\lambda^* \in [0, 1]$. Calculate

$$\zeta'(1/2) = 900 > 0$$

so we discard the upper half.

Bisection Iteration 2. Now we know $\lambda^* \in [0, 1/2]$. Calculate

$$\zeta'(1/4) = 435 > 0$$

so we discard the upper half.

Bisection Iteration 3. Now we know $\lambda^* \in [0, 1/4]$. Calculate

$$\zeta'(1/8) = 203 > 0$$

so we discard the upper half.

Bisection Iteration 4. Now we know $\lambda^* \in [0, 1/8]$. Calculate

$$\zeta'(1/16) = 87 > 0$$

so we discard the upper half.

Bisection Iteration 5. Now we know $\lambda^* \in [0, 1/16]$. Calculate

$$\zeta'(1/32) = 29 > 0$$

so we discard the upper half.

The midpoint of the final interval is $\lambda^* \approx 1/64$, so $\mathbf{x} = (1/64)\mathbf{x}^* + (63/64)\mathbf{x} = [29.99 \quad 20.01]$ and $\mathbf{t} = [39.99 \quad 40.01]$.

Iteration 3. The relative gap is now $\gamma_1 = 0.00014$, so we quit and claim we have found flows that are “good enough” (the difference in travel times between the routes is less than a second).

Alternately, using calculus, we could have identified λ^* during the second iteration as *exactly* 0.40, which would have found the exact equilibrium after only one step.

6.2.4 Large network example

Here we apply FW to the network shown in Figure 6.2, using the same notation as in the MSA example.

Initialization. Path $[1, 3]$ is shortest for OD pair $(1,3)$, and path $[2, 4]$ is shortest for OD pair $(2,4)$, so

$$\mathbf{x}^* = [5000 \ 0 \ 0 \ 0 \ 0 \ 0 \ 10000]$$

and

$$\mathbf{x} = \mathbf{x}^* = [5000 \ 0 \ 0 \ 0 \ 0 \ 0 \ 10000] .$$

Recalculating the travel times, we have

$$\mathbf{t} = [60 \ 10 \ 10 \ 10 \ 10 \ 10 \ 110] .$$

Iteration 1. With the new travel times, the shortest path for $(1,3)$ is now $[1, 5, 6, 3]$, and the new shortest path for $(2,4)$ is $[2, 5, 6, 4]$, so $AEC = 63.33$. If everyone were to take the new shortest paths, the flows would be

$$\mathbf{x}^* = [0 \ 5000 \ 15000 \ 5000 \ 10000 \ 10000 \ 0] .$$

Begin the bisection method to find the right combination of \mathbf{x}^* and \mathbf{x} .

Bisection Iteration 1. Initially $\lambda^* \in [0, 1]$. Calculate

$$\begin{aligned} \zeta'(1/2) &= (0 - 5000) \times (10 + 2500/100) + \dots \\ &\quad + (0 - 10000) \times (10 + 5000/100) = 2050000 > 0 \end{aligned}$$

so we discard the upper half.

Bisection Iteration 2. Now we know $\lambda^* \in [0, 1/2]$. Calculate

$$\begin{aligned} \zeta'(1/4) &= (0 - 5000) \times (10 + 3750/100) + \dots \\ &\quad + (0 - 10000) \times (10 + 7500/100) = 550000 > 0 \end{aligned}$$

so we discard the upper half.

Bisection Iteration 3. Now we know $\lambda^* \in [0, 1/4]$. Calculate

$$\begin{aligned} \zeta'(1/8) &= (0 - 5000) \times (10 + 4375/100) + \dots \\ &\quad + (0 - 10000) \times (10 + 8750/100) = -200000 < 0 \end{aligned}$$

so we discard the lower half.

Bisection Iteration 4. Now we know $\lambda^* \in [1/8, 1/4]$. Calculate

$$\begin{aligned} \zeta'(3/16) &= (0 - 5000) \times (10 + 4062/100) + \dots \\ &\quad + (0 - 10000) \times (10 + 8125/100) = 175000 > 0 \end{aligned}$$

so we discard the upper half.

Bisection Iteration 5. Now we know $\lambda^* \in [1/8, 3/16]$. Calculate

$$\begin{aligned}\zeta'(5/32) &= (0 - 5000) \times (10 + 4219/100) + \dots \\ &\quad + (0 - 10000) \times (10 + 8437/100) = -12500 < 0\end{aligned}$$

so we discard the lower half.

The final interval is $\lambda^* \in [5/32, 3/16]$, so the estimate is $\lambda^* = 11/64$ and

$$\mathbf{x} = \frac{11}{64}\mathbf{x}^* + \frac{53}{64}\mathbf{x} = [4141 \quad 859 \quad 2578 \quad 859 \quad 1719 \quad 1719 \quad 8281].$$

The new travel times are thus

$$\mathbf{t} = [51.4 \quad 18.6 \quad 35.8 \quad 18.6 \quad 27.2 \quad 27.2 \quad 92.8].$$

Iteration 2. With the new travel times, the shortest path for (1,3) is now [1, 3], but the shortest path for (2,4) is still [2, 5, 6, 4]. The relative gap is $AEC = 2.67$ (roughly 30 times smaller than the corresponding point in the MSA algorithm!) We have

$$\mathbf{x}^* = [5000 \quad 0 \quad 10000 \quad 0 \quad 10000 \quad 10000 \quad 0].$$

We begin the bisection method to find the right combination of \mathbf{x}^* and \mathbf{x} .

Bisection Iteration 1. Initially $\lambda^* \in [0, 1]$. Calculate

$$\zeta'(1/2) = 637329 > 0$$

so we discard the upper half.

Bisection Iteration 2. Now we know $\lambda^* \in [0, 1/2]$. Calculate

$$\zeta'(1/4) = 154266 > 0$$

so we discard the upper half.

Bisection Iteration 3. Now we know $\lambda^* \in [0, 1/4]$. Calculate

$$\zeta'(1/8) = 36063 > 0$$

so we discard the upper half.

Bisection Iteration 4. Now we know $\lambda^* \in [0, 1/8]$. Calculate

$$\zeta'(1/16) = 7741 > 0$$

so we discard the upper half.

Bisection Iteration 5. Now we know $\lambda^* \in [0, 1/16]$. Calculate

$$\zeta'(1/32) = 1302 > 0$$

so we discard the upper half.

The final interval is $\lambda^* \in [0, 1/32]$, so the estimate is $\lambda^* = 1/64$ and

$$\mathbf{x} = \frac{1}{64}\mathbf{x}^* + \frac{63}{64}\mathbf{x} = [4154 \quad 845 \quad 2694 \quad 845 \quad 1848 \quad 1848 \quad 8152].$$

The new travel times are thus

$$\mathbf{t} = [51.5 \quad 18.5 \quad 36.9 \quad 18.5 \quad 28.5 \quad 28.5 \quad 91.5].$$

At this point, the average excess cost is around 1.56 min; note that FW is able to decrease the relative gap much faster than MSA. However, we're still quite far from equilibrium if you compute the actual path travel times. In this case, even though we're allowing the step size to vary for each iteration, we are forcing travelers from all OD pairs to shift in the same proportion. In reality, OD pairs farther from equilibrium should see bigger flow shifts, and OD pairs closer to equilibrium should see smaller ones. This can be remedied by more advanced algorithms.

6.2.5 Conjugate Frank-Wolfe

If the distinction between FW and MSA is that FW chooses the step size λ in a more clever way, the distinction between conjugate Frank-Wolfe (CFW) and plain FW is that CFW chooses the target \mathbf{x}^* in a more clever way. To understand why CFW is more clever, we first need to understand why using the all-or-nothing assignment as target can be problematic.

Viewed in terms of the set of feasible link assignments X , the all-or-nothing assignments correspond to corner points of X . That is, Frank-Wolfe must limit itself to the corner points of the feasible region when determining where to move. In Figure 6.4, FW is constrained to follow the trajectory shown by the thin lines, and is unable to take a direct step like that indicated by the thick arrow. While these directions are effective in the early iterations, as the algorithm approaches the equilibrium point its converge slows down dramatically, and “zigzagging” behavior is observed.

So, how can we choose the target solution in a smarter way, so that steps in the direction of the target still move toward equilibrium, while granting more flexibility than the use of an all-or-nothing assignment? Conjugate Frank-Wolfe provides one approach towards doing so.

Understanding CFW requires introducing the concept of conjugacy, which is done here. Temporarily ignoring the context of the traffic assignment problem, assume that we are trying to find the minimum point of a convex quadratic function $f(x_1, x_2)$ of two variables, when there are no constraints. Figure 6.5 shows a few examples of these functions. Any quadratic function of two variables can be written in the form

$$f(\mathbf{x}) = \mathbf{x}^T \mathbf{Q} \mathbf{x} + \mathbf{b}^T \mathbf{x}, \quad (6.17)$$

where $\mathbf{x} = [x_1 \quad x_2]$ and \mathbf{b} are two-dimensional vectors, and \mathbf{Q} is a 2×2 matrix. Figure 6.5 shows the \mathbf{Q} matrix and \mathbf{b} vector corresponding to each example.

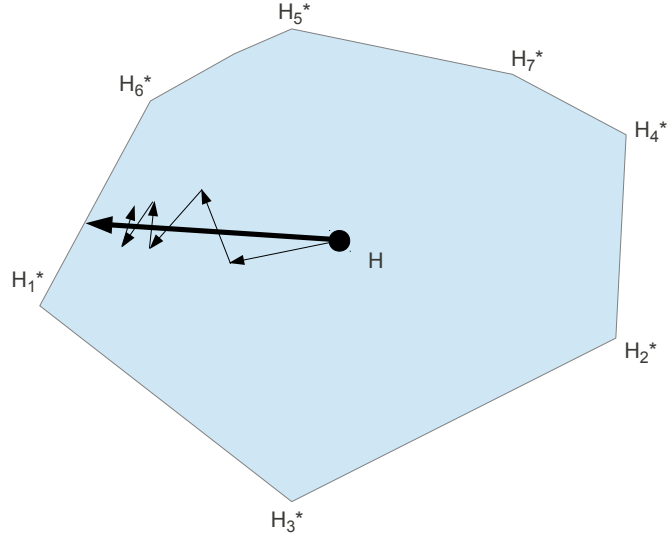


Figure 6.4: Frank-Wolfe can only move towards extreme points of the feasible region.

How would you go about finding the minimum of such a function? Given some initial solution (x_1, x_2) , one idea is to fix x_1 as a constant, and find the value of x_2 which minimizes f . Then, we can fix x_2 , and find the value of x_1 which minimizes f , and so on. This process will converge to the minimum, as shown in Figure 6.6, but in general this convergence is only asymptotic, and the process will never actually reach the minimum. The exception is when \mathbf{Q} is the identity matrix, as in Figure 6.6(a). In this case, the exact optimum is reached in only two steps.

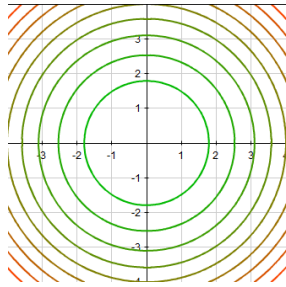
In fact, it is possible to reach the exact optimum in only two steps even when \mathbf{Q} is not the identity matrix, by changing the search directions. The process described above (alternately fixing x_1 , and then x_2) can be thought of as alternating between searching in the direction $[0 \ 1]$, then searching in the direction $[1 \ 0]$. As shown in Figure 6.7, by making a different choice for the two search directions, the minimum can always be obtained in exactly two steps.

This happens if the two directions \mathbf{d}_1 and \mathbf{d}_2 are *conjugate*, that is, if

$$\mathbf{d}_1^T \mathbf{Q} \mathbf{d}_2 = 0 \quad (6.18)$$

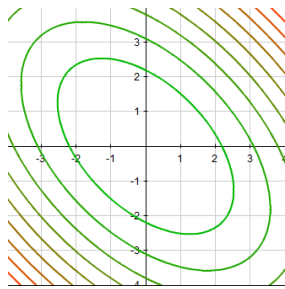
Conjugacy generalizes the concept of orthogonality (or perpendicularity). If \mathbf{Q} is the identity matrix, equation (6.18) reduces to $\mathbf{d}_1 \cdot \mathbf{d}_2 = 0$, the definition of perpendicular vectors.

Now, returning to the traffic assignment problem, we want to use the concept of conjugacy to choose a more intelligent search direction. In particular, we want the target \mathbf{x}^* to be chosen so that the search direction $\mathbf{x}^* - \mathbf{x}$ is conjugate to



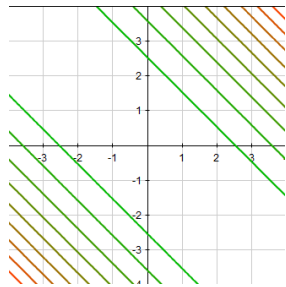
$$\mathbf{Q} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad \mathbf{b} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

(a) $f(x_1, x_2) = x_1^2 + x_2^2$



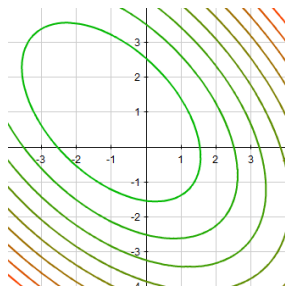
$$\mathbf{Q} = \begin{bmatrix} 1 & 1/2 \\ 1/2 & 1 \end{bmatrix} \quad \mathbf{b} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

(b) $f(x_1, x_2) = x_1^2 + x_2^2 + x_1x_2$



$$\mathbf{Q} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \quad \mathbf{b} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

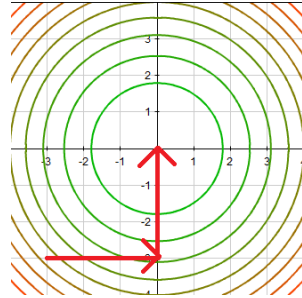
(c) $f(x_1, x_2) = x_1^2 + x_2^2 + 2x_1x_2$



$$\mathbf{Q} = \begin{bmatrix} 1 & 1/2 \\ 1/2 & 1 \end{bmatrix} \quad \mathbf{b} = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$$

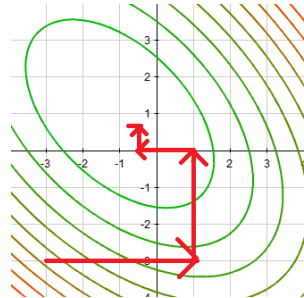
(d) $f(x_1, x_2) = x_1^2 + x_2^2 + x_1x_2 + x_1 - x_2$

Figure 6.5: Four examples of convex quadratic functions of the form $f(\mathbf{x}) = \mathbf{x}^T \mathbf{Q} \mathbf{x} + \mathbf{b}^T \mathbf{x}$



$$\mathbf{Q} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad \mathbf{b} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

(a) $f(x_1, x_2) = x_1^2 + x_2^2$



$$\mathbf{Q} = \begin{bmatrix} 1 & 1/2 \\ 1/2 & 1 \end{bmatrix} \quad \mathbf{b} = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$$

(b) $f(x_1, x_2) = x_1^2 + x_2^2 + x_1x_2 + x_1 - x_2$

Figure 6.6: Searching in orthogonal directions finds the optimum in two steps only if $\mathbf{Q} = \mathbf{I}$.

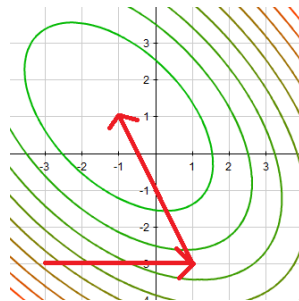


Figure 6.7: Searching in conjugate directions always leads to the optimum in two steps.

the previous search direction. Before the derivation, there are a few differences between traffic assignment and the unconstrained quadratic program used to introduce conjugacy which should be addressed.

- The Beckmann function is not a function of two variables. This is not a huge problem. Finding the unconstrained minimum of a quadratic function of n variables requires n conjugate steps (so, in the above examples with two variables, two steps sufficed), due to a result known as the expanding subspace theorem. Of course...
- The Beckmann function is in general not a quadratic function. (Can you think of a case when it is?) Instead of the matrix \mathbf{Q} , we will instead use the Hessian of the Beckmann function Hf . Therefore, we cannot hope for exact convergence in a finite number of iterations. However, when the solution gets closer and closer to equilibrium, the Beckmann function can be better and better approximated by a quadratic by taking the first two terms of its Taylor series. This is good news, because zigzagging in plain Frank-Wolfe becomes worse and worse as the equilibrium solution is approached.
- The optimization problem (6.3)–(6.6) has constraints. Again, this is not a major problem, since the feasible region is convex. As long as we ensure that the target point \mathbf{x}^* is feasible, any choice of $\lambda \in [0, 1]$ will retain feasibility.

So, how can we make sure that the new target vector \mathbf{x}^* is chosen so that the search direction is conjugate to the previous direction, and that \mathbf{x}^* is feasible? Since X is a convex set, feasibility can be assured by choosing \mathbf{x}^* to be a convex combination of the old target vector ($\mathbf{x}_{\text{old}}^*$) and the all-or-nothing assignment \mathbf{x}^{AON} :

$$\mathbf{x}^* = \alpha \mathbf{x}_{\text{old}}^* + (1 - \alpha) \mathbf{x}^{\text{AON}}, \quad (6.19)$$

for some $\alpha \in [0, 1]$. Choosing $\alpha = 0$ would make the all-or-nothing assignment the target (as in plain FW), while choosing $\alpha = 1$ would make the target in this iteration the same as in the last. In fact, α should be chosen so that the new direction is conjugate to the last, that is,

$$(\mathbf{x}_{\text{old}}^* - \mathbf{x})^T \mathbf{H}(\mathbf{x}^* - \mathbf{x}) = 0, \quad (6.20)$$

where \mathbf{H} is the Hessian of the Beckmann function evaluated at the current solution \mathbf{x} . Substituting (6.19) into (6.20), we can solve for α as follows:

$$(\mathbf{x}_{\text{old}}^* - \mathbf{x})^T \mathbf{H}(\mathbf{x}^* - \mathbf{x}) = 0 \quad (6.21)$$

$$\iff (\mathbf{x}_{\text{old}}^* - \mathbf{x})^T \mathbf{H}(\alpha \mathbf{x}_{\text{old}}^* + (1 - \alpha) \mathbf{x}^{\text{AON}} - \alpha \mathbf{x} - (1 - \alpha) \mathbf{x}) = 0 \quad (6.22)$$

$$\iff (\mathbf{x}_{\text{old}}^* - \mathbf{x})^T \mathbf{H}(\alpha(\mathbf{x}_{\text{old}}^* - \mathbf{x}) + (1 - \alpha)(\mathbf{x}^{\text{AON}} - \mathbf{x})) = 0 \quad (6.23)$$

or, after rearrangement,

$$\alpha[(\mathbf{x}_{\text{old}}^* - \mathbf{x})^T \mathbf{H}(\mathbf{x}_{\text{old}}^* - \mathbf{x}^{\text{AON}})] = -(\mathbf{x}_{\text{old}}^* - \mathbf{x})^T \mathbf{H}(\mathbf{x}^{\text{AON}} - \mathbf{x}) \quad (6.24)$$

$$\iff \alpha = \frac{(\mathbf{x}_{\text{old}}^* - \mathbf{x})^T \mathbf{H}(\mathbf{x}^{\text{AON}} - \mathbf{x})}{(\mathbf{x}_{\text{old}}^* - \mathbf{x})^T \mathbf{H}(\mathbf{x}_{\text{AON}}^* - \mathbf{x}_{\text{old}})} \quad (6.25)$$

Now, for TAP, the Hessian takes a specific form. Since the Beckmann function is

$$f(\mathbf{x}) = \sum_{(i,j) \in A} \int_0^{x_{ij}} t_{ij}(x) dx, \quad (6.26)$$

its gradient is simply the vector of travel times at the current flows

$$\nabla f(\mathbf{x}) = \text{vect}\{t_{ij}(x_{ij})\}, \quad (6.27)$$

and its Hessian is the diagonal matrix of travel time derivatives at the current flows

$$Hf(\mathbf{x}) = \text{diag}\{t'_{ij}(x_{ij})\}. \quad (6.28)$$

So, the matrix products in equation (6.25) can be written out explicitly, giving

$$\alpha = \frac{\sum_{(i,j) \in A} ((x_{\text{old}}^*)_{ij} - x_{ij})(x_{ij}^{\text{AON}} - x_{ij})t'_{ij}}{\sum_{(i,j) \in A} ((x_{\text{old}}^*)_{ij} - x_{ij})(x_{ij}^{\text{AON}} - (x_{\text{old}}^*)_{ij})t'_{ij}}, \quad (6.29)$$

where the derivatives t'_{ij} are evaluated at the current link flows x_{ij} .

Almost there! A careful reader may have some doubts about the formula in (6.29). First, it is possible that the denominator can be zero, and division by zero is undefined. Second, to ensure feasibility of \mathbf{x}^* , we need $\alpha \in [0, 1]$, even though it is not obvious that this formula always lies in this range (and in fact, it need not do so). Furthermore, $\alpha = 1$ is undesirable, because then the current target point is the same as the target point in the last iteration. If the previous line search was exact, there will be no further improvement and the algorithm will be stuck in an infinite loop. Finally, what should you do for the first iteration, when there is no “old” target \mathbf{x}^* ?

To address the first issue, the easiest approach is to simply set $\alpha = 0$ if the denominator of (6.29) is zero (i.e., if the formula is undefined, simply take a plain FW step by using the all-or-nothing solution as the target). As for the second and third issues, if the denominator is nonzero we can project the right-hand side of (6.29) onto the interval $[0, 1 - \epsilon]$ where $\epsilon > 0$ is some small tolerance value. That is, if equation (6.29) would give a value greater than $1 - \epsilon$, set $\alpha = 1 - \epsilon$; if it would give a negative value, use zero. Finally, for the first iteration, simply use the all-or-nothing solution as the target: $\mathbf{x}^* = \mathbf{x}^{\text{AON}}$.

So, to summarize the discussion, choose α in the following way. If it is the first iteration or the denominator of (6.29) is zero, set $\alpha = 0$. Otherwise set

$$\alpha = \text{proj}_{[0, 1 - \epsilon]} \left(\frac{\sum_{(i,j) \in A} t'_{ij}((x_{\text{old}}^*)_{ij} - x_{ij})(x_{ij}^{\text{AON}} - x_{ij})}{\sum_{(i,j) \in A} t'_{ij}((x_{\text{old}}^*)_{ij} - x_{ij})(x_{ij}^{\text{AON}} - (x_{\text{old}}^*)_{ij})} \right) \quad (6.30)$$

Then the target solution \mathbf{x}^* is calculated using (6.19). The value of the step size λ is chosen in the same way as in Frank-Wolfe, by performing a line search (e.g., using bisection or Newton's method) to solve (6.16).

Large network example Here we apply CFW to the network shown in Figure 6.2, using the same notation as in the FW and MSA examples. The tolerance ϵ is chosen to be a small positive constant, 0.01 in the following example.

Initialization. Generate the initial solution by solving an all-or-nothing assignment. Path [1, 3] is shortest for OD pair (1,3), and path [2, 4] is shortest for OD pair (2,4), so

$$\mathbf{x}^{\text{AON}} = [5000 \ 0 \ 0 \ 0 \ 0 \ 0 \ 10000]$$

and

$$\mathbf{x} = \mathbf{x}^{\text{AON}} = [5000 \ 0 \ 0 \ 0 \ 0 \ 0 \ 10000] .$$

Recalculating the travel times, we have

$$\mathbf{t} = [60 \ 10 \ 10 \ 10 \ 10 \ 10 \ 110] .$$

Iteration 1. Proceeding in the same way as in the large network example for Frank-Wolfe, the all-or-nothing assignment in this case is

$$\mathbf{x}^{\text{AON}} = [0 \ 5000 \ 0 \ 10000 \ 15000 \ 5000 \ 10000] .$$

which is used as the target \mathbf{x}^* since this is the first iteration of CFW. Repeating the same line search process, the optimal value of λ is 19/120, producing the new solution

$$\mathbf{x} = [4208 \ 792 \ 8417 \ 1583 \ 2375 \ 792 \ 1583] .$$

Iteration 2. Again as with regular Frank-Wolfe, based on the updated travel times the all-or-nothing assignment is

$$\mathbf{x}^{\text{AON}} = [5000 \ 0 \ 0 \ 10000 \ 10000 \ 0 \ 10000] .$$

However, here FW and CFW take different paths. Rather than using \mathbf{x}^{AON} as the target vector, CFW generates a conjugate search direction. First calculate the right-hand side of (6.29). Since for this problem $t_{ij} = 1/100$ for all links (regardless of the flow), the formula is especially easy to compute using \mathbf{x} and \mathbf{x}^* from the previous iteration and \mathbf{x}^{AON} as just now computed. The denominator of (6.29) is nonzero, and the formula gives -2.37 ; projecting onto the set $[0, 1 - \epsilon]$ thus gives $\alpha = 0$. So, calculating the target \mathbf{x}^* from equation (6.19) with $\alpha = 0$ we have

$$\mathbf{x}^* = [5000 \ 0 \ 0 \ 10000 \ 10000 \ 0 \ 10000]$$

and, using a line search between \mathbf{x} and \mathbf{x}^* , find that $\lambda = 0.0321$ is best, resulting in

$$\mathbf{x} = [4233 \ 766 \ 8146 \ 1853 \ 2620 \ 766 \ 1854] .$$

Iteration 3. With the new flows \mathbf{x} , the travel times are now

$$\mathbf{t} = [52.3 \quad 17.7 \quad 91.5 \quad 28.5 \quad 36.2 \quad 17.7 \quad 28.5]$$

and the all-or-nothing assignment is

$$\mathbf{x}^{\text{AON}} = [5000 \quad 0 \quad 10000 \quad 0 \quad 0 \quad 0 \quad 0] .$$

Calculating the right-hand side of (6.29), we see that the denominator is nonzero, and the formula gives 0.198, which can be used as is since it lies in $[0, 1 - \epsilon]$. So, calculating the target \mathbf{x}^* from equation (6.19) with $\alpha = 0.198$ we have

$$\mathbf{x}^* = [5000 \quad 0 \quad 8024 \quad 1976 \quad 1976 \quad 0 \quad 1976] .$$

Note that unlike any of the other link-based algorithms in this section, the target flows are not an all-or-nothing assignment (i.e., not an extreme point of X). Performing a line search between \mathbf{x} and \mathbf{x}^* , find that $\lambda = 0.652$ is best, resulting in

$$\mathbf{x} = [4733 \quad 267 \quad 8067 \quad 1933 \quad 2200 \quad 267 \quad 1933] .$$

which solves the equilibrium problem exactly, so we terminate.

In this example, CFW found the exact equilibrium solution in three iterations. This type of performance is not typical (even though it is generally faster than regular FW or MSA). In this example, the link performance functions are linear, so the Beckmann function is quadratic. Iterative line searches with conjugate directions lead to the exact solution of quadratic programs in a finite number of iterations, as suggested by the above discussion. This performance cannot be assured with other types of link performance functions.

An even faster algorithm known as *biconjugate* Frank-Wolfe chooses its target so that the search direction is conjugate to both of the previous two search directions. This method converges better than CFW, but is not explained here because the details are a little more complicated even though the idea is the same.

6.3 Path-Based Algorithms

This section introduces equilibrium algorithms which work in the space of path flows H , rather than the space of link flows X . These tend to be faster, especially when high precision solutions are needed, but they require more computer memory. Furthermore, achieving the full potential of these algorithms for rapid convergence requires considerably more programming skill than for link-based algorithms.

Before explaining path-based algorithms, it is worth explaining why link-based algorithms are slow to converge. The following criticisms are specifically aimed at the Frank-Wolfe algorithm, but apply to other link-based algorithms as well.

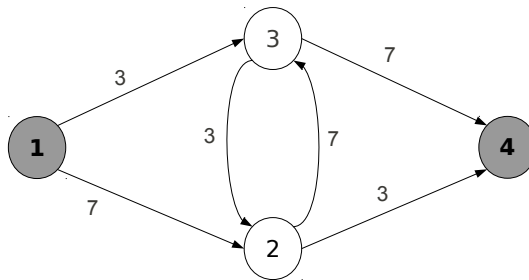


Figure 6.8: A persistent cycle in Frank-Wolfe. (Link flows shown.)

It treats all OD pairs equally. If an OD pair is close to equilibrium, only a small flow shift among its paths is needed, while if an OD pair is far from equilibrium, a larger flow shift is needed. The Frank-Wolfe method uses the same λ shift for all OD pairs regardless of how close or far away each one is from equilibrium.

It uses a restricted set of search directions. If one imagines the space of feasible traffic assignments, the “all-or-nothing” \mathbf{x}^* solutions generated by the Frank-Wolfe algorithm represent extreme points or corners of this region. In other words, the Frank-Wolfe algorithm is only capable of moving towards a corner. Initially, this is fine, but as one nears the optimal solution, this results in extensive zig-zagging when a much more direct path exists (Figure 6.4). Conjugate Frank-Wolfe is a bit better in this regard, but still uses a fairly restrictive set of target points.

It is unable to erase cyclic flows. Consider the network in Figure 6.8, with the flows as shown. Such a flow might easily arise if $[1, 2, 3, 4]$ is the shortest path during the first iteration of Frank-Wolfe, $[1, 3, 2, 4]$ is the shortest path during the second, and $\lambda = 0.3$. With only one OD pair, it is impossible for both links $(2, 3)$ and $(3, 2)$ to be used at equilibrium, as discussed in Section 5.2.3. However, the Frank-Wolfe method will always leave some flow on both links unless $\lambda = 1$ at any iteration (which is exceedingly rare, especially in later iterations when λ is typically very close to zero).

These difficulties can all be avoided by tracking the path flows \mathbf{h} , rather than the link flows \mathbf{x} . The path flows contain much more information, tracking flow by origin and destination, as opposed to link flows which are aggregated together. On balance, the number of elements in the path flow vector is many orders of magnitude larger than that of the link flows, easily numbering in the millions for realistic networks. Algorithms which require us to first list off all paths in the network are not tractable. Instead, path-based algorithms only track the

paths which an OD pair actually uses, that is the set $\hat{\Pi}^{rs} = \{\pi \in \pi^{rs} : h^\pi > 0\}$.⁴ This is often referred to as the set of *working paths* for each OD pair.⁵ A rough description of path-based algorithms can then be described as

1. Initialize $\hat{\Pi}^{rs} \leftarrow \emptyset$ for all OD pairs.
2. Repeat the following steps for each OD pair (r, s) :
 - (a) Find the shortest path $\hat{\pi}_{r,s}$. Add it to $\hat{\Pi}^{rs}$ if it's not already used.
 - (b) Shift travelers among paths to get closer to equilibrium.
 - (c) Update travel times.
3. Drop paths from $\hat{\Pi}^{rs}$ if they are no longer used; return to step 2 unless a convergence criterion is satisfied.

On the surface, this scheme looks quite similar to the link-based methods presented earlier. Why might it converge faster? Recall the three factors described above. First, each OD pair is now being treated independently. With MSA and Frank-Wolfe, the same step-size λ was applied across all links (and therefore, across all OD pairs). If one OD pair is very close to equilibrium, while another is far away, we should probably make a finer adjustment to the first OD pair, and a larger adjustment to the second one. Link-based methods allow no such finesse, and instead bluntly apply the same λ to all origins. In practice, this means that λ becomes very small after a few iterations: we can't move very far without disturbing an OD pair which is already close to equilibrium. As a result of this, it takes a really long time to solve OD pairs which are far from equilibrium. Equivalently, the set of search directions is broader in that we can vary the step size by OD pairs; and lastly, it is quite possible to erase cyclic flows in a path-based context, because the extra precision allows us to take larger steps.

Step 3 is where different path-based algorithms differ. This section describes two path-based algorithms: *gradient projection* and *manifold suboptimization*. The latter is sometimes called *projected gradient*, because both algorithms use the same two ingredients: exploiting the fact that the *gradient* is the direction of steepest ascent (and therefore, in a minimization problem, we should move in the opposite direction to descend as quickly as possible); and having to consider the constraints in the problem by using a *projection* operation to stay in the feasible set. (Recall from Chapter 3 that projection involves finding the point within a set which lies closest to another point.) Where they differ is in the order these steps are applied.

In gradient projection, we first take a step in the opposite direction of the gradient, which will typically result in an infeasible point. The projection is done *after* the flow shift, not before: we do the projection after we make use of

⁴Those familiar with other types of optimization problems might recognize this as a column generation scheme.

⁵You may recognize similarities with the “trial-and-error” method from Chapter 4. Path-based algorithms are essentially a more clever form of this method.

the gradient, so to speak. We then apply a projection operation to return to the feasible set. In manifold suboptimization, we first calculate the gradient, then project the gradient onto the feasible set so that our search direction respects the demand constraint.

6.3.1 Gradient projection

The gradient projection method identifies the direction of steepest descent for the Beckmann function, and calculates a new point in this direction. In case this point is infeasible, a projection operation is applied to return to the closest feasible point. Writing the Beckmann function in terms of path flows \mathbf{h} , rather than link flows as is customary, we have

$$f(\mathbf{h}) = \sum_{(i,j) \in A} \int_0^{\sum_{\pi \in \Pi} \delta_{ij}^{\pi} h^{\pi}} t_{ij}(x) dx, \quad (6.31)$$

and its partial derivative with respect to any path flow variable is

$$\frac{\partial f}{\partial h^{\pi}} = \sum_{(i,j) \in A} \delta_{ij}^{\pi} t_{ij} \left(\sum_{\pi' \in \Pi} \delta_{ij}^{\pi'} h^{\pi'} \right) = \sum_{(i,j) \in A} \delta_{ij}^{\pi} t_{ij}(x_{ij}) = c^{\pi}, \quad (6.32)$$

so the direction of steepest descent \mathbf{s} is the negative gradient:

$$-\nabla_{\mathbf{h}} f = -\text{vect}(c^{\pi}). \quad (6.33)$$

So, a first attempt at gradient projection would be to update $\mathbf{h} \leftarrow \text{proj}_H(\mathbf{h} - \nabla_{\mathbf{h}} f)$. Unfortunately, projecting onto the set H is not particularly easy. If we apply a suitable change of variables, though, the projection can be made much easier.

Define the *basic path* for OD pair (r, s) to be a path $\hat{\pi}_{rs}$ with minimum travel time. All of the other paths are called *nonbasic paths*. We can eliminate the basic path flow variable by expressing it in terms of the nonbasic path flows:

$$h^{\hat{\pi}_{rs}} = d^{rs} - \sum_{\pi \in \hat{\Pi}^{rs}: \pi \neq \hat{\pi}_{rs}} h^{\pi}. \quad (6.34)$$

Substituting this into the path-based Beckmann function (6.31), the partial derivative with respect to one of the nonbasic path flows is now

$$\frac{\partial \hat{f}}{\partial h^{\pi}} = c^{\pi} - c^{\hat{\pi}_{rs}} \quad \forall (r, s) \in Z^2, \pi \in \hat{\Pi}^{rs} - \{\hat{\pi}_{rs}\}, \quad (6.35)$$

denoting the Beckmann function as \hat{f} instead of f because the function has been modified by using (6.34) to eliminate some of the path flow variables.

So, the change in path flows will be the negative of the gradient. Since the gradient is given by (6.35) and $c^{\pi} \geq c^{\hat{\pi}_{rs}}$ (because the basic path is by definition

the shortest one), moving in this direction means that every nonbasic path flow will *decrease*.

Since the transformation (6.34) eliminated the demand satisfaction constraint, the only remaining constraint is that the nonbasic path flow variables be nonnegative. Projecting onto this set is trivial: if any of the nonbasic path flow variables is negative after taking a step, simply set it to zero. At this point, the basic path flow can be calculated through equation (6.34).

Furthermore, the larger the difference in travel times, the larger the corresponding element of the gradient will be. This suggests that more flow be shifted away from paths with higher travel times. We can go a step further, and estimate directly how much flow should be shifted from a nonbasic path to a basic path to equalize the travel times, using Newton's method.

Let Δh denote the amount of flow we shift *away* from non-basic path π and *onto* the basic path $\hat{\pi}$, and let $c_\pi(\Delta h)$ and $c_{\hat{\pi}}(\Delta h)$ denote the travel times on path π and $\hat{\pi}$ after we make such a shift. We want to choose Δh so these costs are equal, that is, so

$$g(\Delta h) = c_\pi(\Delta h) - c_{\hat{\pi}}(\Delta h) = 0; \quad (6.36)$$

that is, g is simply the difference in travel times between the two paths. To apply Newton's method, we need to find the derivative of g with respect to Δh .

Using the relationships between link travel times and path travel times, we have

$$g(\Delta h) = \sum_{(i,j) \in \mathcal{A}} (\delta_{ij}^\pi - \delta_{ij}^{\hat{\pi}}) t_{ij}(x_{ij}(\Delta h)),$$

so

$$g'(\Delta h) = \sum_{(i,j) \in \mathcal{A}} (\delta_{ij}^\pi - \delta_{ij}^{\hat{\pi}}) \frac{dt_{ij}}{dx_{ij}} \frac{dx_{ij}}{d\Delta h}$$

by the chain rule. For each arc, there are four possible cases:

Case I : $\delta_{ij}^\pi = \delta_{ij}^{\hat{\pi}} = 0$, that is, neither path π nor path $\hat{\pi}$ uses link (i, j) . Then

$$(\delta_{ij}^\pi - \delta_{ij}^{\hat{\pi}}) \frac{dt_{ij}}{dx_{ij}} \frac{dx_{ij}}{d\Delta h} = 0 \text{ and this link does not contribute to the derivative.}$$

Case II : $\delta_{ij}^\pi = \delta_{ij}^{\hat{\pi}} = 1$, that is, both paths π and path $\hat{\pi}$ use link (i, j) .

Then $(\delta_{ij}^\pi - \delta_{ij}^{\hat{\pi}}) \frac{dt_{ij}}{dx_{ij}} \frac{dx_{ij}}{d\Delta h} = 0$ and this link again does not contribute to the derivative. (Another way to think of it: since both paths use this arc, its total flow will not change if we shift travelers from one path to another.)

Case III : $\delta_{ij}^\pi = 1$ and $\delta_{ij}^{\hat{\pi}} = 0$, that is, path π uses link (i, j) , but path $\hat{\pi}$ does not. Then

$$(\delta_{ij}^\pi - \delta_{ij}^{\hat{\pi}}) \frac{dt_{ij}}{dx_{ij}} \frac{dx_{ij}}{\Delta h} = \frac{dt_{ij}}{dx_{ij}} \frac{dx_{ij}}{d\Delta h} = -\frac{dt_{ij}}{dx_{ij}} \text{ since } \frac{dx_{ij}}{d\Delta h} = -1.$$

Case IV : $\delta_{ij}^\pi = 0$ and $\delta_{ij}^{\hat{\pi}} = 1$, that is, path $\hat{\pi}$ uses link (i, j) , but path π does not. Then

$$(\delta_{ij}^\pi - \delta_{ij}^{\hat{\pi}}) \frac{dt_{ij}}{dx_{ij}} \frac{dx_{ij}}{\Delta h} = -\frac{dt_{ij}}{dx_{ij}} \frac{dx_{ij}}{d\Delta h} = -\frac{dt_{ij}}{dx_{ij}} \text{ since } \frac{dx_{ij}}{d\Delta h} = 1.$$

Putting it all together, the only terms which contribute to the derivative are the links which are used by either π or $\hat{\pi}$, but not both. Let A_1 , A_2 , A_3 , and A_4 denote the sets of links falling into the four cases listed above. Then

$$g'(\Delta h) = - \sum_{(i,j) \in A_3 \cup A_4} \frac{dt_{ij}}{dx_{ij}}.$$

which is simply the negative sum of the derivatives of these links, evaluated at the current link flows.

Then, starting with an initial guess of $\Delta h = 0$, one step of Newton's method gives an improved guess of

$$\Delta h = 0 - g(0)/g'(0) = \frac{c_\pi - c_{\hat{\pi}}}{\sum_{a \in A_3 \cup A_4} \frac{dt_{ij}}{dx_{ij}}}.$$

That is, the recommended Newton shift is given by the difference in path costs, divided by the sum of the derivatives of the link performance functions for links used by one path or the other, but not both. Therefore, the updated nonbasic and basic path flows are given by

$$h_{\hat{\pi}} \leftarrow h_{\hat{\pi}} + \frac{c_\pi - c_{\hat{\pi}}}{\sum_{a \in A_3 \cup A_4} \frac{dt_{ij}}{dx_{ij}}}$$

and

$$h_\pi \leftarrow h_\pi - \frac{c_\pi - c_{\hat{\pi}}}{\sum_{a \in A_3 \cup A_4} \frac{dt_{ij}}{dx_{ij}}}.$$

This process is repeated for every nonbasic path.

This is demonstrated on the example in Figure 6.2 as follows.

Iteration 1, Step 1. Initially $\hat{\Pi}^{13} = \hat{\Pi}^{24} = \emptyset$.

Iteration 1, Step 2a, OD pair (1,3). Find the shortest path for (1,3): with no travelers on the network, the top link has a travel time of 10. This is not in the set of used paths, so include it: $\hat{\Pi}^{13} = \{[1, 3]\}$.

Iteration 1, Step 2b, OD pair (1,3). Since there is only one used path, we simply have $h_{[1,3]}^{13} = 5000$.

Iteration 1, Step 2c, OD pair (1,3). Update travel times:

$$\mathbf{t} = [60 \quad 10 \quad 10 \quad 10 \quad 10 \quad 10 \quad 10]$$

Iteration 1, Step 2a, OD pair (2,4). Find the shortest path for (2,4): with no travelers on the network, link 7 has a travel time of 10. This is not in the set of used paths, so include it: $\hat{\Pi}^{24} = \{[2, 4]\}$.

Iteration 1, Step 2b. Since there is only one used path, $h_{[2,4]}^{24} = 10000$.

Iteration 1, Step 2c. Update travel times:

$$\mathbf{t} = [60 \quad 10 \quad 10 \quad 10 \quad 10 \quad 10 \quad 110]$$

Iteration 1, Step 3. All paths are used, so return to step 2. The relative gap is $\gamma_1 = 2.11$.

Iteration 2, Step 2a, OD pair (1,3). The shortest path is now $[1,5,6,3]$. This is not part of the set of used paths, so we add it: $\hat{\Pi}^{13} = \{[1,3], [1,5,6,3]\}$.

Iteration 2, Step 2b, OD pair (1,3). The difference in travel times between the paths is 30 minutes; and the sum of the derivatives of links 1, 2, 3, and 4 is 0.04. So we shift $30/0.04 = 750$ vehicles from $[1,3]$ to $[1,5,6,3]$, $h_{[1,3]}^{13} = 4250$ and $h_{[1,5,6,3]}^{13} = 750$.

Iteration 2, Step 2c, OD pair (1,3). Update travel times:

$$\mathbf{t} = [52.5 \quad 17.5 \quad 17.5 \quad 17.5 \quad 10 \quad 10 \quad 110]$$

Note that the two paths have exactly the same cost after only one step! This is because Newton's method is exact for linear functions.

Iteration 2, Step 2a, OD pair (2,4). The shortest path is now $[2,5,6,4]$. This is not part of the set of used paths, so we add it: $\hat{\Pi}^{24} = \{[2,4], [2,5,6,4]\}$.

Iteration 2, Step 2b, OD pair (2,4). The difference in travel times between the paths is 72.5 minutes; and the sum of the derivatives of links 5, 3, 6, and 7 is 0.04. So we shift $72.5/0.04 = 1812.5$ vehicles from $[2,4]$ to $[2,5,6,4]$, $h_{[2,4]}^{24} = 8187.5$ and $h_{[2,5,6,4]}^{24} = 1812.5$.

Iteration 2, Step 2c, OD pair (2,4). Update travel times:

$$\mathbf{t} = [52.5 \quad 17.5 \quad 35.625 \quad 17.5 \quad 28.125 \quad 28.125 \quad 91.875]$$

Note that the two paths again have exactly the same cost. However, the equilibrium for the first OD pair has been disturbed.

Iteration 2, Step 3. All paths are used, so return to step 2. The relative gap is $\gamma_1 = 0.0115$.

Iteration 3, Step 2a, OD pair (1,3). The shortest path is now $[1,3]$, which is already in the set of used paths, so nothing to do here.

Iteration 3, Step 2b, OD pair (1,3). The difference in travel times between the paths is 18.125 minutes; and the sum of the derivatives of links 1, 2, 3, and 4 is 0.04. So we shift $18.125/0.04 = 453$ vehicles from $[1,5,6,3]$ to $[1,3]$, $h_{[1,3]}^{13} = 4703$ and $h_{[1,5,6,3]}^{13} = 297$.

Iteration 3, Step 2c, OD pair (1,3). Update travel times:

$$\mathbf{t} = [57.0 \quad 13.0 \quad 31.1 \quad 13.0 \quad 28.1 \quad 28.1 \quad 91.9]$$

Again the first OD pair is at equilibrium, up to rounding error.

Iteration 3, Step 2a, OD pair (2,4). The shortest path is again [2,5,6,4]. This is already in the set of used paths, so nothing to do here.

Iteration 3, Step 2b, OD pair (2,4). The difference in travel times between the paths is 4.6 minutes; and the sum of the derivatives of links 5, 3, 6, and 7 is 0.04. So we shift $4.6/0.04 = 114$ vehicles from [2,4] to [2,5,6,4], $h_{[2,4]}^{24} = 8073$ and $h_{[2,5,6,4]}^{24} = 1927$.

Iteration 3, Step 2c, OD pair (2,4). Update travel times:

$$\mathbf{t} = [57.0 \quad 13.0 \quad 32.2 \quad 13.0 \quad 29.3 \quad 29.3 \quad 90.7]$$

Iteration 3, Step 3. All paths are used, so return to step 2. The relative gap is $\gamma_1 = 0.00028$.

Note that after three iterations of gradient projection, the relative gap is two orders of magnitude smaller than that from the Frank-Wolfe algorithm. Although not demonstrated here for reasons of space, the performance of gradient projection relative to Frank-Wolfe actually *improves* from here on out. Frank-Wolfe usually does most of its work in the first few iterations, and then converges *very* slowly after that.⁶ On the other hand, gradient projection maintains a steady rate of progress throughout, with a nearly constant proportionate decrease in gap from one iteration to the next.

6.3.2 Manifold suboptimization

The manifold suboptimization algorithm is also based on the Beckmann formulation, applying an algorithm of Rosen from nonlinear optimization. As with gradient projection, moving in the direction of the negative gradient will travel along the direction of steepest descent. Such a direction will tend towards the equilibrium solution, which minimizes the Beckmann function. However, we need to be careful not to leave the region of feasible path flows H , making sure that our search direction still satisfies the demand constraint and retains non-negative path flows. Unlike gradient projection, which fixes an infeasible move by projecting back to the feasible region, manifold suboptimization avoids ever leaving the feasible region in the first place by modifying the steepest descent direction. This algorithm goes by several names in the literature; sometimes called projected gradient and at other times even called gradient projection (even though it is different than the gradient projection method of Section 6.3.1).

⁶It's been said that Frank-Wolfe converges, but just barely.

To avoid confusion, we refer to it by manifold suboptimization, a convention also followed by Bertsekas (2016).

Recall from equation (6.33) that the gradient of the Beckmann function is

$$\nabla_{\mathbf{h}} f = \text{vect}(c^\pi). \quad (6.37)$$

Assuming that the current path flow solution \mathbf{h} is feasible, we must move in a direction that does not violate any of the constraints, only using the working paths $\hat{\Pi}^{rs}$. For instance, if the demand constraint $\sum_{\pi \in \hat{\Pi}^{rs}} h^\pi = d^{rs}$ is satisfied for all OD pairs (r, s) , it must remain so after taking a step in the direction $\Delta \mathbf{h}$:

$$\sum_{\pi \in \hat{\Pi}^{rs}} (h^\pi + \Delta h^\pi) = d^{rs}. \quad (6.38)$$

This in turn implies

$$\sum_{\pi \in \hat{\Pi}^{rs}} \Delta h^\pi = d^{rs} - \sum_{\pi \in \hat{\Pi}^{rs}} h^\pi \quad (6.39)$$

$$\sum_{\pi \in \hat{\Pi}^{rs}} \Delta h^\pi = 0 \quad (6.40)$$

So, we need to find the projection of the steepest descent direction \mathbf{s} onto the space $\Delta H \equiv \{\Delta \mathbf{h} : \sum_{\pi \in \hat{\Pi}^{rs}} \Delta h^\pi = 0\}$. It turns out that this projection has a remarkably simple closed form expression.

Proposition 6.1. *Using $\bar{c}^{rs} = (1/|\hat{\Pi}^{rs}|) \sum_{\pi \in \hat{\Pi}^{rs}} c^\pi$ to denote the average travel time of the working paths for OD pair (r, s) , the direction \mathbf{s}' with components $s'_\pi = c^\pi - \bar{c}^{rs}$ is the projection of the steepest descent direction \mathbf{s} onto the set ΔH .*

Proof. To show that \mathbf{s}' is the projection of \mathbf{s} onto ΔH , we must show that $\mathbf{s}' \in \Delta H$, and that $\mathbf{s} - \mathbf{s}'$ is orthogonal to ΔH . Regarding the first part, we have

$$\sum_{\pi \in \hat{\Pi}^{rs}} s'_\pi = \sum_{\pi \in \hat{\Pi}^{rs}} (c^\pi - \bar{c}^{rs}) = \sum_{\pi \in \hat{\Pi}^{rs}} c^\pi - |\hat{\Pi}^{rs}| \bar{c}^{rs} = 0, \quad (6.41)$$

so $\mathbf{s}' \in \Delta H$.

Regarding the second part, let $\Delta \mathbf{h}$ be any vector in ΔH . We now show that $(\mathbf{s} - \mathbf{s}') \cdot \Delta \mathbf{h} = 0$. We have

$$(\mathbf{s} - \mathbf{s}') \cdot \Delta \mathbf{h} = \sum_{\pi \in \hat{\Pi}^{rs}} [c^\pi - (c^\pi - \bar{c}^{rs})] \Delta h^\pi \quad (6.42)$$

$$= -\bar{c}^{rs} \sum_{\pi \in \hat{\Pi}^{rs}} \Delta h^\pi \quad (6.43)$$

$$= 0 \quad (6.44)$$

since $\Delta \mathbf{h} \in \Delta H$. \square

So, given current path flows \mathbf{h}^{rs} for OD pair (r, s) , we use $\Delta\mathbf{h}^{rs} = \text{vect}(c^\pi - \bar{c}^{rs})$ as the search direction. To update the path flows $\mathbf{h}^{rs} \leftarrow \mathbf{h}^{rs} + \mu\Delta\mathbf{h}^{rs}$, we need an expression for the step size. For any path $\pi \in \hat{\Pi}^{rs}$ for which $\Delta h^\pi < 0$, the new path flows would be infeasible if $\mu > h^\pi/\Delta h^\pi$. Therefore, the largest possible step size is

$$\bar{\mu} = \min_{\pi \in \hat{\Pi}^{rs}: \Delta h^\pi < 0} \frac{h^\pi}{\Delta h^\pi}. \quad (6.45)$$

The actual step size $\mu \in [0, \bar{\mu}]$ should be chosen to minimize the Beckmann function. This can be done either through bisection or one or more iterations of Newton's method.

6.4 Bush-Based Algorithms

As seen in the previous section, path-based algorithms converge faster than link-based algorithms, and allow us find much more accurate equilibrium solutions in a fraction of the time. The prime disadvantage is a huge memory requirement, with potentially millions of paths available for use in large networks. Furthermore, many of these paths are “redundant” in some way, as they overlap: the same sequence of links might be used by many different paths.

Bush-based algorithms (also known as origin-based algorithms) try to address these limitations. Rather than treating each OD pair separately (as link-based algorithms do), bush-based algorithms simultaneously consider every destination associated with a single origin. Instead of considering every possible used path (as path-based algorithms do, and there are potentially very many of these), they maintain a set of links called a *bush*, which are the only links which travelers from that origin are permitted to use. One can think of a bush as the set of links obtained by superimposing all of the paths used by travelers starting from an origin. In particular, a bush must be:

- *Connected*; that is, using only links in the bush, it is possible to reach every node which was reachable in the original network.
- *Acyclic*; that is, no path using only bush links can pass the same node more than once. This is not restrictive, because travelers trying to minimize their own travel time would never cycle back to the same node, and greatly speeds up the algorithm, because acyclic networks are much simpler and admit much faster methods for finding shortest paths and other quantities of interest.

There is no particular reason why bushes have to be “origin-based” rather than “destination-based,” and all of the results in this section can be derived in a parallel way for bushes terminating at a common destination, rather than starting at a common origin.

An example of a bush is shown in panels (a) and (b) of Figure 6.9, where the thickly-shaded links are part of the bush, and the lightly-shaded links are not. The bush in panel (a) is a special type of bush known as a *tree*, which

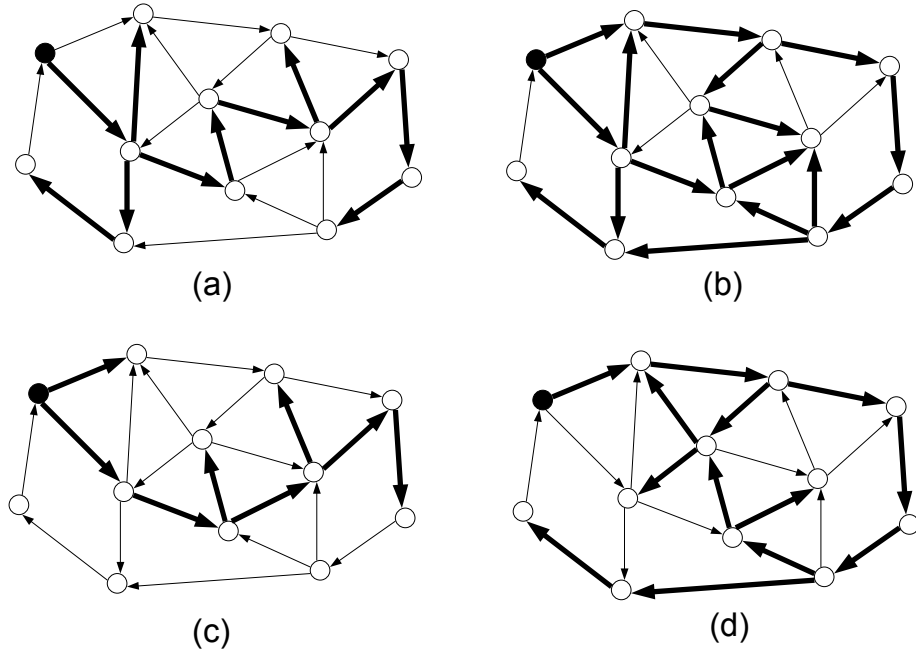


Figure 6.9: Examples of bushes (panels (a) and (b)) and non-bushes (panels (c) and (d)).

has exactly one path from the origin to every node. The thickly-shaded links in panel (c) do not form a bush, because it is not connected; there is no way to reach the nodes at the bottom of the network only using bush links. Likewise, the thick links in panel (d) do not form a bush either, because a cycle exists and it would be possible to revisit some nodes multiple times using the bush links (find them!).

Notice that because a bush is acyclic, it can never include both directions of a two-way link. This implies that at equilibrium, *on every street travelers from the same origin must all be traveling in the same direction*. (This follows from Proposition 5.3 in Section 5.2.3). Interestingly, link-based and path-based algorithms cannot enforce this requirement easily; and this is yet another reason that bush-based algorithms are a good option for solving equilibrium. There is one bush for every origin; this means that if there are z origins and m links, we need to keep track of at most zm values. By contrast, a link-based approach (such as Frank-Wolfe) requires storage of only m values to represent a solution, while a path-based approach could conceivably require $z^2 2^m$ values.⁷

The first well-known origin-based algorithm was developed by Hillel Bar-Gera in his dissertation (circa 2000), and was simply called origin-based assignment (OBA). Bob Dial developed another, simpler method that was published

⁷These values are very approximate, but give you an idea of the scale.

in 2006 as “Algorithm B,” which also seems to work faster than OBA. Yu (Marco) Nie compared both algorithms and developed additional variations by combining features of both, and contributing some ideas of his own. Most recently, Guido Gentile has developed the LUCE algorithm, and Hillel Bar-Gera has provided a new algorithm called TAPAS which simultaneously solves for equilibrium and proportional link flows (approximating entropy maximization). This section focuses on Algorithm B, since it is simpler to explain on its own.

All bush-based algorithms operate according to the same general scheme:

1. Start with initial bushes for each origin (the shortest path tree with free-flow times is often used as a starting point).
2. Shift flows within each bush to bring each origin closer to equilibrium.
3. Improve the bushes by adding links which can reduce travel times, and by removing unused links. Return to step 2.

Step 1 is fairly self-explanatory: collecting the shortest paths from an origin to every destination into one bush results in a connected and acyclic set of links, and we can simply load the travel demand to each destination on the unique path in the bush.

Step 2 is where most bush-based algorithms differ.⁸ This section explains three different methods for shifting flows within a bush: Algorithm B, origin-based assignment (OBA), and linear user cost equilibrium (LUCE). Each of these algorithms is based on labels calculated for bush nodes and links, and Section 6.4.1 defines each of these. The three algorithms themselves are presented in Section 6.4.2.

Step 3 requires a little bit of thought to determine how to adjust the bush links themselves to allow movement towards equilibrium when Step 2 is performed on the new bushes. Section 6.4.3 shows how this can be done.

6.4.1 Bush labels

Algorithm B, OBA, and LUCE all make use of “bush labels” associated with each node and link in a bush. All of these labels can be calculated in a straightforward, efficient manner using the topological order within each bush. As you read this section, pay attention to which of these labels are calculated in ascending topological order (i.e., the formulas only involve nodes with lower topological order) and which are calculated in descending order (based only on nodes with higher topological order.) In the discussion below, assume that a bush is given and fixed, with origin node r . The set of bush links is denoted \mathcal{B} and, for the purposes of this section, assume that the travel times t_{ij} and travel time derivatives t'_{ij} of all bush links are given and constant. All of these labels are only defined for bush links, and the formulas only involve bush links. From the standpoint of Step 2 of bush-based algorithms, non-bush links are completely

⁸If this is starting to sound familiar, good! How to shift flows to move closer to equilibrium is also where link-based and path-based algorithms differ most.

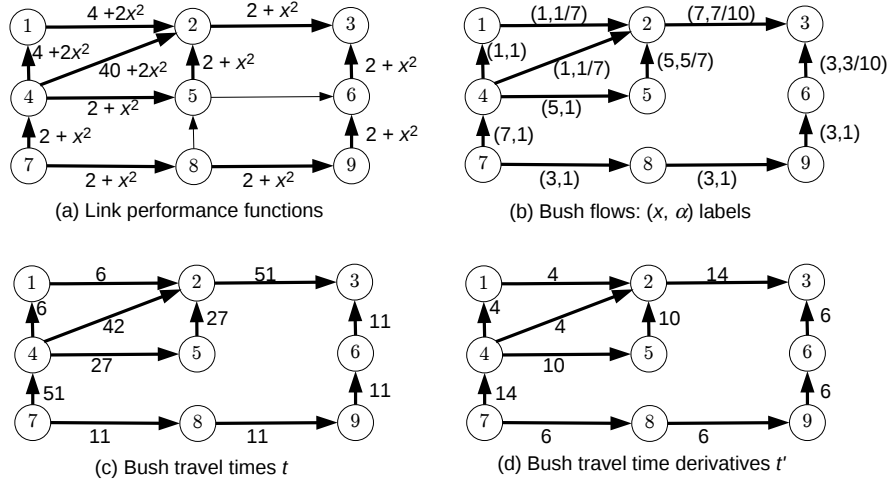


Figure 6.10: Example bush used to demonstrate label calculation.

irrelevant. Table 6.1 shows all of the labels defined in this section, and which labels are used in which algorithms, and in the bush-updating steps described in Section 6.4.3.

We start with two different ways to represent the travel patterns on each bush, starting with x labels. The label $x_{ij}^{\mathcal{B}}$ associated with each link indicates the number of travelers starting at node r (the root of bush \mathcal{B}) and traveling on bush link (i, j) . The superscript \mathcal{B} indicates that we are only referring to the flow on this link associated with the bush \mathcal{B} , and the *total* link flow x_{ij} is the sum of $x_{ij}^{\mathcal{B}}$ across all bushes \mathcal{B} . However, using these superscripts tends to clutter formulas, and often times it is clear that we are only referring to flows within the context of a specific bush. In this case, we can simply write x_{ij} with it being understood that this label refers to the flow on a bush link. Within this section, we are only concerned with a single bush and the superscript will be omitted for brevity.

The network in Figure 6.10(a) will be used as to demonstrate the labels introduced in this section. The thick links comprise the bush, and the link performance functions for all links in the network are shown. The origin in this case is node 7, and the demand is 10 vehicles from node 7 to node 3. You can verify that a topological ordering of the nodes on this bush is 7, 8, 9, 6, 4, 1, 5, 2, 3.

The corresponding label x_i associated with each node indicates the total number of vehicles using node i on the bush \mathcal{B} , including flow which is terminating at node i . The flow conservation equations relating x_{ij} and x_i labels are as follows:

$$x_i = \sum_{(h,i) \in \mathcal{B}} x_{hi} = d^{r_i} + \sum_{(i,j) \in \mathcal{B}} x_{ij} \quad \forall i \in N,$$

where the first expression defines the node flow in terms of *incoming* link flows, and the second in terms of *outgoing* link flows. The two definitions are equivalent.

It is sometimes convenient to refer to the *fraction* of the flow at a node coming from a particular link. For any node i with positive node flow ($x_i > 0$), define α_{hi} to be the proportion of the node flow contributed by the incoming link (h, i) , that is

$$\alpha_{hi} = x_{hi}/x_i. \quad (6.46)$$

Clearly each α_{hi} is nonnegative, and, by flow conservation, the sum of the α_{hi} values entering each node i is one. The definition of α_{hi} is slightly trickier when $x_i = 0$, because the formula (6.46) then involves a division by zero. To accommodate this case, we adopt this rule: *when $x_i = 0$, the proportions α_{hi} may take any values whatsoever, as long as they are nonnegative and sum to one.* It is important to be able to define α_{hi} values even in this case, because the flow-shifting algorithms may cause x_i to become positive, and in this case we need to know how to distribute this new flow among the incoming links.

If we are given α labels for each bush link, it is possible to calculate the resulting node and link flows x_i and x_{ij} , using this recursion:

$$x_i = d^r + \sum_{(i,j) \in \mathcal{B}} x_{ij} \quad \forall i \in N \quad (6.47)$$

$$x_{ij} = \alpha_{ij} x_j \quad \forall (i, j) \in \mathcal{B} \quad (6.48)$$

The sum in (6.47) is empty for the node with the highest topological order. So we can start there, and then proceed with the calculations in backward topological order.

Figure 6.10(b) shows the x and α labels for the example bush. You should verify that the formulas (6.47) and (6.48) are consistent with these labels. The link travel times and link travel time derivatives are shown in panels (c) and (d) of this figure.

As has been used earlier in the text for shortest path algorithms, L is used to denote travel times on shortest paths. The superscript \mathcal{B} can be appended to these labels when it is necessary to indicate that these labels are for shortest paths *specifically on the bush \mathcal{B}* , although it will usually be clear from context which bush is meant. This section will omit such a superscript to avoid cluttering the formulas, and it should be understood that L means the shortest path *only on the bush under consideration*. The same convention will apply to the other labels in this section. There are L labels associated with each node i , and with each link $(i, j) \in \mathcal{B}$: L_i denotes the distance on the shortest path from r to i using only bush links, and L_{ij} is the travel time which would result if you follow the shortest path to node i , then take link (i, j) . These labels are calculated

using these equations:

$$L_r = 0 \quad (6.49)$$

$$L_{ij} = L_i + t_{ij} \quad \forall (i, j) \in \mathcal{B} \quad (6.50)$$

$$L_i = \min_{(h,i) \in \mathcal{B}} \{L_{hi}\} \quad \forall i \neq r \quad (6.51)$$

The U labels are used to denote travel times on longest paths within the bush, and are calculated in a similar way. Like the L labels, U labels are calculated for both nodes and bush links, using the formulas:

$$U_r = 0 \quad (6.52)$$

$$U_{ij} = U_i + t_{ij} \quad \forall (i, j) \in \mathcal{B} \quad (6.53)$$

$$U_i = \max_{(h,i) \in \mathcal{B}} \{U_{hi}\} \quad \forall i \neq r \quad (6.54)$$

The M labels are used to denote the *average* travel times within the bush, recognizing that some travelers will be on longer paths and other travelers will be on shorter ones. The node label M_i represents the average travel time between origin r and node i across all bush paths connecting these nodes, weighted by the number of travelers using each of these paths. The label M_{ij} indicates the average travel time of vehicles after they finish traveling on link (i, j) , again averaging across all of the bush paths starting at the origin r and ending with link (i, j) . These can be calculated as follows:

$$M_r = 0 \quad (6.55)$$

$$M_{ij} = M_i + t_{ij} \quad \forall (i, j) \in \mathcal{B} \quad (6.56)$$

$$M_i = \sum_{(h,i) \in \mathcal{B}} \alpha_{hi} M_{hi} \quad \forall i \neq r \quad (6.57)$$

Figure 6.11 shows the L , U , and M labels corresponding to the example bush in panels (a) and (b). Panel (a) shows the labels associated with links (L_{ij} , M_{ij} , and U_{ij}), while panel (b) shows the labels associated with nodes (L_i , M_i , and U_i).

It is also useful to know how the average travel times (the M labels) will change with a marginal increase in flow on a particular link or through a particular node. The set of D labels are used to represent this. As we will see shortly, these play the role of the travel time derivatives⁹, which are needed in flow shifting rules based on Newton's method (like those described for path-based algorithms in the previous section.) Ideally, the D_{ij} labels would represent the derivative of M_j with respect to α_{ij} . However, there is no known method for calculating these derivatives in an efficient way ("efficient" meaning they can be calculated in a single topological pass). The following formulas for D_{ij} can be used as an approximation that can be calculated in a single topological pass.

⁹For a mnemonic, you can associate the labels L , U , and M with the *lower*, *upper*, and *mean* travel times to nodes, while D refers to the *derivative*.

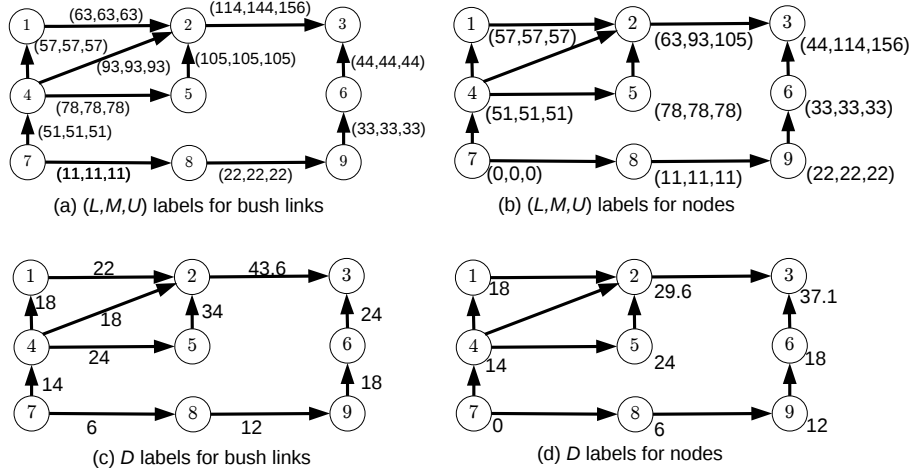


Figure 6.11: Continuation of label calculation demonstration.

Table 6.1: Bush labels used in different algorithms.

Label	B	OBA	LUCE	Updating
x	X			
α		X	X	
L	X			X
U	X			X
M		X	X	
D		X	X	

$$D_r = 0 \tag{6.58}$$

$$D_{ij} = D_i + t'_{ij} \quad \forall (i, j) \in \mathcal{B} \tag{6.59}$$

$$D_i = \sum_{(h,i) \in \mathcal{B}} \alpha_{hi}^2 D_{hi} + \sum_{(h,i) \in \mathcal{B}} \sum_{\substack{(g,i) \in \mathcal{B} \\ (g,i) \neq (h,i)}} \alpha_{hi} \alpha_{gi} \sqrt{D_{hi} D_{gi}} \quad \forall i \neq r \tag{6.60}$$

Figure 6.11 shows the D labels associated with the example bush in panels (c) and (d). Panel (c) shows the D_{ij} labels associated with links, and panel (d) shows the D_i labels associated with nodes.

6.4.2 Shifting flows on a bush

With the labels defined in the previous subsection, we are now ready to state the Algorithm B, OBA, and LUCE flow shifting procedures. Any of these can be used for Step 2 of the generic bush-based algorithm presented at the start

of the section. You will notice that all of these procedures follow the same general form: calculate bush labels (different labels for different algorithms) in *forward* topological order, then scan each node in turn in *reverse* topological order. When scanning a node, use the labels to identify vehicles entering the node from higher-cost approaches, and shift them to paths using lower-cost approaches. Update the x and/or α labels accordingly, then proceed to the previous node topologically until the origin has been reached.

All three of these algorithms also make use of *divergence nodes* (also called *last common nodes* or *pseudo-origins* in the literature) as a way to limit the scope of these updates. While the definition of divergence nodes is slightly different in these algorithms, the key idea is to find the “closest” node which is common to all of the paths travelers are being shifted among. The rest of this subsection details these definitions and the role they play.

Algorithm B

Algorithm B identifies the longest and shortest paths to reach a node, and shifts flows between them to equalize their travel times. That is, when scanning a node i , only two paths (the longest and shortest) are considered. It is easy to determine these paths using the L and U labels, tracing back the shortest and longest paths by identifying the links used for the minimum or maximum in equations (6.51) and (6.54). Once these paths are identified, the divergence node a is the last node common to both of these paths.

The shortest and longest path segments between nodes a and i form a *pair of alternate segments*; let σ_L and σ_U denote these path segments. Within Algorithm B, flow is shifted from the longest path to the shortest path, using Newton’s method to determine the amount of flow to shift:

$$\Delta h = \frac{(U_i - U_a) - (L_i - L_a)}{\sum_{(g,h) \in \sigma_L \cup \sigma_U} t'_{gh}} \quad (6.61)$$

There is also the constraint $\Delta h \leq \min_{(i,j) \in \sigma_U} x_{ij}$ which must be imposed to ensure that all links retain nonnegative flow after the shift. This flow is subtracted from each of the links in the longest path segment σ_U , and added to each of the links in the shortest path segment σ_L . The derivation of this formula parallels that of gradient projection in Section 6.3.1. By shifting flow from the longer path to the shorter path, we will either (i) equalize the travel times on the two paths or (ii) shift all the flow onto the shorter path, and have it still be faster than the longer path. In either case, we move closer to satisfying the equilibrium condition.

The steps of Algorithm B are as follows:

1. Calculate the L and U labels in forward topological order.
2. Let i be the topologically last node in the bush.
3. Scan i by performing the following steps:

- (a) Use the L and U labels to determine the divergence node a and the pair of alternate segments σ_L and σ_U .
 - (b) Calculate Δh using equation (6.61) (capping Δh at $\min_{(i,j) \in \sigma_U} x_{ij}$ if needed).
 - (c) Subtract Δh from the x label on each link in σ_U , and add Δh to the x label on each link in σ_L .
4. If $i = r$, go to the next step. Otherwise, let i be the previous node topologically and return to step 3.
 5. Update all travel times t_{ij} and derivatives t'_{ij} using the new flows x (remembering to add flows from other bushes.)

Demonstrating on the example in Figures 6.10 and 6.11, we start with the L and U labels as shown in Figure 6.11(a) and (b), and start by letting $i = 3$, the last node topologically. The longest path in the bush from the origin ($r = 7$) to node $i = 3$ is $[7,4,5,2,3]$, and the shortest path is $[7,8,9,6,3]$, as can be easily found from the L and U labels. The divergence node is the last node common to both of these paths, which in this case is the origin, so $a = 7$. Equation (6.61) gives $\Delta h = 1.56$, so we shift this many vehicles away from the longest path and onto the shortest path, giving the flows in Figure 6.12(a).

The second-to-last node topologically is node 2, and we repeat this process. The longest and shortest paths from the origin to node 2 in the bush are $[7,4,5,2]$ and $[7,4,1,2]$, respectively.¹⁰ The last node common to both of these paths is 4, so the divergence node is $a = 4$ and we shift flow between the pair of alternate segments $[4,5,2]$ and $[4,1,2]$. Using equation (6.61) gives $\Delta h = 1.50$. Shifting this many vehicles from the longer segment to the shorter one gives the flows in Figure 6.12(b).

The previous node topologically is node 5. Since there is only one incoming bush link to node 5, there is nothing for Algorithm B to do. To see why, notice that the longest and shortest bush paths are $[7,4,5]$ and $[7,4,5]$. The divergence node would be $a = 5$, which is the same as i , and the “pair of alternate segments” is the empty paths $[5]$ and $[5]$. Intuitively, since there is only one way to approach node 5, there are no “alternate routes” to divert incoming flow. In fact, the same is true for all of the previous nodes topologically (1, 4, 6, 9, 8, 7 in the reverse of the order given above.), so there are no more flow shifts on the bush.

Origin-based assignment (OBA)

Rather than considering just two paths at a time, as Algorithm B did, OBA shifts flow among many paths simultaneously. This can be both a blessing (in that its moves affect more paths simultaneously) and a curse (in that its moves are not as sharp as Algorithm B, which can focus on the two paths with the

¹⁰Notice that we have not yet updated the travel time labels based on the shift at node 7. You may do so if you wish, but it is not required for Algorithm B to work, and in all of the examples in this section travel times are not updated until all flow shifts are complete for the bush.

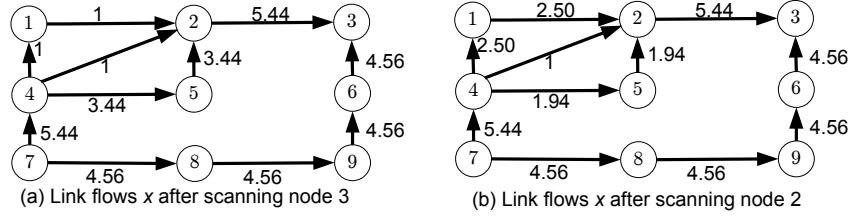


Figure 6.12: Algorithm B flow shifts.

greatest travel time difference). Because OBA shifts flow among many paths, it makes use of the average cost labels M and the derivative labels D , rather than the shortest and longest path costs (and the direct link travel times derivatives) used by Algorithm B. OBA also makes use of a Newton-type shift, dividing a difference in travel times by an approximation of this difference's derivative.

When scanning a node i in OBA, first identify a least-travel time approach, that is, a link (\hat{h}, i) such that $M_{\hat{h}i} \leq M_{hi}$ for all other approaches (h, i) . This link is called the *basic approach* in analogy to the basic path concept used in path-based algorithms. Then, for each nonbasic approach (h, i) , the following amount of flow is shifted from x_{hi} to $x_{\hat{h}i}$:

$$\Delta x_{hi} = \frac{M_{hi} - M_{\hat{h}i}}{D_{hi} + D_{\hat{h}i} - 2D_a}, \quad (6.62)$$

with the constraint that $\Delta x_{hi} \leq x_{hi}$ to prevent negative flows. It can be shown that this formula would equalize the mean travel times on the two approaches if they were linear functions. In reality, they are not, but the formula is still used as an approximation.

Here a is the divergence node, defined for OBA as the node with the highest topological order which is common to *all* paths in the bush from r to i , excluding i itself. This is a reasonable place to truncate the search, since shifting flow among path segments between a and i will not affect the flows on any earlier links in the bush. This is a generalization of the definition used for Algorithm B, needed since there are more than two paths subject to the flow shift.

After applying the shift Δx_{hi} to each of the links entering node i , we have to update flows on the other links between a and i to maintain flow conservation. This is done by assuming that the α values stay the same elsewhere, meaning that any increase or decrease in the flow passing through a node propagates backward to its incoming links in proportion to the contribution each incoming link provides to that total flow. Thus, the x labels can be recalculated using equations (6.47) and (6.48) to links and nodes topologically between a and i .

The steps of OBA are as follows:

1. Calculate the M and D labels in forward topological order.
2. Let i be the topologically last node in the bush.

3. Scan i by performing the following steps:
 - (a) Determine the divergence node a corresponding to node i , and the basic approach (\hat{h}, i) .
 - (b) For each nonbasic approach (h, i) calculate Δx_{hi} using equation (6.62) (with $\Delta x_{hi} \leq x_{hi}$), subtract Δx_{hi} from x_{hi} and add it to $x_{\hat{h}i}$.
 - (c) Update α_{hi} for every bush link terminating at i .
 - (d) Apply equations (6.47) and (6.48) to all nodes and links topologically between a and i , updating their x_{ij} and x_i values.
4. If $i = r$, go to the next step. Otherwise, let i be the previous node topologically and return to step 3.
5. Update all travel times t_{ij} and derivatives t'_{ij} using the new flows x (remembering to add flows from other bushes.)

Again demonstrating on the example in Figure 6.10, we use the M and D labels shown in Figure 6.11. As with Algorithm B, we scan nodes in reverse topological order. Starting with node 3, we compare the labels M_{23} and M_{63} to determine the basic approach. Since M_{63} is lower, this is the least-cost approach and named basic. For OBA, the divergence node corresponding to node 3 is node 7, since it is the only node common to all paths between nodes 7 and 3, except for node 3 itself. Formula (6.62) is then applied to determine the amount of flow Δx_{23} that should be shifted from approach (2,3) to the basic approach (6,3), producing $\Delta x_{23} = 1.48$. Shifting this flow updates the α values as shown in Figure 6.13(b). To maintain flow conservation, we also need to adjust the flow on other links in the bush. Holding the α proportions fixed at all other links, these changes at node 3 are propagated back proportional to the original flows, producing the link flows in Figure 6.13(a).

Next scanning node 2, we examine M_{12} , M_{42} , and M_{52} to determine the basic approach: since M_{12} is smallest, it is deemed the basic approach. The divergence node a corresponding to node 2 is node 4, since all paths from 7 to 2 pass through node 4 (and no other node with higher topological order). We now apply two shifts, calculating Δx_{42} and Δx_{52} with equation (6.62). This equation gives $\Delta x_{42} = 2.5$, but since $x_{42} = 0.79$ it is only possible to move 0.79 units of flow from (4,2) to (1,2). For approach (5,2), $\Delta x_{52} = 1.50$, so we move 1.50 units of flow from (5,2) to (1,2). Assuming that the α values at all other nodes remains constant, propagating this change back produces the x and α labels in Figure 6.13(c) and (d). Notice that we only had to update flows between the divergence node ($a = 4$) and the node being scanned ($i = 2$), since no other links would be affected by shifting flow among approaches to node 2.

As with Algorithm B, no changes are made for the remaining nodes scanned, since they only have one incoming link. This incoming link is trivially the “basic” approach, and there are no nonbasic approaches to shift flow from.

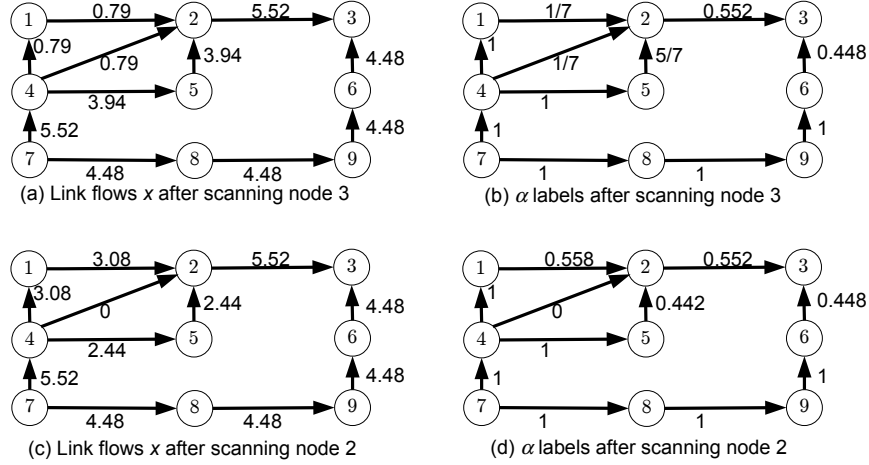


Figure 6.13: OBA flow shifts.

Linear user cost equilibrium (LUCE)

The LUCE algorithm is similar in structure to OBA, using the same definition of a divergence node and the same sets of labels (M and D). Where it differs is in how nodes are scanned. When LUCE scans a node, it attempts to solve a “local” user equilibrium problem, based on linear approximations to the approach travel times. To be concrete, when node i is scanned, the labels M_{hi} and D_{hi} respectively indicate the average travel time for travelers arriving at i via (h, i) , and the derivative of this average travel time as additional vehicles arrive via this link. Both of these are calculated based on the current bush flows, so if the flow on approach (h, i) changes by Δx_{hi} , then the new average travel time M'_{hi} can be approximated by

$$M'_{hi} \approx M_{hi} + \Delta x_{hi} D_{hi}. \quad (6.63)$$

The LUCE algorithm chooses Δx_{hi} values for each approach according to the following principles:

- Flow conservation must be obeyed, that is, $\sum_{(h,i) \in \mathcal{B}} \Delta x_{hi} = 0$.
- No link flow can be made negative, that is, $x_{hi} + \Delta x_{hi} \geq 0$ for all $(h, i) \in \mathcal{B}$.
- The M'_{hi} values should be equal and minimal for any approach with positive flow after the shift, that is, if $x_{hi} + \Delta x_{hi} > 0$, then M'_{hi} must be less than or equal to the M' label for any other approach to i .

The Δx_{hi} values satisfying this principles can be found using a “trial and error” algorithm, like that introduced in Section 4.2.1. Choose a set of approaches (call

it \mathcal{A}), and set $\Delta x_{hi} = -x_{hi}$ for all (h, i) not in this set \mathcal{A} . For the remaining approaches, solve the linear system of equations which set M'_{hi} equal for all $(h, i) \in \mathcal{A}$ and which have $\sum_{(h,i) \in \mathcal{A}} \Delta x_{hi} = 0$. The number of equations will equal the number of approaches in \mathcal{A} . After obtaining such a solution, you can verify whether the three principles are satisfied. Principle (a) will always be satisfied. If principle (b) is violated, approaches with negative $x_{hi} + \Delta x_{hi}$ should be removed from \mathcal{A} . If principle (c) is violated, then some approach not in \mathcal{A} has a lower M' value, and that approach should be added to \mathcal{A} . In either of the latter two cases, the entire process should be repeated with the new \mathcal{A} set.

The steps of LUCE are as follows:

1. Calculate the M and D labels in forward topological order.
2. Let i be the topologically last node in the bush.
3. Scan i by performing the following steps:
 - (a) Determine the divergence node a corresponding to node i , and the basic approach (\hat{h}, i) .
 - (b) Use the process described above to find Δx_{hi} values satisfying the local equilibrium principles (a)–(c) above, and add Δx_{hi} to x_{hi} for each approach to i .
 - (c) Update α_{hi} for every bush link terminating at i .
 - (d) Use equations (6.47) and (6.48) to all nodes and links topologically between a and i , updating their x_{ij} and x_i values.
4. If $i = r$, go to the next step. Otherwise, let i be the previous node topologically and return to step 3.
5. Update all travel times t_{ij} and derivatives t'_{ij} using the new flows x (remembering to add flows from other bushes.)

Applying LUCE to the same example, we again start by scanning node 2. Assuming that both approaches (2,3) and (6,3) will continue to be used, we solve the following equations simultaneously for Δx_{23} and Δx_{63} :

$$\begin{aligned} M_{23} + D_{23}\Delta x_{23} &= M_{63} + D_{63}\Delta x_{63} \\ \Delta x_{23} + \Delta x_{63} &= 0 \end{aligned}$$

Substituting the M and D labels from Figure 6.11 and solving, we obtain $\Delta x_{23} = -1.48$ and $\Delta x_{63} = +1.48$. Updating flows as in OBA (using the divergence node $a = 7$ and assuming all α values at nodes other than 3 are fixed) gives the x and α labels shown in Figure 6.14(a) and (b). Notice that this step is exactly the same as the first step taken by OBA. The interpretation of LUCE and solving a local, linearized equilibrium problem provides insight into how the OBA flow shift formula (6.62) was derived.

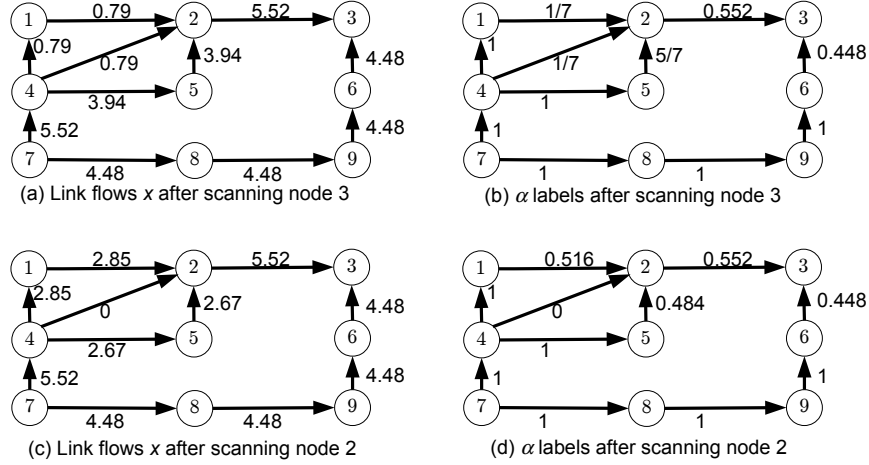


Figure 6.14: LUCE flow shifts.

The shift is slightly different when there are three approaches to a node, as happens when we proceed to scan node 2. First assuming that all three approaches will be used, we solve these three equations simultaneously, enforcing flow conservation and that the travel times on the three approaches should be the same:

$$\begin{aligned} M_{12} + D_{12}\Delta x_{12} &= M_{42} + D_{42}\Delta x_{42} \\ M_{52} + D_{52}\Delta x_{52} &= M_{42} + D_{42}\Delta x_{42} \\ \Delta x_{12} + \Delta x_{42} + \Delta x_{52} &= 0 \end{aligned}$$

Substituting values from Figure 6.11 and solving these equations simultaneously gives $\Delta x_{12} = 2.82$, $\Delta x_{42} = -1.85$, and $\Delta x_{52} = -0.97$. This is problematic, since $x_{42} = 0.79$ and it is impossible to reduce its flow further by 1.85. This means that approach (4,2) should not be used, so we fix $\Delta x_{42} = -0.79$ as a constant and re-solve the system of equations for Δx_{12} and Δx_{52} :

$$\begin{aligned} M_{12} + D_{12}\Delta x_{12} &= M_{42} - 0.79D_{42} \\ M_{52} + D_{52}\Delta x_{52} &= M_{42} - 0.79D_{42} \\ \Delta x_{12} - 0.79 + \Delta x_{52} &= 0 \end{aligned}$$

This produces $\Delta x_{12} = 2.06$, and $\Delta x_{52} = -1.27$, alongside the fixed value $\Delta x_{42} = -0.79$. Updating flows on other links as in OBA produces the x and α labels shown in Figure 6.14(b) and (c). No other shifts occur at lower topologically-ordered nodes, because there is only one incoming link, and flow conservation demands that no flow increase or decrease take place.

6.4.3 Improving bushes

After shifting flow for the current bushes as described in the previous section, the next step is to determine whether the bushes themselves need to change (adding or dropping links) to allow us to move closer to equilibrium on the entire network.

To begin, any link which is unused can be dropped from a bush without disturbing the solution (unless it is needed for connectivity); and it can always be added back later if we need to. As far as which links to add, one approach is to add any “shortcut” links, defined based on the bush travel times. If we solve each bush precisely to equilibrium in Step 2, then the distance from the origin r to each node i is the same on any used path; call this L_i . A “shortcut” link (i, j) is any link for which $L_i + t_{ij} < L_j$, where the L labels have been re-calculated after eliminating the zero-flow bush links. Such a link can justifiably be called a shortcut, because it provides a faster path to node j than currently exists in the bush.

If this is our rule for adding links to the bush, it is easy to see that no cycles are created. Reasoning by contradiction, assume that a cycle is created, say, $[i_1, i_2, \dots, i_k, i_1]$. If link travel times are strictly positive, it is easy to see that $L_{i_{j+1}} > L_{i_j}$ for any successive pair of nodes in the cycle (either (i_j, i_{j+1}) was in the previous bush, in which case $L_{i_j} + t_{i_j i_{j+1}} = L_{i_{j+1}}$ by equilibrium; or (i_j, i_{j+1}) was just added, in which case $L_{i_j} + t_{i_j i_{j+1}} < L_{i_{j+1}}$). Applying this identity cyclically, we have $L_{i_1} < L_{i_2} < \dots < L_{i_k} < L_{i_1}$, a contradiction since we cannot have $L_{i_1} < L_{i_1}$.

The requirement that we solve each bush exactly to equilibrium in step 2 is vital to ensuring no cycles are created. If this is not the case (and in practice, we can only solve equilibrium approximately), another criterion is needed. To see why, the argument used to show that no cycles would be created relied critically on the assumption the difference in minimum cost labels for adjacent nodes in the cycle was exactly the travel time of the connecting link (except possibly for a link just added), which is only possible if all used paths have equal and minimal travel time. Another one, which is almost as easy to implement, is to add links to the bush based on the *maximum* travel time to a node using bush links, as opposed to the *minimum* travel time. Although calculating longest paths in general networks is difficult, in acyclic networks it can be done just as efficiently as finding shortest paths (Section 2.4.1).

Assume that we have re-calculated the longest path labels U_i to each bush node, after eliminating unused bush links as described above. We now define a “modified shortcut” link as any link for which $U_i + t_{ij} < U_j$ (this is the same definition as a shortcut link but with maximum costs labels used instead of minimum cost labels). Using a similar argument, we can show that adding modified shortcut links cannot create cycles, even if the bush is not at equilibrium. Arguing again by contradiction, assume that the cycle $[i_1, i_2, \dots, i_k, i_1]$ is created by adding modified shortcut links, and consider in turn each adjacent pair of labels L_{i_j} and $L_{i_{j+1}}$. By the definition of maximum cost, for any link which was in the bush before we have $U_{i_j} + t_{i_j i_{j+1}} = U_{i_{j+1}}$. For any link just

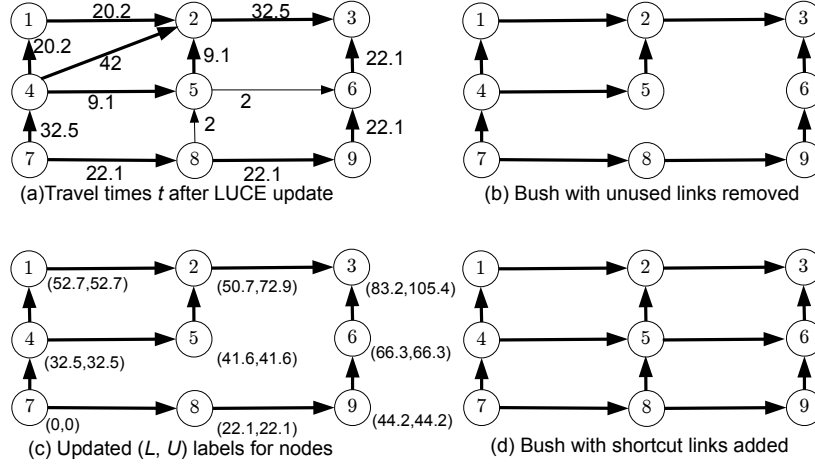


Figure 6.15: Updating a bush by removing unused links and adding shortcuts.

added, we have $U_{i_j} + t_{i_j i_{j+1}} < U_{i_{j+1}}$). Applying this identity cyclically, we have $U_{i_1} \leq U_{i_2} \leq \dots \leq U_{i_k} \leq U_{i_1}$. Furthermore, since there were no cycles in the bush during the previous iteration, at least one of these links must be new, and for this link the inequality must be strict.

As an example, consider the bush updates which occur after performing the LUCE example in the previous section. Figure 6.15 shows the updated link travel times in panel (a), and the remaining bush links after zero-flow links are removed in panel (b). Re-calculating L and U labels with the new travel times and bush topology gives the values in Figure 6.15(c). At this point the three unused bush links (4,2), (5,6), and (8,5), are examined to determine whether $U_i + t_{ij} < U_j$ for any of them. This is true for (5,6) and (8,5), since $41.6 + 2 < 66.3$ and $22.1 + 2 < 41.6$, but false for (4,2) since $32.5 + 42 \geq 72.9$. So, (5,6) and (8,5) are added to the bush, as shown in Figure 6.15(d). From here, one can return to the flow shifting algorithm to update flows further.

6.5 Likely Path Flow Algorithms

Recall from Section 5.2.2 that there is generally an infinite number of path flow vectors \mathbf{h} which satisfy the principle of user equilibrium. This contrasts with the fact that the equilibrium link flow vector $\hat{\mathbf{x}}$ is unique as long as link performance functions are increasing. That section introduced the principle of entropy maximization as a way to select a equilibrium path flow believed to be the most likely to occur in practice. The entropy-maximizing link flows solve

the optimization problem

$$\max_{\mathbf{h}} \quad - \sum_{(r,s) \in Z^2} \sum_{\pi \in \hat{\Pi}^{rs}} h_{\pi} \log(h_{\pi}/d^{rs}) \quad (6.64)$$

$$\text{s.t.} \quad \sum_{\pi \in \Pi} \delta_{ij}^{\pi} h_{\pi} = \hat{x}_{ij} \quad \forall (i,j) \in A \quad (6.65)$$

$$\sum_{\pi \in \hat{\Pi}^{rs}} h_{\pi} = d^{rs} \quad \forall (r,s) \in Z^2 \quad (6.66)$$

$$h_{\pi} \geq 0 \quad \forall \pi \in \Pi \quad (6.67)$$

That section also introduced the proportionality condition, showing in Theorem 5.4 that the ratio of the flows on any two paths connecting the same OD pair depends *only* on the pairs of alternate segments where the paths differ, not what that OD pair happens to be, or on any links where the paths coincide. The proportionality condition is easier to verify, and entropy-maximizing solutions must satisfy proportionality. Proportionality does not imply entropy maximization, but in practical terms they seem nearly equivalent. Therefore, this section focuses on finding proportional solutions, hence the term “likely path flow algorithms” rather than “*most* likely.”

The proof that entropy maximization implies proportionality was based on the Lagrangian of the entropy-maximization problem:

$$\begin{aligned} \mathcal{L}(\mathbf{h}, \boldsymbol{\beta}, \boldsymbol{\gamma}) = & - \sum_{(r,s) \in Z^2} \sum_{\pi \in \hat{\Pi}^{rs}} h_{\pi} \log\left(\frac{h_{\pi}}{d^{rs}}\right) + \sum_{(i,j) \in A} \beta_{ij} \left(\hat{x}_{ij} - \sum_{\pi \in \Pi} \delta_{ij}^{\pi} h_{\pi} \right) \\ & + \sum_{(r,s) \in Z^2} \gamma_{rs} \left(d^{rs} - \sum_{\pi \in \hat{\Pi}^{rs}} h_{\pi} \right). \end{aligned} \quad (6.68)$$

After some algebraic manipulation, we obtained the formula

$$h^{\pi} = K_{rs} \exp\left(- \sum_{(i,j) \in A} \delta_{ij}^{\pi} \beta_{ij}\right), \quad (6.69)$$

where K_{rs} is a proportionality constant associated with the OD pair (r, s) corresponding to path π . The value of K_{rs} can be found from the constraint that total path flows must equal demand, $\sum_{\pi \in \hat{\Pi}^{rs}} h^{\pi} = d^{rs}$. The rest of the equation shows that the entropy-maximizing path flows are determined solely by the Lagrange multipliers β_{ij} associated with each link, and therefore that the ratio between two path flows for the same OD pair only depends on the links where they differ.

Here we describe three ways to calculate likely path flows. The first method is “primal,” and operates directly on the path flow vector \mathbf{h} itself. The second method is “dual,” operating on the Lagrange multipliers $\boldsymbol{\beta}$ in the entropy maximization problem, which can then be used to determine the path flows. Both

of these methods presume that an equilibrium link flow solution $\hat{\mathbf{x}}$ has already been found. The third method, “traffic assignment by paired alternative segments” (TAPAS), is an algorithm which simultaneously solves for equilibrium and likely paths.

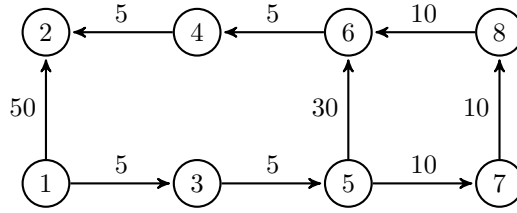
Primal and dual methods for likely path flows have two major issues in common:

1. The sets of equilibrium paths $\hat{\Pi}_{r,s}$ must be determined in some way. If $\hat{\mathbf{x}}$ is exactly a user equilibrium, we can identify the shortest path for each OD pair, and then set $\hat{\Pi}_{r,s}$ to be all the paths with that same cost between r and s . In practice, though, we cannot solve for the equilibrium link flows exactly, but only to a finite precision. So we cannot guarantee that all of the used paths for an OD pair have the same travel time, and we cannot guarantee that the unused paths are actually unused at the true equilibrium solution.
2. The system of constraints (6.65) and (6.66) has many more variables than equations; the number of equilibrium paths is typically much larger than the number of OD pairs and network links. For primal methods, we need to find and specify these “degrees of freedom” so we know how to adjust path flows while keeping the solution feasible (matching equilibrium link flows and the OD matrix). For dual methods, it means that many of the link flow constraints (6.65) are redundant, and their corresponding β_{ij} Lagrange multipliers are not needed. If these redundant constraints and Lagrange multipliers are included, then there are infinitely many β values that correspond to entropy maximization.

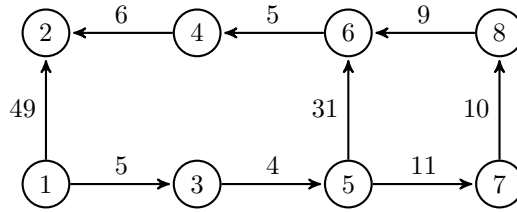
To address the first issue, we can include all paths in $\hat{\Pi}$ that are within some threshold ϵ of the shortest-path travel time for each OD pair. Some care must be taken to ensure that the resulting path sets are “consistent,” in that travelers from different origins and destinations consider the same sets of alternatives, when their path sets overlap. Figure 6.16 illustrates the difficulties in obtaining a consistent solution. The top panel of the figure shows the travel times at the true equilibrium solution, and the bottom panel shows the travel times at an approximate equilibrium solution, such as the one obtained after stopping one of the algorithms in this chapter after a finite number of iterations.

In this figure, assume there are two OD pairs, one from node 1 to node 2, and another from node 3 to node 4. At the true equilibrium solution, there are three equal-cost paths between nodes 1 and 2 — $[1, 2]$, $[1, 3, 5, 6, 4, 2]$, and $[1, 3, 5, 7, 8, 6, 4, 2]$ — and two equal-cost paths between nodes 3 and 4 — $[3, 5, 6, 4]$ and $[3, 5, 7, 8, 6, 4]$. Travelers whose paths cross nodes 5 and 6 consider two possible alternative routes between these nodes ($[3, 5, 6, 4]$ and $[3, 7, 8, 6, 4]$), regardless of which OD pair they came from. So defining $\hat{\Pi}_{12}$ and $\hat{\Pi}_{34}$ to include the equal-cost paths listed above is consistent.

Now consider the bottom panel of the figure. There is now a unique shortest path for each OD pair — $[1, 2]$ and $[3, 5, 6, 4]$ — but this does not reflect the true equilibrium, simply the imprecision in an approximate solution. The threshold



(a) Equilibrium link travel times



(b) Link travel times at an approximate equilibrium solution

Figure 6.16: Link travel times at the true equilibrium solution (top) and at an approximate equilibrium solution (bottom).

ϵ can be set to reflect this. Assume that $\epsilon = 1$, so that any path within 1 minute of the shortest path travel time is included in $\hat{\Pi}$. This choice gives the path sets $\hat{\Pi}_{12} = \{[1, 2], [1, 3, 5, 6, 4, 2]\}$ and $\hat{\Pi}_{34} = \{[3, 5, 6, 4], [3, 5, 7, 8, 6, 4]\}$. This choice is *not* consistent, because it assumes travelers passing between nodes 5 and 6 consider different choices depending on their OD pair: travelers starting at node 1 only consider the segment $[5, 6]$, while travelers starting at node 3 consider both $[5, 6]$ and $[5, 7, 8, 6]$ as options. If travelers are choosing routes to minimize cost, it should not matter what their origin or destination is. Increasing ϵ to a larger value would address this problem, but in a larger network would run the risk of including paths which are not used at the true equilibrium solution.

So some care must be taken in how ϵ is chosen. In the approximate equilibrium solution, we can define the *acceptance gap* g_a to be the greatest difference between a used path's travel time and the shortest path travel time for its OD pair, and the *rejection gap* g_r to be the smallest difference between the travel time of an unused path, and the shortest path for an OD pair. In Figure 6.17, the OD pairs are from 1 to 6 and 4 to 9, and the links are labeled with their travel times. The thick lines show the links used by these OD pairs, and the thin lines show unused links. For travelers between nodes 1 and 6, the used paths have travel times of 21 (shortest) and 22 minutes, and the unused path has a travel time of 26 minutes. For travelers between nodes 4 and 9, the used paths have travel times 18 (which is shortest) and 20 minutes, while the unused path has a travel time of 21 minutes. The acceptance gap g_a for this solution is 2 minutes (difference between 20 and 18), and the rejection gap is 5 (difference between 26 and 21).

It is possible to show that if ϵ is at least equal to the acceptance gap, but less

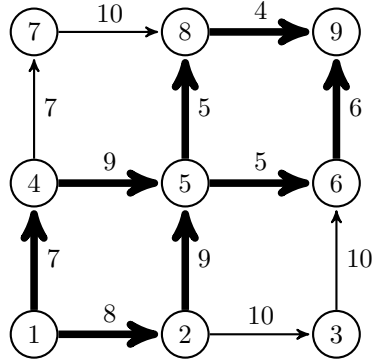


Figure 6.17: Example demonstrating rejection and acceptance gaps. Thick links are parts of used paths at the equilibrium solution. Link labels are travel times.

than half of the rejection gap, then the resulting sets of paths $\hat{\Pi}_{r,s}$ are consistent for the proportionality condition. That is, we need

$$g_a \leq \epsilon \leq \frac{g_r}{2}. \quad (6.70)$$

In the network of Figure 6.17, any ϵ choice between 2 and 2.5 will thus lead to a consistent set of paths.

Exercise 21 gives a more formal definition of consistency and asks you to prove this statement. It is not possible to choose such an ϵ unless the equilibrium problem is solved with enough precision for the acceptance gap to be less than half of the rejection gap. To fully maximize entropy, rather than just satisfying proportionality, demands a more stringent level of precision in the equilibrium solution.

The second issue involves redundancies in the set of equations enforcing the OD matrix and equilibrium link flow constraints. Properly resolving this issue requires using linear algebra to analyze the structure of the set of equations (and this in fact is the key to bridging the gap between proportionality and entropy maximization), but for proportionality a simpler approach is possible.

A redundancy in a system of equations can be interpreted as a “degree of freedom,” a dimension along which a solution can be adjusted. Consider the network in Figure 6.18, which has two OD pairs (A to B, and C to D). The equilibrium link flows are shown in the figure, along with the link IDs and an indexing of the eight paths. The OD matrix and link flow constraints are

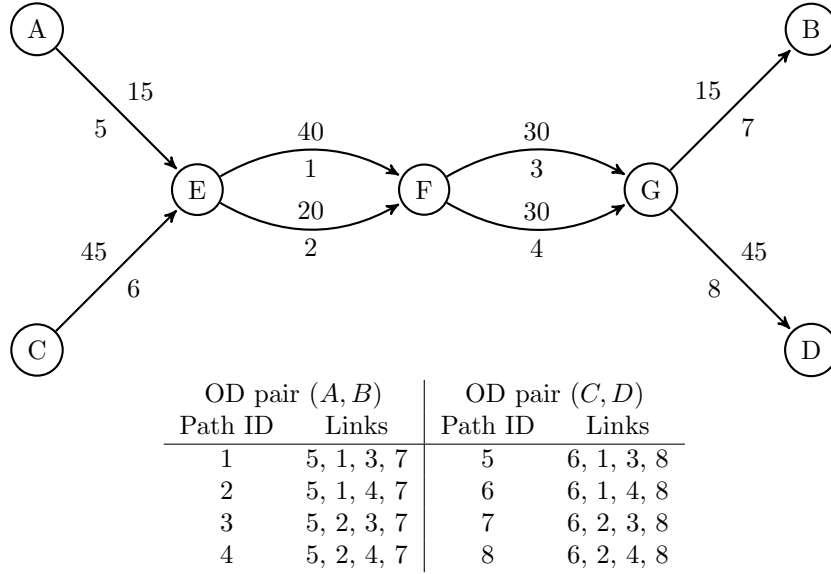


Figure 6.18: Example demonstrating redundancies and “degrees of freedom” when choosing a path flow solution. Links are labeled with equilibrium flows (above) and link IDs (below).

reflected in the six equations

$$h_1 + h_2 + h_3 + h_4 = 15 \tag{6.71}$$

$$h_5 + h_6 + h_7 + h_8 = 45 \tag{6.72}$$

$$h_1 + h_2 + h_5 + h_6 = 40 \tag{6.73}$$

$$h_3 + h_4 + h_7 + h_8 = 20 \tag{6.74}$$

$$h_1 + h_3 + h_5 + h_7 = 30 \tag{6.75}$$

$$h_2 + h_4 + h_6 + h_8 = 30 \tag{6.76}$$

Equations (6.71) and (6.72) reflect the constraints that the total demand among all paths from A to B, and from C to D, must equal the respective values in the OD matrix. Equations (6.73)–(6.76) reflect the constraints that the flow on links 1–4 must match their equilibrium values. Similar equations for links 5–8 are omitted, since they are identical to (6.71) and (6.72), as can easily be verified.

This system has eight variables but only six equations, so there must be at least two independent variables — in fact, there are four, since some of the six equations are redundant. For example, adding equations (6.71) and (6.72) gives you the same result as adding equations (6.73) and (6.74), so one of them — say, (6.74) can be eliminated. Likewise, equation (6.76) can be eliminated, since adding (6.71) and (6.72) is the same as adding (6.75) and (6.76). These choices are not unique, and there are other equivalent ways of expressing the

same redundancies.

Each of these redundancies corresponds to an independent way to adjust the path flows without affecting either total path flows between OD pairs, or total link flows. A primal method uses these redundancies to adjust the path flows directly, increasing the entropy of the solution without sacrificing feasibility of the original path flow solution. A dual method uses these redundancies to eliminate unnecessary link flow constraints — for instance, equations (6.74) and (6.76) in the example above — so that the entropy-maximizing β_{ij} values are unique.

6.5.1 Primal method

To find a proportional solution with a primal method, we need (1) the equilibrium path set $\hat{\Pi}$; (2) the equilibrium link flows $\hat{\mathbf{x}}$; (3) an initial path flow solution \mathbf{h}^0 , and (4) a list of “redundancies” in the link flow constraints, each of which corresponds to a way to change path flows while maintaining feasibility. The algorithm then adjusts the \mathbf{h} values, increasing the entropy at each iteration, until termination.¹¹

The equilibrium path set and link flows were already described above. Depending on the algorithm used to solve for the equilibrium link flows, a path flow solution may already be available. If you solved for equilibrium with a path-based algorithm, its solution already contains the flows on each path. If you used a bush-based algorithm, a corresponding path flow can be found using the “within origins” adjustment technique described below. If you used a link-based algorithm, it is not as easy to directly identify a path flow solution from the final output — but it may be possible to track a path flow solution as the algorithm progresses (for instance, in Frank-Wolfe, each “target” all-or-nothing solution can be clearly identified with a path flow solution, and this can be averaged with previous path flow solutions using the same λ values).

To achieve proportionality, it is not necessary to identify *all* of the redundancies in the link flow constraints (it would be needed to fully maximize entropy). It is enough to identify *pairs of alternate segments* between two nodes which are used by multiple OD pairs. In Figure 6.18, links 1 and 2 are alternate segments between nodes E and F, and links 3 and 4 are alternate segments between nodes F and G.

The primal algorithm alternates between two steps: (1) adjusting \mathbf{h} to achieve proportionality for each origin r ; and (2) for each pair of alternative segments, adjusting \mathbf{h} to achieve proportionality between those nodes. These steps are described below; you can verify that each one of these steps preserves feasibility of the solution, and increases entropy. If the solution remains unchanged after performing both of these steps, then proportionality has been achieved and we terminate. Other stopping criteria can be introduced, by measuring the deviation from proportionality and terminating once this deviation

¹¹There are alternative solution representations that can make this algorithm much faster, but would make the explanations more complicated. You are encouraged to think about how to implement this algorithm efficiently, without having to list all paths explicitly.

is sufficiently small.

Proportionality within origins

Section 5.2.3 described how a user equilibrium solution can be described by the total flow from each origin r on each link (i, j) , denoted x_{ij}^r , and how at equilibrium the links with positive x_{ij}^r values must form an acyclic subnetwork, a bush. It is easy to obtain such a solution if we have a path flow solution \mathbf{h} :

$$x_{ij}^r = \sum_{s \in Z} \sum_{\pi \in \hat{\Pi}_{rs}} \delta_{ij}^{\pi} h^{\pi}. \quad (6.77)$$

With these values, we can calculate the fraction of the flow from origin r approaching any node j from one specific link entering that node (i, j) :

$$\alpha_{ij}^r = x_{ij}^r / \sum_{(h,j) \in \Gamma^{-1}(j)} x_{ij}^r \quad (6.78)$$

with α_{ij}^r defined arbitrarily if the denominator is zero. (This use of α is the same as in the bush-based algorithms described in Section 6.4).

We can ensure that the proportionality condition holds between *all* paths associated with an origin r by updating the path flows according to the formula

$$h^{\pi} \leftarrow d^{rs} \prod_{(i,j) \in \pi} \alpha_{ij}^r \quad \forall s \in Z, \pi \in \hat{\Pi}_{rs}, \quad (6.79)$$

that is, by applying the *aggregate* approach proportions across all paths from this origin to each individual path. One can show updating the path flows with this formula will not change either the total OD flows or link flows, maintaining feasibility, and will also increase entropy if there is any change in \mathbf{h} .

This process is repeated for each origin r .

Proportionality between origins

To achieve proportionality between different origins, we consider pairs of alternate segments between two nodes, used by multiple OD pairs. In Figure 6.18, an example of a pair of alternate segments is links 1 and 2. In a larger network, these alternate segments can contain multiple links. These links connect the same two nodes (E and F), and are parts of equal-cost paths for both OD pairs (A, B) and (C, D) . If we move some flow from link 1 to link 2 from OD pair (A, B) , and move the equivalent amount of flow from link 2 to link 1 from OD pair (C, D) , we will not disturb the total link flows or OD flows, but will change \mathbf{h} and allow us to increase entropy.

Let $\sigma = \{\sigma_1, \sigma_2\}$ be a pair of alternate segments, and let $Z^2(\sigma)$ be the set of OD pairs which have paths using both of the alternate segments in σ . For each

segment σ_i , the segment flow $g(\sigma_i)$ is defined as the sum of flows on all paths using that segment:

$$g(\sigma_i) = \sum_{(r,s) \in Z^2(\sigma)} \sum_{\pi \in \hat{\Pi}_{r,s}; \sigma_i \subseteq \pi} h^\pi, \quad (6.80)$$

where the notation $\sigma_i \subseteq \pi$ means that all of the links in the segment σ_i are in the path π .

If the equilibrium path set $\hat{\Pi}$ is consistent in the sense of (6.70), then any path π which uses one segment of the pair has a “companion” path which is identical, except it uses the other segment of the pair, denoted $\pi^c(\sigma)$. For example, in Figure 6.18, if σ is the pair of alternate segments between nodes E and F, the companion of path 1 is path 3, and the companion of path 6 is path 8.

To achieve proportionality between origins for the alternate segments in σ , we calculate the ratios between $g(\sigma_i)$ values and apply the same ratios to the path flows for each OD pair using this set of alternate segments:

$$h^\pi \leftarrow (h^\pi + h^{\pi^c(\sigma)}) \frac{g(\sigma_i)}{g(\sigma_1) + g(\sigma_2)} \quad \forall (r,s) \in Z^2(\sigma), i \in \{1,2\}, \pi \in \hat{\Pi}_{r,s} \wedge \pi \supseteq \sigma_i. \quad (6.81)$$

It is again possible to show that updating path flows with this formula leaves total OD flows and link flows fixed, and can only increase entropy.

Example

This section shows how the primal algorithm can solve the example in Figure 6.18. Assume that the initial path flow solution is $h_1 = 15$, $h_5 = 15$, $h_6 = 10$, $h_8 = 20$, and all other path flows zero. As shown in the first column of Table 6.2, this solution satisfies the OD matrix and the resulting link flows match the equilibrium link flows, so it is feasible. The entropy of this solution, calculated using (6.64), is 47.7.¹² Two pairs of alternate segments are identified: links 1 and 2 between nodes E and F, and links 3 and 4 between nodes F and G.

This table summarizes the progress of the algorithm in successive columns; you may find it helpful to refer to this table when reading this section. The bottom section of the table shows the origin-based link flows corresponding to the path flow solution, calculated using (6.77).

The first iteration applies the within-origin formula (6.79) to origin A, and to origin C. To apply the formula to origin A, the origin-based proportions are first calculated with equation (6.78): $\alpha_1^A = 1$, $\alpha_2^A = 0$, $\alpha_3^A = 1$, and $\alpha_4^A = 0$, and thus $h_1 \leftarrow 15$, $h_2 \leftarrow 0$, $h_3 \leftarrow 0$, and $h_4 \leftarrow 0$. (There is no change.) For origin C, we have $\alpha_1^C = 5/9$, $\alpha_2^C = 4/9$, $\alpha_3^C = 1/3$, and $\alpha_4^C = 2/3$, and thus $h_5 \leftarrow 8\frac{1}{3}$, $h_6 \leftarrow 16\frac{2}{3}$, $h_7 \leftarrow 6\frac{2}{3}$, and $h_8 \leftarrow 13\frac{1}{3}$. The entropy of this new solution has increased to 59.6.

¹²When computing this formula, $0 \log 0$ is taken to be zero, since $\lim_{x \rightarrow 0^+} x \log x = 0$.

Next, we apply the between-origin formula to the pair of alternate segments between nodes F and G. Paths 1, 3, 5, and 7 use link 3; and their companion paths (2, 4, 6, and 8, respectively) use link 4. The segment flow for link 3 is the sum of the path flows that use it (30), and similarly the segment flow for link 4 is also 30. Thus formula (6.81) requires paths 1 and 2 to have equal flow, paths 3 and 4 to have equal flow, and so on for each path and its companion. Redistributing the flow between paths and companions in this way gives the result in the third column of Table 6.2, and the entropy has increased to 72.5.

The between-origin formula is applied a second time to the other pair of alternate segments, between nodes E and F. Paths 1, 2, 5, and 6 use link 1, and the companion paths using link 2 are 3, 4, 7, and 8, respectively. The segment flows for links 1 and 2 are 40 and 20, so for each path and its companion, the path using link 1 should have twice the flow of the path using link 2. The fourth column of Table 6.2 shows the results, and the entropy has increased again to 79.8.

This solution achieves proportionality (and in fact maximizes entropy). The proportionality conditions can either be checked directly, or noticed when running the algorithm a second time does not change any path flows. For larger networks with a more complicated structure, the algorithm generally requires multiple iterations, and only converges to proportionality in the limit.

6.5.2 Dual method

An alternative approach involves the optimality conditions directly. Recall from equation (6.69) that entropy-maximizing (and thus proportional) flow on each path is $K_{rs} \exp\left(-\sum_{(i,j) \in A} \delta_{ij}^{\pi} \beta_{ij}\right)$, where K_{rs} is an OD-specific constant chosen so that the path flows sum to the total demand d_{rs} . We can adopt this condition as a *formula* for \mathbf{h} , and no matter what values are chosen for β_{ij} , the path flows we calculate satisfy proportionality. The difficulty is that they will generally not be feasible, unless the link flows \mathbf{x} corresponding to \mathbf{h} happen to equal their equilibrium values $\hat{\mathbf{x}}$. A *dual* algorithm tries to adjust these β_{ij} values until $\mathbf{x} = \hat{\mathbf{x}}$, at which point we terminate with proportional (and in fact entropy-maximizing) path flows.

This contrasts with the *primal* approach in the previous section, which always maintained feasibility (the link flows in Table 6.2 never changed from the equilibrium values) and worked toward optimality, expressed in equation (6.69). The algorithm in this section always maintains optimality, and works toward feasibility. Unlike the primal algorithm, the path flows in the dual algorithm are not feasible until termination.

The idea behind the algorithm is simple enough: start with initial values for β_{ij} on each link; calculate \mathbf{h} from equation (6.69); calculate \mathbf{x} from \mathbf{h} ; and see which links have too much flow or too little flow. Adjust the β_{ij} values accordingly, and iterate until the flow on every link is approximately equal to its equilibrium value. There are a few details to take care of: how to find an initial solution, when to terminate, how to adjust the β_{ij} values, and dealing

Table 6.2: Demonstration of primal proportionality algorithm.

		Initial solution	Within-origin	F/G PAS	E/F PAS
Entropy		47.7	59.6	72.5	79.8
Path flows	h_1	15	15	7.5	5
	h_2	0	0	7.5	5
	h_3	0	0	0	2.5
	h_4	0	0	0	2.5
	h_5	15	8.33	12.5	15
	h_6	10	16.67	12.5	15
	h_7	0	6.67	10	7.5
	h_8	20	13.33	10	7.5
OD flows	(A, B)	15	15	15	15
	(C, D)	45	45	45	45
Total link flows	x_1	40	40	40	40
	x_2	20	20	20	20
	x_3	30	30	30	30
	x_4	30	30	30	30
From origin A	x_1^A	15	15	15	10
	x_2^A	0	0	0	5
	x_3^A	15	15	7.5	7.5
	x_4^A	0	0	7.5	7.5
From origin B	x_1^B	25	25	25	30
	x_2^B	20	20	20	15
	x_3^B	15	15	22.5	22.5
	x_4^B	30	30	22.5	22.5

with redundancies in the system of constraints. The first two details are fairly simple: any initial solution will do; $\beta = \mathbf{0}$ is simplest. Terminate when \mathbf{x} is “close enough” to $\hat{\mathbf{x}}$ according to some measure.

For adjusting the β_{ij} values, notice from equation (6.69) that increasing β_{ij} will decrease x_{ij} , and vice versa. So a natural update rule is

$$\beta_{ij} \leftarrow \beta_{ij} + \alpha(x_{ij} - \hat{x}_{ij}). \quad (6.82)$$

where α is a step size, and $x_{ij} - \hat{x}_{ij}$ is the difference between the link flows currently implied by β , and the equilibrium values. In addition to this intuitive interpretation, this search direction is also proportional to the gradient of the least-squares function

$$\phi(\mathbf{x}) = \sum_{(i,j) \in A} (x_{ij} - \hat{x}_{ij})^2. \quad (6.83)$$

This function is zero only at a feasible solution, and (6.82) is a steepest descent direction in terms of \mathbf{x} .¹³

Equation (6.83) can also be used to set the step size α . One can select a trial sequence of α values (say, 1, 1/2, 1/4, 1/8, ...), evaluating each α value in turn and stopping once the new \mathbf{x} values reduce (6.83). A more sophisticated step size rule chooses α using Newton’s method, to approximately maximize entropy. Newton’s method also has the advantage of scaling the step size based on the effect changes in β have on link flows. Exercise 23 develops this approach in more detail.

A last technical detail concerns redundancies in the system of link flow equations, as discussed at the end of Section 6.5. Redundancies in the link flow equations mean that the β_{ij} values maximizing entropy may not be unique. To resolve this issue, redundant link flow constraints can be removed, and their β_{ij} values left fixed at zero. Practical experience shows that this can significantly speed convergence.

Example

The dual method is now demonstrated on the same example as the primal algorithm; see Figure 6.18. You may find it helpful to refer to Table 6.3 when reading this section to track the progress of the algorithm. The format is similar to Table 6.2, except for additional rows showing the β_{ij} values.

This table summarizes the progress of the algorithm in successive columns; you may find it helpful to refer to this table when reading this section. The bottom section of the table shows the origin-based link flows corresponding to the path flow solution, calculated using (6.77).

To begin, as discussed at the end of Section 6.5, two of the link flow constraints are redundant, and their β_{ij} values are fixed at zero. Assume that links 2 and 4 are chosen for this purpose, so $\beta_2 = \beta_4 = 0$ throughout the algorithm. (The algorithm would perform similarly for other choices of the two redundant

¹³To be precise, it is not a steepest descent direction in terms of β . An alternative derivation of equation (6.82) is explored in Exercise 22.

links; note that we are also continuing to ignore the link flow constraints associated with links 5–8, since these are identical to the OD matrix constraints (6.71) and (6.72).)

Initially, $\beta_1 = \beta_3 = 0$. This means that $\exp\left(-\sum_{(i,j) \in A} \delta_{ij}^\pi \beta_{ij}\right) = 1$ for all paths, so $h_1 = h_2 = h_3 = h_4 = K_{AB}$ and $h_5 = h_6 = h_7 = h_8 = K_{CD}$. To ensure that the sum of each OD pairs' path flows equals the total demand, we need $K_{AB} = 3.75$ and $K_{CD} = 11.25$, and equation (6.69) gives the flows on each path, as shown in the Iteration 0 column of Table 6.3.

The table also shows the link flows corresponding to this solution: links 1–4 all have 30 vehicles, whereas the equilibrium solution has $x_1 = 40$ and $x_2 = 20$. The least-squares function (6.83) has the value $(40 - 30)^2 + (20 - 30)^2 = 200$. Trying an initial step size of $\alpha = 1$ would give $\beta_1 = 0 + 1 \times (30 - 40) = -10$. The other β_{ij} values are unchanged: β_3 remains at zero because it has the correct link flow, while β_2 and β_4 are permanently fixed at zero because their link flow constraints were redundant. Re-applying equation (6.69) with this new value of β_1 (and recalculating K_{AB} and K_{CD} to satisfy the OD matrix) would give $x_1 = 60$, $x_2 = 0$, and $x_3 = x_4 = 30$. This has a larger least-squares function than before (800 vs. 200), so we try again with $\alpha = 1/2$. This is slightly better (the least-squares function is 768), but still worse than the current solution. After two more trials, we reach $\alpha = 1/8$, which produces a lower mismatch (88).

This step size is accepted, and we proceed to the next iteration. The path and link flows are shown in the Iteration 1 column of Table 6.3. The flows on links 1 and 2 are closer to their equilibrium values than before. Continuing as before, we find that $\alpha = 1/8$ is again the acceptable step size with the new link flow values, so $\beta_1 \leftarrow -1.25 + \frac{1}{8}(36.2 - 40) = -0.42$, producing the values in the Iteration 2 column. Over additional iterations, the algorithm converges to the final values shown in the rightmost column.

It is instructive to compare the dual algorithm in Table 6.3 with the primal algorithm in Table 6.2. Notice how the dual algorithm always maintains proportionality, and the link flows gradually converge to their equilibrium algorithms. By contrast, the primal algorithm maintains the link flows at their equilibrium values, and gradually converges to proportionality. The entropy also does not change monotonically, and at times it is higher than the maximum entropy value — this can only happen for an infeasible solution.

6.5.3 Traffic assignment by paired alternative segments

The primal and dual methods described in the preceding sections assumed that user equilibrium link flows were already available, and then found likely path flows as a post-processing step. Traffic assignment by paired alternative segments (TAPAS) is an algorithm which finds the equilibrium solution and proportional path flows simultaneously. Interestingly, accomplishing both tasks at once does not slow down the algorithm. TAPAS is in fact among the fastest of the traffic assignment algorithms currently known.

Table 6.3: Demonstration of dual proportionality algorithm.

		Iteration 0	1	2	...	∞
Entropy		83.2	73.4	81.9		79.8
	β_1	0	-1.25	-0.42		-0.692
	β_3	0	0	0		0
Path flows	h_1	3.75	5.83	4.53		5
	h_2	3.75	5.83	4.53		5
	h_3	3.75	1.67	2.97		2.5
	h_4	3.75	1.67	2.97		2.5
	h_5	11.25	17.49	13.58		15
	h_6	11.25	17.49	13.58		15
	h_7	5.01	8.92	6.53		7.5
	h_8	5.01	8.92	6.53		7.5
OD flows	(A, B)	15	15	15		15
	(C, D)	45	45	45		45
Total link flows	x_1	30	46.6	36.2		40
	x_2	30	13.4	23.8		20
	x_3	30	30	30		30
	x_4	30	30	30		30
From origin A	x_1^A	7.5	11.7	9.1		10
	x_2^A	7.5	3.3	5.9		5
	x_3^A	7.5	7.5	7.5		7.5
	x_4^A	7.5	7.5	7.5		7.5
From origin B	x_1^B	22.5	35.0	27.1		30
	x_2^B	22.5	10.0	17.8		15
	x_3^B	22.5	22.5	22.5		22.5
	x_4^B	22.5	22.5	22.5		22.5

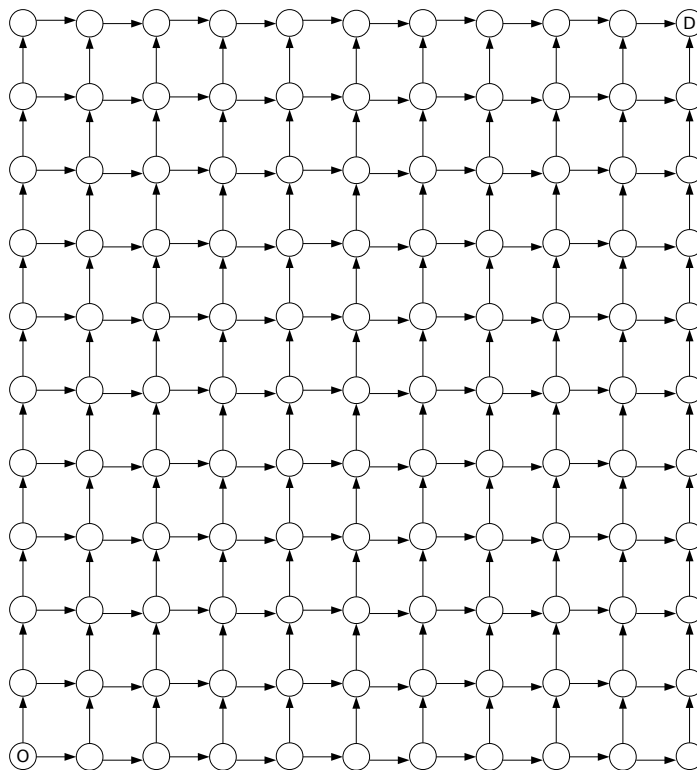


Figure 6.19: A grid network consisting of 100 city blocks.

Recall from Section 6.4 that path- and bush-based algorithms find the equilibrium solution by shifting flow from longer paths to shorter ones, and that these paths often differ on a relatively small set of links (in gradient projection, we denoted these by the set $A_3 \cup A_4$; in bush-based algorithms, by the concept of a divergence node). The main insights of TAPAS are that these algorithms tend to shift flow repeatedly between the same sets of links, and that these links are common to paths used by between different origins and destinations. As a result, it makes sense to store these path segments from one iteration to the next, rather than having to expend effort finding them again and again. Furthermore, since these links are common to multiple origins, we can apply proportionality concepts at the same time to find a high-entropy path flow solution.

Pairs of alternative segments can also form a concise representation of the equilibrium conditions. In the grid network of Figure 6.19, the number of paths between the origin in the lower-left and the destination in the upper-right is rather large (in fact there are 184,756) even though the network is a relatively modestly-sized grid of ten rows and columns. If all paths are used at equilibrium, expressing the equilibrium condition by requiring the travel times on all paths

be equal requires 184,755 equations.

The network can also be seen as a hundred “city blocks,” each of which can either be traversed in the clockwise direction (north, then east) or in the counterclockwise direction (east, then north). If travel times on all paths in the network are identical, then the travel time around each block must be the same in the clockwise and counterclockwise directions. In fact, the converse is true as well: if the travel time is the same around every block in both orientations, then the travel times on all paths in the network are the same as well. We can thus express equality of all network paths with only 100 equations!

Furthermore, other nodes in the network may serve as origins and destinations, not just nodes at two corners. Expressing equilibrium in terms of path travel time equality requires additional equations for each new OD pair, but the same 100 equations expressing equality of travel times around each block are sufficient no matter how many nodes serve as origins or destinations.

This discussion implies that most of the 184,755 path travel-time equations are redundant. It is not trivial to identify these linear dependencies, but methods based on paired alternative segments are a way to do so. The two ways to travel around each block can be seen as a pair of alternative segments between their southwestern and northeastern nodes, and the set of all such pairs of alternative segments is “spanning” in the sense that one can shift flow between any two paths with the same origin and destination by shifting flow between a sequence of pairs of alternative segments. There are some subtleties involving the equivalence of equilibrium conditions on pairs of alternative segments and on paths (see Exercise 24), but this example shows how they can often simplify the search for a user equilibrium solution.

The TAPAS algorithm represents network flows aggregated by origin, with x_{ij}^r denoting the flow on link (i, j) which started at origin r , following equation (5.30). The algorithm also involves a set of pairs of alternative segments (PASs). Each PAS ζ is defined by two path segments σ_1^ζ and σ_2^ζ starting and ending at the same nodes, and by a list of *relevant origins* Z_ζ indicating a subset of zones which have flow on both path segments at the current solution. The steps of the algorithm involve maintaining a set of PASs (creating new ones, updating relevant origins, and optionally discarding inactive ones), and adjusting the link flows x_{ij}^r to move towards user equilibrium and proportionality. Notice that TAPAS does not store the path flows \mathbf{h} explicitly, for computational reasons. Instead, the path flows are represented *implicitly*, obtained from the x_{ij}^r values using the proportional split formulas previously introduced in Section 6.5.1:

$$\alpha_{ij}^r = x_{ij}^r / \sum_{(h,j) \in \Gamma^{-1}(j)} x_{ij}^r \quad (6.84)$$

where

$$x_{ij}^r = \sum_{s \in Z} \sum_{\pi \in \hat{\Pi}_{r,s}} \delta_{ij}^\pi h^\pi. \quad (6.85)$$

An example of a PAS is shown in Figure 6.20. For each link, the upper and lower labels give the flows on that link from Origin 1 and Origin 2, respectively.

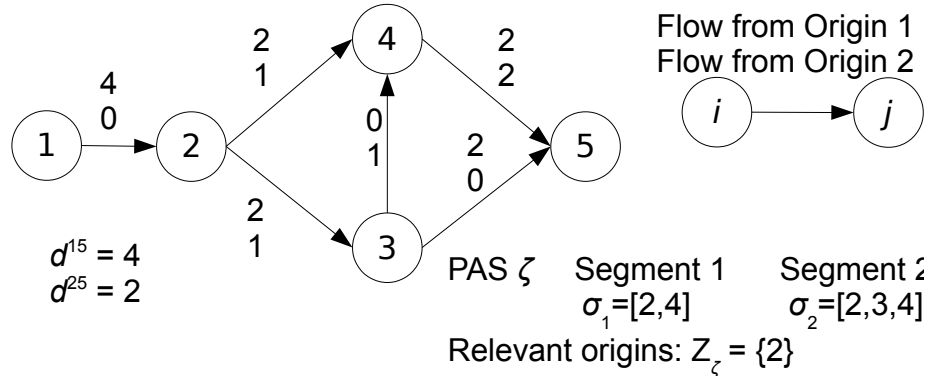


Figure 6.20: Example for demonstrating pairs of alternative segments.

There are two path segments: $[2, 4]$ and $[2, 3, 4]$, and there is one relevant origin (Origin 2). You might expect that this PAS is also relevant to Origin 1; and indeed at the ultimate equilibrium solution this will be true. However, in large networks it is not immediately obvious which PASs are relevant to which origins, and the TAPAS algorithm must discover this during its steps.

There are three main components to the algorithm: PAS management, flow shifts, and proportionality adjustments. PAS management involves identifying new PAS, updating the lists of relevant origins, and removing inactive ones. Flow shifts move the solution closer to user equilibrium, by shifting vehicles from longer paths to shorter ones. Proportionality adjustments maintain the total link flows at their current values, but adjust the origin-specific link flows to increase the entropy of the path flow solution implied by (6.84). One possible way to perform these steps is as follows; the rest of this subsection fills out the details of each step.

1. Find an initial origin-disaggregated solution, and initialize the set of PASs to be empty.
2. Update the set of PASs by determining whether new ones should be created, or whether existing ones are relevant to more origins.
3. Perform flow shifts within existing PASs.
4. Perform proportionality adjustments within existing PASs.
5. Check for convergence, and return to step 2 unless done.

This algorithmic description may appear vague. Like many of the fastest algorithms currently available, the performance of the algorithm depends on successfully balancing these three components of the algorithm. The right amount of time to spend on each component is network- and problem-specific, and implementations that make such decisions adaptively, based on the progress of the algorithm, can work well.

Updating the set of pairs of alternative segments

In its second step, the TAPAS algorithm must update the set of PASs. Since flow shifts mainly occur within PASs, finding the user equilibrium solution relies on being able to identify new alternative routes which are shorter than the ones currently being used. Given the origin-disaggregated solution \mathbf{x}^r for each origin r , we can search for routes in the following way.

Solving a shortest path algorithm over the entire network produces a tree rooted at an origin r , containing paths to every node. (Note that already having an origin-disaggregated solution can greatly accelerate the process of finding shortest paths; see Exercise 48 from Chapter 2.) At equilibrium, essentially all of these links should be used.¹⁴ By comparing this tree to the links which are used in the disaggregate solution \mathbf{x}^r , we can identify any links in the shortest path tree not currently being used by an origin r even though there is flow to their head nodes.

More specifically, let \hat{A}^r denote the set of these links. A link (i, j) is in \hat{A}^r if (i, j) is in the shortest path tree rooted at r , if $x_{ij}^r = 0$ (it is currently unused by origin r), and if there is some other link (h, j) for which x_{hj}^r (flow from origin r is reaching the head node j in another way). We then look for a PAS ζ whose two segments σ_1^ζ and σ_2^ζ end with the links (h, j) and (i, j) , which will allow us to shift flow onto the shortest path segment.

Two possibilities exist: either such a PAS already exists (in which case we add origin r to the relevant set Z_ζ if it is not already listed), or we create a new one. To create a new PAS, we must choose two path segments σ_1^ζ and σ_2^ζ which start and end at the same node. The first segment σ_1^ζ should consist of links with positive flow from origin r , and the second one σ_2^ζ should consist of links from the shortest path tree. We also know they must both end at node j , but must choose an appropriate node for them to start (a divergence node). As discussed in Section 6.4.2, there are several ways to choose divergence nodes. For TAPAS, an ideal divergence node results in short path segments σ_1^ζ and σ_2^ζ . Short segments both result in faster computation, and intuitively are more likely to be relevant to more origins.

Putting these concepts together, we can search for a divergence node a for which there is a segment σ_1^ζ starting at a , ending with link (h, j) , and only using links with positive flow from origin r ; and a segment σ_2^ζ starting at a , ending with link (i, j) , and only using links in the current shortest path tree. Among all such divergence nodes and segments, we want one for which the two path segments are short.

As an example, consider again the network from Figure 6.20. The link performance functions and current travel times are shown in Figure 6.21. The bold links show the shortest path tree rooted at Origin 1, and see that there are two links used by this origin which are not part of this tree: links (2, 4) and (3, 5). For link (2, 4), we see that the two segments of the PAS a from the previous

¹⁴The only exceptions would be to links leading to nodes not used by this origin, or if multiple paths are tied for being shortest with one of them having zero flow.

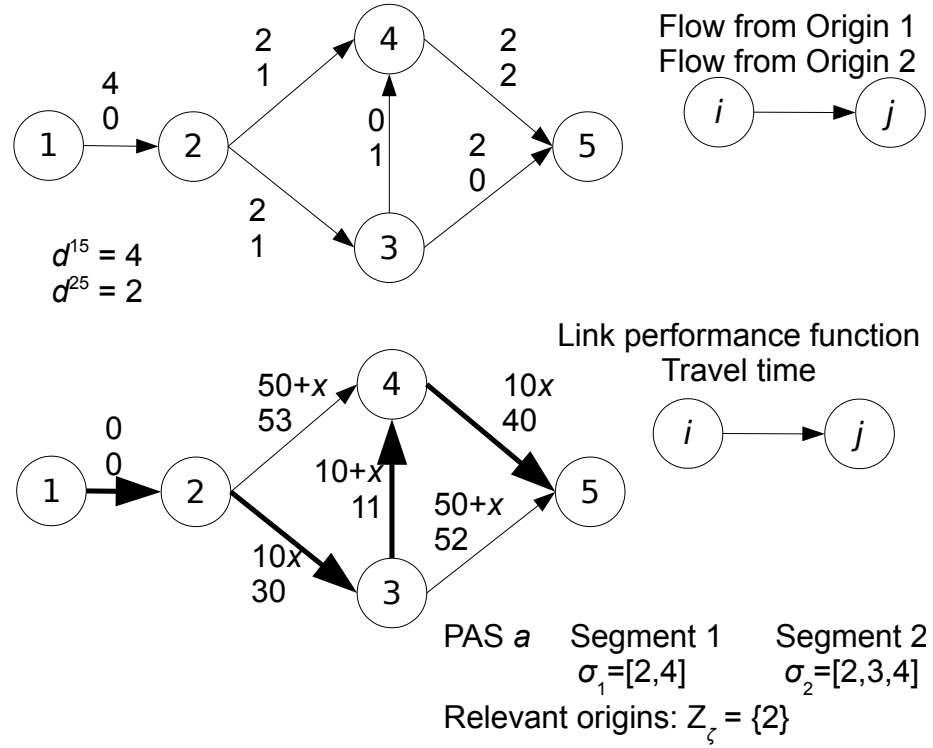


Figure 6.21: Creating a new PAS. The top panel shows the origin-specific flows, the bottom panel the link performance functions and current times. Bold links are the shortest path tree for Origin 1.

example include a segment of links used by this origin $[2, 4]$, and a segment of links from the shortest path tree $[2, 3, 4]$ that have a common divergence node 2. At this point, we declare Origin 1 relevant to this PAS by adding it to the set Z_a .

For link $(3, 5)$, we need to create a new PAS. There are two possibilities for choosing segments: one choice is $[3, 5]$ and $[3, 4, 5]$; and the other choice is $[2, 3, 5]$ and $[2, 3, 4, 5]$ (both involve a segment of used links and a segment from the shortest path tree, starting at a common divergence node). The first one is preferred, because it has fewer links — and in fact the common link $(2, 3)$ in the second PAS is irrelevant, since shifting flow between segments will not change flow on such a link at all). By being shorter, there are potentially more relevant origins. If node 3 were also an origin, it could be relevant to the first choice of segments, but not the second. Therefore we create a new PAS b , and set $\sigma_b^1 = [3, 5]$, $\sigma_b^2 = [3, 4, 5]$, and $Z_b = \{1\}$. (The choice of which segment is the first and second one is arbitrary.)

Repeating the same process with the shortest path tree from Origin 2, we

verify that it is relevant to PAS a (which it already is), and add it as relevant to PAS b , so $Z_a = Z_b = \{1, 2\}$. (Both origins are now relevant to both PASs).

Flow shifts

TAPAS uses flow shifts to find an equilibrium solution. There are two types of flow shifts: the most common involves shifting flow between the two segments on an existing PAS. The second involves identifying and eliminating cycles of used links for particular origins.

For the first type, assume we are given a PAS ζ , and without loss of generality assume that the current travel time on the first segment σ_1^ζ is greater than that on the second σ_2^ζ . We wish to shift flow from the first segment to the second one to either equalize their travel times, or to shift all the flow to the second path if it is still shorter. The total amount of flow we need to shift to equalize the travel times is approximately given by Newton's method:

$$\Delta h = \frac{\sum_{(i,j) \in \sigma_1^\zeta} t_{ij} - \sum_{(i,j) \in \sigma_2^\zeta} t_{ij}}{\sum_{(i,j) \in \sigma_1^\zeta} t'_{ij} + \sum_{(i,j) \in \sigma_2^\zeta} t'_{ij}}. \quad (6.86)$$

We must also determine whether such a shift is feasible (would shifting this much flow force an x_{ij}^r value to become negative?) and, unlike the algorithms earlier in this chapter, how much of this flow shift comes from each of the relevant origins in Z_ζ .

To preserve feasibility, for any relevant origin r , we must subtract the same amount from x_{ij}^r for each link in the longer segment, and add the same amount to each link in the shorter segment. Call this amount Δh^r . The non-negativity constraints require $\Delta h^r \leq \min_{(i,j) \in \pi_1^\zeta} \{x_{ij}^r\}$; let $\overline{\Delta h}^r$ denote the right-hand side of this inequality, which must hold for every relevant origin.

If

$$\Delta h \leq \sum_{r \in Z_\zeta} \overline{\Delta h}^r, \quad (6.87)$$

then the desired shift is feasible, and we choose the origin-specific shifts Δh^r to be proportional to their maximum values $\overline{\Delta h}^r$ to help maintain proportionality. If this shift is not feasible, then we shift as much as we can by setting $\Delta h^r = \overline{\Delta h}^r$ for each relevant origin.

An example of such a shift is shown in Figure 6.22, continuing the example from before. The left side of the figure shows the state of the network prior to the flow shift. The top panel shows the current origin-specific link flows; the middle panel the current travel times and travel time derivatives; and the bottom panel shows the structure of both PASs. Starting with PAS a , we first calculate the desired total shift from equation (6.86):

$$\Delta h = \frac{53 - (30 + 11)}{2 + (10 + 1)} = 1. \quad (6.88)$$

For origin 1, we can subtract at most 2 units of flow from segment 1, and for origin 2, we can subtract at most 1. Removing any more would result in negative

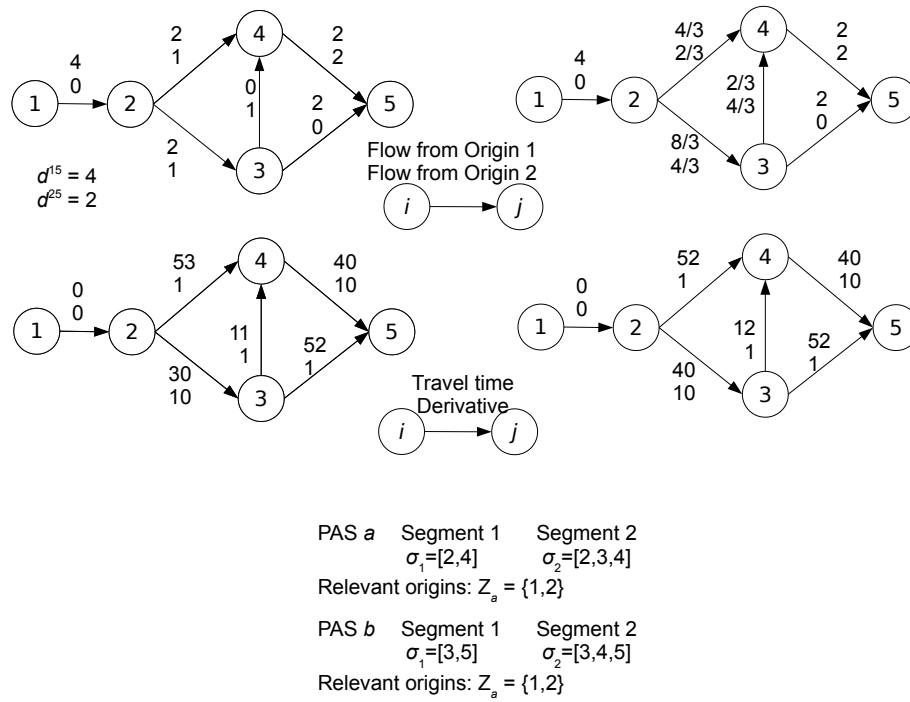


Figure 6.22: Example of flow shifts using TAPAS; left panel shows initial flows and times, right panel shows flows and times after a shift for PAS *a*.

origin flows on link (2, 4). We thus split Δh in proportion to these maximum allowable values, yielding

$$\Delta h^1 = 2/3 \quad \Delta h^2 = 1/3 \quad (6.89)$$

and producing the solution shown in the right half of Figure 6.22. Since the link performance functions are linear, Newton's method is exact, and travel times are equal on the two segments of PAS a .

Moving to the second PAS, we see that it is at equilibrium as well, and no flow shift is done: the numerator of equation (6.86) is zero. In fact, the entire network is now at equilibrium, but the origin-based link flows do not represent a proportional solution. Proportionality adjustments are discussed below.

The second kind of flow shift involves removing flow from cycles. In TAPAS, there may be occasions where cycles are found among the links with positive flow ($x_{ij}^r > 0$). Such cycles can be detected using the topological ordering algorithm described in Section 2.2.

In such cases, we can subtract flow from every link in the cycle, maintaining feasibility and reducing the value of the Beckmann function. (See Exercise 26). Let \bar{X} denote the minimum value of x_{ij}^r among the links in such a cycle. After subtracting this amount from every x_{ij}^r in the cycle, the solution is closer to equilibrium and the cycle of positive-flow links no longer exists.

Figure 6.23 shows an example of how this might happen. This network has only a single origin, and two PASs. Applying the flow shift formula, we move 1 vehicle from segment [1, 2] to [1, 3, 2], and 1 vehicle from segment [2, 4] to segment [2, 3, 4]. This produces the flow solution in the lower-left of the figure, which contains a cycle of flow involving links (2, 3) and (3, 2). If we subtract 1 unit of flow from both of those links, we have the flow solution in the lower-right. This solution is feasible and has a lower value of the Beckmann function, as you can verify.

Proportionality adjustments

TAPAS uses proportionality adjustments to increase the entropy of the path flow solution. Note that the path flow solution is not explicitly stored, since the number of used paths can grow exponentially with network size. Rather, a path flow solution is implied by the bush, using the procedure described in Section 6.5.1, and the definitions of α and h in equations (6.84) and (6.85). That is, we calculate approach proportions α_{ij}^ζ using the formula

In a proportionality adjustment, we shift flows between segments in the PAS *without* changing the total flow on each link, so some origins will shift flow from the links in σ_1^ζ to those in σ_2^ζ , while other origins will shift flow from σ_2^ζ to σ_1^ζ . Following the previous section, we will use Δh^r to denote the amount of flow shifted from each link in σ_1^ζ to each link in σ_2^ζ , using negative numbers to indicate flow shifting from σ_2^ζ to σ_1^ζ . To maintain total link flows at their

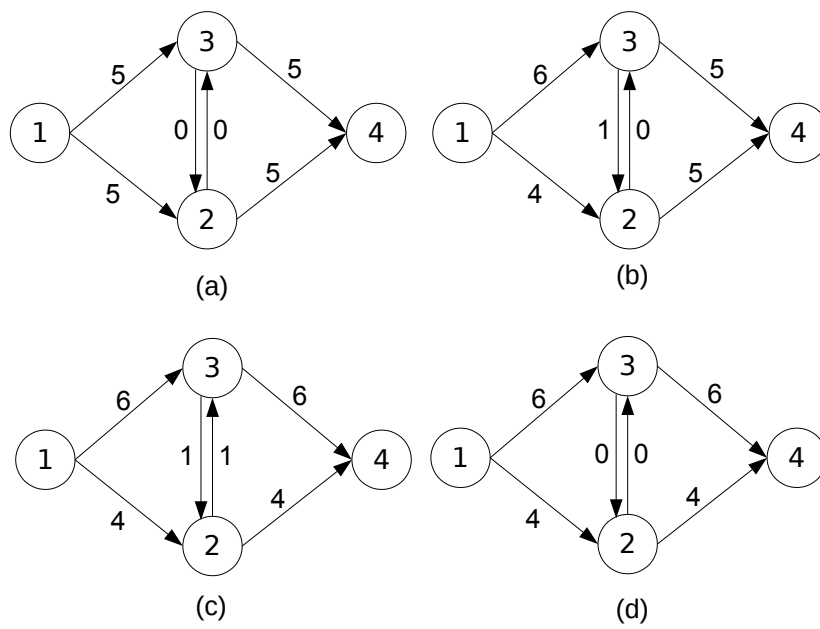


Figure 6.23: (a) Initial flow solution; (b) After a flow shift at a PAS ending at 3; (c) After a flow shift at a PAS ending at 4; (d) After removing a cycle of flow.

current levels, we will require

$$\sum_{r \in Z_\zeta} \Delta h^r = 0. \quad (6.90)$$

To describe the problem more formally, let b denote the node at the downstream end of the PAS, and let x_b denote the total flow through this node as in Equation (6.4.1). We can calculate the flow on the segments σ_1^ζ and σ_2^ζ for each relevant origin r with the formulas

$$g^r(\sigma_1^\zeta) = x_b^r \prod_{(i,j) \in \sigma_1^\zeta} \frac{x_{ij}^r}{x_j^r} \quad (6.91)$$

$$g^r(\sigma_2^\zeta) = x_b^r \prod_{(i,j) \in \sigma_2^\zeta} \frac{x_{ij}^r}{x_j^r}, \quad (6.92)$$

$$(6.93)$$

assuming positive flow through all nodes in the segment ($x_j^r > 0$).¹⁵ If proportionality were satisfied, we would have

$$\frac{g^r(\sigma_1^\zeta)}{g^r(\sigma_1^\zeta) + g^r(\sigma_2^\zeta)} = \frac{\sum_{r' \in Z_\zeta} g^{r'}(\sigma_1^\zeta)}{\sum_{r' \in Z_\zeta} (g^{r'}(\sigma_1^\zeta) + g^{r'}(\sigma_2^\zeta))} \quad (6.94)$$

for all relevant origins.

After applying segment shifts of size Δh^r , the new segment flows will be given by

$$g^r(\sigma_1^\zeta) = x_b^r \prod_{(i,j) \in \pi_1} \frac{x_{ij}^r - \Delta h^r}{x_j^r - \Delta h^r [r \neq j]} \quad (6.95)$$

$$g^r(\sigma_2^\zeta) = x_b^r \prod_{(i,j) \in \pi_2} \frac{x_{ij}^r + \Delta h^r}{x_j^r + \Delta h^r [r \neq j]}, \quad (6.96)$$

$$(6.97)$$

using brackets for an indicator function. We aim to find Δh^r values satisfying constraint (6.90) and (6.94), where the segment flows are computed with equations (6.95) and (6.96).

Solving this optimization problem exactly is a bit difficult because equations (6.84) and (6.85) are nonlinear. A good approximation method is developed in Exercise 27. A simpler heuristic is to adapt the “proportionality between origins” technique from Section 6.5.1 and approximate the (nonlinear) formulas (6.95) and (6.96) by the (linear) formulas

$$g^r(\sigma_1^\zeta) \approx g_0^r(\sigma_1^\zeta) - \Delta h^r \quad (6.98)$$

$$g^r(\sigma_2^\zeta) \approx g_0^r(\sigma_2^\zeta) + \Delta h^r \quad (6.99)$$

¹⁵What would happen if this were not true?

where $g_0^r(\sigma_1^\zeta)$ and $g_0^r(\sigma_2^\zeta)$ are the *current* segment flows (with zero shift).

Substituting equations (6.98) and (6.99) into (6.94) and simplifying, we obtain

$$\Delta h^r = g_0^r(\sigma_1^\zeta) - (g_0^r(\sigma_1^\zeta) + g_0^r(\sigma_2^\zeta)) \frac{\sum_{r' \in Z_\zeta} g_0^{r'}(\sigma_1^\zeta)}{\sum_{r' \in Z_\zeta} (g_0^{r'}(\sigma_1^\zeta) + g_0^{r'}(\sigma_2^\zeta))} \quad (6.100)$$

as a flow shift to heuristically move toward proportionality.

This heuristic is in fact exact if the PAS is “isolated” in the sense that flow does not enter or leave the segments in the middle, so $\alpha_{ij}^r = 1$ for all links in σ_1^ζ and σ_2^ζ except for the last links of each segment.

To illustrate how this procedure works, consider the example in Figure 6.24. In this example, there are multiple destinations in addition to multiple origins. The figure shows the path flow solution implied by the link flow solution at the left. We emphasize that TAPAS does not maintain the path flow solution explicitly, and the path flows are constructed from the link flows using equation (6.84). For instance, the flow on path [1, 3, 4] from Origin 1 is calculated as $3 \times \frac{1}{3} \times \frac{1}{1} = 1$. This solution does not satisfy proportionality. All the vehicles from Origin 1 passing between nodes 2 and 4 use segment [2,4], while all of those from Origin 2 passing between these nodes use segment [2,3,4]. Since the total flow on the two segments are equal (two vehicles on each), flow from both origins should split equally between the two segments.

Applying equation (6.100) gives the shifts shown in Figure 6.25. This figure also shows the new origin-specific link flows and implied path flows. In this case, the link flows on the segments of PAS *a* now satisfy proportionality. This is a case where the PAS is isolated, because no vehicles entered and left the segments in the middle, and the heuristic formula (6.100) is exact.

To show how the formula is inexact for a non-isolated PAS, consider the modification of this example shown in Figure 6.26. The only change is that node 3 is now a destination for Origin 1, with a demand of 1, and that as a result the flow from Origin 1 on link (1,3) is increased by one vehicle. Repeating the same process as above, and applying (6.100), we again one swap one vehicle between each pair of segments. This results in the situation in Figure 6.27. The origin-specific link flows are the same as in Figure 6.25, except for the additional vehicle from Origin 1 on link (1,3). But the implied path flows are quite different! This is because the additional vehicle shifted onto link (3,4) was “split” between incoming links (1,3) and (2,3), according to equation (6.84), rather than allocated solely to (2,3). (Again, TAPAS does not store the flows on individual paths, and must calculate them implicitly using this formula.) As a result, proportionality is still not satisfied: between segments [2,4] and [2,3,4], Origin 1 splits in the ratio of 3:2, whereas Origin 2 splits in the ratio 1:1. This is closer to proportionality from before, but not exact. Repeated applications of the heuristic shift formula will converge to a proportional solution, in this case.

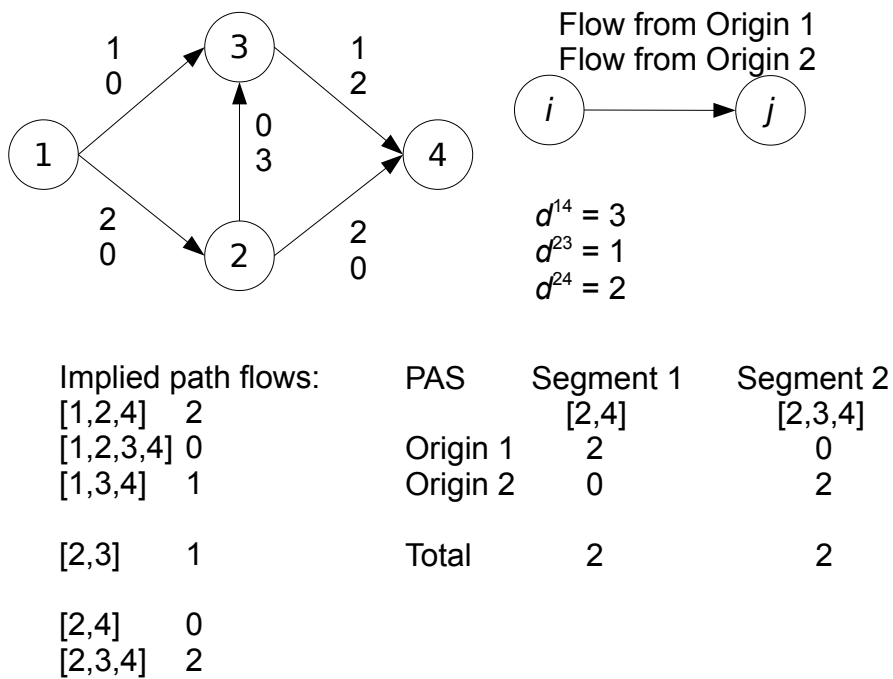


Figure 6.24: Example of a proportionality adjustment in TAPAS.

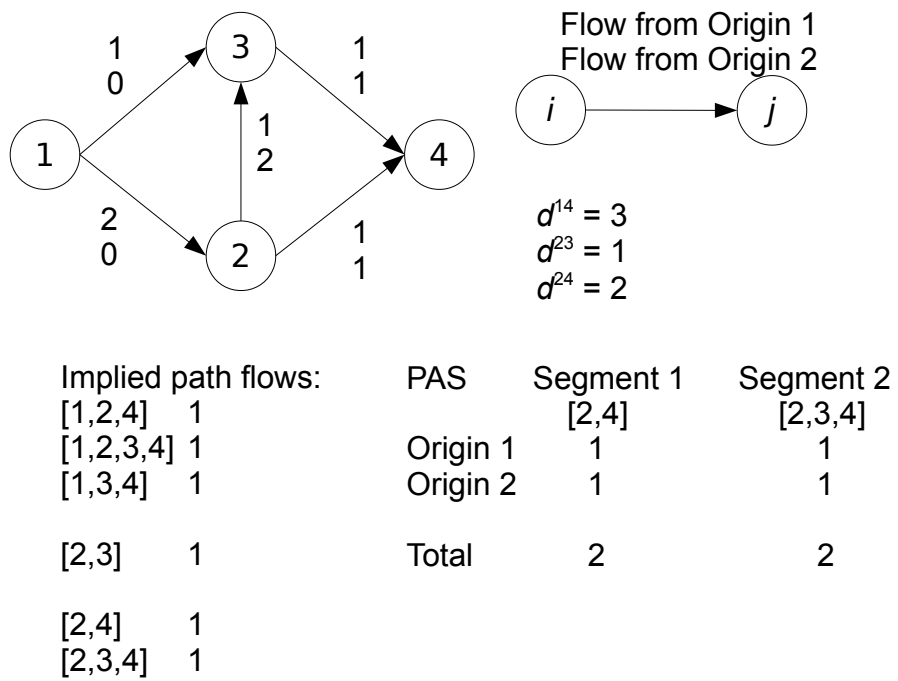


Figure 6.25: Updated flows after a proportionality adjustment for PAS a .

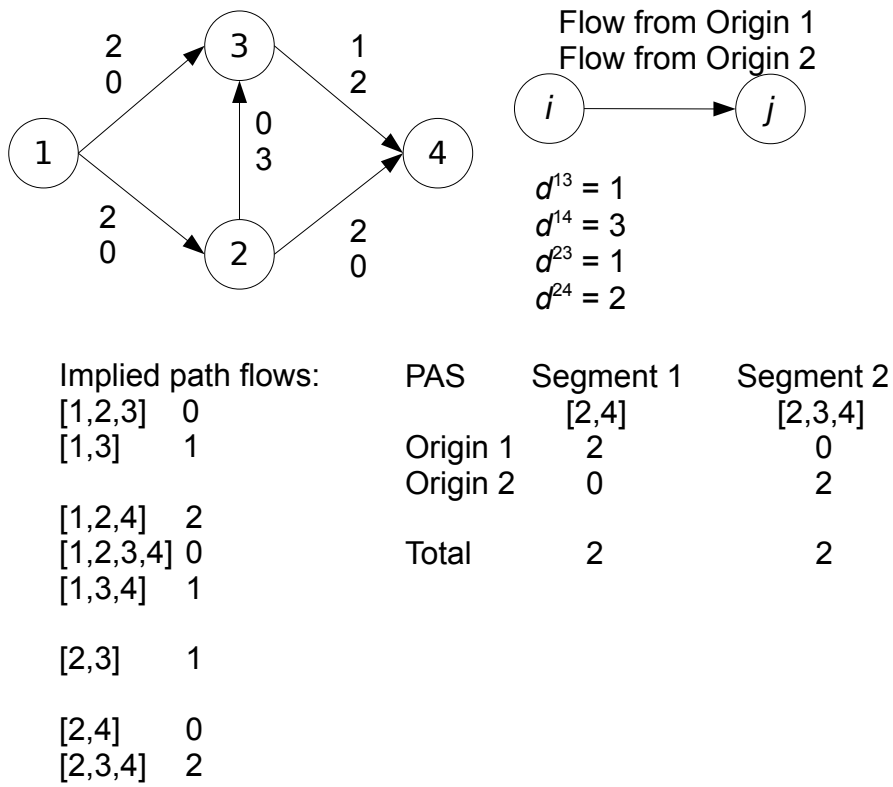


Figure 6.26: Example of a proportionality adjustment for a non-isolated PAS.

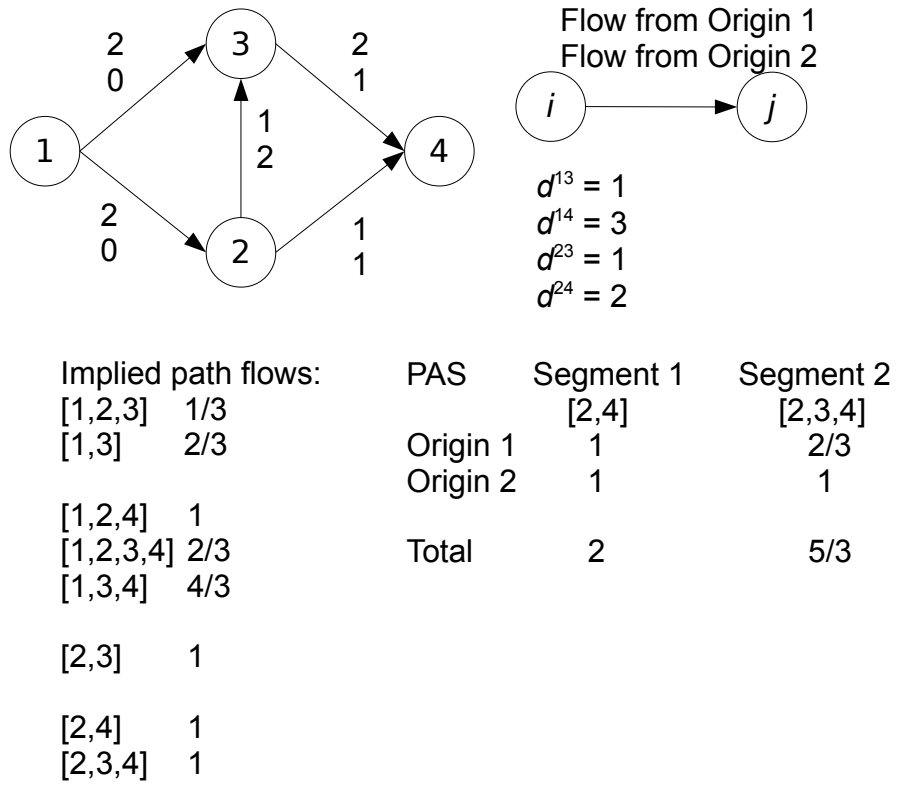


Figure 6.27: The heuristic formula is not always exact for a non-isolated PAS.

6.6 Historical Notes and Further Reading

(These sections are incomplete in this beta version of the text, and will be substantially expanded in the complete first edition.)

Many gap measures have been proposed. For the relative gap variants γ_1 , γ_2 , and γ_3 defined in the text, see Chiu et al. (2010), Patriksson (1994), and Boyce et al. (2004) for more detail. The average excess cost and maximum excess cost were proposed in Bar-Gera (2002). In studying the Philadelphia network, Boyce et al. (2004) found that freeway link flows stabilized once a relative gap of 10^{-4} was reached.

Extensive reviews of algorithms for the traffic assignment problem are found in Patriksson (1994) and Florian and Hearn (1995). For the specific methods discussed in this text, the method of successive averages and Frank-Wolfe are both instances of the more general “convex combinations” method, and can in fact be applied to any convex optimization problem (Bertsekas, 2016). The Frank-Wolfe method itself was proposed in Frank and Wolfe (1956), and the conjugate version in Mitradijeva and Lindberg (2013). Notable link-based algorithm not presented in this chapter is the simplicial decomposition methods of Smith (1983) and Lawphongpanich and Hearn (1984); but see Chapter 8 for discussion of this method in the setting of equilibrium with link interactions.

The gradient projection and manifold suboptimization algorithms were presented in Jayakrishnan et al. (1994) and Florian et al. (2009), respectively. Another notable path-based algorithm not described here is the disaggregate simplicial decomposition method of Larsson and Patriksson (1992).

Nie (2010a) described a general framework for bush-based algorithms, uniting earlier work on origin-based assignment (Bar-Gera, 2002), Algorithm B (Dial, 2006), and local user cost equilibrium (Gentile, 2014). See Xie et al. (2013) for a discussion about the close relationships between origin-based assignment and local user cost equilibrium. Interestingly, many of the concepts in bush-based algorithms were anticipated in the study of routing in telecommunications networks; see Gallager (1977) and Bertsekas et al. (1984) for examples of such work.

The primal method for maximizing path flow entropy is described at greater length in Bar-Gera (2006). The dual method is the conjugate gradient method of Larsson et al. (2001); another dual method not described here is iterative balancing; see Bell and Iida (1997). Traffic assignment by paired alternative segments was presented in Bar-Gera (2010).

6.7 Exercises

1. [32] One critique of the BPR link performance function is that it allows link flows to exceed capacity. An alternative link performance “function” is $t_{ij} = t_{ij}^0 / (u_{ij} - x_{ij})$ if $x_{ij} < u_{ij}$, and ∞ otherwise, where t_{ij}^0 and u_{ij} are the free-flow time and capacity of link (i, j) . First show that $t_{ij} \rightarrow \infty$ as $x_{ij} \rightarrow u_{ij}$. How would using this kind of link performance function affect

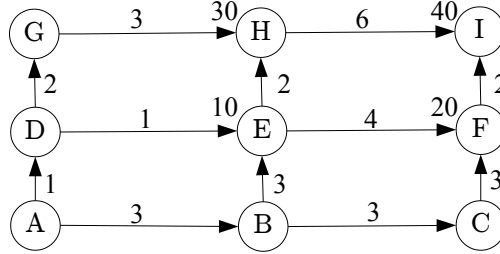


Figure 6.28: Network for Exercise 6.

the solution algorithms discussed in this chapter?

2. [33] Show that the relative gap γ_1 and average excess cost are always nonnegative, and equal to zero if and only if the link or path flows satisfy the principle of user equilibrium.
3. [61] Some relative gap definitions require a lower bound \underline{f} on the value of the Beckmann function at optimality. Let \mathbf{x} denote the current solution, $f(\mathbf{x})$ the value of the Beckmann function at the current solution, and $TSTT(\mathbf{x})$ and $SPTT(\mathbf{x})$ the total system travel time and shortest path travel time at the current solution, respectively. Show that $f(\mathbf{x}) + SPTT(\mathbf{x}) - TSTT(\mathbf{x})$ is a lower bound on the Beckmann function at user equilibrium.
4. [10] What is the value of the lower bound $\underline{f} = f(\mathbf{x}) + SPTT(\mathbf{x}) - TSTT(\mathbf{x})$ if \mathbf{x} satisfies the principle of user equilibrium?
5. [23] Let (\mathbf{h}, \mathbf{x}) and (\mathbf{g}, \mathbf{y}) be two feasible solutions to the Beckmann formulation (6.3)–(6.6), and let $\lambda \in [0, 1]$. Show that $(\lambda\mathbf{h} + (1-\lambda)\mathbf{g}, \lambda\mathbf{x} + (1-\lambda)\mathbf{y})$ is also feasible, directly from the constraints (without appealing to convexity.)
6. [35] In the network in Figure 6.28, all trips originate at node A. The links are labeled with the current travel times, and the nodes are labeled with the number of trips whose destination is that node.
 - (a) Find the shortest paths from node A to all other nodes, and report the cost and backnode labels upon termination.
 - (b) What would be the target link flow solution $\hat{\mathbf{x}}$ in the method of successive averages or the Frank-Wolfe algorithm?
7. [73] All-or-nothing assignments $\hat{\mathbf{x}}$ play a major role in link-based algorithms. A naïve way to calculate these is to start with zero flows on each link; then find the shortest path from each origin r to each destination s ;

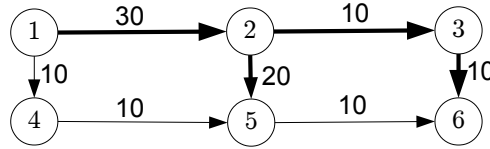


Figure 6.29: Network for Exercise 8, boldface links indicate previous $\hat{\mathbf{x}}$ target.

then add d^{rs} to each link in this path. This method may require adding up to $|Z|^2$ terms for each link, in case every shortest path uses the same link. Formulate a more efficient algorithm which requires solving one shortest path problem per origin, and which requires adding no more than $|Z|$ terms for each link. (*Hint: Do not wait until the end to calculate $\hat{\mathbf{x}}$ and find a way to build \mathbf{x} as you go.*)

8. [42] Consider the network in Figure 6.29, with a single origin and two destinations. Each link has the link performance function $10 + x^2$, and the boldface links indicate the links used in the previous $\hat{\mathbf{x}}$ target. Report the new target link flows $\hat{\mathbf{x}}$, the step size λ , and the new resulting link flows, according to (a) Frank-Wolfe and (b) conjugate Frank-Wolfe.
9. [47] Consider the network in Figure 6.30, where 8 vehicles travel from node 1 to node 4. Each link is labeled with its delay function. For each of the algorithms listed below, report the resulting link flows, average excess cost, and value of the Beckmann function.
 - (a) Perform three iterations of the method of successive averages.
 - (b) Perform three iterations of the Frank-Wolfe algorithm.
 - (c) Perform three iterations of conjugate Frank-Wolfe.
 - (d) Perform three iterations of gradient projection.
 - (e) Perform three iterations of manifold suboptimization.
 - (f) Perform three iterations of Algorithm B (for each iteration, do one flow update and one bush update)
 - (g) Perform three iterations of origin-based assignment (for each iteration, do one flow update and one bush update)
 - (h) Perform three iterations of linear user cost equilibrium (for each iteration, do one flow update and one bush update)
 - (i) Compare and discuss the performance of these algorithms.
10. [48] Consider the network in Figure 6.31. The cost function on the light links is $3 + (x_a/200)^2$, and the delay function on the thick links is $5 + (x_a/100)^2$. 1000 vehicles are traveling from node 1 to 9, and 1000 vehicles from node 4 to node 9. For each of the algorithms listed below, report the resulting link flows, average excess cost, and value of the Beckmann function.

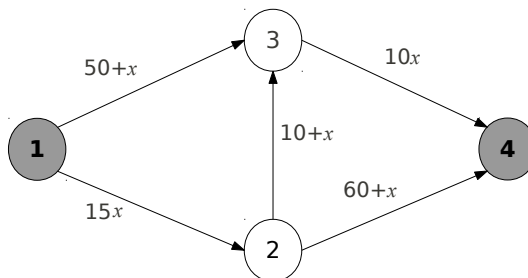


Figure 6.30: Network for Exercise 9.

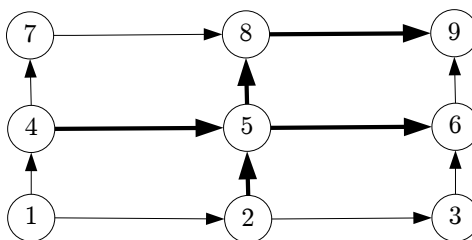


Figure 6.31: Network for Exercise 10

- (a) Perform three iterations of the method of successive averages.
 - (b) Perform three iterations of the Frank-Wolfe algorithm.
 - (c) Perform three iterations of conjugate Frank-Wolfe.
 - (d) Perform three iterations of projected gradient.
 - (e) Perform three iterations of gradient projection.
 - (f) Perform three iterations of Algorithm B (for each iteration, do one flow update and one bush update)
 - (g) Perform three iterations of origin-based assignment (for each iteration, do one flow update and one bush update)
 - (h) Perform three iterations of linear user cost equilibrium (for each iteration, do one flow update and one bush update)
 - (i) Compare and discuss the performance of these algorithms.
11. [43] The method of successive averages, as presented in the text, uses the step size $\lambda_i = 1/(i + 1)$ at iteration i . Other choices of step size can be used, and Exercise 12 shows that the algorithm converges whenever $\lambda_i \in [0, 1]$, $\sum \lambda_i = \infty$ and $\sum \lambda_i^2 < \infty$. Which of the following step size choices guarantee convergence?
- (a) $\lambda_i = 1/(i + 2)$

- (b) $\lambda_i = 4/(i + 2)$
- (c) $\lambda_i = 1/i^2$
- (d) $\lambda_i = 1/(\log i)$
- (e) $\lambda_i = 1/\sqrt{i}$
- (f) $\lambda_i = 1/i^{2/3}$

12. [65] (*Proof of convergence for the method of successive averages.*) Consider the method of successive averages applied to the vector of link flows. This produces a sequence of link flow vectors $\mathbf{x}^1, \mathbf{x}^2, \mathbf{x}^3, \dots$ where \mathbf{x}^i is the vector of link flows at iteration i . We can also write down the sequence f^1, f^2, f^3, \dots of the values taken by the Beckmann function for $\mathbf{x}^1, \mathbf{x}^2$, etc. To show that this algorithm converges to the optimal solution, we have to show that either $\mathbf{x}^i \rightarrow \hat{\mathbf{x}}$ or $f^i \rightarrow \hat{f}$ as $i \rightarrow \infty$, where $\hat{\mathbf{x}}$ is the user equilibrium solution and \hat{f} the associated value of the Beckmann function. This exercise walks through one proof of this fact, for any version of the method of successive averages for which $\lambda_i \in [0, 1]$, $\sum \lambda_i = \infty$ and $\sum \lambda_i^2 < \infty$.

- (a) Assuming that the link performance functions are differentiable, show that for any feasible \mathbf{x} and \mathbf{y} there exists $\theta \in [0, 1]$ such that

$$f(\mathbf{y}) = f(\mathbf{x}) + \sum_{(i,j) \in A} t_{ij}(x_{ij})(y_{ij} - x_{ij}) + \frac{1}{2} \sum_{(i,j) \in A} t'_{ij}((1 - \theta)x_{ij} + \theta y_{ij})(y_{ij} - x_{ij})^2. \quad (6.101)$$

- (b) Setting $\mathbf{x} = \mathbf{x}^i$ and $\mathbf{y} = \mathbf{x}^{i+1}$, recast equation (6.101) into an expression for the difference in the values of the Beckmann function between two consecutive iterations of MSA, in terms of λ_i and \mathbf{x}_i^* .
 - (c) Sum the resulting equation over an infinite number of iterations to obtain a formula for the limiting value \hat{f} of the sequence f^1, f^2, \dots
 - (d) Use the facts that $\sum \lambda_i = \infty$, $\sum \lambda_i^2 < \infty$, and that \hat{f} has a finite value to show that the limiting values of $SPTT(\mathbf{x}^i)$ and $TSTT(\mathbf{x}^i)$ must be equal, implying that the limit point is a user equilibrium.
13. [34] The derivation leading to (6.16) assumed that the solution to the restricted VI was not at the endpoints $\lambda = 0$ or $\lambda = 1$. Show that if you are solving (6.16) using either the bisection method from Section 3.3.2, or Newton's method (with a "projection" step ensuring $\lambda \in [0, 1]$), you will obtain the correct solution to the restricted VI even if it is at an endpoint.
14. [55] (*Linking Frank-Wolfe to optimization.*) At some point in the Frank-Wolfe algorithm, assume that the current link flows are \mathbf{x} and the target

link flows \mathbf{x}^* have just be found, and we need to find new flows $\mathbf{x}'(\lambda) = \lambda\mathbf{x}^* + (1 - \lambda)\mathbf{x}$ for some $\lambda \in [0, 1]$. Let $z(\lambda)$ be the value of the Beckmann function at $\mathbf{x}'(\lambda)$.

- (a) Using the multi-variable chain rule, we can show that z is differentiable and $z'(\lambda)$ is the dot product of the gradient of the Beckmann function evaluated at $\mathbf{x}'(\lambda)$ and the direction $\mathbf{x}^* - \mathbf{x}$. Calculate the gradient of the Beckmann function and use this to write out a formula for $z'(\lambda)$.
 - (b) Is z a convex function of λ ?
 - (c) Show that $z'(0) = 0$ only if \mathbf{x} is an equilibrium, and that otherwise $z'(0) < 0$.
 - (d) Assume that the solution of the restricted variational inequality in the Frank-Wolfe algorithm is for an “interior” point $\lambda^* \in (0, 1)$. Show that $z'(\lambda^*) = 0$.
 - (e) Combine the previous answers to show that the Beckmann function never increases after an iteration of the Frank-Wolfe algorithm (and always decreases strictly if not at an equilibrium).
15. [74] (*Proof of convergence for Frank-Wolfe.*) Exercise 14 shows that the sequence of Beckmann function values f^1, f^2, \dots from subsequent iterations of Frank-Wolfe is nonincreasing. Starting from this point, show that this sequence has a limit, and that the resulting limit corresponds to the global minimum of the Beckmann function (demonstrating convergence to equilibrium.) Your solution may require knowledge of real analysis.
 16. [11] When is the Beckmann function quadratic?
 17. [33] Identify conjugate directions for the following quadratic programs:
 - (a) $f(x_1, x_2) = x_1^2 + x_2^2$
 - (b) $f(x_1, x_2) = x_1^2 + x_2^2 + \frac{2}{3}x_1x_2$
 - (c) $f(x_1, x_2) = 2x_1^2 + 3x_2^2 + \frac{1}{9}x_1x_2$

18. [38] The *biconjugate* Frank-Wolfe method chooses a target vector \mathbf{x}^* so that the search direction \mathbf{x}^* is conjugate to the last two search directions, rather than just the last one. Let \mathbf{x}^{AON} reflect the all-or-nothing solution at the current point, \mathbf{x}_{-1}^* the target vector used at the last iteration, and \mathbf{x}_{-2}^* the target vector used two iterations ago. Also let λ_{-1} be the step size used for the last iteration, and define

$$\mu = - \frac{\sum_{(i,j) \in A} (\lambda_{-1}(x_{-1}^*)_{ij} + (1 - \lambda_{-1})(x_{-2}^*)_{ij} - x_{ij})(x_{ij}^{\text{AON}} - x_{ij})t'_{ij}}{\sum_{(i,j) \in A} (\lambda_{-1}(x_{-1}^*)_{ij} + (1 - \lambda_{-1})(x_{-2}^*)_{ij} - x_{ij})((x_{-2}^*)_{ij} - (x_{-1}^*)_{ij})t'_{ij}} \quad (6.102)$$

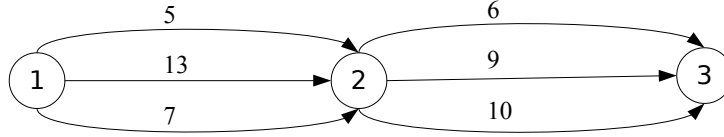


Figure 6.32: Bush for Exercise 20 with current flows.

and

$$\nu = \frac{\mu\lambda_{-1}}{1 - \lambda_{-1}} - \frac{\sum_{(i,j) \in A} ((x_{-1}^*)_{ij} - x_{ij})(x_{ij}^{AON} - x_{ij})t'_{ij}}{\sum_{(i,j) \in A} ((x_{-1}^*)_{ij} - x_{ij})^2 t'_{ij}}. \quad (6.103)$$

Show that the formula

$$\mathbf{x}^* = \frac{1}{1 + \mu + \nu} (\mathbf{x}^{AON} + \nu\mathbf{x}_{-1}^* + \mu\mathbf{x}_{-2}^*) \quad (6.104)$$

gives both a feasible target point \mathbf{x}^* , and one conjugate to the two previous search directions based on the Hessian of the Beckmann function at the current solution.

19. [41] Section 6.4.2 includes an example for Algorithm B, and Figure 6.12 shows the bush link flows at the end of a flow shifting operation. Perform a second iteration of flow shifting on the same bush, recalculating L and U labels, and scanning all nodes in reverse topological order. Report the new link flows and travel times. Also report the new bush after eliminating unused links and adding shortcuts. Does your answer depend on whether you use the L or U labels to define “shortcuts”?
20. [43] Figure 6.32 shows a bush with the current link flows labeled. Each link has delay function $5 + 2x^2$. Calculate the L , U , M , D , and α labels for all links and nodes in the bush, and identify the divergence node for each bush node.
 - (a) Perform one step of flow shifting using Algorithm B.
 - (b) Perform one step of flow shifting using OBA (starting from the original flows).
 - (c) Perform one step of flow shifting using LUCE (starting from the original flows).
 - (d) Report the maximum excess cost before and after each of these flow shifts. Which algorithm reduced this gap measure by the most?
21. [73]. This exercises walks through a proof of the formula (6.70) for choosing the threshold ϵ for finding proportional path flows. A set of paths

$\hat{\Pi} = \cup \hat{\Pi}_{r,s}$ is 2-consistent if there are no paths $\pi_1, \pi_2, \pi_3, \pi_4$ satisfying the following conditions: (1) π_1 and π_3 are in $\hat{\Pi}$; (2) at least one of π_2 and π_4 is not in $\hat{\Pi}$; (3) π_1 and π_2 connect the same OD pair; (4) π_3 and π_4 connect the same OD pair; and (5) π_1 and π_3 use the same links as π_2 and π_4 , exactly the same number of times. For instance, in Figure 6.16, the path set $\hat{\Pi} = \{[1, 2], [1, 3, 5, 6, 4, 2], [3, 5, 6, 4], [3, 5, 7, 8, 6, 4]\}$ is not 2-consistent, because we can choose $\pi_1 = [1, 3, 5, 6, 4, 2]$, $\pi_2 = [1, 3, 5, 7, 8, 6, 4, 2]$, $\pi_3 = [3, 5, 6, 4]$, and $\pi_4 = [3, 5, 7, 8, 6, 4]$. Conditions (1)–(4) are clearly satisfied. For condition (5), look at any link in the network, and count the number of times that link is used in π_1 and π_3 , and the number of times it is used in π_2 and π_4 ; every link is either unused in both pairs, used in exactly one path in both pairs, or used in both paths in each pair. Now assume that $g_a \leq \epsilon \leq g_r/2$, as in (6.70), and choose $\hat{\Pi}$ to be all paths whose travel time is within ϵ of the shortest for its OD pair. Show that there are no four paths π_1, \dots, π_4 satisfying all of the conditions in the previous paragraph. (Hint: argue by contradiction, and apply each of the conditions, using the assumptions about the acceptance and rejection gaps to bound each path's travel time relative to the shortest path travel time.)

22. [38]. The proof of Theorem 5.4 started from the entropy-maximizing Lagrangian (5.31), which Lagrangianized both the link flow constraints (with multipliers β_{ij}) and the OD matrix constraints (with multipliers γ_{rs}). Alternatively, we can Lagrangianize only the link flow constraints, and replace (h_π/d^{rs}) with h_π (why?), giving the equation

$$\hat{\mathcal{L}}(\mathbf{x}, \boldsymbol{\beta}) = \sum_{\pi \in \Pi} h_\pi \log h_\pi + \sum_{(i,j) \in A} \beta_{ij} \left(\hat{x}_{ij} - \sum_{\pi \in \Pi} \delta_{ij}^\pi h_\pi \right). \quad (6.105)$$

Show that the gradient of this alternative Lagrangian with respect to $\boldsymbol{\beta}$ has components given by (6.82). That is, show that

$$\frac{\partial \hat{\mathcal{L}}}{\partial \beta_{ij}} = x_{ij} - \hat{x}_{ij}$$

where x_{ij} is computed from the current path flows h_π .

23. [59]. The dual algorithm step (6.82) can be compactly written as $\boldsymbol{\beta} \leftarrow \boldsymbol{\beta} + \alpha \Delta \boldsymbol{\beta}$, where $\Delta \beta_{ij} = x_{ij} - \hat{x}_{ij}$. Let $f(\alpha)$ denote the value of the alternative Lagrangian (6.105) after a step of size α is taken, and the new $\boldsymbol{\beta}$ and \mathbf{h} values are calculated. Newton's method can be used to find an α value which approximately minimizes $f(\alpha)$, maximizing entropy in the direction $\Delta \boldsymbol{\beta}$. The Newton step is $\alpha = -f'(0)/f''(0)$.

- (a) Show that $f'(0) = \|\nabla_{\boldsymbol{\beta}} \hat{\mathcal{L}}\|^2$. (See Exercise 22.)
 (b) Show that $f''(0) = \nabla_{\boldsymbol{\beta}}^T \hat{\mathcal{L}} H_{\boldsymbol{\beta}} \hat{\mathcal{L}} \nabla_{\boldsymbol{\beta}} \hat{\mathcal{L}}$, where $H_{\boldsymbol{\beta}}$ is the Hessian of the alternate Lagrangian with respect to $\boldsymbol{\beta}$.

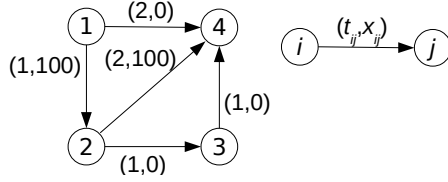


Figure 6.33: Network for Exercise 24.

(c) Show that

$$\frac{f'(0)}{f''(0)} = \frac{\sum_{(i,j) \in A} (x_{ij} - \hat{x}_{ij})^2}{\sum_{(i,j) \in A} \sum_{(k,\ell) \in A} (x_{ij} - \hat{x}_{ij})(x_{k\ell} - \hat{x}_{k\ell}) \sum_{\pi \in \Pi} h^\pi \delta_{ij}^\pi \delta_{k\ell}^\pi}.$$

24. [30]. In the discussion surrounding Figure 6.19, we argued that satisfying the equilibrium conditions around a “spanning” set of PASs (one for each block) was sufficient for establishing equilibrium on the entire network. Consider the network in Figure 6.33, where the demand from origin 1 to destination 4 is 100 vehicles, and there are two PASs: one between segments $[1, 4]$ and $[1, 2, 3, 4]$, and another between segments $[2, 4]$ and $[2, 3, 4]$. These are spanning, in the sense that by shifting flows between these two PASs we can obtain any feasible path flow solution from any other. They also satisfy the equilibrium conditions: for the first PAS, because there is no flow on either segment¹⁶; for the second, because the travel times are equal on the two segments. Yet the network is not at equilibrium, since $[1, 4]$ is the only shortest path and it is unused. Explain this apparent inconsistency.
25. [62]. Develop one or more algorithms to find “short” segments when generating a new PAS. These methods should require a number of steps that grows at most linearly with network size.
26. [22]. Show that the cycle-removing procedure described in the TAPAS algorithm maintains feasibility of the solution (flow conservation at each node, and non-negativity of link flows), and that the Beckmann function decreases strictly (assuming link performance functions are positive).
27. [68]. This exercise develops a technique for approximately solving equations (6.90) and (6.94), better than the heuristic given in the text.
- (a) Define $\Delta h^r(\rho)$ to be the amount of flow that needs to be shifted from origin r 's flows on σ_1 to σ_2 , to adjust the proportion $g^r(\sigma_1)^\zeta / (g^r(\sigma_1)^\zeta + g^r(\sigma_2)^\zeta)$ to be exactly ρ . A negative value of this function indicates

¹⁶There is flow on link $(1,2)$, but not on the entire segment $[1, 2, 3, 4]$.

shifting flow in the reverse direction, from σ_2 to σ_1 . Show that this function is defined over $[0, 1]$, and its range is $[\underline{\Delta}, \overline{\Delta}]$, where $\underline{\Delta} = -\min_{(i,j) \in \sigma_2} x_{ij}^r$ and $\overline{\Delta} = \min_{(i,j) \in \sigma_1} x_{ij}^r$. Furthermore show that $\Delta h^r(0) = \underline{\Delta}$ and $\Delta h^r(1) = \overline{\Delta}$.

- (b) Show that equation (6.94) is satisfied if the flow shift $\Delta h^r(\rho)$ is applied to all relevant origins $r \in Z_\zeta$.
- (c) Therefore, it is enough to find a value of ρ for which

$$U(\rho) \equiv \sum_{r \in Z_\zeta} \Delta h^r(\rho) = 0,$$

in order to satisfy (6.90). The function U is continuous and defined on the interval $[0, 1]$. Show that $U(0) \leq 0$ and $U(1) \geq 0$, ensuring that a zero exists in this interval.

- (d) Develop a quadratic approximation for $\Delta h^r(\rho)$ based on three known points: $\Delta h^r(0) = \underline{\Delta}$, $\Delta h^r(1) = \overline{\Delta}$, and $\Delta h^r(\rho_r) = 0$, where ρ_r is the current proportion $g^r(\sigma_1)^\zeta / (g^r(\sigma_1)^\zeta + g^r(\sigma_2)^\zeta)$.
- (e) By summing these, develop a quadratic approximation for $U(\rho)$, and give an explicit formula for its root.

Chapter 7

Sensitivity Analysis and Applications

This chapter shows how a sensitivity analysis can be conducted for the traffic assignment problem (TAP), identifying how the equilibrium assignment will change if the problem parameters (such as the OD matrix or link performance functions) are changed. This type of analysis is useful in many ways: it can be used to determine the extent to which errors or uncertainty in the input data create errors in the output data. It can be used as a component in so-called “bilevel” optimization problems, where we seek to optimize some objective function while enforcing that the traffic flows remain at equilibrium. This occurs most often in the network design problem, where one must determine how to improve network links to reduce total costs, and in the OD matrix estimation problem, where one attempts to infer the OD matrix from link flows, or improve upon an existing estimate of the OD matrix.

After exploring the sensitivity analysis problem using the familiar Braess network, the first objective in the chapter is calculating derivatives of the equilibrium link flows with respect to elements in the OD matrix. It turns out that this essentially amounts to solving another, easier, traffic assignment problem with different link performance functions and constraints. The remainder of the chapter shows how these derivatives can be used in the network design and OD matrix estimation problems, which are classic transportation examples of bilevel programs.

7.1 Sensitivity Analysis Preliminaries

Figure 7.1 shows the Braess network. When this network was first introduced, the demand between node 1 and node 4 was $d_{14} = 6$, and the equilibrium solution was found to be $x_{13} = x_{23} = x_{24} = 2$ and $x_{12} = x_{34} = 4$, with a travel time of 92 minutes on all three paths. What if, instead, the demand d_{14} took another value? Figure 7.2 presents four plots showing how the equilibrium

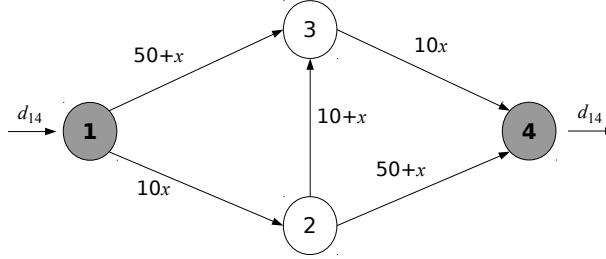


Figure 7.1: Braess network with varying demand from 1 to 4.

solution varies according to the demand level. Panel (a) shows the flows x_{12} and x_{34} , panel (b) shows the flows x_{13} and x_{24} , panel (c) shows the flow x_{23} , and panel (d) shows the shortest path travel time between nodes 1 and 4, at the corresponding equilibrium solution. You can check that when $d_{14} = 6$, the original equilibrium solution is shown in this figure.

Instead of the OD matrix, we also could have changed the link performance functions in the network. Now assume that d_{14} is fixed at its original value of 6, but that the link performance function on link (2,3) can vary. Let $t_{23}(x_{23}) = y + x_{23}$, where y is the free-flow time, resulting in the network shown in Figure 7.3. In the base solution $y = 10$, but conceivably the “free-flow time” could be changed. If the speed limit were increased, y would be lower; if traffic calming were implemented, y would be higher. In an extreme case, if the link were closed entirely you could imagine y takes an extremely large value, large enough that no traveler would use the path. One can also effectively decrease y by providing incentives for traveling on this link (a direct monetary payment, a discount at an affiliated retailer, etc.), and conceivably this incentive could be so large that y is negative. The resulting sensitivity analysis is provided in Figure 7.4

Examining the plots in Figure 7.2 and 7.4, we see that the relationships between the equilibrium solution (link flows and travel times) and the demand or free-flow time are all piecewise linear. Each “piece” of these piecewise linear functions corresponds to a particular subset of the paths being used — for instance, in Figure 7.2, when the demand is lowest, only the middle path is used. When the demand is highest, only the two outer paths are used. When the demand is at a moderate level, all three paths are used. Within each of these regions, the relationship between the demand and the equilibrium solution is linear. These pieces meet at so-called *degenerate* solutions, where the equilibrium solution does not use all of the minimum travel-time paths. (For instance, when $d_{14} = 40/11$ the equilibrium solution requires all drivers to be assigned to the middle path, even though all three have equal travel times.)

In general networks involving nonlinear link performance functions, these relationships cannot be expected to stay linear. However, they are still defined by piecewise functions, with each piece corresponding to a certain set of paths

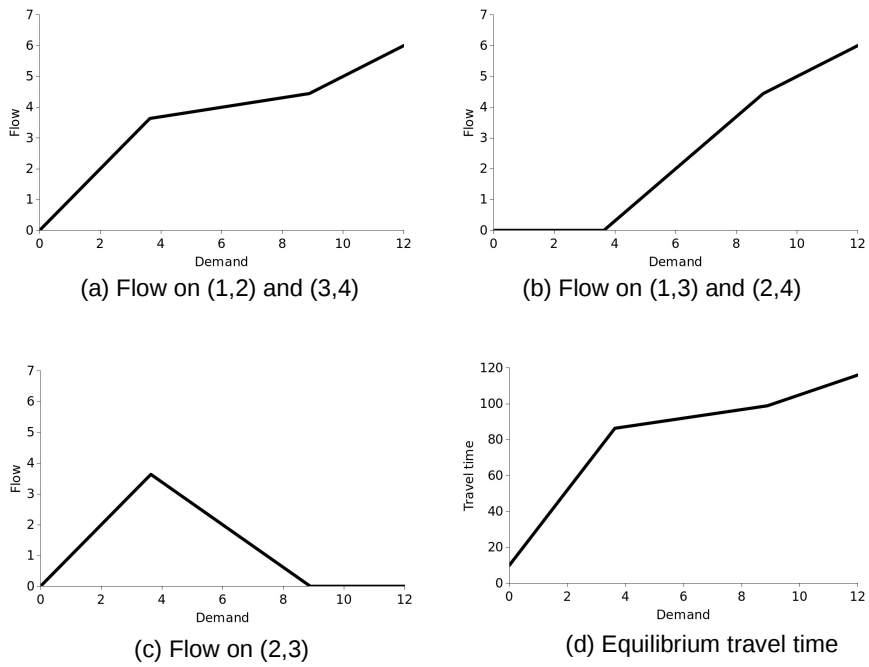


Figure 7.2: Sensitivity analysis of the Braess network to d_{14} .

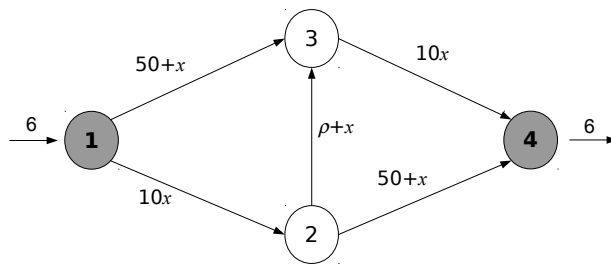


Figure 7.3: Braess network with varying free-flow time on (2,3).

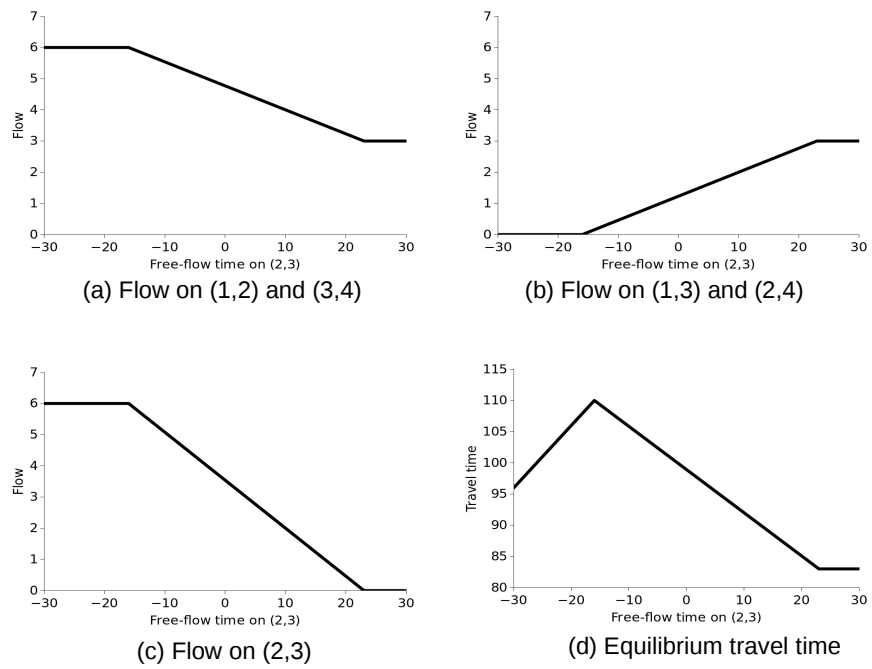


Figure 7.4: Sensitivity analysis of the Braess network to y .

being used, and with the pieces meeting at degenerate solutions. The goal of the sensitivity analyses in this chapter is to identify derivatives of the equilibrium solution (link flows and travel times) at a given point. For these derivatives to be well-defined, we therefore assume that *the point at which our sensitivity analysis occurs is not degenerate*. That is, all minimum-travel time paths have positive flow. This assumption is not too restrictive, because there are only a finite number of degenerate points; for instance, if we pick the demand value at random, the probability of ending up at a degenerate point is zero.

This sensitivity analysis is still local, because the information provided by a derivative grows smaller as we move farther away from the point where the derivative is taken. For a piecewise function, the derivative provides no information whatsoever for pieces other than the one where the derivative was taken.

In this chapter, we show how this kind of sensitivity analysis can be used in two different ways. In the network design problem, this type of sensitivity analysis can be used to determine where network investments are most valuable. In Figure 7.4, the fact that the equilibrium travel time increases when y decreases (around the base solution $y = 10$) highlights the Braess paradox: investing money to improve this link will actually increase travel times throughout the network. If we were to conduct a similar analysis for other links in the network, we would see that the equilibrium travel time would decrease with improvements to the link. In the OD matrix estimation problem, we can use this sensitivity analysis to help calibrate an OD matrix to given conditions.

7.2 Calculating Sensitivities

This section derives sensitivity formulas showing the derivative of the equilibrium link flows and travel times with respect to two parameters: (1) a change in an entry of the OD matrix d^{rs} , and (2) a change to a parameter in the link performance functions (such as the free-flow time or capacity in a BPR function). In this section, assume that we are given some initial OD matrix or link performance functions, and the corresponding equilibrium solution. For our purposes, it will be most convenient if this equilibrium solution is expressed in bush-based form, that is, with vectors $\hat{\mathbf{x}}^r$ showing the flow on each link corresponding to each origin r . In this case, the non-degenerate condition requires that unused links are not part of the equilibrium bushes, that is, if $\hat{x}_{ij}^r = 0$ for any link (i, j) and any origin r , then link (i, j) does not correspond to any shortest path starting from node r — in terms of the distance labels L_i^r , we have $\hat{x}_{ij}^r > 0$ if and only if $L_j^r = L_i^r + t_{ij}$. Let $\mathcal{B}^r = \{(i, j) \in A : \hat{x}_{ij}^r > 0\}$ denote the equilibrium bush for origin r .

The non-degeneracy assumption is important, because one can show that *if the change to the OD matrix or link performance functions is small and the original equilibrium solution is non-degenerate, all of the equilibrium bushes remain unchanged*. Equivalently, even after drivers shift flows to find the new equilibrium, the set of used paths will remain the same as it was before. Fur-

thermore, one can show that the equilibrium solution is differentiable, and the derivatives of the equilibrium link flows or travel times with respect to values in the OD matrix or link performance function parameters can be interpreted as the sensitivities of the equilibrium solution.

There are several ways to calculate the values of these derivatives: historically, the first researchers used matrix-based formulas, and subsequent researchers generalized these formulas using results from the theory of variational inequalities. We adopt a different approach, using the bush-based solution representation, because it leads to an easy solution method and is fairly straightforward. This approach is based on the fact that the equilibrium solution (travel times t_{ij} and bush flows x_{ij}^r) must satisfy the following equations for each origin r :

$$L_j^r - L_i^r - t_{ij}(\hat{x}_{ij}^r) = 0 \quad \forall (i, j) \in \mathcal{B}^r \quad (7.1)$$

$$L_r^r = 0 \quad (7.2)$$

$$\sum_{(h,i) \in \Gamma^{-1}(i)} \hat{x}_{hi}^r - \sum_{(i,j) \in \Gamma(i)} \hat{x}_{ij}^r = d^{ri} \quad \forall i \in N \setminus \{r\} \quad (7.3)$$

$$\sum_{(h,r) \in \Gamma^{-1}(r)} \hat{x}_{hr}^r - \sum_{(r,j) \in \Gamma(r)} \hat{x}_{rj}^r = - \sum_{s \in Z} d^{rs} \quad (7.4)$$

$$\hat{x}_{ij}^r = 0 \quad \forall (i, j) \notin \mathcal{B}^r \quad (7.5)$$

Equations (7.1)–(7.2) reflect the equilibrium condition, and equations (7.3)–(7.4) represent flow conservation. The number of equations for each origin is no more than the sum of the number of links and nodes in the network.

Furthermore, these conditions must remain true even as the problem data (OD matrix and link performance functions) are perturbed. Since derivatives of the equilibrium solution exist under the non-degeneracy assumption, we can differentiate equations (7.1)–(7.4) to identify the relationships which must hold true among these derivatives. For brevity, in this chapter we use ξ_{ij}^r to denote the derivative of x_{ij}^r , and Λ_i to denote the derivative of L_i . These derivatives are taken with respect to either an OD matrix entry or a link performance function parameters, as described separately below.

7.2.1 Changes to the OD matrix

Assume first that we change a single entry in the OD matrix corresponding to origin \hat{r} and destination \hat{s} , so $\xi_{ij}^r = dx_{ij}^r / dd^{\hat{r}\hat{s}}$ and $\Lambda_i = dL_i / dd^{\hat{r}\hat{s}}$. Then differentiating each of equations (7.1)–(7.4) with respect to $d^{\hat{r}\hat{s}}$ gives the following

equations for each $r \in Z$:

$$\Lambda_j^r - \Lambda_i^r - t'_{ij} \sum_{r' \in Z} \xi_{ij}^{r'} = 0 \quad \forall (i, j) \in \mathcal{B}^r \quad (7.6)$$

$$\Lambda_r^r = 0 \quad (7.7)$$

$$\sum_{(h,i) \in \Gamma^{-1}(i)} \xi_{hi}^r - \sum_{(i,j) \in \Gamma(i)} \xi_{ij}^r = \begin{cases} 1 & \text{if } r = \hat{r} \text{ and } i = \hat{s} \\ 0 & \text{otherwise} \end{cases} \quad \forall i \in N \setminus r \quad (7.8)$$

$$\sum_{(h,r) \in \Gamma^{-1}(r)} \xi_{hr}^r - \sum_{(r,j) \in \Gamma(r)} \xi_{rj}^r = \begin{cases} -1 & \text{if } r = \hat{r} \\ 0 & \text{otherwise} \end{cases} \quad (7.9)$$

$$\xi_{ij}^r = 0 \quad \forall (i, j) \notin \mathcal{B}^r \quad (7.10)$$

where t'_{ij} is the derivative of the link performance function, evaluated at the current equilibrium solution $\hat{\mathbf{x}}$ (and thus treated as a constant in these equations). Equations (7.6) enforce the fact that the equilibrium bushes must remain the same. That is, the shortest path labels L_i and travel times t_{ij} must change in such a way that every link on the bush is part of a minimum travel time path to its head node. Equations (7.8) and (7.9) enforce flow conservation. For all bushes except for \hat{r} , the total flow from the origin to each destination is the same, so flow is allowed to redistribute among the bush links, but the flows starting or ending at a node cannot change. For the bush corresponding to \hat{r} , a unit increase in demand from \hat{r} to \hat{s} must be reflected by an additional vehicle leaving \hat{r} and an additional vehicle arriving at \hat{s} .

All together, the system of equations (7.6)–(7.9) involves variables Λ_i^r for each origin r and node i , and ξ_{ij}^r for each origin r and link (i, j) . Furthermore, for each origin, it contains an equation for each link and each node. Therefore, this linear system of equations can be solved to obtain the sensitivity values.¹

However, there is an easier way to solve for Λ_i^r and ξ_{ij}^r . Using the techniques in Section 3.3, you can show that the equations (7.6)–(7.9) are exactly the optimality conditions to the following minimization problem:

$$\begin{aligned} \min_{\xi^r, \Lambda^r} \quad & \frac{1}{2} \sum_{(i,j) \in A} t'_{ij} \left(\sum_{r \in Z} \xi_{ij}^r \right)^2 \\ & + \sum_{r \in Z} \sum_{i \in N} \Lambda_i^r \left(\sum_{(h,i) \in \Gamma^{-1}(i)} \xi_{hi}^r - \sum_{(i,j) \in \Gamma(i)} \xi_{ij}^r - \Delta_{ri} \right) \end{aligned} \quad (7.11)$$

$$\text{s.t.} \quad \Lambda_r^r = 0 \quad \forall r \in Z \quad (7.12)$$

$$\xi_{ij}^r = 0 \quad \forall (i, j) \notin \mathcal{B}^r \quad (7.13)$$

¹A careful reader will note that one of the flow conservation equations for each origin is redundant, but this is of no consequence to what follows.

where Δ_{ri} represents the right-hand side of equation (7.8) or (7.9), that is, $\Delta_{\hat{r}\hat{s}} = 1$, $\Delta_{\hat{r}\hat{r}} = -1$, and $\Delta_{ri} = 0$ otherwise.

This optimization problem can be put in a more convenient form by interpreting Λ_i^r as the Lagrange multiplier for the flow conservation equation corresponding to node i in bush r , and the second term in (7.11) as the Lagrangianization of this equation. Furthermore, defining $\xi_{ij} = \sum_{r \in Z} \xi_{ij}^r$, the optimization problem can be recast in the following equivalent form:

$$\min_{\xi^r} \int_0^{\xi_{ij}} t'_{ij} \xi \, d\xi \quad (7.14)$$

$$\text{s.t.} \quad \sum_{(h,i) \in \Gamma^{-1}(i)} \xi_{hi}^r - \sum_{(i,j) \in \Gamma(i)} \xi_{ij}^r = \Delta_{ri} \quad \forall i \in N, r \in Z \quad (7.15)$$

$$\xi_{ij}^r = 0 \quad \forall (i,j) \notin \mathcal{B}^r \quad (7.16)$$

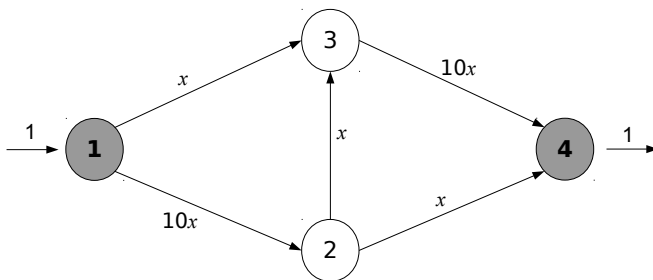
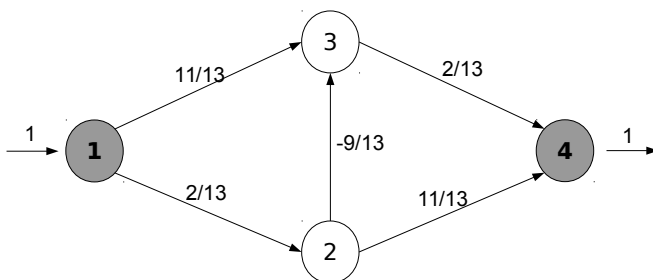
This is essentially a traffic assignment problem (TAP) in bush-based form (see Chapter 6), with the following changes:

- The original link performance functions have been replaced by *linear* link performance functions with slope equal to the derivative of the original link performance function at the original equilibrium solution. (Remember that t'_{ij} is a constant, the value of the link performance function derivative at the equilibrium solution.)
- The equilibrium bushes for each origin are *fixed* at the bushes for the original equilibrium solution.
- The only entry in the OD matrix is one unit of demand from \hat{r} to \hat{s} .
- There are no non-negativity conditions. This is because the solution variables ξ_{ij} represent changes in the original link flows, and it is possible for these changes to be negative as well as positive (cf. Figure 7.2).

If you have access to an implementation of a bush-based algorithm for solving TAP, it is easy to modify the program to take account of these distinctions, and to find the link flow sensitivities ξ_{ij}^r . From here, the values of Λ_i^r can be found by solving a shortest path problem with link travel times $\frac{dt_{ij}}{dx_{ij}} \xi_{ij}$.

As a demonstration, we use the Braess network of Figure 7.1, working around the base demand $d_{14} = 6$ and base equilibrium solution $\hat{x}_{12} = \hat{x}_{34} = 4$, $\hat{x}_{13} = \hat{x}_{23} = \hat{x}_{24} = 2$. At this level of demand, all paths are used and the equilibrium bush contains all of the links in the original network. Furthermore, at this equilibrium solution the derivatives of the link performance functions are $t'_{12} = t'_{34} = 10$ and $t'_{13} = t'_{23} = t'_{24} = 1$. The linear link performance functions based on these derivatives are shown in Figure 7.5. The OD matrix is replaced by a single unit of flow traveling from 1 (our \hat{r}) to 4 (our \hat{s}).

Solving the problem without non-negativity constraints produces the ξ_{ij} values shown in Figure 7.6. Substituting these into the link performance functions in Figure 7.5 shows that the equilibrium is satisfied: all paths have equal travel

Figure 7.5: Modified traffic assignment problem for sensitivity to d_{14} Figure 7.6: Solution to traffic assignment problem for sensitivity to d_{14}

times of $31/13$. This is exactly the slope of the piece of the equilibrium travel time (Figure 7.2d) around $d_{14} = 6$, that is, the equilibrium travel time in the sensitivity problem gives the derivative of the equilibrium travel time in the original network.

7.2.2 Changes to a link performance function

Now assume that we change a parameter y (which may represent the free-flow time, capacity, or any other parameter) in the link performance function corresponding to link (i, j) , so $\xi_{ij}^r = dx_{ij}^r/dy$ and $\Lambda_i = dL_i/dy$. This means that the link performance function t_{ij} now depends on both its flow x_{ij} , and the parameter y , so we write $t_{ij}(x, y)$. We will write $t'_{ij,x}$ to mean the partial derivative with respect to link flow², and $t'_{ij,y}$ to mean the partial derivative with respect to the improvement parameter y . Then differentiating each of equations (7.1)–(7.4)

²This is what we wrote as t'_{ij} in the previous section, when the link performance function only depended on x_{ij} .

with respect to y gives the following equations for each $r \in Z$:

$$\Lambda_j^r - \Lambda_i^r - t'_{ij,x} \sum_{r' \in Z} \xi_{ij}^{r'} - t'_{ij,y} = 0 \quad \forall (i, j) \in \mathcal{B}^r \quad (7.17)$$

$$\Lambda_r^r = 0 \quad (7.18)$$

$$\sum_{(h,i) \in \Gamma^{-1}(i)} \xi_{hi}^r - \sum_{(i,j) \in \Gamma(i)} \xi_{ij}^r = 0 \quad \forall i \in N \setminus \{r\} \quad (7.19)$$

$$\sum_{(h,r) \in \Gamma^{-1}(r)} \xi_{hr}^r - \sum_{(r,j) \in \Gamma(r)} \xi_{rj}^r = 0 \quad (7.20)$$

$$\xi_{ij}^r = 0 \quad \forall (i, j) \notin \mathcal{B}^r \quad (7.21)$$

where as before, $t'_{ij,x}$ and $t'_{ij,y}$ are evaluated at the current, equilibrium solution $\hat{\mathbf{x}}$. Equations (7.17) enforce the fact that the equilibrium bushes must remain the same, taking into account both the change in travel time on (i, j) due to the change in its link performance function as well as changes in all links' travel times from travelers shifting paths. Equations (7.19) and (7.20) enforce flow conservation. These equations are simpler than for the case of a change to the OD matrix, because the total number of vehicles on the network remains the same, and these vehicles can only shift amongst the paths in the bush. There is no change in the flow originating or terminating at any node, and the ξ variables must form a circulation.

As with a change in an OD matrix entry, the system of equations (7.17)–(7.20) is a linear system involving, for each origin, variables for each node and link. Repeating the same steps as before, this system of equations can be seen as the optimality conditions for the following optimization problem:

$$\min_{\xi^r} \int_0^{\xi_{ij}} (t'_{ij,x} \xi + t'_{ij,y}) d\xi \quad (7.22)$$

$$\text{s.t.} \quad \sum_{(h,i) \in \Gamma^{-1}(i)} \xi_{hi}^r - \sum_{(i,j) \in \Gamma(i)} \xi_{ij}^r = 0 \quad \forall i \in N, r \in Z \quad (7.23)$$

$$\xi_{ij}^r = 0 \quad \forall (i, j) \notin \mathcal{B}^r \quad (7.24)$$

This is essentially a traffic assignment problem in bush-based form, with the following changes:

- The original link performance functions have been replaced by *affine* link performance functions with slope equal to the derivative of the original link performance function at the original equilibrium solution, and intercept equal to the derivative of the link performance function with respect to the parameter y .
- The equilibrium bushes for each origin are *fixed* at the bushes for the original equilibrium solution.
- All entries in the OD matrix are zero.

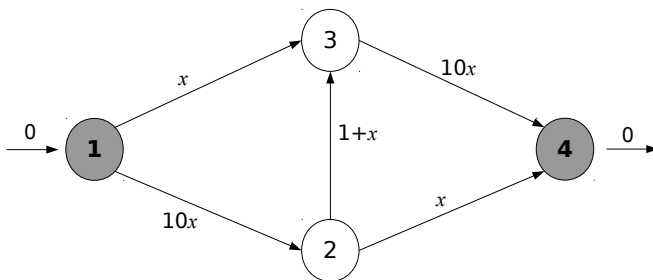


Figure 7.7: Modified traffic assignment problem for sensitivity to y_{23}

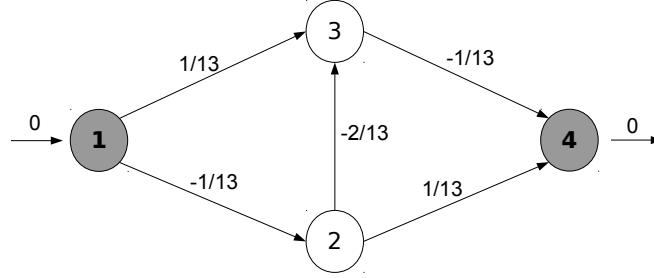
- There are no non-negativity conditions.

As a demonstration, we use the Braess network of Figure 7.3, working around the base free-flow time $y = 10$ and base equilibrium solution $\hat{x}_{12} = \hat{x}_{34} = 4$, $\hat{x}_{13} = \hat{x}_{23} = \hat{x}_{24} = 2$. That is, we replace the link performance function $t_{23}(x_{23}) = 10 + x_{23}$ with the function $t_{23}(x_{23}, y) = y + x_{23}$ and see what happens when y varies. At the base value $y = 10$, all paths are used and the equilibrium bush contains all of the links in the original network. Furthermore, at this equilibrium solution the derivatives of the link performance functions with respect to flows are $t'_{12,x} = t'_{34,x} = 10$ and $t'_{13,x} = t'_{23,x} = t'_{24,x} = 1$. For link (2,3), we add the constant term $t'_{23,y} = 1$. The link performance functions based on these derivatives are shown in Figure 7.7. The OD matrix is set equal to zero, since there is no change in the total demand through the network.

Solving the problem without non-negativity constraints produces the ξ_{ij} values shown in Figure 7.8. Substituting these into the link performance functions in Figure 7.7 shows that the equilibrium is satisfied: all paths have equal travel times of $-9/13$. Again, this is the slope of the piece of the equilibrium travel time (Figure 7.4d) around $y = 10$, that is, the equilibrium travel time in the sensitivity problem gives the derivative of the equilibrium travel time in the original network.

7.3 Network Design Problem

In the network design problem, one must determine how best to spend funds on improving links in the transportation network. This is a challenging optimization problem, and in cases of any practical interest one cannot hope to identify a globally optimal investment policy. This is mainly because *in transportation systems, the planner cannot compel travelers to choose a particular path*. Instead, after any improvement is made to the links, flows will redistribute according to the principle of user equilibrium. The goal is to find the best investment policy, knowing and anticipating how travelers will respond once the network has been

Figure 7.8: Solution to traffic assignment problem for sensitivity to y_{23}

changed.

Specifically, assume that the link performance functions for each link (i, j) now depend on the amount of money y_{ij} invested in that link (perhaps increasing its capacity through widening, or decreasing its free-flow time) as well as on the flow x_{ij} . One example of such a link performance function is

$$t_{ij}(x_{ij}, y_{ij}) = t_{ij}^0 \left(1 + \alpha \left(\frac{x_{ij}}{u_{ij} + K_{ij}y_{ij}} \right)^\beta \right) \quad (7.25)$$

where K_{ij} represents the capacity improvement if a single unit of money is invested on it.

A planning agency may have many different objectives and constraints when determining how to improve a network. This section develops and explores one specific variation of the network design problem, but there are many other variations which have been proposed in the literature. The version presented here is a fairly standard one, which can be extended in a number of different ways. In this variation, the objective of the planning agency is to minimize the total cost, given by the sum of total system travel time $TSTT$ (converted to monetary units by a conversion factor Θ , which also reflects duration of the analysis horizon and discounting) and the costs of the network improvements themselves. The optimization problem is

$$\min_{\mathbf{x}, \mathbf{y}} f(\mathbf{x}, \mathbf{y}) = \Theta \sum_{(i,j) \in A} x_{ij} t_{ij}(x_{ij}, y_{ij}) + \sum_{(i,j) \in A} y_{ij} \quad (7.26)$$

$$\text{s.t.} \quad \mathbf{x} \in \arg \min_{\mathbf{x} \in X} \sum_{(i,j) \in A} \int_0^{x_{ij}} t_{ij}(x, y_{ij}) dx \quad (7.27)$$

$$y_{ij} \geq 0 \quad \forall (i, j) \in A \quad (7.28)$$

Most of this problem is familiar: the objective (7.26) is to minimize the sum of total system travel time (converted to units of money) and construction cost, and constraint (7.28) requires that money can only be spent on links (not “recovered”

from them with a negative y_{ij} value). Also note that since $\mathbf{y} = \mathbf{0}$ is a feasible solution (corresponding to the “do-nothing” alternative), in the optimal solution to this problem the cost savings (in the form of reduced $TSTT$) must at least be equal to the construction costs, guaranteeing that the optimal investment policy has greater benefit than cost. The key equation here is (7.27), which requires that the link flows \mathbf{x} satisfy the principle of user equilibrium by minimizing the Beckmann function. In other words, *one of the constraints of the network design problem is itself an optimization problem*. This is why the network design problem is called a bilevel program. This type of problem is also known as a mathematical program with equilibrium constraints. This class of problems is extremely difficult to solve, because the feasible region is typically nonconvex.

To see why, consider two feasible solutions $(\mathbf{x}^1, \mathbf{y}^1)$ and $(\mathbf{x}^2, \mathbf{y}^2)$ to the network design problem. The link flows \mathbf{x}^1 are the equilibrium link flows under investment policy \mathbf{y}^1 , and link flows \mathbf{x}^2 are the equilibrium link flows under investment policy \mathbf{y}^2 . If the feasible region were a convex set, then any weighted average of these two solutions would themselves be feasible. Investment policy $\frac{1}{2}\mathbf{y}^1 + \frac{1}{2}\mathbf{y}^2$ still satisfies all the constraints on \mathbf{y} (all link investments are nonnegative). However, the equilibrium link flows under this policy cannot be expected to be the average of \mathbf{x}^1 and \mathbf{x}^2 , because the influence of y_{ij} on t_{ij} can be nonlinear and the sets of paths which are used in \mathbf{x}^1 and \mathbf{x}^2 can be completely different. In other words, the equilibrium link flows after averaging two investment policies need not be the average of the equilibrium link flows under those two policies separately.

Unfortunately, solving optimization problems with nonconvex feasible regions is a very difficult task. Therefore, solution methods for the network design problem are almost entirely heuristic in nature. These heuristics can take many forms; one popular approach is to adapt a metaheuristic method, such as those discussed in Section C.6.

Another approach is to develop a more tailored heuristic based on specific insights about the network design problem. This approach, being more educational, is adopted here. Specifically, we can use the sensitivity analysis from the previous sections to identify derivatives of the objective function f with respect to each link investment y_{ij} , and use this to move in a direction which reduces total cost.

Specifically, notice that constraint (7.27) actually makes \mathbf{x} a function of \mathbf{y} , since the solution to the user equilibrium problem is unique in link flows. That is, the investment policy \mathbf{y} determines the equilibrium link flows \mathbf{x} exactly. So, the objective function can be made a function of \mathbf{y} alone, written $f(\mathbf{x}(\mathbf{y}), \mathbf{y})$. The derivative of this function with respect to an improvement on any link is then

$$\frac{\partial f}{\partial y_{ij}} = \Theta \sum_{(k,\ell) \in A} \frac{\partial f}{\partial x_{k\ell}} \frac{\partial x_{k\ell}}{\partial y_{ij}} + 1 \quad (7.29)$$

or, substituting the derivative of (7.26) with respect to each link flow,

$$\frac{\partial f}{\partial y_{ij}} = \Theta \left\{ \sum_{(k,\ell) \in A} \frac{\partial x_{k\ell}}{\partial y_{ij}} \left(t_{k\ell}(x_{k\ell}, y_{k\ell}) + x_{k\ell} \frac{\partial t_{k\ell}}{\partial x_{k\ell}}(x_{k\ell}, y_{k\ell}) \right) + x_{ij} \frac{\partial t_{ij}}{\partial y_{ij}} \right\} + 1. \quad (7.30)$$

In turn, the partial derivatives $\frac{\partial x_{k\ell}}{\partial y_{ij}}$ can be identified using the technique of Section 7.2.2 as the marginal changes in link flows throughout the network when the link performance function of (i, j) is perturbed.

The vector of all the derivatives (7.30) forms the gradient of f with respect to \mathbf{y} . This gradient is the direction of steepest *ascent*, that is, the direction in which f is increasing fastest. Since we are solving a minimization problem, we should move in the opposite direction. Taking such a step, and ensuring feasibility, gives the updating equation

$$\mathbf{y} \leftarrow [\mathbf{y} - \mu \nabla_{\mathbf{y}} f]^+ \quad (7.31)$$

where μ is a step size to be determined, and the $[\cdot]^+$ operation is applied to each component of the vector. This suggests the following algorithm:

1. Initialize $\mathbf{y} \leftarrow \mathbf{0}$.
2. Calculate the link flows $\mathbf{x}(\mathbf{y})$ by solving the traffic assignment problem with link performance functions $\mathbf{t}(\mathbf{x}, \mathbf{y})$.
3. For each link (i, j) determine $\frac{\partial f}{\partial y_{ij}}$ by solving the sensitivity problem described in Section 7.2.2 and using (7.30).
4. Update \mathbf{y} using (7.31) for a suitable step size μ .
5. Test for convergence, and return to step 2 if not converged.

Two questions are how μ should be chosen in step 4, and how convergence should be tested in step 5. The difficulty in step 4 is that the derivatives provided by a sensitivity analysis are only local, and in particular if μ is large enough that (7.31) changes the set of used paths, the derivative information is meaningless. However, if μ is small enough one will see a decrease in the objective function if at all possible. So, one could start by testing a sequence of μ values (say, 1, 1/2, 1/4, ...), evaluating the resulting \mathbf{y} values, \mathbf{x} values, and f , stopping as soon as f decreases from its current value. (Note that this is a fairly computationally intensive process, since the traffic assignment problem must be solved for each μ to get the appropriate \mathbf{x} value.) Other options include using a stricter stopping criterion such as the Armijo rule, which would ensure that the decrease is “sufficiently large”; or using bisection to try to choose the value of μ which minimizes f in analogy to Frank-Wolfe. All of these methods require solving multiple traffic assignment problems at each iteration.

Regarding convergence in step 5, one can either compare the progress made in decreasing f over the last few iterations, or the changes in the investments

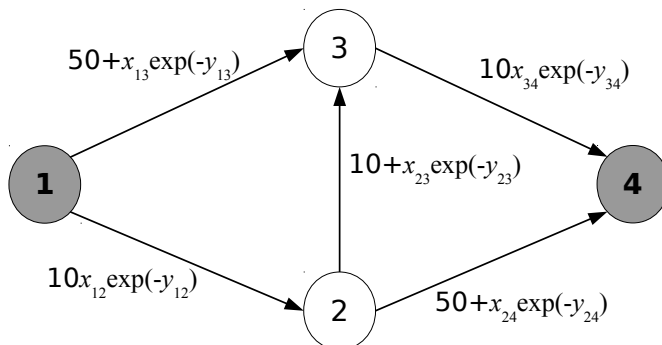


Figure 7.9: Network design problem example.

\mathbf{y} . This choice of stopping criterion is different in nature than those used in Chapter 6 for solution methods to TAP. In that case, we can prove theoretical convergence to the equilibrium solution, and we can design our stopping criteria directly on the equilibrium condition. The method described above, by contrast, is *not* proven to converge to the global optimum, and can get stuck in solutions which are locally optimal but not globally so. (This often happens in nonconvex optimization problems.) By terminating the algorithm when no more progress is made, we are checking that we have found a local optimum, but cannot guarantee that we have found a global one.

As stated above, the network design problem does not have a convex feasible region, and even if one were to eliminate \mathbf{x} by writing \mathbf{x} as a function of \mathbf{y} , the resulting function can be shown to be nonconvex and have multiple local optimal solutions. Therefore, the method described above is not guaranteed to converge to a global optimum solution. This is perhaps a bit disappointing; but, as stated above, at present there is no way to ensure global optimality within a reasonable amount of computation time. Therefore, alternative heuristics can only be compared by the quality of solutions obtained for a given quantity of computational effort.

For an example, consider the Braess network shown in Figure 7.9, where the link performance functions are shown and $\Theta = \frac{1}{20}$. Notice now that the coefficients of x_{ij} in the link performance functions now depend on the amount of money y_{ij} invested as well, with decreasing marginal returns as more money is spent. When $\mathbf{y} = \mathbf{0}$, the solution to the user equilibrium problem $\mathbf{x}(\mathbf{y})$ is the now-familiar solution to the Braess network, which divides flow evenly among all three paths in the network. The total system travel time is 552 vehicle-minutes, so the value of the objective (7.26) function is $\frac{1}{20}(552) + 0 = 27.6$. This completes the first two steps of the algorithm.

For the third step, we must solve five sensitivity problems, one for each link, to determine $\frac{\partial f}{\partial y_{ij}}$. These five problems are shown in Figure 7.10, where we have substituted the initial values $\mathbf{y} = \mathbf{0}$ and the equilibrium link flows \mathbf{x} . In all of

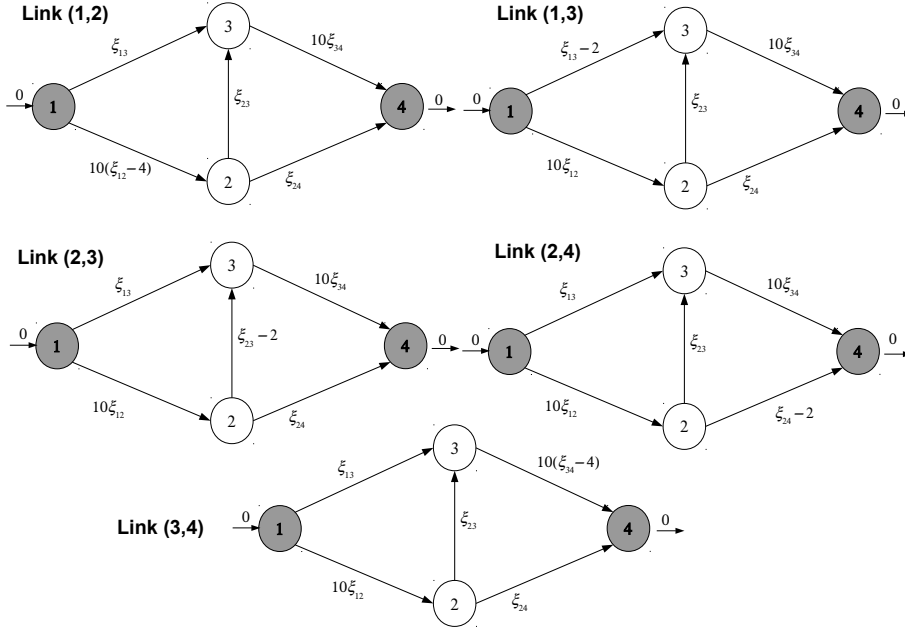


Figure 7.10: Five sensitivity problems for network design.

these, notice that the demand is zero, and the link performance functions are affine. Within each problem, the link being improved has a slightly different link performance function, accounting for the effect of the link improvement. The other links' performance functions only reflect their change due to shifting flows. For instance, in the problem in the upper left, link (1,3) is not improved, so its link performance function is simply $\xi_{13} \frac{\partial t_{13}}{\partial x_{13}} = 10\xi_{12} \exp(-y_{13}) = 10\xi_{12}$ since $y_{13} = 0$. Since link (1,2) is being improved, in addition to the term $\xi_{12} \frac{\partial t_{12}}{\partial x_{12}} = \xi_{12} \exp(-y_{12}) = \xi_{14}$, we add the constant term $\frac{\partial t_{12}}{\partial y_{12}} = -x_{14} \exp(-y_{12}) = -4$ since $x_{12} = 4$ and $y_{12} = 0$.

The solutions (in terms of ξ_{ij}) to the five sensitivity problems are shown in Table 7.1, as can be verified by substituting these ξ values into the networks in Figure 7.10. Substituting these ξ values into equation (7.30), along with the current values of the travel times and link flows, gives the gradient

$$\nabla_y f = \begin{bmatrix} y_{12} \\ y_{13} \\ y_{23} \\ y_{24} \\ y_{34} \end{bmatrix} = \begin{bmatrix} -0.85 \\ 0.49 \\ 1.42 \\ 0.49 \\ -0.85 \end{bmatrix} \tag{7.32}$$

Each component of the gradient shows how the objective of the network de-

Table 7.1: Solutions to Braess sensitivity problems.

Link	Sensitivity problem				
	(1,2)	(1,3)	(2,3)	(2,4)	(3,4)
(1,2)	3.36	-0.17	0.15	0.014	-0.28
(1,3)	-3.36	0.17	-0.15	-0.014	0.28
(2,3)	3.08	-0.15	0.31	-0.15	3.08
(2,4)	0.28	-0.014	-0.15	0.17	-3.36
(3,4)	-0.28	0.014	0.15	-0.17	3.36

sign problem will change if a unit of money is spent improving a particular link. In the derivative formula (7.30), the term in parentheses represents the marginal change in total system travel time, and the addition of unity at the end of the formula represents the increase in total expenditures. If the derivative (7.30) is negative, then the reduction in total system travel time from a marginal improvement in the link will outweigh the investment cost. If it lies between zero and one for a link, then a marginal investment in the link will reduce total system travel time, but the cost of the improvement will outweigh the value of the travel time savings. If it is greater than one, then total system travel time would actually increase if the link is improved, as in the Braess paradox. So, in this example, the gradient (7.32) shows that improvements on links (1,2) and (3,4) will be worthwhile; improvements on links (1,3) and (2,4) would reduce TSTT but not by enough to outweigh construction cost; and an improvement on link (2,3) would actually be counterproductive and worsen congestion.

So, proceeding to step 4 of the network design algorithm, we choose a trial value $\mu = 1$ and apply (7.31) to obtain a candidate solution where $y_{12} = y_{34} = 0.85$ and all other y_{ij} values remain at zero. Re-solving equilibrium with the new link performance functions, the new equilibrium solution is to load all vehicles on the middle path, that is, $x_{12} = x_{23} = x_{34} = 6$ and $x_{13} = x_{24} = 0$. The total system travel time is now 405 vehicle-minutes, so the objective is $\frac{1}{20}(405) + 2(0.85) = 22.0$. This reduces the objective from its current value of 27.6, so we accept the step size $\mu = 1$, and return to step 2.

Notice that solving the network design problem requires solving a very large number of traffic assignment subproblems: once for each iteration to determine \mathbf{x} ; modified sensitivity problems for each link to calculate derivatives; and again multiple times per iteration to identify μ . Solving practical problems can easily require solution of thousands or even millions of traffic assignment problems. In bilevel programs such as network design, having an efficient method for solving traffic assignment problems is critical. Path-based and bush-based algorithms can be efficiently “warm-started,” making them good choices for this application.

7.4 OD Matrix Estimation

Practical application of the traffic assignment problem requires two main inputs: information about the physical network (roadway topology, link performance functions) as well as information about travel demand patterns (the OD matrix). The former is much easier to obtain and validate. Standard link performance functions (such as the BPR function) can be used only knowing the free-flow time and capacity of links, which can be easily estimated; and in principle, this information can be directly inferred from field measurements and traffic sensors. The OD matrix is quite a bit harder to estimate, for several reasons. First, there is no practical way to directly observe the complete OD matrix; at best we can work with a sample of travelers who consent to reveal their travel patterns. Second, at least until recently, it was difficult to obtain this information without travelers explicitly reporting their origins and destinations — traffic sensors tell you what is happening on a specific link, but not where the people are coming from or where they are going. Recently, GPS and Bluetooth technologies have become more commonplace and can in principle provide origins and destinations automatically (a rough estimate could even be obtained from cellular phone traces) — but there are major privacy issues associated with using such data, as well as a number of data inference issues involved in translating these data into an OD matrix. Third, the number of entries in an OD matrix is much larger than the number of links in the network: the number of OD pairs is generally proportional to the *square* of the number of nodes, while the number of links is generally proportional to the number of nodes themselves. It is typical to see practical networks which have tens of thousands of links, but millions of OD pairs.

Therefore, it is natural to find ways to determine an OD matrix which should be used for traffic assignment. One approach, used in the field of travel demand modeling, is based on developing behavioral models of how households make travel choices. Using demographic and other features, there are models to estimate the total number of trips made by households over some period of time, their destinations, their mode choices, and so forth. Another approach is to attempt to infer an OD matrix from traffic counts on specified links in the network. Both methods have advantages and disadvantages: travel demand models provide more insights about underlying behavior (critical when developing long-range forecasts of future demand), and can directly lead to an OD matrix. Unfortunately, the data needed to calibrate demand models is more expensive and cumbersome to work with, often involving recruiting survey participants.

Traffic sensors, on the other hand, collect data automatically, inexpensively, and without privacy issues, but there is a major dimensionality issue. Since the number of OD pairs is much larger than the number of links, one cannot hope to uniquely determine the OD matrix solely from link counts; the problem is massively underdetermined. In a network where all nodes are zones, it is trivial to find an OD matrix which perfectly matches link counts, by creating an OD matrix where all trips travel from one node to an adjacent node. Even though this matrix is entirely unrealistic, there is no deviation whatsoever from

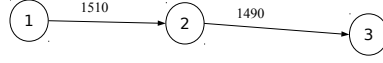


Figure 7.11: Observed link volumes from traffic sensors along a freeway.

the counts. Even worse, due to unavoidable errors in traffic count records (Figure 7.11), sometimes this trivial matrix will match counts much better than a more “realistic” matrix! In this figure, it is likely that the total flow on the freeway is approximately 1500 vehicles from node 1 to node 3; but this does not match the counts as well as 1510 vehicles from 1 to 2, and 1490 from 2 to 3. Simply matching counts does not provide the behavioral insight needed to identify what a “realistic” matrix is.

This section provides a method to reconcile both approaches. The idea is that an initial OD matrix \mathbf{d}^* is already available from a travel demand model. While this matrix is hopefully close to the true value, it also contains sampling and model estimation errors and can never be fully accurate. However, there are sensors on a subset of links $\bar{A} \subset A$, and there is a vector \mathbf{x}^* of traffic volume counts on these links. The intent is to use these traffic counts to try to improve the initial OD matrix \mathbf{d}^* . The following optimization problem expresses this:

$$\min_{\mathbf{d}, \mathbf{x}} f(\mathbf{d}, \mathbf{x}) = \Theta \sum_{(r,s) \in Z^2} (d_{rs} - d_{rs}^*)^2 + (1 - \Theta) \sum_{(i,j) \in \bar{A}} (x_{ij} - x_{ij}^*)^2 \quad (7.33)$$

$$\text{s.t.} \quad \mathbf{x} \in \arg \min_{\mathbf{x} \in X(\mathbf{d})} \sum_{(i,j) \in A} \int_0^{x_{ij}} t_{ij}(x) dx \quad (7.34)$$

$$d_{rs} \geq 0 \quad \forall (r,s) \in Z^2 \quad (7.35)$$

where Θ is a parameter ranging from zero to one and $X(\mathbf{d})$ is the set of feasible link flows when the OD matrix is \mathbf{d} .

The objective function (7.33) is of the least-squares type, and attempts to minimize both the deviation between the final OD matrix \mathbf{d} and the initial estimate \mathbf{d}^* , and the deviation between the equilibrium link flows \mathbf{x} associated with the OD matrix \mathbf{d} , and the actual observations \mathbf{x}^* on the links with sensors. The hope is to match traffic counts reasonably well, while not wandering into completely unrealistic OD matrices. The factor Θ is used to weight the importance of matching the initial OD matrix estimate, and the link flows. It can reflect the relative degree of confidence in \mathbf{d}^* and \mathbf{x}^* ; if the travel demand was obtained from high-quality data and a large sample size, whereas the traffic count data is old and error-prone, a Θ value close to one is appropriate. Conversely, if the travel demand model is less trustworthy but the traffic count data is highly reliable, a lower Θ value is appropriate. In practice, a variety of Θ values can be chosen, and the resulting tradeoffs between matching the initial estimate and link flows can be seen.

As indicated by the constraint (7.34), this optimization problem is also a bilevel program, because the mapping from an OD matrix \mathbf{d} to the resulting

link flows \mathbf{x} involves the equilibrium process. The fact that the optimization problem is bilevel also means that we cannot expect to find the global optimum OD matrix, and that heuristics should be applied. A sensitivity-based heuristic, like the one used for the network design problem, would determine how the link flows would shift if the OD matrix is perturbed, and use this information to find a “descent direction” which would reduce the value of the objective.

Following the same technique as in Section 7.3, the constraint (7.34) defines \mathbf{x} uniquely as a function of \mathbf{d} due to the uniqueness of the link flow solution to the traffic assignment problem. (Also note that the dependence on \mathbf{d} appears through the feasible region, requiring that the minimization take place over $X(\mathbf{d})$.) So, we can rewrite the objective function as a function of \mathbf{d} alone, by defining $\mathbf{x}(\mathbf{d})$ as the equilibrium link flows in terms of the OD matrix and transforming the objective to $f(\mathbf{d}, \mathbf{x}(\mathbf{d}))$. Then, the partial derivative of the objective with respect to any entry d^{rs} in the OD matrix is given by

$$\frac{\partial f}{\partial d^{rs}} = 2\Theta(d_{rs} - d_{rs}^*) + 2(1 - \Theta) \sum_{(i,j) \in \bar{A}} (x_{rs} - x_{rs}^*) \frac{\partial x_{ij}}{\partial d^{rs}} \quad (7.36)$$

where the partial derivatives $\frac{\partial x_{ij}}{\partial d^{rs}}$ are found from the sensitivity formulas in Section 7.2.1.

The vector of all the derivatives (7.36) forms the gradient of f with respect to \mathbf{d} . So, taking a step in the opposite direction, and ensuring that the values in the OD matrix remain non-negative provide the following update rule:

$$\mathbf{d} \leftarrow [\mathbf{d} - \mu \nabla_{\mathbf{d}} f]^+ \quad (7.37)$$

where μ is a step size to be determined, and the $[\cdot]^+$ operation is applied to each component of the vector. This leads to the following algorithm for OD matrix estimation:

1. Initialize $\mathbf{d} \leftarrow \mathbf{d}^*$.
2. Calculate the link flows $\mathbf{x}(\mathbf{d})$ by solving the traffic assignment problem with OD matrix \mathbf{d} .
3. For each OD pair (r, s) determine $\frac{\partial f}{\partial d^{rs}}$ by solving the sensitivity problem in Section 7.2.1 and using (7.36).
4. Update \mathbf{d} using (7.37) for a suitable step size μ .
5. Test for convergence, and return to step 2 if not converged.

The comments about the step size μ and convergence criteria from the network design problem (Section 7.3) apply equally as well here: μ can be determined using an iterative line search procedure, choosing smaller values until the objective function decreases, and the algorithm can be terminated when it fails to make additional substantial progress in reducing the objective.

This procedure is demonstrated using the network in Figure 7.12. In this network, traffic counts are available on three of the links in the network, and

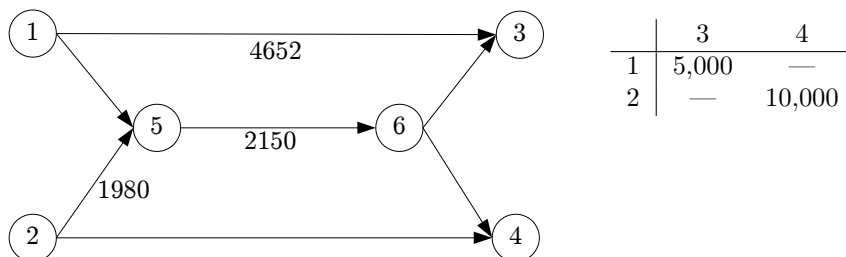


Figure 7.12: Example network for OD matrix estimation. (Observed link volumes and initial OD matrix shown.)

Table 7.2: Solutions to OD matrix estimation sensitivity problems.

Link	Sensitivity problem	
	d_{13}	d_{24}
(1,3)	0.733	0.067
(1,5)	0.267	-0.067
(2,4)	0.067	0.733
(2,5)	-0.067	0.267
(5,6)	0.200	0.200
(6,3)	0.267	-0.067
(6,4)	-0.067	0.267

an initial OD matrix is available based on a travel demand model. The link performance function on every link is $t_{ij} = 10 + x_{ij}/100$, and Θ is given as $\frac{1}{10}$.

We begin by initializing the OD matrix to the initial matrix \mathbf{d}^* , and solving a traffic assignment problem.

As the reader can verify, the equilibrium link flows on the three links with available counts are $x_{13} = 4733\frac{1}{3}$, $x_{25} = 1933\frac{1}{3}$, and $x_{56} = 2200$. The first sum in the objective (7.33) gives the fit of the OD matrix to the initial matrix (zero) and the second sum gives the fit of the equilibrium link flows to the observed link flows (11292). When weighted with $\Theta = 0.1$, this gives an initial objective value of 10164. Next, we solve two sensitivity problems, one for d_{13} and one for d_{24} . (If there was reason to believe there were trips between 1 and 4, and between 2 and 3, we would solve sensitivity problems for those OD pairs as well.) These are shown in Figure 7.13. The link performance functions are the same in both problems; the only difference is in the demand. The solutions to these sensitivity problems is shown in Table 7.2.

We now have all the information needed to calculate the gradient of the objective function, using equation (7.36). Substituting the values of x_{ij} , ξ_{ij} , the

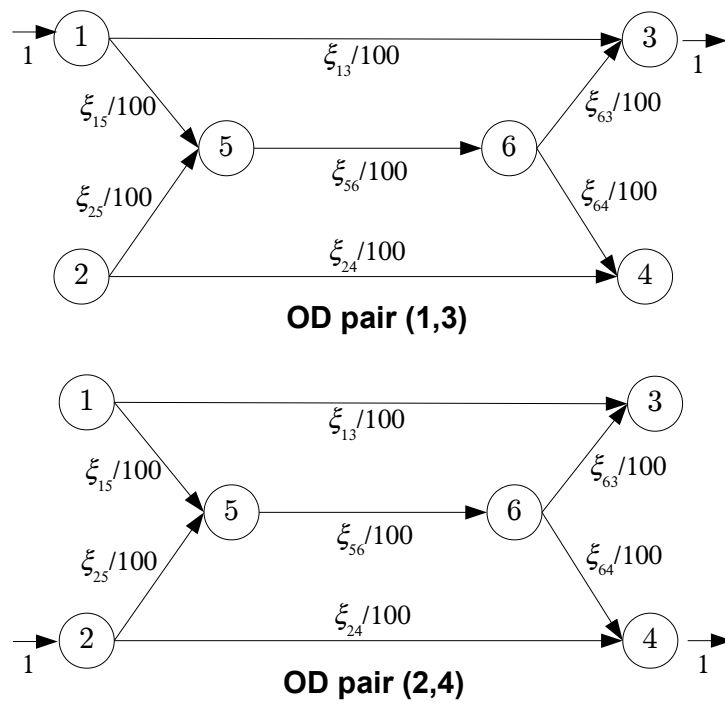


Figure 7.13: Two sensitivity problems for OD matrix estimation.

observed link flows, and the initial OD matrix, we calculate

$$\nabla_{\mathbf{d}} f = \begin{bmatrix} \frac{\partial f}{\partial d_{13}} \\ \frac{\partial f}{\partial d_{24}} \end{bmatrix} = \begin{bmatrix} 131 \\ 5.36 \end{bmatrix} \quad (7.38)$$

Taking a trial step in this direction with $\mu = 1$, the updating rule (7.37) gives the candidate OD matrix $d_{13} = 4869$, $d_{24} = 9995$. The resulting equilibrium link flows include $x_{13} = 4637$, $x_{25} = 1941$, and $x_{56} = 2150$, so the “fit” of the equilibrium link flows and traffic counts has improved from 11293 to 2293. The “fit” of the OD matrix has worsened from 0 to 17179, but with the weight $\Theta = 0.1$, the overall objective still decreases from 10164 to 3782. Therefore, we accept the step size $\mu = 1$, and return to step 3 to continue updating the OD matrix.

7.5 Historical Notes and Further Reading

(These sections are incomplete in this beta version of the text, and will be substantially expanded in the complete first edition.)

The derivation of the equilibrium sensitivity analysis in this chapter follows that in Boyles (2012) and Jafari and Boyles (2016). There are alternative ways to derive the same results, using the implicit function theorem (Tobin and Friesz, 1988; Cho et al., 2000; Yang and Bell, 2007) or results from sensitivity of variational inequalities (Patriksson, 2004; Lu, 2008). In particular, for application of the latter approach to the network design problem, see Josefsson and Patriksson (2007).

7.6 Exercises

1. [63] Given a nondegenerate equilibrium solution in a network with a single origin and continuous link performance functions, show that the equilibrium bushes remain unchanged in a small neighborhood of the current OD matrix.
2. [65] Given a nondegenerate equilibrium solution in a network with a single origin and differentiable link performance functions, show that the derivatives dx_{ij}/dd^{rs} exist at the current equilibrium solution.
3. [56] Extend the results in Exercises 1 and 2 to networks with multiple origins.
4. [84] Show that the entropy-maximizing path flow solution is a continuous function of the OD matrix.
5. [26] Verify that the optimality conditions for (7.11)–(7.13) include the conditions (7.6)–(7.10).

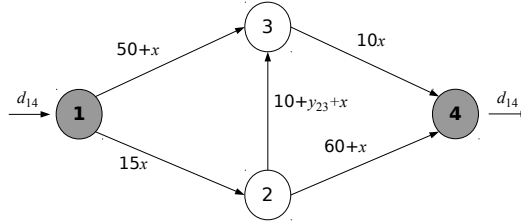


Figure 7.14: Network for use in Exercises 6 and 7.

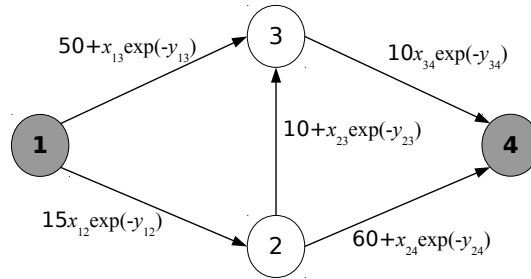


Figure 7.15: Network for use in Exercises 8 and 9.

6. [33] In the modified Braess network of Figure 7.14, find the sensitivity of each link's flow to the demand d^{14} .
7. [35] In the modified Braess network of Figure 7.14, find the sensitivity of each link's flow to the link performance function parameter y^{23} . What value of this parameter minimizes the equilibrium travel time? Suggest what sort of real-world action would correspond to adjusting this parameter to this optimal value.
8. [44] In the network design problem for Figure 7.15, give the gradient of the objective function at the initial solution $\mathbf{y} = \mathbf{0}$. Assume that $\Theta = \frac{1}{20}$.
9. [48] Continue Exercise 8 by performing three iterations of the algorithm given in the text. What is the resulting total system travel time and construction cost?
10. [25] Write out the network design optimization problem for the network in Figure 7.16, with $\Theta = 1$. Show that the feasible region for this problem is not convex by constructing a counterexample.
11. [79] Design a heuristic for the network design problem, based on simulated annealing as discussed in Section C.6.1. Compare the performance of this

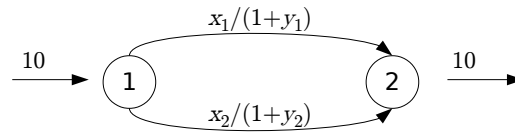


Figure 7.16: Network for use in Exercise 10.

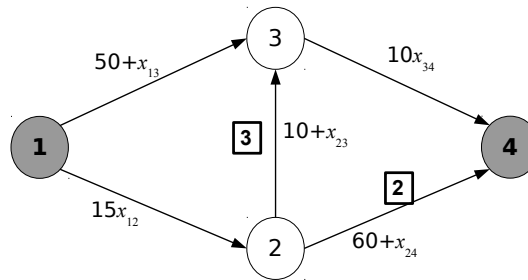


Figure 7.17: Network and observed link flows (in boxes) for Exercise 13.

heuristic with the algorithm given in the text for several networks.

12. [79] Design a heuristic for the network design problem, based on genetic algorithms as discussed in Section C.6.2. Compare the performance of this heuristic with the algorithm given in the text for several networks.
13. [47] In the OD matrix estimation problem of Figure 7.17, give the gradient of the objective function at the initial solution $d_{14}^* = 6$, $d_{24}^* = 4$. What is the value of the objective function if $\Theta = \frac{1}{2}$?
14. [49] Continue Exercise 13 by performing three iterations of the algorithm given in the text. What is the resulting OD matrix and objective function value?
15. [59] Generate five additional OD matrices corresponding to the network in Figure 7.17, with different values of Θ . Which of these OD matrices seems most reasonable to you, and why?
16. [79] Design a heuristic for the OD matrix estimation problem, based on simulated annealing as discussed in Section C.6.1. Compare the performance of this heuristic with the algorithm given in the text for several networks.
17. [79] Design a heuristic for the OD matrix estimation problem, based on genetic algorithms as discussed in Section C.6.2. Compare the perfor-

mance of this heuristic with the algorithm given in the text for several networks.

18. [76] Perform the following “validation” exercise: create a small network with a given OD matrix, and find the equilibrium solution. Then, given the equilibrium link flows, try to compute your original OD matrix using the algorithm given in the text. Do you get your original OD matrix back?

Chapter 8

Extensions of Static Assignment

The basic traffic assignment problem (TAP) was defined in Chapter 5 as follows: we are given a network $G = (N, A)$, link performance functions $t_{ij}(x_{ij})$, and the demand values d^{rs} between each origin and destination. The objective is to find a feasible vector of path flows (or link flows) which satisfy the principle of user equilibrium, that is, that every path with positive flow has the least travel time among all paths connecting that origin and destination. We formulated this as a VI (find $\hat{\mathbf{h}} \in H$ such that $\mathbf{c}(\hat{\mathbf{h}}) \cdot (\hat{\mathbf{h}} - \mathbf{h}) \leq 0$ for all $\mathbf{h} \in H$) and as the solution to the following convex optimization problem:

$$\min_{\mathbf{x}, \mathbf{h}} \sum_{(i,j) \in A} \int_0^{x_{ij}} t_{ij}(x) dx \quad (8.1)$$

$$\text{s.t. } x_{ij} = \sum_{\pi \in \Pi} h^\pi \delta_{ij}^\pi \quad \forall (i, j) \in A \quad (8.2)$$

$$\sum_{\pi \in \Pi^{rs}} h^\pi = d^{rs} \quad \forall (r, s) \in Z^2 \quad (8.3)$$

$$h^\pi \geq 0 \quad \forall \pi \in \Pi \quad (8.4)$$

This formulation remains the most commonly used version of traffic assignment in practice today. However, it is not difficult to see how some of the assumptions may not be reasonable. This chapter shows extensions of the basic TAP which relax these assumptions. This is the typical course of research: the first models developed make a number of simplifying assumptions, in order to capture the basic underlying behavior. Then, once the basic behavior is understood, researchers develop progressively more sophisticated and realistic models which relax these assumptions.

This chapter details three such extensions. Section 8.1 relaxes the assumption that the OD matrix is known and fixed, leading to an *elastic demand* formulation. Section 8.2 relaxes the assumption that the travel time on a link

depends only on the flow on that link (and not on any other link flows, even at intersections). Section 8.3 relaxes the assumption that travelers have accurate knowledge and perception of all travel times in a network, leading to the important class of *stochastic user equilibrium* models.

For simplicity, all of these variations are treated independently of each other. That is, the OD matrix is assumed known and fixed in all sections *except* Section 8.1, and so forth. This is done primarily to keep the focus on the relevant concept of each section, but also to guard the reader against the temptation to assume that a model which relaxes *all* of these assumptions simultaneously is necessarily better than one which does not. While realism is an important characteristic of a model, it is not the only relevant factor when choosing a mathematical model to describe an engineering problem. Other important concerns are computation speed, the existence of enough high-quality data to calibrate and validate the model, transparency, making sure the sensitivity of the model is appropriate to the level of error in input data, ease of explanation to decision makers, and so on. All of these factors should be taken into account when choosing a model, and you can actually do worse off by choosing a more “realistic” model when you don’t have adequate data for calibration — the result may even give the impression of “false precision” when in reality your conclusions cannot be justified.

8.1 Elastic Demand

The assumption that the OD matrix is known and fixed can strain credibility, particularly when considering long time horizons (20–30 years) or when projects are major enough to influence travel decisions at all levels, not just route choice. For instance, consider the “induced demand” phenomenon where major expansion of roadway capacity ends up increasing the amount of demand. This is partly due to changes in route choice (which the basic TAP accounts for), but is also due to changes in other kinds of travel choices, such as departure time, mode, destination, or trip frequency.

Therefore, it is desirable to develop a model which can relax the assumption of an exogenous OD matrix known *a priori*. This section describes how TAP can be extended to accommodate this relaxation. This is called the *elastic demand* formulation of TAP. The elastic demand model is at once more and less useful than basic TAP: more useful because it can provide a more accurate view of the impacts of transportation projects; less useful because it is harder to calibrate.

8.1.1 Demand functions

The new idea in the traffic assignment problem with elastic demand is the *demand function*, which relates the demand for travel between an origin and destination to the travel time between these zones. Specifically, let D^{rs} be a function relating the demand between r and s to the travel time κ^{rs} on the

shortest path between these zones.¹ Generally, D^{rs} is a nonincreasing function — as the travel time between r and s increases, the demand for travel between these nodes is lower. It will also be highly useful to assume that D^{rs} is invertible as well, which will require it to be strictly decreasing. The inverse demand function will give the travel time between r and s corresponding to a given demand level.

As an example, let $D^{rs}(\kappa^{rs}) = 200 \exp(-\kappa^{rs}/100)$. If the travel time between r and s is $\kappa^{rs} = 10$ by the shortest path, then the demand is $d^{rs} = 181$. If the travel time was 20 minutes between these zones, the demand will be 163.7, which is lower as fewer drivers choose to travel between r and s . The inverse demand function is $D_{rs}^{-1}(d^{rs}) = 100 \log(200/d^{rs})$, and can be used to calculate the κ^{rs} value corresponding to a given d^{rs} : when the demand is 181, the shortest path travel time is $\kappa^{rs} = D_{rs}^{-1}(181) = 10$, and so on.² Or, if $D^{rs} = 50 - \frac{1}{2}\kappa^{rs}$, then $D_{rs}^{-1} = 100 - 2d^{rs}$ and a travel time of 10 minutes corresponds to a demand of 45 vehicles as can be seen by substituting either number into the corresponding equation.

An attentive reader may have noticed a potential issue with the demand function $D^{rs} = 50 - \frac{1}{2}\kappa^{rs}$, namely that the demand would be negative if $\kappa^{rs} > 100$. In reality, the demand would simply equal zero if the travel time exceeded 100. We could patch this by redefining D^{rs} as $[50 - \frac{1}{2}\kappa^{rs}]^+$, but then D^{rs} is no longer invertible. Instead, we can allow D^{rs} to take negative values, but replace the relation $d^{rs} = D^{rs}(\kappa^{rs})$ with $d^{rs} = [D^{rs}(\kappa^{rs})]^+$. This allows us to have D^{rs} be strictly decreasing (and thus invertible), but still allow the travel demand to be zero when costs are sufficiently high. While this “trick” may seem a bit trivial (or at least not very useful), it will eventually allow us to formulate the elastic demand equilibrium problem as a variational inequality and convex program, as shown below.

The demand function can be used to define a *consumer surplus* CS representing the benefits of mobility in a region, defined by

$$CS = \sum_{(r,s) \in Z^2} \left(\int_0^{d^{rs}} D_{rs}^{-1}(y) dy - d^{rs} \kappa^{rs} \right) \quad (8.5)$$

(We are using y for the dummy variable of integration instead of d because $\int D^{-1}(d) dd$ is notationally awkward.) The interpretation of this formula is as follows. Each driver has a certain travel time threshold: if the travel time is greater than this threshold, the trip will not be made, and if the travel time is less than this threshold, the trip will be made. Different drivers have different

¹An alternative is to have D^{rs} be a function of the *average* travel time on the used paths between r and s , not the shortest. At equilibrium it doesn't matter because all used paths have the same travel time as the shortest, but in the process of finding an equilibrium the alternative definition of the demand function can be helpful. For our purposes, though, definition in terms of the shortest path time is more useful because it facilitates a link-based formulation.

²In the text, we typically indicate OD pairs with a superscript, as in d^{rs} , and link variables with a subscript, as in x_{ij} . In elastic demand, we will often need to refer to inverse functions, and writing $(D^{rs})^{-1}$ is clumsy. For this reason, OD pairs may also be denoted with a subscript, as in D_{rs}^{-1} . This is purely for notational convenience and carries no significance.

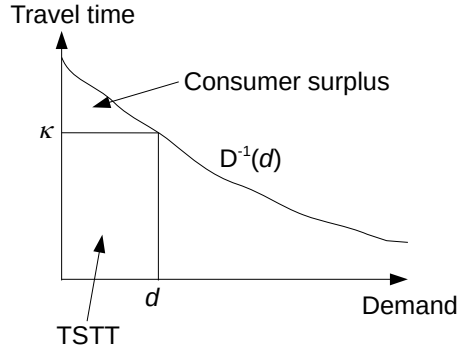


Figure 8.1: Relationship between inverse demand function, consumer surplus, and total system travel time.

thresholds, and the demand function represents the aggregation of these thresholds: when the travel time is κ , $D(\kappa)$ represents the number of travelers whose threshold is κ or higher. If my threshold is, say, 15 minutes and the travel time is 10 minutes, the difference (5 minutes) can be thought of as the “benefit” of travel to me: the trip is worth 15 minutes of my time, but I was able to travel for only 10. Adding this up for all travelers provides the total benefits of travel, which is what CS represents. Figure 8.1 shows the connection between this concept and equation (8.5): assume that drivers are numbered in decreasing order of their threshold values. Then $D^{-1}(1)$ gives the threshold value for the first driver, $D^{-1}(2)$ gives the threshold value for the second driver, and so forth. At equilibrium all drivers experience a travel time of κ , so the benefit to the first driver is $D^{-1}(1) - \kappa$, the benefit to the second driver is $D^{-1}(2) - \kappa$, and so forth. Adding over all drivers gives equation (8.5).

8.1.2 Gartner’s transformation

Before moving to variational inequality and optimization formulations of the elastic demand problem, we’ll take a short digression and show how the elastic demand problem can be cleverly transformed into a traditional equilibrium problem with fixed demand. This transformation works if each demand function D^{rs} is bounded above. Repeat the following for each OD pair (r, s) . Let \bar{d}_{rs} be such an upper bound for OD pair (r, s) . Create a new link (r, s) directly connecting origin r to destination s , and make its link performance function $t_{rs}(x_{rs}) = D_{rs}^{-1}(\bar{d}_{rs} - x_{rs})$ where x_{rs} is the flow on that new link. If D_{rs} is decreasing, then $D_{rs}^{-1}(\bar{d}_{rs} - x_{rs})$ is increasing in x_{rs} so this is a valid link performance function.

Now, solve a *fixed* demand problem where the demand from each origin to each destination is \bar{d}_{rs} . An equilibrium on this network corresponds to an elastic

demand equilibrium on the original network as follows: the flows on the links common to both networks represent flows on the actual traffic network; the flow on the new direct connection links represent drivers who choose not to travel due to excess congestion. Think of \bar{d}_{rs} as the total number of people who might possibly travel from r to s ; those that actually complete their trips travel on the original links and those who choose not to travel choose the direct connection link. At equilibrium, all used paths connecting r to s (including the direct connection link) have the same travel time κ_{rs} ; therefore, the flow on the direct connection link x_{rs} must be such that $D_{rs}^{-1}(\bar{d}_{rs} - x_{rs}) = \kappa_{rs}$, or equivalently $x_{rs} = \bar{d}_{rs} - D_{rs}(\kappa_{rs})$, which is exactly the number of drivers who choose not to travel when the equilibrium times are κ_{rs} . That is, the demand $d^{rs} = \bar{d}_{rs} - x_{rs}$.

The downside of this approach is that it requires creating a large number of new links. In a typical transportation network, the number of links is proportional to the number of nodes and of the same order of magnitude (so a network of 1,000 nodes may have 3–4,000 links). However, the number of OD pairs is roughly proportional to the *square* of the number of nodes, since every node could potentially be both an origin and a destination. So, a network with 1,000 nodes could have roughly 1,000,000 OD pairs. Implementing the Gartner transformation requires creating a new link for every one of these OD pairs, which would result in 99.9% of the network links being the artificial arcs for the transformation!

8.1.3 Variational inequality formulation

Because the elastic demand problem can be expressed as a version of the regular traffic assignment problem through the Gartner transformation, we immediately have a variational inequality formulation of the elastic demand equilibrium problem. Partition the vectors of link flows and travel times into regular and direct-connect (Gartner transformation) links, using \mathbf{x} to represent regular link flows, \mathbf{x}^{\rightarrow} flows on direct-connect links, and \mathbf{t} and \mathbf{t}^{\rightarrow} similarly. Then the variational inequality is

$$\begin{bmatrix} \mathbf{t}(\bar{\mathbf{x}}) \\ \mathbf{t}^{\rightarrow}(\bar{\mathbf{x}}^{\rightarrow}) \end{bmatrix} \cdot \begin{bmatrix} \bar{\mathbf{x}} - \mathbf{x} \\ \bar{\mathbf{x}}^{\rightarrow} - \mathbf{x}^{\rightarrow} \end{bmatrix} \leq 0 \quad (8.6)$$

Since $x_{rs} = \bar{d}_{rs} - d_{rs}$ and $t_{rs}(x_{rs}) = D_{rs}^{-1}(\bar{d}_{rs} - x_{rs})$, the variational inequality can be written in terms of the link flows and OD demands as

$$\begin{bmatrix} \mathbf{t}(\bar{\mathbf{x}}) \\ -\mathbf{D}^{-1}(\bar{\mathbf{d}}) \end{bmatrix} \cdot \begin{bmatrix} \bar{\mathbf{x}} - \mathbf{x} \\ \bar{\mathbf{d}} - \mathbf{d} \end{bmatrix} \leq 0 \quad (8.7)$$

which is the customary form.

8.1.4 Optimization formulation

The Gartner transformation can also lead directly to a convex programming formulation of the elastic demand problem, in a similar way as the variational inequality was derived in the previous section. However, it is also instructive to derive the convex programming formulation from first principles.

As discussed above, the demand is related to the demand function by $d^{rs} = [D^{rs}(\kappa^{rs})]^+$. Put another way, *the demand must always be at least as much as the demand function; further, if the demand is greater than zero then it must equal the demand function*. Thinking laterally, you might notice this is similar to the principle of user equilibrium: the travel time on any path must always be at least as large as the shortest path travel time; further, if the demand is positive then the path travel time must equal the shortest path travel time. When deriving the Beckmann function, we showed that the latter statements could be expressed by $c^\pi \geq \kappa^{rs}$ and $h^\pi(c^\pi - \kappa^{rs}) = 0$ (together with the nonnegativity condition $h^\pi \geq 0$). The same “trick” applies for the relationship between demand and the demand function: $d^{rs} \geq D^{rs}(\kappa^{rs})$, $d^{rs}(d^{rs} - D^{rs}(\kappa^{rs})) = 0$, and the nonnegativity condition $d^{rs} \geq 0$.

It will turn out to be easier to express the latter conditions in terms of the inverse demand functions D^{-1} , rather than the “forward” functions D , because the convex objective function we will derive will be based on the Beckmann function. The Beckmann function involves link performance functions (with units of time). Since the inverse demand functions also are measured in units of time, it will be easier to combine them with the link performance functions than the regular demand functions (which have units of vehicles). Expressed in terms of the inverse demand functions, the conditions above become $\kappa^{rs} \geq D_{rs}^{-1}(d^{rs})$, $d^{rs}(D_{rs}^{-1}(d^{rs}) - \kappa^{rs}) = 0$, and $d^{rs} \geq 0$.

So, this is the question before us. What optimization problem has the following as its optimality conditions?

$$c^\pi \geq \kappa^{rs} \quad \forall (r, s) \in Z^2, \pi \in \Pi^{rs} \quad (8.8)$$

$$h^\pi(c^\pi - \kappa^{rs}) = 0 \quad \forall (r, s) \in Z^2, \pi \in \Pi^{rs} \quad (8.9)$$

$$\kappa^{rs} \geq D_{rs}^{-1}(d^{rs}) \quad \forall (r, s) \in Z^2 \quad (8.10)$$

$$d^{rs}(D_{rs}^{-1}(d^{rs}) - \kappa^{rs}) = 0 \quad \forall (r, s) \in Z^2 \quad (8.11)$$

$$\sum_{\pi \in \Pi^{rs}} h^\pi = d^{rs} \quad \forall (r, s) \in Z^2 \quad (8.12)$$

$$h^\pi \geq 0 \quad \forall \pi \in \Pi \quad (8.13)$$

$$d^{rs} \geq 0 \quad \forall (r, s) \in Z^2 \quad (8.14)$$

The Beckmann formulation is a good place to start, since it already includes (8.8), (8.9), (8.12), and (8.13). So let’s start by conjecturing that the Lagrangian takes the form

$$\mathcal{L}(\mathbf{h}, \mathbf{d}, \boldsymbol{\kappa}) = \sum_{(i,j) \in A} \int_0^{\sum_{\pi \in \Pi} \delta_{ij}^\pi h^\pi} t_{ij}(x) dx + \sum_{(r,s) \in Z^2} \kappa^{rs} \left(d^{rs} - \sum_{\pi \in \Pi^{rs}} h^\pi \right) + F(\mathbf{d}) \quad (8.15)$$

where $F(\mathbf{d})$ is some function involving the OD matrix. (Note also that \mathcal{L} is now a function of \mathbf{d} in addition to \mathbf{h} and $\boldsymbol{\kappa}$, since the demand is a decision variable.) You can check that the optimality conditions related to $\boldsymbol{\kappa}$ and \mathbf{h} are already included in the list of optimality conditions above. Assuming that there

is a nonnegativity constraint on the demand, (8.14) follows immediately as well. What's left is to show that the conditions $\partial\mathcal{L}/\partial d^{rs} \geq 0$ and $d^{rs}(\partial\mathcal{L}/\partial d^{rs}) = 0$ correspond to (8.10) and (8.11).

Calculating from (8.15), we have

$$\frac{\partial\mathcal{L}}{\partial d^{rs}} = \kappa^{rs} + \frac{\partial F}{\partial d^{rs}},$$

so if $\frac{\partial F}{\partial d^{rs}} = -D_{rs}^{-1}(d^{rs})$, we are done (both equations will be true). Integrating, $F(\mathbf{d}) = -\sum_{(r,s) \in Z^2} \int_0^{d^{rs}} D_{rs}^{-1}(\omega) d\omega$ gives us what we need. De-Lagrangianizing the “no vehicle left behind” constraint, we obtain the optimization problem associated with the elastic demand problem:

$$\min_{\mathbf{x}, \mathbf{h}, \mathbf{d}} \sum_{(i,j) \in A} \int_0^{x_{ij}} t_{ij}(x) dx - \sum_{(r,s) \in Z^2} \int_0^{d^{rs}} D_{rs}^{-1}(\omega) d\omega \quad (8.16)$$

$$\text{s.t. } x_{ij} = \sum_{\pi \in \Pi} h^\pi \delta_{ij}^\pi \quad \forall (i,j) \in A \quad (8.17)$$

$$\sum_{\pi \in \Pi^{rs}} h^\pi = d^{rs} \quad \forall (r,s) \in Z^2 \quad (8.18)$$

$$h^\pi \geq 0 \quad \forall \pi \in \Pi \quad (8.19)$$

$$d^{rs} \geq 0 \quad \forall (r,s) \in Z^2 \quad (8.20)$$

8.1.5 Solution method

This section shows how the Frank-Wolfe algorithm can be used to solve the optimization problem (8.16)–(8.20). This is certainly not the only choice, and it is worthwhile for you to think about how other algorithms from Chapter 6 could also be used instead of Frank-Wolfe. The implementation of this algorithm is quite similar to how Frank-Wolfe works for the basic traffic assignment problem, with three changes. First, since the OD matrix is a decision variable along with the link flows, we must keep track of both \mathbf{d} as well as \mathbf{x} ; therefore, in addition to the target link flows \mathbf{x}^* we will have a target OD matrix \mathbf{d}^* , and in addition when we update the link flows we must update the OD matrix as well. Each of these pairs of flows and OD matrices should be consistent with each other, in that the link flows \mathbf{x} must be a feasible network loading when the demand is \mathbf{d} (both before and after updating), and similarly \mathbf{x}^* must correspond to \mathbf{d}^* . Luckily, this will not be difficult.

The second change to the Frank-Wolfe algorithm is how the restricted variational inequality is solved. Instead of solving $\mathbf{t}(\bar{\mathbf{x}}') \cdot (\mathbf{x}^* - \mathbf{x}) \leq 0$ for $\bar{\mathbf{x}}' = \lambda \mathbf{x}^* + (1 - \lambda)\mathbf{x}$, $\lambda \in [0, 1]$, we must solve the variational inequality (8.7) in $\bar{\mathbf{x}}'$ and $\bar{\mathbf{d}}'$. As before, the usual solution involves an “interior” $\lambda \in (0, 1)$, in which

case we must solve the equation

$$\sum_{(i,j) \in A} t_{ij}(\lambda x_{ij}^* + (1 - \lambda)x_{ij})(x_{ij}^* - x_{ij}) - \sum_{(r,s) \in Z^2} D_{rs}^{-1}(\lambda d_{rs}^* + (1 - \lambda)d_{rs})(d_{rs}^* - d_{rs}) = 0 \quad (8.21)$$

in λ . This is simply the variational inequality (8.7) written out in terms of its components, substituting $\lambda \mathbf{x}^* + (1 - \lambda)\mathbf{x}$ for $\bar{\mathbf{x}}'$ and $\lambda \mathbf{d}^* + (1 - \lambda)\mathbf{d}$ for $\bar{\mathbf{d}}'$.

Third, the stopping criterion (relative gap or average excess cost) needs to be augmented with a measure of how well the OD matrix matches the values from the demand functions. A simple measure is the *total misplaced flow* defined as $TMF = \sum_{(r,s) \in Z^2} |d^{rs} - [D^{rs}(\kappa^{rs})]^+|$. The total misplaced flow is always nonnegative, and is zero only if all of the entries in the OD matrix are equal to the values given by the demand function (or zero if the demand function is negative). We should keep track of both total misplaced flow and one of the equilibrium convergence measures (relative gap or average excess cost), and only terminate the algorithm when both of these are sufficiently small.

Implementing these changes, the Frank-Wolfe algorithm for elastic demand is as follows:

1. Choose some initial OD matrix \mathbf{d} and initial link flows \mathbf{x} corresponding to that OD matrix.
2. Find the shortest path between each origin and destination, and calculate convergence measures (total misplaced flow, and either relative gap or average excess cost). If both are sufficiently small, stop.
3. Improve the solution:
 - (a) Calculate a target OD matrix \mathbf{d}^* using the demand functions: $d_{rs}^* = [D^{rs}(\kappa^{rs})]^+$ for all OD pairs (r, s) .
 - (b) Using the target matrix \mathbf{d}^* , find the link flows if everybody were traveling on the shortest paths found in step 2, store these in \mathbf{x}^* .
 - (c) Solve the restricted variational inequality by finding λ such that (8.21) is true.
 - (d) Update the OD matrix and link flows: replace \mathbf{x} with $\lambda \mathbf{x}^* + (1 - \lambda)\mathbf{x}$ and replace \mathbf{d} with $\lambda \mathbf{d}^* + (1 - \lambda)\mathbf{d}$.
4. Return to step 1.

This algorithm can also be linked to the convex programming formulation described above. Given a current solution (\mathbf{x}, \mathbf{d}) , it can be shown that the derivative of the Beckmann function in the direction towards $(\mathbf{x}^*, \mathbf{d}^*)$ is nonpositive (and strictly negative if the current solution does not solve the elastic demand problem), and that the solution of the restricted variational inequality (8.21) minimizes the objective function along the line joining (\mathbf{x}, \mathbf{d}) to $(\mathbf{x}^*, \mathbf{d}^*)$. The algebra is a bit tedious and is left as an exercise at the end of the chapter.

8.1.6 Example

Here we solve the small example of Figure 8.2 with the Frank-Wolfe algorithm, using the average excess cost to measure how close we are to equilibrium. The demand function is $D(\kappa) = 50 - \kappa$, so its inverse function is $D^{-1}(d) = 50 - d$.

Initialization. Arbitrarily set $d = 50$, then arbitrarily load all 50 vehicles onto the two links; say $\mathbf{x} = [50 \ 0]$.

Iteration 1. The link travel times are now $\mathbf{t} = [60 \ 20]$, so the shortest path travel time $\kappa = 20$ and the demand function indicates that the demand should be $D = 50 - 20 = 30$. The average excess cost is $(60 \times 50 - 20 \times 50)/50 = 40$, and the total misplaced flow is $|50 - 30| = 20$. The target demand is what the demand function indicates $d^* = 30$, and this flow should all be loaded on the bottom path, so $\mathbf{x}^* = [0 \ 30]$. Solving the equation

$$(10 + 50(1 - \lambda))(-50) + (20 + 30\lambda)(30) - (50 - (30\lambda + 50(1 - \lambda)))(-20) = 0$$

we obtain $\lambda = 12/19 \approx 0.632$ so the new demand and flows are $d = 37.36$ and $\mathbf{x} = [18.42 \ 18.95]$.

Iteration 2. The link travel times are now $\mathbf{t} = [28.42 \ 38.95]$, so $\kappa = 28.42$, $D = 21.58$, $AEC = 5.19$, and $TMF = 15.78$. Both convergence measures have decreased from the first iteration, particularly the average excess cost. Thus, $d^* = 21.58$, $\mathbf{x}^* = [21.58 \ 0]$, and we solve

$$(10 + 21.58\lambda + 18.42(1 - \lambda))(21.58 - 18.42) + (20 + 18.95(1 - \lambda))(0 - 18.95) - (50 - (21.58\lambda + 37.36(1 - \lambda)))(21.58 - 37.36) = 0$$

so $\lambda = 0.726$ and the new demand and flows are $d = 25.9$ and $\mathbf{x} = [20.71 \ 5.19]$

Iteration 3 . The link travel times are now $\mathbf{t} = [30.71 \ 25.19]$, so $\kappa = 25.19$, $D = 24.81$, $AEC = 4.42$, and $TMF = 1.80$. Assuming that these are small enough to terminate, we are done.

8.2 Link Interactions

This section develops another extension of TAP, in which we relax the assumption that the travel time on a link depends only on the flow on that link. There are a few reasons why this kind of extension may be useful:

Junction interactions: At an intersection, the delay on a particular approach often depends on flows from competing approaches. As an example, consider a freeway onramp which has to yield to mainline traffic at a merge. Because merging traffic must find an acceptably large gap in the main

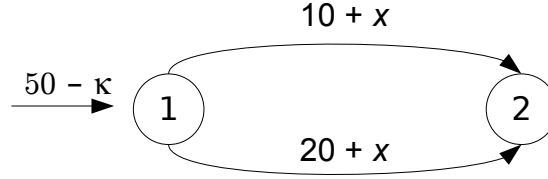


Figure 8.2: Example network for elastic demand equilibrium.

lanes, the travel time on the onramp depends on the flow on the main lanes as well as the flow on the onramp. Similar arguments hold at arterial junctions controlled by two-way or four-way stops, at signalized intersections with permissive phases (e.g., left-turning traffic yielding to gaps in oncoming flow), or at actuated intersections where the green times are determined in real-time based on available flow. The basic TAP cannot model the link interactions characterizing these types of links.

Overtaking traffic: On rural highways, overtaking slow-moving vehicles often requires finding a (fairly large) gap in oncoming flow. If the oncoming flow is small, the effect of slow-moving vehicles on average travel time is negligible. However, as the oncoming flow becomes larger and larger, the ability to overtake is diminished and traffic speeds will tend to be determined by the slowest-moving vehicle on the highway. Since traffic moving in different directions on the same highway is modeled with different links, a link interaction model is needed to capture this effect.

Multiclass flow: Consider a network model where there are two types of vehicles (say, passenger cars and semi trucks, or passenger cars and buses). Presumably these vehicles may choose routes differently or even have a different roadway network available to them (heavy vehicles are prohibited from some streets, and buses must drive along a fixed route). This type of situation can be modeled by creating a “two-layer” network, with the two layers representing the links available to each class. However, where these links represent the same physical roadway, the link performance functions should be connected to each other (truck volume influences passenger car speed and vice versa) even if they are not identical (truck speed need not be the same as passenger car speed). Link interaction models therefore allow us to model multiclass flow as well.

However, there are a few twists to the story, some of which are explored below. Section 8.2.1 presents a mathematical formulation of the link interactions model, but shows that a convex programming formulation is not possible except

in some rather unlikely cases. Section 8.2.2 explores the properties of the link interactions model, in particular addressing the issue of uniqueness — even though link flow solutions to TAP are unique under relatively mild assumptions, this is not generally true when there are link interactions. Section 8.2.3 gives us two solution methods for the link interactions model, the diagonalization method and simplicial decomposition. Diagonalization is easier to implement, but simplicial decomposition is generally more powerful.

8.2.1 Formulation

In the basic TAP, the link performance function for link (i, j) was a function of x_{ij} alone, that is, we could write $t_{ij}(x_{ij})$. Now, t_{ij} may depend on the flow on multiple links. For full generality, our notation will allow t_{ij} to depend on the flows on *any or all* other links in the network: the travel time is given by the function $t_{ij}(x_1, x_2, \dots, x_{ij}, \dots, x_m)$ or, more compactly, $t_{ij}(\mathbf{x})$ using vector notation. We assume these are given to us. Everything else is the same as in vanilla TAP: origin-destination demand is fixed, and we seek an equilibrium solution where all used paths have equal and minimal travel time.

Now, how to formulate the equilibrium principle? It's not hard to see that the variational inequality for TAP works equally well here:

$$\mathbf{c}(\bar{\mathbf{h}}) \cdot (\bar{\mathbf{h}} - \mathbf{h}) \leq 0 \quad \forall \mathbf{h} \in H \quad (8.22)$$

where the only difference is that the link performance functions used to calculate path travel times C are now of the form $t_{ij}(\mathbf{x})$ rather than $t_{ij}(x_{ij})$. But this is of no consequence. Path flows $\bar{\mathbf{h}}$ solve the variational inequality if and only if

$$\mathbf{c}(\bar{\mathbf{h}}) \cdot \bar{\mathbf{h}} \leq \mathbf{c}(\bar{\mathbf{h}}) \cdot \mathbf{h} \quad (8.23)$$

for any other feasible path flows \mathbf{h} . That is, if the travel times were fixed at their current values, then it is impossible to reduce the total system travel time by changing any drivers' route choices. This is only possible if all used paths have equal and minimal travel time. Similarly, the link-flow variational inequality

$$\mathbf{t}(\bar{\mathbf{x}}) \cdot (\bar{\mathbf{x}} - \mathbf{x}) \leq 0, \quad (8.24)$$

where \mathbf{x} is any feasible link flow, also represents the equilibrium problem with link interactions.

The ease of translating the variational inequality formulation for the case of link interactions may give us hope that a convex programming formulation exists as well. The feasible region is the same, all we need is to find an appropriate objective function. Unfortunately, this turns out to be a dead end. For example, the obvious approach is to amend the Beckmann function in some way, for instance, changing

$$\sum_{(i,j) \in A} \int_0^{x_{ij}} t_{ij}(x) dx \quad \text{to} \quad \sum_{(i,j) \in A} \int_{\mathbf{0}}^{\mathbf{x}} t_{ij}(\mathbf{y}) d\mathbf{y} \quad (8.25)$$

where the simple integral in the Beckmann function is replaced with a line integral between the origin $\mathbf{0}$ and the current flows \mathbf{x} . Unfortunately, this line integral is in general not well-defined, since its value depends on the path taken between the origin and \mathbf{x} .

The one exception is if the vector of travel times $\mathbf{t}(\mathbf{x})$ is a gradient map (that is, it is a conservative vector field). In this case, the fundamental theorem of line integrals implies that the value of this integral is *independent* of the path taken between $\mathbf{0}$ and \mathbf{x} . For $\mathbf{t}(\mathbf{x})$ to be a gradient map, its Jacobian must be symmetric. That is, for every pair of links (i, j) and (k, ℓ) , we need the following condition to be true:

$$\frac{\partial t_{ij}}{\partial x_{k\ell}} = \frac{\partial t_{k\ell}}{\partial x_{ij}} \quad (8.26)$$

That is, *regardless of the current flow vector \mathbf{x}* , the marginal impact of another vehicle added to link (i, j) on the travel time of (k, ℓ) must equal the marginal impact of another vehicle added to link (k, ℓ) on the travel time of (i, j) . This condition is very strong. Comparing with the motivating examples used to justify studying link interactions, the symmetry condition is not usually satisfied: the impact of an additional unit of flow on the mainline on the onramp travel time is much greater than the impact of an additional unit of onramp flow on mainline travel time. The impact of semi truck flow on passenger car travel time is probably greater than the impact of passenger car flow on truck travel time at the margin. Symmetry may perhaps hold in the case of overtaking on a rural highway, but even then it is far from clear. So, when modeling link interactions we cannot hope for condition (8.26) to hold. If it does so, consider it a happy accident: the function (8.25) is then an appropriate convex optimization problem.

8.2.2 Properties

This section explores the properties of the link interaction equilibrium problem defined by the variational inequality (8.22). The first question concerns existence of an equilibrium. Because (8.22) is essentially the same variational inequality derived for TAP, the arguments used to derive existence of an equilibrium (based on Brouwer's theorem) carry over directly and we have the same result:

Proposition 8.1. *If each link performance function $t_{ij}(\mathbf{x})$ is continuous in the vector of link flows \mathbf{x} , then at least one solution exists satisfying the principle of user equilibrium.*

However, uniqueness turns out to be trickier. Consider a network of two parallel links where the demand is 6 vehicles, and the link performance functions are $t_1 = x_1 + 2x_2$ and $t_2 = 2x_1 + x_2$. Setting the two links' travel times equal to each other and using $x_1 + x_2 = 6$, it is easy to see that one equilibrium is the solution $x_1 = x_2 = 3$ when the travel time on both links is 9. However, this is not the only equilibrium solution: if all of the drivers were to choose link 1,

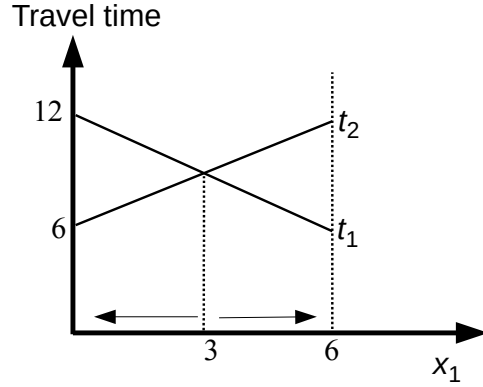


Figure 8.3: Change in path travel times as x_1 varies, “artificial” two-link network.

then $x_1 = 6$, $x_2 = 0$, $t_1 = 6$, and $t_2 = 12$. The top link is the only used path, but it has the least travel time so this solution also satisfies the principle of user equilibrium. Likewise, if $x_1 = 0$ and $x_2 = 6$, then $t_1 = 12$ and $t_2 = 6$ and again the only used path has the least travel time. Therefore, this network has three equilibrium solutions; compare with Figure 8.3.

To make this situation less artificial, we can change the link performance functions to represent a more realistic scenario. Assume that the rate of demand is 1800 vehicles per hour, and that link 1 has a constant travel time of 300 seconds independent of the flow on either link. Link 2 is shorter with a free-flow time of 120 seconds, but must yield to link 1 using gap acceptance principles. In traffic operations, gap acceptance is often modeled with two parameters: the *critical gap* t_c , and the *follow-up gap* t_f . The critical gap is the smallest headway required in the main stream for a vehicle to enter. Given that the gap is large enough for one vehicle to enter the stream, the follow-up gap is the incremental amount of time needed for each additional vehicle to enter. For this example, let t_c be 4 seconds and t_f be 2 seconds. Then, assuming that flows on both links 1 and 2 can be modeled as Poisson arrivals, the travel time on link 2 can be derived as

$$t_2(x_1, x_2) = \frac{1}{u} + \frac{\Lambda}{4} \left[\frac{x_2}{u} - 1 + \sqrt{\left(\frac{x_2}{u} - 1\right)^2 + \frac{8x_2}{u^2\Lambda}} \right] \quad (8.27)$$

where Λ is the length of the analysis period and u is the capacity of link 2 defined by

$$u = \frac{x_2 \exp(-x_1 t_c)}{1 - \exp(-x_1 t_f)} \quad (8.28)$$

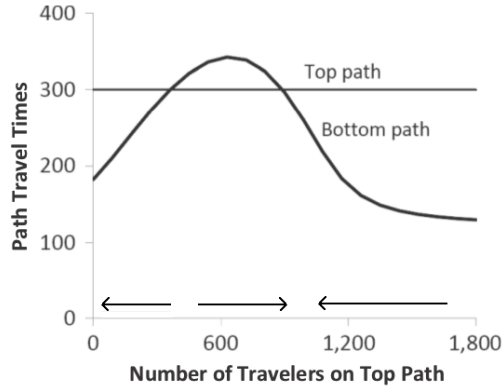


Figure 8.4: Change in path travel times as x_1 varies, “realistic” two-link network.

Figure 8.4 shows the travel times on the two paths as x_1 varies. Again, there are three equilibria: (1) $x_1 = 0$, $x_2 = 1800$, where $t_1 = 300$ and $t_2 = 182$; (2) $x_1 = 362$, $x_2 = 1438$, where $t_1 = t_2 = 300$; and (3) $x_1 = 892$, $x_2 = 908$, where $t_1 = t_2 = 300$.

So, even in realistic examples we cannot expect equilibrium to be unique when there are link interactions. The practical significance is that it raises doubt about which equilibrium solution should be used for project evaluation or ranking. For instance, consider a candidate project which would improve the free-flow time on link 1 from 300 to a smaller value; this would correspond to lowering the horizontal line in Figure 8.4. If we are at one of the equilibria where the travel times are equal, such a change will indeed reduce the travel times experienced by drivers. However, if we are at the equilibrium where the top path is unused, such a change will have no impact whatsoever.

While a complete study of the methods used to distinguish among multiple equilibria is beyond the scope of this section, a simple stability criterion is explained here: an equilibrium solution is *stable* if small perturbations to the solution would incentivize drivers to move back towards that initial equilibrium — that is, if we reassign a few drivers to different paths, the path travel times will change in such a way that those drivers would want to move back to their original paths. By contrast, an *unstable* equilibrium does not have this property: if a few drivers are assigned to different paths, the path travel times will change in such a way that even more drivers would want to switch paths, and so on until another equilibrium is found.

In the simple two-link network we’ve been looking at, stability can be identified using graphs such as those in Figures 8.3 and 8.4. The arrows on the bottom axis indicate the direction of the path switching which would occur for a given value of x_1 . When the arrow is pointing to the left, $t_1 > t_2$ so travelers want to switch away from path 1 to path 2, resulting in a decrease in x_1 (a move

further to the left on the graph). When $t_2 > t_1$, travelers want to switch away from path 2 to path 1, resulting in an increase in x_1 , indicated by an arrow pointing to the right. At the equilibrium solutions, there is no pressure to move in any feasible direction. So, for the first example, the only stable equilibria are the “extreme” solutions with all travelers on either the top or bottom link. The equilibrium with both paths used is unstable in the sense that any shift from one path to another amplifies the difference in travel times and encourages even more travelers to shift in that direction. In the second example, the first and third equilibria are stable, but the second is unstable.

Based on these two examples, an intuitive explanation for the presence of stability with link interactions can be provided. For the regular traffic assignment problem with increasing link performance functions, shifting flow away from a path π and onto another path π' always decreases the travel time on π and increases the travel time on π' . Therefore, if the paths have different travel times, flow will shift in a way that always tends to equalize the travel times on the two paths. Even where there are link interactions, the same will hold true *if the travel time on a path is predominantly determined by the flow on that path*. However, when the link interactions are very strong, the travel time on a path may depend more strongly by the flow on a different path. In the first example, notice that each link’s travel time is influenced more by the *other* link’s flow than its own. In the second example, for certain ranges of flow the travel time on the merge path is influenced more by the flow on the priority path. In such cases, there is no guarantee that moving flow from a higher-cost path to a lower-cost path will tend to equalize their travel times. In the first example, we have an extreme case where moving flow to a path *decreases* its travel time while *increasing* the travel time of the path the flow moved away from!

To make this idea more precise, the following section introduces the mathematical concept of strict monotonicity.

Strict Monotonicity

Let $\mathbf{f}(\mathbf{x})$ be a vector-valued function whose domain and range are vectors of the same dimension. For instance, $\mathbf{t}(\mathbf{x})$ maps the vector of link flows to the vector of link travel times; the dimension of both of these is the number of links in the network. We say that \mathbf{f} is *strictly monotone* if for any two distinct vectors \mathbf{x} and \mathbf{y} in its domain, the dot product of $\mathbf{f}(\mathbf{x}) - \mathbf{f}(\mathbf{y})$ and $\mathbf{x} - \mathbf{y}$ is strictly positive.

For example, let $\mathbf{f}(\mathbf{x})$ be defined by $f_1(x_1, x_2) = 2x_1$ and $f_2(x_1, x_2) = 2x_2$. Then for any distinct vectors \mathbf{x} and \mathbf{y} , we have

$$\begin{aligned} (\mathbf{f}(\mathbf{x}) - \mathbf{f}(\mathbf{y})) \cdot (\mathbf{x} - \mathbf{y}) &= ([2x_1 \quad 2x_2] - [2y_1 \quad 2y_2]) \cdot ([x_1 \quad x_2] - [y_1 \quad y_2]) \\ &= 2((x_1 - y_1)^2 + (x_2 - y_2)^2) \end{aligned}$$

Since $\mathbf{x} \neq \mathbf{y}$, the right-hand side is always greater than zero, so f is monotone. As another example, let the function $\mathbf{g}(\mathbf{x})$ be defined by $g_1(x_1, x_2) = 2x_2$

and $g_2(x_1, x_2) = 2x_1$. If we choose $\mathbf{x} = [0 \ 1]$ and $\mathbf{y} = [1 \ 0]$, then

$$(\mathbf{g}(\mathbf{x}) - \mathbf{g}(\mathbf{y})) \cdot (\mathbf{x} - \mathbf{y}) = ([2 \ -2]) \cdot ([-1 \ 1]) = -4$$

so \mathbf{g} is not strictly monotone. Note that proving strict monotonicity requires a general argument valid for *any* distinct vectors \mathbf{x} and \mathbf{y} ; showing that a function is not strictly monotone only requires a single counterexample.

Warning! It is very common for students to think that the link performance functions are strictly monotone if they are strictly increasing functions of the flow on each link. This is not true: in the first example in this section, all link performance functions are increasing in each flow variable but if we compare $\mathbf{x} = [0 \ 6]$ and $\mathbf{y} = [5 \ 1]$, we have

$$(\mathbf{t}(\mathbf{x}) - \mathbf{t}(\mathbf{y})) \cdot (\mathbf{x} - \mathbf{y}) = ([12 \ 6] - [7 \ 11]) \cdot ([0 \ 6] - [5 \ 1]) = -50$$

so these link performance functions are not strictly monotone. Roughly speaking, strict monotonicity requires the diagonal terms of the Jacobian of \mathbf{t} to be large compared to the off-diagonal terms. The precise version of this “roughly speaking” fact is the following:

Proposition 8.2. *If \mathbf{f} is a continuously differentiable function whose domain is convex, then \mathbf{f} is strictly monotone if and only if its Jacobian is positive definite at all points in the domain.*

With this definition of monotonicity in hand, we can provide the uniqueness result we’ve been searching for:

Proposition 8.3. *Consider an instance of the traffic assignment problem with link interactions. If the link performance functions $\mathbf{t}(\mathbf{x})$ are continuous and strictly monotone, then there is exactly one user equilibrium solution.*

Proof. Since $\mathbf{t}(\mathbf{x})$ is continuous, we are guaranteed existence of at least one equilibrium solution from Brouwer’s theorem; let $\hat{\mathbf{x}}$ be such an equilibrium and let $\tilde{\mathbf{x}}$ be any other feasible link flow solution. We need to show that $\tilde{\mathbf{x}}$ cannot be an equilibrium. Arguing by contradiction, assume that $\tilde{\mathbf{x}}$ is in fact an equilibrium. Then it would solve the variational inequality (8.24), so

$$\mathbf{t}(\tilde{\mathbf{x}}) \cdot (\tilde{\mathbf{x}} - \hat{\mathbf{x}}) \leq 0$$

Adding a clever form of zero to the left hand side, this would imply

$$(\mathbf{t}(\tilde{\mathbf{x}}) - \mathbf{t}(\hat{\mathbf{x}})) \cdot (\tilde{\mathbf{x}} - \hat{\mathbf{x}}) + \mathbf{t}(\hat{\mathbf{x}}) \cdot (\tilde{\mathbf{x}} - \hat{\mathbf{x}}) \leq 0 \quad (8.29)$$

But since the link performance functions are strictly monotone, the first term on the left-hand side is strictly positive. Furthermore, since $\hat{\mathbf{x}}$ is an equilibrium the variational inequality (8.24) is true, so $\mathbf{t}(\hat{\mathbf{x}}) \cdot (\hat{\mathbf{x}} - \tilde{\mathbf{x}}) \leq 0$, which implies that the second term on the right-hand side is nonnegative. Therefore, the left-hand side of (8.29) is strictly positive, which is a contradiction. Therefore $\tilde{\mathbf{x}}$ cannot satisfy the principle of user equilibrium. □

8.2.3 Algorithms

This section presents two algorithms for the traffic assignment problem with link interactions. If the link performance functions are strictly monotone, it can be shown that both of these algorithms converge to the unique equilibrium solution. Otherwise, it is possible that these algorithms may not converge, although they will typically do so if they start sufficiently close to an equilibrium. In any case, these algorithms may be acceptable heuristics even when strict monotonicity does not hold.

Diagonalization

The diagonalization method is a variation of Frank-Wolfe, which differs only in how the step size λ is found. Recall that the Frank-Wolfe step size is found by solving the equation

$$\sum_{(i,j) \in A} t_{ij}(\lambda x_{ij}^* + (1 - \lambda)x_{ij})(x_{ij}^* - x_{ij}) = 0 \quad (8.30)$$

since this minimizes the Beckmann function along the line segment connecting \mathbf{x} to \mathbf{x}^* . Since there is no corresponding objective function when there are asymmetric link interactions, it is not clear that a similar approach will necessarily work. (And in any case, t_{ij} is no longer a function of x_{ij} alone, so the formula as stated will not work.)

To make this formula logical, construct a temporary link performance function $\tilde{t}_{ij}(x_{ij})$ which only depends on its own flow. This is done by assuming that the flow on all other links is constant: $\tilde{t}_{ij}(x_{ij}) = t_{ij}(x_1, \dots, x_{ij}, \dots, x_m)$. For example, if $t_1(x_1, x_2, x_3) = x_1 + x_2^2 + x_3^3$ and the current solution is $x_1 = 1$, $x_2 = 2$, and $x_3 = 3$, then $\tilde{t}_1(x_1) = 31 + x_1$, since this is what we would get if x_2 and x_3 were set to constants at their current values of 2 and 3, respectively.

The step size λ is then found by adapting the Frank-Wolfe formula, using $\tilde{t}_{ij}(x_{ij})$ in place of $t_{ij}(\mathbf{x})$. That is, in the diagonalization method λ solves

$$\sum_{(i,j) \in A} \tilde{t}_{ij}(\lambda x_{ij}^* + (1 - \lambda)x_{ij})(x_{ij}^* - x_{ij}) = 0 \quad (8.31)$$

At each iteration, new \tilde{t} functions are calculated based on the current solution. The complete algorithm is as follows:

1. Find the shortest path between each origin and destination, and calculate the relative gap (unless it is the first iteration). If the relative gap is sufficiently small, stop.
2. Shift travelers onto shortest paths:
 - (a) Find the link flows if everybody were traveling on the shortest paths found in step 1, store these in \mathbf{x}^* .

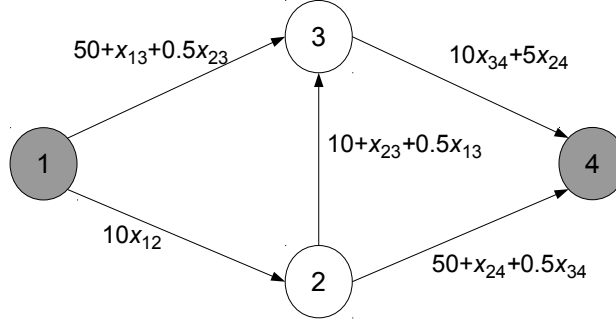


Figure 8.5: Braess network with link interactions.

- (b) If this is the first iteration, set $\mathbf{x} \leftarrow \mathbf{x}^*$ and move to step 3. Otherwise, continue with step c.
 - (c) Using the current solution \mathbf{x} , form the diagonalized link performance functions $\tilde{t}_{ij}(x_{ij})$ for each link.
 - (d) Find λ which solves equation (8.31).
 - (e) Update $\mathbf{x} \leftarrow \lambda \mathbf{x}^* + (1 - \lambda)\mathbf{x}$.
3. Calculate the new link travel times and the relative gap. Increase the iteration counter i by one and return to step 1.

As an example, consider the modified Braess network shown in Figure 8.5. At each merge node, the travel time on each of the incoming links depends on the flow on *both* links which merge together. The link flow vectors are indexed $\mathbf{x} = [x_{12} \ x_{13} \ x_{23} \ x_{24} \ x_{34}]$.

Iteration 1. In the first iteration, load all travelers onto shortest paths, so $\mathbf{x} = \mathbf{x}^* = [6 \ 0 \ 6 \ 0 \ 6]$, $\mathbf{t} = [60 \ 53 \ 16 \ 53 \ 60]$ and the average excess cost is 23.

Iteration 2. The all-or-nothing loading on shortest paths given \mathbf{t} is $\mathbf{x}^* = [0 \ 6 \ 0 \ 0 \ 6]$. The diagonalized link performance functions are obtained by assuming the flows on all other links are constant at \mathbf{x} : $\hat{t}_{12} = 10x_{12}$, $\hat{t}_{13} = 53 + x_{13}$, $\hat{t}_{23} = 10 + x_{23}$, $\hat{t}_{24} = 53 + x_{24}$, and $\hat{t}_{34} = 10x_{34}$. So we solve the equation (8.31) for λ :

$$(53 + 6\lambda)6 + 10(6(1 - \lambda))(-6) + (10 + 6(1 - \lambda))(-6) = 0$$

omitting terms where $x_{ij} = x_{ij}^*$ because they are zero in (8.31). The solution is $\lambda = 23/72$, so we update $\mathbf{x} = [4\frac{1}{12} \ 1\frac{11}{12} \ 4\frac{1}{12} \ 0 \ 6]$ and $\mathbf{t} = [40\frac{5}{6} \ 53\frac{23}{24} \ 15\frac{1}{24} \ 53 \ 60]$ (using the regular cost functions, not the diagonalized ones.) The average excess cost is now 21.43.

Iteration 3. The all-or-nothing assignment is $\mathbf{x}^* = [6 \ 0 \ 0 \ 6 \ 0]$ and the diagonalized link performance functions are $\hat{t}_{12} = 10x_{12}$, $\hat{t}_{13} = 52\frac{1}{24} + x_{13}$, $\hat{t}_{23} = 10\frac{23}{24} + x_{23}$, $\hat{t}_{24} = 53 + x_{24}$, and $\hat{t}_{34} = 10x_{34}$. The solution to (8.31) is $\lambda = 0.284$, which gives $\mathbf{x} = [4.63 \ 1.37 \ 2.92 \ 1.70 \ 4.30]$, $\mathbf{t} = [46.3 \ 52.8 \ 13.6 \ 53.9 \ 51.5]$, so the average excess cost is 6.44.

and so on until convergence is reached.

Simplicial decomposition

An alternative to diagonalization is the simplicial decomposition algorithm. This algorithm is introduced at this point (rather than in Chapter 6) for several reasons. First, historically it was the first provably convergent algorithm for the equilibrium problem with link interactions. Second, although it is an improvement on Frank-Wolfe, for the basic TAP it is outperformed by the path-based and bush-based algorithms presented in that chapter. However, like those algorithms, it overcomes the “zig-zagging” difficulty that Frank-Wolfe runs into (cf. Figure 6.4).

The price of this additional flexibility is that more computer memory is needed. Frank-Wolfe and MSA are exceptionally economical in that they only require two vectors to be stored: the current link flows \mathbf{x} and the target link flows \mathbf{x}^* . In simplicial decomposition, we will “remember” all of the target link flows found in earlier iterations, and exploit this longer-term memory by allowing “combination” moves towards several of these previous targets simultaneously. In the algorithm, the set \mathcal{X} is used to store all target link flows found thus far.

A second notion in simplicial decomposition is that of a “restricted equilibrium.” Given a set $\mathcal{X} = \{\mathbf{x}_1^*, \mathbf{x}_2^*, \dots, \mathbf{x}_k^*\}$ and a current link flow solution \mathbf{x} , we say that \mathbf{x} is a restricted equilibrium if it solves the variational inequality

$$\mathbf{t}(\mathbf{x}) \cdot (\mathbf{x} - \mathbf{x}') \leq 0 \quad \forall \mathbf{x}' \in X(\mathbf{x}, \mathcal{X}) \quad (8.32)$$

where $X(\mathbf{x}, \mathcal{X})$ means the set of link flow vectors which are obtained by a convex combination of \mathbf{x} and any of the target vectors in \mathcal{X} .³ Equivalently, \mathbf{x} is a restricted equilibrium if none of the targets in \mathcal{X} lead to improving directions in the sense that the total system travel time would be reduced by moving to some $\mathbf{x}_i^* \in \mathcal{X}$ while fixing the travel times at their current values. That is,

$$\mathbf{t}(\mathbf{x}) \cdot (\mathbf{x} - \mathbf{x}_i^*) \leq 0 \quad \forall \mathbf{x}_i^* \in \mathcal{X} \quad (8.33)$$

At a high level, simplicial decomposition works by iterating between adding new target vectors to \mathcal{X} , and then finding a restricted equilibrium using the current vectors in \mathcal{X} . In this light, Frank-Wolfe can be seen as a special case of simplicial decomposition, where \mathcal{X} only consists of the current target vector (forgetting any from past iterations), because if there is only one target $\mathbf{x}^* \in \mathcal{X}$

³A flow vector \mathbf{x}' is a convex combination of \mathbf{x} and the target vectors in \mathcal{X} if there exist nonnegative constants $\lambda_0, \lambda_1, \dots, \lambda_k$ such that $\mathbf{x}' = \lambda_0\mathbf{x} + \lambda_1\mathbf{x}_1^* + \dots + \lambda_k\mathbf{x}_k^*$.

then the restricted equilibrium in $X(\mathbf{x}, \mathcal{X})$ is simply the restricted equilibrium along the line segment connecting \mathbf{x} to \mathbf{x}^* .

In practice, it is too expensive to exactly find a restricted equilibrium at each iteration. Instead, several “inner iteration” steps are taken to move towards a restricted equilibrium with the current set \mathcal{X} before looking to add another target. In each inner iteration, the current solution \mathbf{x} is adjusted to $\mathbf{x} + \mu\Delta\mathbf{x}$, where μ is a step size and $\Delta\mathbf{x}$ is a direction which moves toward restricted equilibrium. Smith (1984) shows that one good choice for this direction is

$$\Delta\mathbf{x} = \frac{\sum_{\mathbf{x}_i^* \in \mathcal{X}} [\mathbf{t}(\mathbf{x}) \cdot (\mathbf{x} - \mathbf{x}_i^*)]^+ (\mathbf{x}_i^* - \mathbf{x})}{\sum_{\mathbf{x}_i^* \in \mathcal{X}} [\mathbf{t}(\mathbf{x}) \cdot (\mathbf{x} - \mathbf{x}_i^*)]^+} \quad (8.34)$$

This rather intimidating-looking formula is actually quite simple. It is nothing more than a weighted average of the directions $\mathbf{x}_i^* - \mathbf{x}$ (potential moves toward each target in \mathcal{X}), where the weight for each potential direction is the extent to which it improves upon the current solution: $[\mathbf{t}(\mathbf{x}) \cdot (\mathbf{x} - \mathbf{x}_i^*)]$ is the reduction in total system travel time obtained by moving from \mathbf{x} to \mathbf{x}_i^* while holding travel times constant. If this term is negative, there is no need to move in that direction, so the weight is simply set to zero. The denominator is simply the sum of the weights, which serves as a normalizing factor.

The step size μ is chosen through trial-and-error. One potential strategy is to iteratively test μ values in some sequence (say, $1, 1/2, 1/4, \dots$) until we have found a solution acceptably closer to restricted equilibrium than \mathbf{x} .⁴ “Acceptably closer” can be calculated using the *Smith gap*

$$\gamma_S = \sum_{\mathbf{x}_i^* \in \mathcal{X}} ([\mathbf{t}(\mathbf{x}) \cdot (\mathbf{x} - \mathbf{x}_i^*)]^+)^2 \quad (8.35)$$

which is similar to the gap measures described in Chapter 6 in that it is zero if and only if \mathbf{x} is a restricted equilibrium, and positive otherwise. It reflects a restricted equilibrium because the summation is only over the vectors in \mathcal{X} : if any of them reflect an “improvement” over the current solution, in that total travel time would be reduced (if link travel times were held fixed), the corresponding term in square brackets is positive, and $\gamma_S > 0$. It can be shown that taking a small enough step in the direction (8.34) will reduce the Smith gap, and a line search or trial-and-error method can be used to determine what an acceptable step size is. Squaring the term in brackets ensures that γ_S is differentiable, which plays a role in deriving the convergence rate of this method.

The Smith gap could thus be converted into a stopping criterion for the (unrestricted) traffic assignment problem, by extending the sum to include every possible all-or-nothing assignment, not just the ones in \mathcal{X} , but this idea will not be explored further in this book.

Putting all of this together, the simplicial decomposition algorithm can be stated as:

⁴Those familiar with nonlinear optimization may see parallels between this and the Armijo rule.

1. Initialize the set $\mathcal{X} \leftarrow \emptyset$
2. Find shortest paths for all OD pairs.
3. Form the all-or-nothing assignment \mathbf{x}^* based on shortest paths.
4. If \mathbf{x}^* is already in \mathcal{X} , stop.
5. Add \mathbf{x}^* to \mathcal{X} .
6. **Subproblem:** Find a restricted equilibrium \mathbf{x} using only the vectors in \mathcal{X} .
 - (a) Find the improvement direction $\Delta \mathbf{x}$ using equation (8.34).
 - (b) Update $\mathbf{x} \leftarrow \mathbf{x} + \mu \Delta \mathbf{x}$, with μ sufficiently small (to reduce γ_S).
 - (c) Update travel times.
 - (d) Return to step 1 of subproblem unless γ_S is small enough.
7. Return to step 2.

Below we apply this algorithm to the example in Figure 8.5, choosing μ via trial and error from the sequence $1/2, 1/4, \dots$ and stopping at the first value that reduces γ_S . Other ways of choosing μ are also possible.

Iteration 1. We set

$$\mathbf{x}_1^* = [6 \quad 0 \quad 6 \quad 0 \quad 6]$$

and $\mathcal{X} = \{x_1^*\}$. For the subproblem, the only possible solution is $\mathbf{x} = \mathbf{x}_1^*$, which has $\gamma_S = 0$ (it is trivially a restricted equilibrium) and travel times are

$$\mathbf{t} = [60 \quad 53 \quad 16 \quad 53 \quad 60] .$$

Iteration 2. The new all-or-nothing assignment is

$$\mathbf{x}_2^* = [0 \quad 6 \quad 0 \quad 0 \quad 6] ,$$

and $\mathcal{X} = \{\mathbf{x}_1^*, \mathbf{x}_2^*\}$. For the first iteration of the subproblem, notice that $\mathbf{t} \cdot (\mathbf{x} - \mathbf{x}_1^*) = 0$ and $\mathbf{t} \cdot (\mathbf{x} - \mathbf{x}_2^*) = 138$, so Smith's formula (8.34) reduces to

$$\Delta \mathbf{x} = \frac{0}{138}(\mathbf{x}_1^* - \mathbf{x}) + \frac{138}{138}(\mathbf{x}_2^* - \mathbf{x}) = [-6 \quad 6 \quad -6 \quad 0 \quad 0] .$$

Taking a step of size $\mu = 1/2$ gives us

$$\mathbf{x} = [3 \quad 3 \quad 3 \quad 0 \quad 6] .$$

The new travel times are

$$\mathbf{t} = [30 \quad 54\frac{1}{2} \quad 14\frac{1}{2} \quad 53 \quad 60] ,$$

so $\mathbf{t} \cdot \mathbf{x} = 657$, $\mathbf{t} \cdot \mathbf{x}_1^* = 627$, and $\mathbf{t} \cdot \mathbf{x}_2^* = 687$, so

$$\gamma_S = ([657 - 627]^+)^2 + ([657 - 687]^+)^2 = 30^2 + 0^2 = 900 .$$

Assume this is "small enough" to complete the subproblem.

Iteration 3. The new all-or-nothing assignment is

$$\mathbf{x}_3^* = [6 \quad 0 \quad 0 \quad 6 \quad 0]$$

and $\mathcal{X} = \{\mathbf{x}_1^*, \mathbf{x}_2^*, \mathbf{x}_3^*\}$. For the first iteration of the subproblem, calculate $\mathbf{t} \cdot (\mathbf{x} - \mathbf{x}_1^*) = 30$, $\mathbf{t} \cdot (\mathbf{x} - \mathbf{x}_2^*) = -30$, and $\mathbf{t} \cdot (\mathbf{x} - \mathbf{x}_3^*) = 159$. So (8.34) gives

$$\begin{aligned} \Delta \mathbf{x} &= \frac{30}{189}(\mathbf{x}_1^* - \mathbf{x}) + \frac{0}{189}(\mathbf{x}_2^* - \mathbf{x}) + \frac{159}{189}(\mathbf{x}_3^* - \mathbf{x}) \\ &= [3 \quad -3 \quad -2.05 \quad 5.05 \quad -5.05] . \end{aligned}$$

Taking a step of size $\mu = 1/2$ would give

$$\mathbf{x} = [4.5 \quad 1.5 \quad 1.98 \quad 2.52 \quad 3.48]$$

and

$$\mathbf{t} = [45 \quad 52.5 \quad 12.7 \quad 54.3 \quad 47.4]$$

which has $\gamma_S = 1308$. Assume that this is no longer “small enough” to return to the master problem, so we begin a second subproblem iteration. Smith’s formula (8.34) now gives $\Delta \mathbf{x} = 0.2(\mathbf{x} - \mathbf{x}_2^*) + 0.8(\mathbf{x} - \mathbf{x}_3^*)$, and the trial solution $\mathbf{x} + \frac{1}{2}\Delta \mathbf{x}$ has $\gamma_S = 1195$, which is an improvement. In this case, choosing a smaller μ would work even better; for instance $\mu = 1/4$ would reduce the Smith gap to 557. There is thus a tradeoff between spending more time on finding the “best” value of μ , or spending more time on finding new search directions and vectors for \mathcal{X} . Balancing these is an important question for implementation.

The algorithm can continue from this point or terminate if this average excess cost is small enough.

8.3 Stochastic User Equilibrium

This section describes another extension to the basic TAP. To this point in the text, we have been using the principle of equilibrium to determine link and path flows, requiring all used paths between the same origin and destination to have equal and minimal travel time. We derived this principle by assuming that all drivers choose the least-travel time path between their origin and destination. However, this assumption implicitly requires *drivers to have perfect knowledge of the travel times on all routes in the network*. In reality, we know this is not true: do you know the travel times on literally *all* routes between an origin and destination? And can you accurately distinguish between a route with a travel time of 16 minutes, and one with a travel time of 15 minutes and 59 seconds? Relaxing these assumptions leads us to the *stochastic user equilibrium* (SUE) model.

In SUE, rather than requiring that each driver follow the true shortest path between each origin and destination, we assume that drivers follow the path

they *believe* to be shortest, but allow for some perception error between their belief and the actual travel times. An alternative, mathematically equivalent, interpretation (explained below) is that drivers do in fact perceive travel times accurately, but care about factors other than travel time. This section explains the development of the SUE model. The mathematical foundation for the SUE model is in discrete choice concepts, which are briefly reviewed in Section 8.3.1. The specific application of discrete choice to the route choice decision is taken up in Section 8.3.2.

These sections address the “individual” perspective of logit route choice. The next steps to creating the SUE model are an efficient network loading model (a way to find the choices of *all* drivers efficiently), and then finally the equilibrium model which combines the network loading with updates to travel times, to account for the mutual dependence between travel times and route choices. These are undertaken in Sections 8.3.4 and 8.3.5, respectively. For the most part, this discussion assumes a relatively simplistic model for perception errors in travel times; Section 8.3.7 briefly discusses how more general situations can be handled.

8.3.1 Discrete choice modeling

This section provides a brief overview of discrete choice concepts. The application to route choice is in the following section. Discrete choice is a large area of scholarly inquiry in and of itself, and so the discussion is restricted to what is needed in the chapter.

Consider an individual who must make a choice from a set of options. For instance, when purchasing groceries, you must choose one store from a set of alternatives (the grocery stores in your city). When dressing in the morning, you must choose one set of clothes among all of the clothing you own. And, more relevant to transportation, when choosing to drive from one point to another, you must choose one route among all of the possible routes connecting your origin to your destination.

Mathematically, let I be a finite set of alternatives. Each alternative $i \in I$ is associated with a *utility* value U_i representing the amount of happiness or satisfaction you would have if you were to choose option i . We assume that you would choose an alternative $i^* \in \arg \max_i \{U_i\}$ which maximizes the utility you receive. Now, the utility U_i consists of two parts: an *observable utility* V_i , and an *unobservable utility* denoted by the random variable ϵ :

$$U_i = V_i + \epsilon_i \tag{8.36}$$

The difference between observable and unobservable utility can be explained in different ways. One interpretation is that V_i represents the portion of the utility that is due to objective factors visible to the modeler; when choosing a grocery store, that might include the distance from your home, the price, the variety of items stocked, etc. The unobserved utility consists of subjective factors that the modeler cannot see (or chooses not to include in the model),

even though they are real insofar as they affect your choice. In the grocery store example, this might include your opinion on the taste of the store brands, the cleanliness of the store, and so on. Then, by modeling the unobserved utility as a random variable ϵ , we can express choices in terms of probabilities. (The modeler does not know all of the factors affecting your choice, so they can only speak of probabilities of choosing different options based on what is observable.)

A second interpretation is that the observed utility V_i actually represents all of the factors that you care about. However, for various reasons you are incapable of knowing all of these reasons with complete accuracy. (You probably have a general sense of the prices of items at a grocery store, but very few know the exact price of every item in a store's inventory.) Then the random variable ϵ_i represents the *error* between the true utility (V_i) and what you believe the utility to be (U_i). Either interpretation leads to the same mathematical model.

Depending on the distribution we choose for the random variables ϵ_i , different discrete models are obtained. A classic is the *logit* model, which is obtained when the unobserved utilities ϵ_i are assumed to be independent across alternatives, and to have Gumbel distributions with zero mean and identical variance. Under this assumption, the probability of choosing alternative i is given by

$$p_i = \frac{\exp(\theta V_i)}{\sum_{j \in I} \exp(\theta V_j)} \quad (8.37)$$

where θ is a nonnegative parameter reflecting the relative magnitude of the observed utility relative to the unobserved utility. Notice what happens in this formula as θ takes extreme values: if $\theta = 0$, then all terms in the numerator and denominator are 1, and the probability of choosing any alternative is exactly the same. (Interpretation: the unobserved utility ϵ is much more important than the observed utility, so the observed utility has no impact on the decision made. Since the unobserved utility has the same distribution for every alternative, each is equally likely.) Or, if θ grows large, then the denominator of (8.37) will be dominated by whichever terms have the largest observed utility V_i . If there is some alternative i^* for which $V_{i^*} > V_i$ for all $i \neq i^*$ (the observed utility for i^* is strictly greater than any other alternative), then as $\theta \rightarrow \infty$, the probability of choosing i^* approaches 1 and the probability of choosing any other alternative approaches 0. Another important consequence of (8.37) is that the probability of choosing any alternative is always strictly positive; there is *some* chance that the unobserved utility will be large enough that any option could be chosen.

While the logit model is nice in that we have a closed-form expression (8.37) for the probabilities of choosing any alternative, the logit assumptions are very strong — particularly the assumption that the ϵ_i are independent and identically distributed. The exercises at the end of the chapter explore some examples demonstrating how these assumptions can lead to unreasonable results. Another common assumption is that the ϵ are drawn from a multivariate normal distribution, which allows for correlation among the unobserved utilities for different alternatives. This leads to the *probit* choice model, which is more flexible and arguably realistic. However, unlike the logit model, the probit model does not

have a closed-form expression for probabilities like (8.37). Instead, Monte Carlo sampling methods are used to estimate choices.

The majority of this section is focused on logit-based models. While probit models are more general and arguably more realistic, the logit model has two major advantages from the perspective of a book like this. First, computations in logit models can often be done analytically, simplifying explanations and making it possible to give examples you can easily verify. This helps you better understand the main ideas in stochastic user equilibrium and build intuition. Second, logit models admit faster solution algorithms, algorithms which scale relatively well with network size. This is an important practical advantage for logit models. Nevertheless, Section 8.3.7 provides some discussion on probit and other models and what needs to change from the logit discussion below.

8.3.2 Logit route choice

This section specializes the discrete choice framework from the previous section to route choice in networks. Consider a traveler leaving origin r for destination s . They must choose one of the paths π connecting r to s , that is, they must make a choice from the set Π_{rs} . The most straightforward way to generalize the principle of user equilibrium to account for perception errors is to set the observed utility equal to the negative of path travel time, so

$$U^\pi = -c^\pi + \epsilon^\pi \quad (8.38)$$

with the negative sign indicating that *maximizing utility* for drivers means *minimizing travel time*. Assuming that the ϵ^π are independent, identically distributed Gumbel random variables, we can use the logit formula (8.37) to express the probability that path π is chosen:

$$p^\pi = \frac{\exp(-\theta c^\pi)}{\sum_{\pi' \in \Pi_{rs}} \exp(-\theta c_{\pi'})} \quad (8.39)$$

The comments in the previous section apply to the interpretation of this formula. As θ approaches 0, drivers' perception errors are large relative to the path travel times, and each path is chosen with nearly equal probability. (The errors are so large, the choice is essentially random.) As θ grows large, perception errors are small relative to path travel times, and the path with lowest travel time is chosen with higher and higher probability. At any level of θ , there is a strictly positive probability that each path will be taken.

For concreteness, the route choice discussion so far corresponds to the second interpretation of SUE, where the unobserved utility represents perception errors in the utility. The first interpretation would mean that ϵ^π represents factors other than travel time which affect route choice (such as comfort, quality of scenery, etc.). Either of these interpretations is mathematically consistent with the discussion in Section 8.3.1.

The fact that the denominator of (8.39) includes a summation over *all* paths connecting r to s is problematic, from a computation standpoint. The number

of paths can grow exponentially with network size. Any use of stochastic user equilibrium in a practical setting, therefore, requires a way to compute link flows without explicitly calculating the sum in (8.39).

This is done by carefully defining which paths are in the choice set for travelers. The notation Π^{rs} in (8.39) in this book means the set of all acyclic paths connecting origin r to destination s . Following Chapter 6, we will use the notation $\hat{\Pi}^{rs}$ to define the set of paths being considered by travelers in SUE; these sets are sometimes called sets of *reasonable paths*. With a suitable definition of this set, using $\hat{\Pi}^{rs}$ in place of Π^{rs} in equation (8.39) leads to tractable computation schemes.

Two possibilities are common: selecting an acyclic subset of links, and choosing $\hat{\Pi}^{rs}$ to contain the paths using these links only; or setting $\hat{\Pi}^{rs}$ to consist of literally *all* paths (even cyclic ones) connecting origin r to destination s . Both of these are discussed next. The key to both of these definitions of $\hat{\Pi}^{rs}$ is that we can determine how many of the travelers passing through a given node came from each of the available incoming links, *without needing to know the specific path they are on*. This is known as the Markov property, and is discussed at more length in the optional Section 8.3.3.

Totally acyclic paths

For a particular origin r and destination s , choose a set of allowable links \hat{A}^{rs} , and let $\hat{\Pi}^{rs}$ consist of all paths starting at r , ending at s , and only containing links from \hat{A}^{rs} . We require two conditions on the set of allowable links:

1. There is at least one path from r to s using allowable links; this ensures that $\hat{\Pi}^{rs}$ is nonempty.
2. The set of allowable links contains no cycle; this ensures that all paths in $\hat{\Pi}^{rs}$ are acyclic.

We say a set of paths $\hat{\Pi}^{rs}$ is *totally acyclic* if it can be generated from an allowable link set satisfying these conditions.

This definition is closely related to the idea of a bush from Section 5.2.3. If \hat{A}^{rs} contains a path from r to every destination, it is also a bush; and if furthermore the sets \hat{A}^{rs} are the same for all destinations s , we can do the network loading for all the travelers leaving origin r simultaneously, rather than separately for each destination.

Note that there are collections of acyclic paths which are not totally acyclic. In the network in Figure 8.6, if we choose $\hat{\Pi}^{14} = \{[1, 2, 3, 4], [1, 3, 2, 4]\}$, both paths in this set are acyclic, but the set of allowable links needs to include every link in the network:

$$\hat{A}^{14} = \{(1, 2), (1, 3), (2, 3), (2, 4), (3, 2), (3, 4)\}$$

This set contains the cycle $[2, 3, 2]$, so it is not possible to generate a reasonable path set containing $[1, 2, 3, 4]$ and $[1, 3, 2, 4]$ from an acyclic set of allowable links.

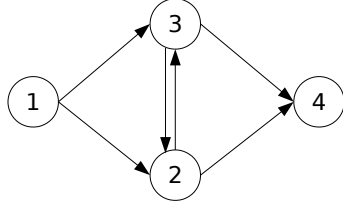


Figure 8.6: A set of acyclic paths need not be totally acyclic.

The advantage of totally acyclic path sets is that we can define a topological order on the nodes (see Section 2.2). With this topological order, we can efficiently make computations using the logit formula (8.39) without having to enumerate all the paths. This procedure is described in the next section.

We next describe two ways to form sets of totally acyclic paths. For each link, define a positive value c_{ij}^0 for each link which is constant and independent of flow — examples include the free-flow travel time or distance on the link. For each origin r and node i , let L_i^r denote the length of the shortest path from r to i , using the quantities c_{ij}^0 as the link costs. Likewise, for each destination s and node i , let ℓ_i^s denote the length of the shortest path from i to s , again using the quantities c_{ij}^0 as the link costs.

Consider the following sets of paths:

1. The set of all paths, for which the head node of each link is further away from the origin than the tail node, based on the quantities c_{ij}^0 . That is, the sets $\hat{\Pi}^r$ containing all paths starting at r and satisfying $L_j^r > L_i^r$ for each link (i, j) in the path.
2. The set of all paths, for which the head node of each link is closer to the destination than the tail node, based on the quantities c_{ij}^0 . That is, the sets $\hat{\Pi}^s$ containing all paths ending at s satisfying $\ell_j^s < \ell_i^s$ for each link (i, j) in the path. (This is like the first one, but oriented toward the destination, rather than the origin.)
3. The set of paths which satisfy both of the above criteria: $\hat{\Pi}^{rs} = \hat{\Pi}^r \cap \hat{\Pi}^s$.

For instance, if c_{ij}^0 reflects the physical length on each link, then for a given origin, $\hat{\Pi}^r$ would consist of all of the paths which start at that origin and always move away from it, never “doubling back.” Likewise, $\hat{\Pi}^s$ would consist of all paths which always move closer to their destination node s , without any diversions that lead it away. The third option has paths which both continually move away from their origin and toward their destination. Exercise 15 asks you to show that all three possibilities for $\hat{\Pi}$ are totally acyclic.

Figure 8.7 illustrates these three definitions. The top of the figure shows a network with node A as origin and node F as destination, and the links

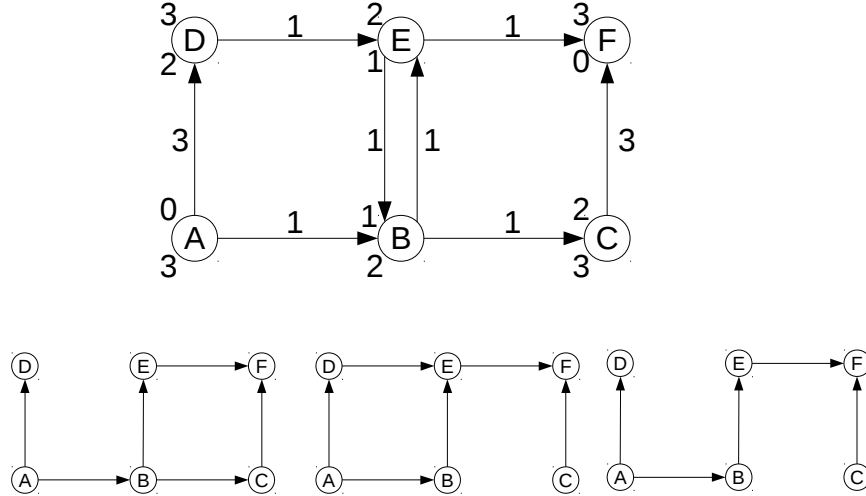


Figure 8.7: Demonstration of three alternative definitions of totally acyclic paths. Top panel shows c_{ij}^0 values on links, shortest path distances L_i from A above nodes, and shortest path distances ℓ_i to F below nodes.

are labeled with their c_{ij}^0 values. The nodes are labeled with their L_i values (above each node) and ℓ_i values below. The bottom of the figure shows the links satisfying each of the three criteria ($L_j > L_i$; $\ell_j < \ell_i$; and both of these simultaneously). The paths in these networks are the allowable paths in the original network. Notice that in all cases, there are no cycles in these links (even though the original network had the cycle [2,5,2]).

Of these principles, the third imposes stricter conditions on which paths are in the reasonable set. The first and second are weaker, and includes some paths which may not seem reasonable to you. For instance, the spiral path in Figure 8.8 satisfies the first condition, since the distance from the origin is always increasing. However, it does not satisfy the third condition, since at times the distance to the destination increases as well.

However, a major advantage of the first two principles is that we can aggregate travelers by origin or destination. With the first principle, the destination of travelers can be ignored for routing purposes — if a path is reasonable for a travel from an origin r to a node i , that path segment is reasonable for travel to any node beyond i as well. This allows us to aggregate travelers by origin (as in Section 5.2.3) and calculate a “one-to-all” path set for each origin, rather than having separate path sets for each OD pair. A similar destination-based aggregation is possible with the second principle.

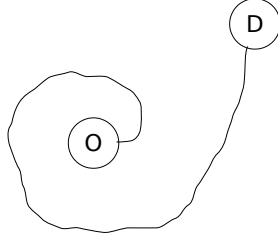


Figure 8.8: Is this path reasonable?

Full cyclic path set

Instead of restricting the path set to create a totally acyclic collection, an alternative is to have $\hat{\Pi}^{rs}$ consist of literally all paths from origin r to destination s , *even including cycles*. This will often mean these sets are infinite. For example, consider the network in Figure 8.6. Under this definition, the (cyclic) path $[1, 2, 3, 2, 4]$ is part of $\hat{\Pi}^{rs}$, as is $[1, 2, 3, 2, 3, 2, 4]$, and so on.

Including cyclic paths, especially paths with arbitrarily many repetitions of cycles, may seem counterintuitive. There are several reasons why this definition of $\hat{\Pi}^{rs}$ is nevertheless useful. One reason is that requiring total acyclicity is in fact quite a strong condition. In Figure 8.6, there is no totally acyclic path set that includes both $[1, 2, 3, 4]$ and $[1, 3, 2, 4]$ as paths — if we want to allow one path as reasonable, then by symmetry the other should be reasonable as well. But any path set including both of those includes both links $(2, 3)$ and $(3, 2)$, which form a cycle.

So, it is desirable to have an alternative to total acyclicity that still does not require path enumeration. As shown in the following section, it is possible to compute the link flows resulting from (8.39) without having to list all the paths, if all cyclic paths are included. The intuition is that all travelers at a given node can be treated identically in terms of which link they move to next: in Figure 8.6, we can split the vehicles arriving at node 2 between links $(2, 3)$ and $(2, 4)$ without having to distinguish whether they came via link $(1, 2)$, as if on the path $[1, 2, 4]$ or $[1, 2, 3, 4]$, or whether they came via link $(3, 2)$, as if on the path $[1, 3, 2, 4]$, or even $[1, 2, 3, 2, 4]$.

Furthermore, some modelers are philosophically uncomfortable with including restrictions like those in the previous section, without evidence that those rules really represent traveler behavior. Determining which sets of paths travelers actually consider (and why) is complicated, and still not well-understood.⁵ One school of thought is that it is therefore better to impose no restrictions at all, essentially taking an “agnostic” position with respect to the sets $\hat{\Pi}^{rs}$, rather than imposing restrictions which may not actually represent real behavior.

⁵Emerging data sources, such as Bluetooth readers, are providing more complete information on observed vehicle trajectories. This may provide more insight on this subject.

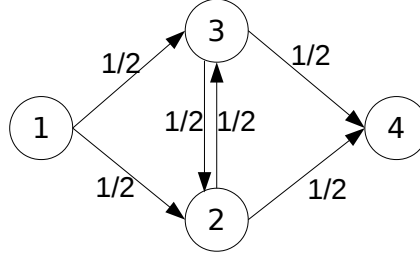


Figure 8.9: Flows from stochastic network loading when all links have unit cost, $\theta = \log 2$, and the full cyclic path set is allowed.

As an example, assume that every link in Figure 8.6 has the same travel time of 1 unit, and that $\theta = \log 2$. Then there are two paths of length 2 ($[1,2,4]$ and $[1,3,4]$), two paths of length 3 ($[1,2,3,4]$ and $[1,3,2,4]$), two paths of length 4 ($[1,2,3,2,4]$ and $[1,3,2,3,4]$), and so on. Therefore the denominator in the logit formula is

$$\sum_{\pi \in \tilde{\Pi}} \exp(-\theta c^\pi) = \frac{1}{2} + \frac{1}{2} + \frac{1}{4} + \frac{1}{4} + \frac{1}{8} + \frac{1}{8} + \cdots = 2 \quad (8.40)$$

and the probability of choosing one of the length-2 paths is $(1/2)/2 = 1/4$, the probability of choosing one of the length-3 paths is $1/8$, and so on. The flows on each link can be calculated by multiplying these path flows by the number of times that path uses a link. For instance, to calculate the flow on link (1,2), observe that it is used by paths $[1,2,4]$, $[1,2,3,4]$, $[1,2,3,2,4]$, $[1,2,3,2,3,4]$, and so on, with respective probabilities $1/4$, $1/8$, $1/16$, $1/32$, etc. Thus the total flow on this link is the sum of these, or $1/2$. The flows on links (1,3), (2,4), and (3,4) are also found to be $1/2$ by the same technique. Calculating the flow on links (2,3) and (3,2) is trickier, because some paths use these multiple times. For example, path $[1,2,3,2,3,4]$ uses link (2,3) twice, so even though the probability of selecting this path is $1/2^5 = 1/32$, it actually contributes twice this ($1/16$) to the flow. It is possible to show that the flow on these links is also $1/2$, giving the final flows in Figure 8.9.

As this example shows, direct calculations involving this path set usually involve summing infinite series. As will show in Section 8.3.4, there is an alternative method that allows us to make these computations without explicitly calculating such sums.

8.3.3 (*) The Markov property and the logit formula

(This optional section gives mathematical reasons why totally acyclic path sets, and complete path sets, both allow for efficient calculation of the logit formula.)

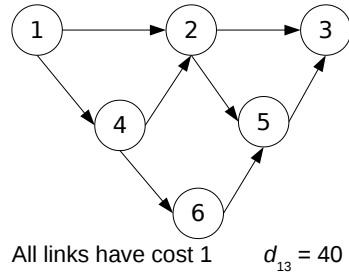


Figure 8.10: Network for demonstrating the Markov property.

Both path set definitions above (“totally acyclic paths” and the “full cyclic path set”) allow for the computation of the logit formula to be disaggregated by node and by link, without having to enumerate the paths in the network. The key to this is the *Markov property*. An informal statement of this property is that if we randomly select a traveler passing through a node, and want to know the probability that they leave that node by a particular link, there is no information provided by knowing which link they used to arrive to that node.

For example, consider the network in Figure 8.10, where the demand is $d_{13} = 40$ vehicles and $\theta = \log 2$. All links have unit cost. Assume first that all paths in this network are allowed. Then the right panel of Figure 8.11 shows the flow on each path, and the left panel shows the flow on each link.

In this network 18 vehicles pass through node 2. Suppose we pick one of them at random, and want to know the probability that the next link in this vehicle’s path is (2,3), as opposed to (2,5). From examining the path flows in Figure 8.11, we can see that this probability is $(8 + 4)/(8 + 4 + 4 + 2) = 2/3$ (ignoring the flow on path [1,4,6,5,3] in the denominator, since these trips do not pass through node 2). Now, suppose that these vehicles also reported the segment of their path that led them to node 2 — that is, they also report whether they came via segment [1,2] or segment [1,4,2]. Does this change our answers in any way?

If we know they came from segment [1,2], then they are either on path [1,2,3] or [1,2,5,3], and the probability that they continue on (2,3) is $8/(8+4) = 2/3$. If we know they came from segment [1,4,2], then they are either on path [1,4,2,3] or path [1,4,2,5,3], and the probability that they continue on (2,3) is $4/(4+2)$, which is still $2/3$. So knowing the first segment of their trip does not provide any additional information as to the remaining segment.

The situation changes if we modify the allowable path set to only include three paths, [1,2,3], [1,2,5,3], and [1,4,2,3].⁶ Here Figure 8.12 shows the corresponding path flows and link flows.

Let us ask the same question of the travelers passing through node 2. With-

⁶A natural way this path set might arise is to include paths that are only within a small threshold of the shortest path cost, thus including paths with cost 2 or 3 but excluding paths with cost 4.

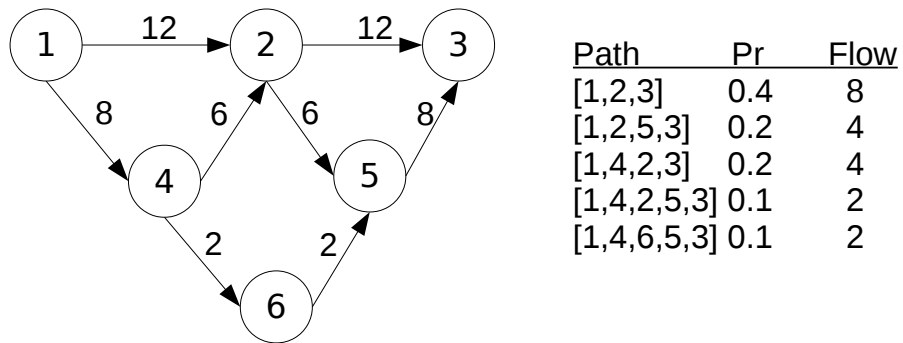


Figure 8.11: Link and path flows when all paths are allowed.

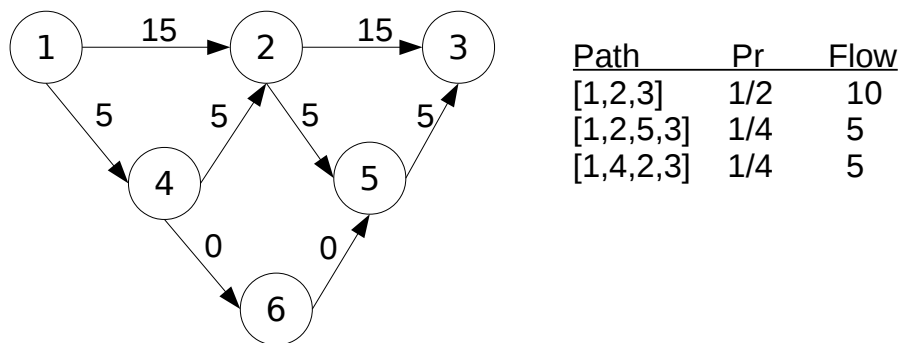


Figure 8.12: Link and path flows when only a subset of paths is allowed.

out knowing anything further, the probability that they continue on link (2,3) is $(10 + 5)/20 = 3/4$. However, if we know they came from [1,2], then the probability that they continue on (2,3) is $10/15 = 2/3$. If we know they came from [1,4,2], then the probability that they continue on (2,3) is 1, and there is no other option! So in this case, knowing the first segment of the path *does* give us additional information about the rest of their journey.

It turns out that the Markov property will be very useful, and will allow us to efficiently evaluate the logit formula without enumerating paths. Informally, we can do computations using just the link flows (the left panels in Figures 8.11 and 8.12) without having to use the path flows (the right panels of these figures) — we can get the path flows on the right from the link flows on the left. In large networks, the link-based representation is much more compact and efficient.

A more formal statement of this property is as follows. To keep the formulas clean, assume that there is a single origin r and destination s ; in a general network, we can apply the same logic separately to each OD pair. In logit assignment, the path set $\hat{\Pi}$ is said to satisfy the Markov property if there exist values P_{ij} for each link such that

$$p^\pi = \frac{\exp(-\theta c^\pi)}{\sum_{\pi' \in \hat{\Pi}} \exp(-\theta c^{\pi'})} = \prod_{(i,j) \in \pi} P_{ij}^{\delta_{ij}^\pi} \quad (8.41)$$

where δ_{ij}^π is the number of times path π uses link (i, j) .

That is, that the probability of a traveler selecting any path can be computed by multiplying P_{ij} values across its links. The P_{ij} values can be interpreted as conditional probabilities: given that a traveler is passing through i , they express the probability that their path leaves that node through link (i, j) .

The segment substitution property

To make the connection between totally acyclic and complete path sets and the Markov property, we first show that both path set choices satisfy the *segment substitution property*. Given any path $\pi = [r, i_1, i_2, \dots, s]$, a *segment* σ is any contiguous subset of one or more of its nodes. For example, the path [1, 2, 3, 2, 4] contains [1, 2, 3], [2, 3], [2], and [2, 3, 2, 4]. It does not contain the segment [1, 4]; even though both of those nodes are in the path, they do not appear consecutively. Note that a segment can consist of a single link, such as [2, 3], or even a single node, such as [2]. We use the notation \oplus to indicate joining segments, so [1, 2, 3, 2, 4] = [1, 2, 3] \oplus [3, 2, 4]. If two segments are being joined, the end node of the first must match the starting node of the second.

The set of reasonable paths $\hat{\Pi}$ satisfies the segment substitution property if, for any pair of reasonable paths which pass through the same two nodes, the paths formed by exchanging the segments between those nodes are also reasonable. That is, if $\pi = \sigma_1 \oplus \sigma_2 \oplus \sigma_3$ is reasonable, and if there is another reasonable path $\pi' = \sigma'_1 \oplus \sigma'_2 \oplus \sigma'_3$ with σ_2 and σ'_2 starting and ending at the same node, then the paths $\sigma_1 \oplus \sigma'_2 \oplus \sigma_3$ and $\sigma'_1 \oplus \sigma_2 \oplus \sigma'_3$ are also reasonable.

With the allowable path set in Figure 8.12, the segment substitution property is not satisfied. There are paths $[1,2,5,3]$ and $[1,4,2,3]$, both passing through nodes 1 and 2, which can be decomposed in the following way:

$$[1, 2, 5, 3] = [1] \oplus [1, 2] \oplus [2, 5, 3] \quad (8.42)$$

$$[1, 4, 2, 3] = [1] \oplus [1, 4, 2] \oplus [2, 3] \quad (8.43)$$

Notice that the corresponding pairs of segments on the right-hand sides all start and end at the same nodes. We can generate two new paths by “crossing” the middle segments of (8.42) and (8.43): $[1,2,3]$ and $[1,4,2,5,3]$. The first of these is allowable, but the second is not.

By contrast, you can verify that segment substitution is satisfied for the allowable path set in Figure 8.11. No matter which pairs of paths you choose, swapping the segments results in another allowable path.

A reasonable path set cannot have the Markov property unless it satisfies the segment substitution property, as shown in the above example. Without the segment substitution property, we could not know how the vehicles at node i would split without knowing the specific paths they were on. Segment substitution ensures that all travelers passing through node i are considering the same set of outgoing links (and, indeed, the same set of path segments continuing on to the destination).

It is fairly easy to show that both path set definitions considered above — sets of totally acyclic paths, and the set of all paths (even cyclic ones) — satisfy the segment substitution property; see Exercise 16.

Decomposing the logit formula

Given a reasonable path set $\hat{\Pi}$ satisfying the segment substitution property, and any two nodes a and b , let Σ_{ab} denote the set of segments which start and end at these nodes, and are part of a reasonable path. Define the quantity

$$V_{ab} = \sum_{\sigma \in \Sigma_{ab}} \exp(-\theta c^\sigma) \quad (8.44)$$

where $c^\sigma = \sum_{(i,j) \in \sigma} c_{ij}$ is the travel time of a segment. In particular, V_{rs} is a sum over all of the reasonable paths from origin to destination, and corresponds to the denominator of the logit formula. Thus, the probability that a traveler chooses a particular path π is simply $\exp(-\theta c^\pi)/V_{rs}$, and the flow on this path is $d^{rs} \exp(-\theta c^\pi)/V_{rs}$. Also note the special case V_{ii} where the start and end nodes are the same. In this case Σ_{ii} consists only of the single-node segment $[i]$ with zero cost, so $V_{ii} = 1$.

Furthermore, note that the numerator of the logit formula can be factored by segment, so that if $\pi = \sigma_1 \oplus \sigma_2 \oplus \sigma_3$, we have

$$\exp(-\theta c^\pi) = \exp(-\theta(c^{\sigma_1} + c^{\sigma_2} + c^{\sigma_3})) = \exp(-\theta c^{\sigma_1}) \exp(-\theta c^{\sigma_2}) \exp(-\theta c^{\sigma_3}). \quad (8.45)$$

To calculate the flow on a link x_{ij} , we need to add the flow from all of the reasonable paths which use this link. Every reasonable path using link (i, j) takes the form $\sigma_1 \oplus [i, j] \oplus \sigma_2$, where σ_1 goes from the origin r to node i and σ_2 goes from node j to the destination s .⁷ Therefore

$$x_{ij} = \sum_{\pi \in \hat{\Pi}^{rs}} \delta_{ij}^{\pi} h^{\pi} \quad (8.46)$$

$$= \frac{d^{rs}}{V_{rs}} \sum_{\pi \in \hat{\Pi}^{rs}} \delta_{ij}^{\pi} \exp(-\theta c^{\pi}) \quad (8.47)$$

$$= \frac{d^{rs}}{V_{rs}} \sum_{\sigma_1 \in \Sigma_{ri}} \sum_{\sigma_2 \in \Sigma_{js}} \exp(-\theta(c^{\sigma_1} + t_{ij} + c^{\sigma_2})) \quad (8.48)$$

$$= \frac{d^{rs}}{V_{rs}} \sum_{\sigma_1 \in \Sigma_{ri}} \sum_{\sigma_2 \in \Sigma_{js}} \exp(-\theta c^{\sigma_1}) \exp(-\theta t_{ij}) \exp(-\theta c^{\sigma_2}) \quad (8.49)$$

$$= \frac{d^{rs}}{V_{rs}} \left(\sum_{\sigma_1 \in \Sigma_{ri}} \exp(-\theta c^{\sigma_1}) \right) \exp(-\theta t_{ij}) \left(\sum_{\sigma_2 \in \Sigma_{js}} \exp(-\theta c^{\sigma_2}) \right) \quad (8.50)$$

$$= d^{rs} \frac{V_{ri} \exp(-\theta t_{ij}) V_{js}}{V_{rs}} \quad (8.51)$$

The third equality groups the sum over paths according to the starting and ending segment. This equation for x_{ij} will be used extensively in the rest of this section.

Similarly, to find the number of vehicles passing through a particular node i (call this x_i), observe that every path through i can be divided into a segment from r to i , and a segment from i to s . Grouping the paths according to these segments and repeating the algebraic manipulations above gives the formula

$$x_i = d^{rs} \frac{V_{ri} V_{is}}{V_{rs}} \quad (8.52)$$

With equations (8.51) and (8.52) in hand, it is easy to show the Markov property holds in any reasonable path set with the segment substitution property. Define

$$P_{ij} = \frac{x_{ij}}{x_i} = \frac{\exp(-\theta t_{ij}) V_{js}}{V_{is}} \quad (8.53)$$

and then multiply these values together for the links in a path, say, $\pi =$

⁷If the link starts at the origin or ends at the destination, we may have $i = r$ or $j = s$, in which case σ_1 or σ_2 will consist of a single node, $[r]$ or $[s]$.

$[r, i_1, i_2, \dots, i_k, s]$:

$$\prod_{(i,j) \in \pi} P_{ij} = \prod \frac{\exp(-\theta t_{ij}) V_{js}}{V_{is}} \quad (8.54)$$

$$= \exp \left(-\theta \sum_{(i,j) \in \pi} t_{ij} \right) \frac{V_{i_1 s} V_{i_2 s} \cdots V_{i_k s} V_{s s}}{V_{r s} V_{i_1 s} V_{i_2 s} \cdots V_{i_k s}} \quad (8.55)$$

$$= \exp(-\theta c^\pi) \frac{V_{s s}}{V_{r s}} \quad (8.56)$$

But $V_{s s} = 1$ and $V_{r s}$ is simply the denominator in the logit formula, so this product is exactly p^π . Therefore the logit path flow assignment in the set of reasonable paths satisfies the Markov property.

8.3.4 Stochastic network loading

The stochastic network loading problem is to determine the flows on each link x_{ij} , given their travel times t_{ij} , according to a particular discrete choice model. This section describes how to do so for the logit model, using formula (8.39). For stochastic network loading, we assume that these travel times are fixed and constant, and therefore unaffected by the path and link flows we calculate.

In the larger stochastic user equilibrium problem (where travel times can depend on flows), stochastic network loading plays the role of a subproblem in an iterative algorithm. This is analogous to how shortest paths and all-or-nothing loadings are often used as a subproblem in the classical traffic assignment problem: although travel times do depend on link flows, to find an equilibrium we can solve a number of shortest path problems, temporarily fixing the link costs at particular values.

Throughout this section, we are assuming a single origin r and destination s to simplify the notation. If there are many origins and destinations, these procedures should be repeated for each, and the link flows added to obtain the total link flows. (There is no harm in doing so, since we are assuming link travel times are constant, and therefore the different OD pairs do not interact with each other.) Depending on the choice of path set, it may be possible to aggregate all travelers from the same origin or destination, and load them at once. This is more efficient than doing a separate loading for each OD pair, but requires a more limited definition of the reasonable path sets.

In principle, the stochastic network loading procedure is straightforward. Given the link travel times t_{ij} , we can calculate the path travel times c^π as in Section 4.1. The path flows h^π can then be calculated from the logit formula (8.39), from which the link flows can be calculated by addition, again as in Section 4.1.

While conceptually straightforward, this procedure faces the practical difficulty that the logit formula requires summations over the set of all reasonable paths $\hat{\Pi}^{rs}$, which can grow exponentially with network size. In a realistic-sized

network, this renders the above procedure unusable, or at least computationally taxing.

This section describes how the link flows can be calculated *without* explicitly enumerating paths or using the logit formula. This is possible for both of the $\bar{\Pi}$ definitions discussed in the previous section: a totally acyclic set of paths, or including the full path set, even including all cycles.

These two procedures have several features in common. As was described in Section 8.3.3, both of these path set definitions have the Markov property. This section derived an important formula, repeated here:

$$x_{ij} = d^{rs} \frac{V_{ri} \exp(-\theta t_{ij}) V_{js}}{V_{rs}}, \quad (8.57)$$

where V_{ab} is the sum of $\exp(-\theta c^\sigma)$ for all reasonable path segments σ starting at node a and ending at node b . This formula is important because it allows us to calculate the flow on each link without having to enumerate all of the paths that use that link.

It is also possible to show (see Exercise 19) that we can replace t_{ij} with $L_i + t_{ij} - L_j$, where \mathbf{L} is a vector of node-specific constants in this formula; this reduces numerical errors in computations. It is common to use shortest path distances at free-flow (hence the use of the notation L). Doing this for the link and segment costs helps avoid numerical issues involved with calculating exponentials of large values. With this re-scaling, we define the *link likelihood* as

$$L_{ij} = \exp(\theta(L_j - L_i - t_{ij})) \quad (8.58)$$

and thus

$$x_{ij} = d^{rs} \frac{V_{ri} L_{ij} V_{js}}{V_{rs}}. \quad (8.59)$$

It remains to describe how the V_{ab} values can be efficiently calculated. This section shows how this can be done for both totally acyclic path sets, and the complete set of all paths (even cyclic ones).

In the case of a totally acyclic path set, the relevant V values can be calculated in a single pass over the network in topological order, and then the link flows calculated in a second pass over the network in reverse topological order. In the case of the full cyclic path set, the cycles create dependencies in the link weights and in the link flow formulas which prevent them from being directly evaluated. But we can still calculate them explicitly using matrix techniques.

Totally acyclic paths

The defining feature of a totally acyclic path set is that the collection of links used by reasonable paths has no cycles. This allows us to define a topological order on the nodes, so that each allowable link connects a lower-numbered node to a higher-numbered one. In acyclic networks, it is often easy to perform calculations recursively, in increasing or decreasing topological order. Stochastic network loading is one of these cases, using Dial's method.

Define W_i and W_{ij} to be the *weight* of a node and link, respectively. The node weight W_i is a shorthand for V_{ri} , and W_{ij} is a shorthand for $V_{ri}L_{ij}$. These can be calculated recursively, using the following procedure:

1. Calculate the link likelihoods L_{ij} using equation (8.58) for all allowable links; set $L_{ij} = 0$ for any link not in the allowable set.
2. Set the current node i to be the origin r , and initialize its weight: $W_r = 1$.
3. For all links (i, j) leaving node i , set $W_{ij} = W_i L_{ij}$.
4. If i is the destination, stop. Otherwise, set i to be the next node in topological order.
5. Calculate $W_i = \sum_{(h,i) \in \Gamma^{-1}(i)} W_{hi}$ by summing the weights of the incoming links.
6. Return to step 3.

With the node and link weights in hand, we proceed to calculate the flows on each link x_{ij} and the flow through each node x_i . Using the link weights, we can rewrite equation (8.52) for the flow through each node as

$$x_i = d^{rs} \frac{V_{ri} V_{is}}{V_{rs}} = d^{rs} \frac{W_i V_{is}}{W_s}. \quad (8.60)$$

Combining with equation (8.59) for link flow, we have

$$x_{hi} = d^{rs} \frac{V_{rh} L_{hi} V_{is}}{V_{rs}} = x_i \frac{W_{hi}}{W_i} \quad (8.61)$$

so if we know the flows to some node i , we can calculate the flows to its incoming links (h, i) . So, the link flows can be calculated in reverse topological order:

1. Initialize all flows to zero: $x_i = 0$ for all nodes, and $x_{ij} = 0$ for all links.
2. Set the current node i to be the destination s , and initialize its flow: $x_i = d^{rs}$, since all vehicles must reach the destination.
3. For all links (h, i) entering node i , set $x_{hi} = x_i W_{hi} / W_i$.
4. If i is the origin, stop. Otherwise, set i to be the previous node in topological order.
5. Compute $x_i = \sum_{(i,j) \in \Gamma(i)} x_{ij}$ as the sum of the flows on outgoing links.
6. Return to step 3.

As an example, Dial's method is demonstrated on the network shown in Figure 8.13, where the demand is 2368 vehicles from node 1 to node 5 and $\theta = 1$. For convenience, the results of the calculations in this example are shown in Tables 8.1 and 8.2. As a preliminary step, we calculate the shortest

path from node 1 to all other nodes at free-flow conditions (assuming $\mathbf{x} = \mathbf{0}$) using standard techniques. Details are omitted since the technique is familiar, and the resulting shortest path labels L_i are shown in Table 8.1.

Next, we must identify the allowable links. This example will adopt the first principle from the previous section, where the allowable links (i, j) are those for which $L_i < L_j$. Every link is thus allowable except for $(2, 3)$, because this link connects node 2 (with shortest path label $L_2 = 2$) to node 3 ($L_3 = 1$). Therefore, link $(2, 3)$ will be excluded from the remaining steps of Dial's method, since no vehicles will use this link.

The next step is to calculate the link likelihoods L_{ij} for the allowable links $(i, j) \in A_B$. So, $L_{12} = \exp(2 - 0 - 2) = 1$, $L_{35} = \exp(3 - 1 - 3) = e^{-1} \approx 0.368$, and so forth. Weights are now calculated in forward topological order. The topological ordering for the reasonable bush is 1, 3, 2, 4, and 5 in that order. Notice that the original network has a cycle $[2, 3, 2]$ and so no topological order can exist on the full network. But when restricted to the allowable set, the cycle disappears and node 3 must come before node 2 topologically. For the origin, $W_1 = 1$ by definition. The weights on links leaving node 1 can now be calculated: $W_{12} = W_1 L_{12} = 1$, and $W_{13} = 1 \times 1 = 1$. Proceeding to node 3 (the next in topological order), W_3 is calculated as the sum of the weights on incoming links: $W_3 = W_{13} = 1$ and thus $W_{32} = 1 \times 1 = 1$ and $W_{35} = 1 \times e^{-1} \approx 0.368$. Node 2 is next in topological order, and its weight is the sum of the weights on its incoming links: $W_2 = W_{12} + W_{32} = 2$, so $W_{24} = 2$. Node 4 is next, and we have $W_4 = 2$ and $W_{45} = 2$. Finally, the weight of node 5 is $W_5 = W_{35} + W_{45} = 2.368$.

Link and node flows are now calculated in *reverse* topological order, starting with node 5 and then proceeding to nodes 4, 2, 3, and 1. For node 5, the node flow X_5 is simply the demand destined to this node (since there are no outgoing links), so $X_5 = 2368$. This flow is now distributed among the two incoming links $(3, 5)$ and $(4, 5)$ in proportion to their weights. So, $x_{35} = X_5 W_{35} / (W_{35} + W_{45}) = 2368 \times 0.368 / (2 + 0.368) = 368$ and $x_{45} = 2000$. Proceeding upstream, the node flow at 4 is simply the flow on link $(4, 5)$, the only outgoing link (since no vehicles have node 4 as their destination), and $X_4 = 2000$. There is only one incoming link $(2, 4)$, so $x_{24} = 2000$. (This follows trivially from the formula in step 3c since $W_4 = W_{24}$. Thus $X_2 = 2000$. Since there are two incoming reasonable links to node 2 with equal weight, they receive equal flow (again, following from the formula in 3c), setting $x_{32} = x_{12} = 1000$. Then $X_3 = x_{32} + x_{35} = 1368$, and $x_{13} = X_3 = 1368$. Finally, the node flow at the origin 1 is $X_1 = 1368 + 1000 = 2368$, as it should be.

Dial's method is now complete, having calculated the flows on each link. This method is completely consistent with (8.39), if we were to restrict attention to the reasonable paths in the network. If we were to use this formula directly, we would first enumerate the three reasonable paths $[1, 3, 5]$, $[1, 3, 2, 4, 5]$, and $[1, 2, 4, 5]$ and calculate their costs: $C^{[1,3,5]} = 4$, $C^{[1,3,2,4,5]} = C^{[1,2,4,5]} = 3$. Equation (8.39) then gives

$$p^{[1,3,5]} = \frac{e^{-4}}{e^{-4} + e^{-3} + e^{-3}} = 0.155 \quad p^{[1,3,2,4,5]} = \frac{e^{-3}}{e^{-4} + e^{-3} + e^{-3}} = 0.422$$

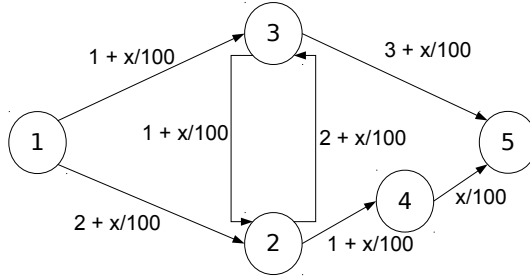


Figure 8.13: Network for demonstration of Dial's method.

Table 8.1: Node calculations in Dial's method example.

Node i	L_i	W_i	X_i
1	0	1	2368
2	2	2	1368
3	1	1	2000
4	3	2	2000
5	3	$2 + e^{-1}$	2368

$$p^{[1,2,4,5]} = \frac{e^{-3}}{e^{-4} + e^{-3} + e^{-3}} = 0.422$$

as the path choice proportions. Multiplying each of these by the total demand (2368) gives path flows

$$h^{[1,3,5]} = 368 \quad h^{[1,3,2,4,5]} = 1000 \quad h^{[1,2,4,5]} = 1000$$

As you can verify, these path flows correspond to the same link flows shown in Table 8.2.

Table 8.2: Link calculations in Dial's method example.

Link (i, j)	L_{ij}	W_{ij}	x_{ij}
(1,2)	1	1	1000
(1,3)	1	1	1368
(2,3)	0	0	0
(2,4)	1	2	2000
(3,2)	1	1	1000
(3,5)	e^{-1}	e^{-1}	368
(4,5)	1	2	2000

Full cyclic path set

We can also efficiently calculate the link flows from logit network loading, if the set of reasonable paths contains all paths, even cyclic ones. Again we will use formula (8.59). If we have an efficient way to compute V_{ab} for all pairs of nodes a and b , we can substitute them into this formula to directly obtain the link flows.

Since the set of reasonable paths contains cycles, we cannot calculate these V values inductively on topological order, as was done above. Rather, a different approach is needed.

Let \mathbf{V} be the $n \times n$ matrix whose components are V_{ab} . Recall that V_{ab} is defined as the sum of $\exp(-\theta c^\sigma)$ for all segments σ starting at node a and ending at node b . We will calculate this sum by dividing the sum into segments of the same length.

We begin by calculating the part of V_{ab} which corresponds to the segments of length one (that is, the segments consisting of a single link.) If there are no parallel links in the network, then this is simply L_{ab} : there is at most one such segment, which must be $\sigma = [a, b]$, and if it exists $\exp(-\theta c^\sigma) = \exp(-\theta t_{ab}) = L_{ab}$. If it does not exist, then $L_{ab} = 0$, which is again the part of V_{ab} corresponding to segments of length one (which is empty if no such segment exists.) We can proceed similarly if there are parallel links; see Exercise 20.

Now, let \mathbf{L} be the $n \times n$ matrix whose components are L_{ab} , and form the matrix product \mathbf{L}^2 . Its components are

$$(L^2)_{ab} = \sum_{c \in N} L_{ac} L_{cb} \quad (8.62)$$

by the definition of matrix multiplication.⁸

For a given node c , the product $L_{ac} L_{cb}$ is zero unless there is a link both from a to c , and from c to b . So, we can restrict the sum in (8.62) to be over nodes c for which there are links (a, c) and (c, b) — which is precisely the nodes c for which there is a segment $[a, c, b]$ of length two. Furthermore, for such segments,

$$\exp(-\theta c^{[a,c,b]}) = \exp(-\theta[t_{ac} + t_{cb}]) = \exp(-\theta t_{ac}) \exp(-\theta t_{cb}) = L_{ac} L_{cb}. \quad (8.63)$$

So, *the components of \mathbf{L}^2 are exactly the portion of the sums defining V_{ab} for segments of length two!*

We demonstrate this using the example from Figure 8.6, recalling that $d^{14} = 1$, $\theta = \log 2$, and that all links have unit cost. In this network, the matrices \mathbf{L} and \mathbf{L}^2 take the form

$$\mathbf{L} = \begin{bmatrix} 0 & 1/2 & 1/2 & 0 \\ 0 & 0 & 1/2 & 1/2 \\ 0 & 1/2 & 0 & 1/2 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad \mathbf{L}^2 = \begin{bmatrix} 0 & 1/4 & 1/4 & 1/2 \\ 0 & 1/4 & 0 & 1/4 \\ 0 & 0 & 1/4 & 1/4 \\ 0 & 0 & 0 & 0 \end{bmatrix}. \quad (8.64)$$

⁸The parentheses are intentional: $(L^2)_{ab}$ is the component of matrix \mathbf{L}^2 in row a and column b . This is *not* the same as $(L_{ab})^2$, the square of the component of matrix \mathbf{L} in row a and column b .

Looking at the first row of \mathbf{L}^2 , we see that $(L^2)_{12} = 1/4$, because there is one segment which starts at node 1, ends at node 2, and contains two links ($[1,3,2]$) and $\exp(-\theta c^{[1,3,2]}) = 1/4$. Similarly $(L^2)_{13} = 1/4$, and $(L^2)_{14} = 1/2$ because there are two segments starting at node 1, ending at node 2, and containing two links ($[1,2,4]$ and $[1,3,4]$). The sum $\exp(-\theta c^{[1,2,4]}) + \exp(-\theta c^{[1,3,4]})$ is indeed $1/2$. In this matrix, $(L^2)_{23} = 0$, because there are no segments of length two starting at node 2 and ending at node 3.

Proceeding a step further, we have

$$(L^3)_{ab} = \sum_{c \in N} (L^2)_{ac} L_{cb}. \quad (8.65)$$

By the same logic, we see that this sum expresses the component of V_{ab} corresponding to segments of length three. Every segment of length three connecting a to b consists of a segment of length two connecting a to some node c , followed by a link (c, b) . Group the sum of all segments of length three by this final link, and note that $(L^2)_{ac}$ already contains the relevant portion of the sum for the first segment.

Thus, by induction, the components of matrix \mathbf{L}^n contains the portion of the sum defining \mathbf{V} corresponding to segments of length n . Therefore

$$\mathbf{V} = \mathbf{L}^0 + \mathbf{L}^1 + \mathbf{L}^2 + \mathbf{L}^3 + \dots, \quad (8.66)$$

where the infinite sum is needed since we allow paths with an arbitrary number of cycles. Assuming that this sum exists, we can calculate it as follows:

$$\mathbf{V} = \mathbf{I} + \mathbf{L} + \mathbf{L}^2 + \mathbf{L}^3 + \dots \quad (8.67)$$

$$= \mathbf{I} + \mathbf{L}(\mathbf{I} + \mathbf{L} + \mathbf{L}^2 + \dots) \quad (8.68)$$

$$= \mathbf{I} + \mathbf{L}\mathbf{V}. \quad (8.69)$$

Therefore $\mathbf{V} - \mathbf{L}\mathbf{V} = \mathbf{I}$, or

$$\mathbf{V} = (\mathbf{I} - \mathbf{L})^{-1}. \quad (8.70)$$

After calculating the matrix \mathbf{V} with this formula, we can directly read off its components V_{ab} and use them to calculate the link flows using (8.59).

To complete the example, we have

$$\mathbf{V} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 4/3 & 2/3 & 1 \\ 0 & 2/3 & 4/3 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (8.71)$$

and, for example, the flow on link (2,3) is given by

$$x_{23} = d^{14} \frac{V_{12} L_{23} V_{34}}{V_{14}} = 1 \cdot \frac{1}{2} \cdot 1 = \frac{1}{2}. \quad (8.72)$$

Repeating this process will give the flow on every link — and unlike in the previous section, does not involve summing an infinite series term-by-term. The formula (8.59) handles all of the paths.

8.3.5 Stochastic user equilibrium

Stochastic network loading methods, described in the previous section, are the analogue of the all-or-nothing assignments we used to find \mathbf{x}^* in the method of successive averages, or Frank-Wolfe, in Chapter 6. Recall that in those methods, we identified \mathbf{x}^* by finding the link flows that would be observed if all the travel times were held constant at the current values. In TAP, all drivers aim to take the shortest path, so \mathbf{x}^* was obtained by loading all flow onto the shortest paths at the current travel times. In logit SUE, drivers do not always take the shortest path, but instead choose paths by (8.39). The methods in the previous section thus calculate an “ \mathbf{x}^* ” in the sense that it reflects the link flows which would arise if travel times were held constant.

The full development of the SUE model requires relaxing the assumption of constant travel times, in the same way as finding a traditional user equilibrium requires more than a single shortest path computation. SUE is very easy to define as a fixed point problem. Let $\mathbf{C}(\mathbf{h})$ denote the vector of path travel times as a function of the vector of path flows (this is the same as before). However, the logit formula directly gives us a complementary function $\mathbf{H}(\mathbf{c})$ giving path flows as a function of path travel times, with components

$$h^\pi = d^{rs} \frac{\exp(-\theta c^\pi)}{\sum_{\pi' \in \Pi^{rs}} \exp(\theta c_{\pi'})} \quad (8.73)$$

where (r, s) is the OD pair corresponding to path π . Therefore, the SUE problem can be expressed as follows: find a feasible path flow vector \mathbf{h}^* such that $\mathbf{h}^* = \mathbf{H}(\mathbf{C}(\mathbf{h}^*))$. This is a standard fixed-point problem. Clearly \mathbf{H} and \mathbf{C} are continuous functions if the link performance functions are continuous, and the feasible path set is compact and convex, so Brouwer’s theorem immediately gives existence of a solution to the SUE problem.

Notice that this was much easier than showing existence of an equilibrium solution to the original traffic assignment problem! For that problem, there was no equivalent of (8.73). Travelers were all using shortest paths, but if there were two or more shortest paths there was no rule for how those ties should be broken. As a result, we had to reformulate the problem as a variational inequality and introduce an auxiliary function based on movement of a point under a force. For the SUE problem, there is no need for such machinations, and we can write down the fixed point problem immediately.

However, fixed-point theorems do not offer much help in terms of actually finding the SUE solution. It turns out that convex programming and variational inequality formulations exist as well, and we will get to them shortly. But first, as a practical note, we mention that our definition of the reasonable path set $\hat{\Pi}$ should be specified *without reference to the final travel times*. The reason is that until we have found the equilibrium solution, we do not know what the link and path travel times will be. If the sets of reasonable paths vary from iteration to iteration, as the flows and travel times change, there may be problems with convergence or solution consistency. This is why the methods described above for generating totally acyclic path sets relied on constants c_{ij}^0

which were independent of flow: constants such as physical length or free-flow times. If using the full cyclic path set, there is no concern; at all iterations every path is reasonable.

Path-flow formulation and the method of successive averages

Consider the following optimization problem, developed by Caroline Fisk:

$$\min_{\mathbf{x}, \mathbf{h}} \theta \sum_{(i,j) \in A} \int_0^{x_{ij}} t_{ij}(x) dx + \sum_{\pi \in \hat{\Pi}} h^\pi \log h^\pi \quad (8.74)$$

$$\text{s.t. } x_{ij} = \sum_{\pi \in \hat{\Pi}} h^\pi \delta_{ij}^\pi \quad \forall (i, j) \in A \quad (8.75)$$

$$\sum_{\pi \in \hat{\Pi}^{rs}} h^\pi = d^{rs} \quad \forall (r, s) \in Z^2 \quad (8.76)$$

$$h^\pi \geq 0 \quad \forall \pi \in \hat{\Pi} \quad (8.77)$$

Comparing with Beckmann's formulation from Chapter 5, we see that it is identical except that the sums on path variables are now over the set of reasonable paths, rather than all paths, and that the objective is first scaled by θ , and then an additional term is added. We show below that these modifications ensure that the optimal path flows satisfy the logit formula (8.39). This objective function is strictly convex in the *path flows* (see Exercise 23), so the SUE path flow solution is unique. This is a different situation than the classical traffic assignment problem, which has a unique equilibrium solution in link flows, but not in path flows.

First, we show that the non-negativity constraint (8.77) can be ignored without any problem, and in fact that at optimality the flow on each path is *strictly* positive. The $\log h^\pi$ term in the objective is not defined at all if a path flow is negative; if $h^\pi = 0$ we can define $h^\pi \log h^\pi = 0$, since this is the limiting value by l'Hôpital's rule. But the derivative of $h \log h$ becomes infinitely steep as h approaches zero: $\frac{d}{dh}(h \log h) = \log h + 1 \rightarrow -\infty$ as $h \rightarrow 0$. Therefore the objective function cannot be minimized at $h = 0$; no matter what the change in the other terms in the objective would be, it is better to have a very slightly positive h value than a zero one. This is a very useful property, since the optimality conditions are much simpler if there are no non-negativity constraints.

Next, as with Beckmann's formulation, we Lagrangianize the demand constraint (8.76), and substitute (8.75) in place of \mathbf{x} , to obtain a Lagrangian in terms of path flow variables only:

$$\begin{aligned} \mathcal{L}(\mathbf{h}, \boldsymbol{\kappa}) = & \theta \sum_{(i,j) \in A} \int_0^{\sum_{\pi \in \hat{\Pi}} h^\pi \delta_{ij}^\pi} t_{ij}(x) dx + \sum_{\pi \in \hat{\Pi}} h^\pi \log h^\pi + \\ & \sum_{(r,s) \in Z^2} \kappa^{rs} \left(d^{rs} - \sum_{\pi \in \hat{\Pi}^{rs}} h^\pi \right) \end{aligned} \quad (8.78)$$

Since there are no non-negativity conditions, it is enough to find the stationary points of the Lagrangian, that is, the points where $\frac{\partial \mathcal{L}}{\partial h} = 0$. Therefore, we require

$$\frac{\partial \mathcal{L}}{\partial h^\pi} = \theta c^\pi + 1 + \log h^\pi - \kappa^{rs} = 0, \quad (8.79)$$

or, solving for h^π ,

$$h^\pi = \exp(\kappa^{rs} - 1) \exp(-\theta c^\pi). \quad (8.80)$$

To find the value of the Lagrange multiplier κ^{rs} , we substitute (8.80) into the demand constraint (8.76), and find that κ^{rs} must be chosen such that

$$\exp(\kappa^{rs} - 1) = \frac{d^{rs}}{\sum_{\pi' \in \hat{\Pi}^{rs}} \exp(-\theta c^{\pi'})}. \quad (8.81)$$

Substituting into (8.80) gives the logit formula, and therefore the path flows solving Fisk's convex optimization problem are those solving logit stochastic user equilibrium.

This convex optimization problem can be solved using the method of successive averages. In this way, SUE can be solved quite simply, using a familiar method from the basic TAP. The only change is that \mathbf{x}^* is calculated using a method from the previous section, rather than by finding an all-or-nothing assignment loading all flow onto shortest paths. The algorithm is as follows:

1. Choose an initial feasible link assignment \mathbf{x} .
2. Update link travel times based on \mathbf{x} .
3. Calculate target flows \mathbf{x}^* :
 - (a) For each OD pair (r, s) , use a method from the previous section to calculate OD-specific flows \mathbf{x}_{rs}^* .
 - (b) Calculate $\mathbf{x}^* \leftarrow \sum_{(r,s) \in Z^2} \mathbf{x}_{rs}^*$
4. Update $\mathbf{x} \leftarrow \lambda \mathbf{x}^* + (1 - \lambda) \mathbf{x}$ for some $\lambda \in (0, 1]$.
5. If \mathbf{x} and \mathbf{x}^* are sufficiently close, terminate. Otherwise, return to step 2.

(It is often possible to do the calculations in step 3 per origin or per destination, rather than separately for each OD pair.)

Notice that the termination criteria is slightly different than before. With the classical traffic assignment problem, we argued that it was a *bad* idea to compare the current solution to the previous solution and terminate when they are sufficiently small. Here, we are not doing that, but are doing something which looks similar: comparing the current solution \mathbf{x} with the “target” solution \mathbf{x}^* . This works because, unlike in the regular traffic assignment problem, the mapping $\mathbf{H}(\mathbf{C})$ is always continuous and well-defined. This means that \mathbf{x}^* varies slowly with \mathbf{x} : small changes in link flows mean small changes in link and path travel times, which means small changes in path flows from (8.73). With classic traffic assignment, a small change in link flows could mean a shift in the shortest

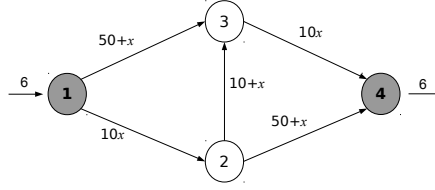


Figure 8.14: Network for demonstrating method of successive averages in stochastic assignment.

path, which would result in a dramatic change in \mathbf{x}^* , moving all flow from an OD pair to that new shortest path. Furthermore, when there are ties there are multiple possible \mathbf{x}^* values. So, in this case we had to introduce auxiliary convergence measures like the relative gap or average excess cost. SUE is simpler in that we can simply compare the current and target solutions — in fact, as defined earlier, the relative gap and average excess cost do not make sense, since the equilibrium principle is no longer defined by all travelers being on the shortest path. For these reasons, the method of successive averages works much better for SUE than for deterministic assignment.

To demonstrate this algorithm, consider the Braess paradox network shown in Figure 8.14, where the demand from node 1 to node 4 is 6 vehicles, $\theta = 0.01$, and all paths are allowable. Assume that we choose the initial solution \mathbf{x} by performing Dial's method on this network, using the *free-flow* travel times. Table 8.3 shows the calculations; in this table, we first calculate the L_i values for nodes (based on shortest paths), then the link likelihoods L_{ij} ; then the node and link weights W_i and W_{ij} , and finally the node and link flows x_i and x_{ij} . This gives initial link flows of $\mathbf{x} = [4.282 \ 1.718 \ 2.563 \ 1.718 \ 4.282]$. Recalculating the link performance functions with these flows gives new link travel times, which we put into Dial's method again, giving the result $\mathbf{x}^* = [3.976 \ 2.024 \ 1.951 \ 2.024 \ 3.976]$. *Notice that \mathbf{x}^* and \mathbf{x} are quite close to each other!* This is a very different situation than when the method of successive averages is applied to the classical traffic assignment problem (compare with the examples in Section 6.2.1), where \mathbf{x}^* was an extreme-point solution quite far away from \mathbf{x} .

So the link flows \mathbf{x} are updated by averaging \mathbf{x}^* into the old \mathbf{x} values, using a weight of $\lambda = 1/2$, producing the results in the rightmost column of Figure 8.14. The process is repeated until convergence.

Disaggregate link-flow formulation and Frank-Wolfe

For the classical TAP, the Frank-Wolfe algorithm was much faster than the method of successive averages, because it chose λ adaptively, to maximize the reduction in the objective function at each iteration. In theory, it is possible to do the same thing with the convex program described above. In practice, it is

Table 8.3: Method of successive averages for stochastic user equilibrium

	Initialization			Iteration 1			
Nodes	L_i	W_i	x_i	L_i	W_i	x_i	
1	0	1	6	0	1	6	
2	0	1	4.282	42.8	1	3.976	
3	10	1.670	4.282	51.7	1.964	3.976	
4	10	2.341	6	94.5	2.964	6	
Links	L_{ij}	W_{ij}	x_{ij}	L_{ij}	W_{ij}	x_{ij}^*	x_{ij}
(1,2)	1	1	4.282	1	1	3.976	4.129
(1,3)	0.670	0.670	1.718	1	1	2.024	1.871
(2,3)	1	1	2.563	0.964	0.964	1.951	2.257
(2,4)	0.670	0.670	1.718	1	1	2.024	1.871
(3,4)	1	1.670	4.282	1	1.964	3.976	4.129

harder because this program makes use of the path-flow variables \mathbf{h} . With the classical traffic assignment problem, we could express solutions and the objective solely in terms of the link flows \mathbf{x} . The addition of the $h \log h$ terms to the SUE objective function renders this impossible. Furthermore, the number of paths grows very quickly with network size (and if we are choosing the reasonable path set to include fully cyclic paths, the number of paths is usually infinite).

The method of successive averages avoids this problem by not actually referring to the objective function at any point — if you review the steps above, you see that the objective function is never calculated. Its role is implicit, guaranteeing that the algorithm will eventually converge, since the direction $\mathbf{x}^* - \mathbf{x}$ is one along which the objective is decreasing. If we can find an efficient way to evaluate the objective function, then we can develop an analogue to the Frank-Wolfe method for classical assignment.

It turns out that we can reformulate the objective function in terms of the destination-aggregated flows on each link, x_{ij}^s (see Section 5.2.3), using the Markov property of the logit loading. There is an equivalent disaggregation by origin (if we reverse our interpretation of the Markov property; see Exercise 18.) Recall the following results from Section 8.3.3:⁹

- There exist values P_{ij}^s for each link (i, j) and destination s , such that

$$h^\pi = d^{rs} \prod_{(i,j) \in \pi} (P_{ij}^s)^{\delta_{ij}^\pi} \quad (8.82)$$

for any path π connecting origin r to destination s .

- The P_{ij}^s values can be interpreted as the conditional probability that a vehicle arriving at node i and destined for node s will have link (i, j) as

⁹Technically, this section only proved them for the case of a single origin-destination pair, but they hold for each destination as well.

the next link in its path. Therefore,

$$P_{ij}^s = \frac{x_{ij}^s}{x_i^s} \quad (8.83)$$

for each destination s , node $i \neq s$, and outgoing link (i, j) .

Using these properties, we derive an equivalent formula for the additional term to the Beckmann function.

Proposition 8.4. *Let \mathbf{h} be a feasible path flow vector satisfying the Markov property, and let x_{ij}^s and x_i^s be the corresponding destination-aggregated link and node flows. Then*

$$\sum_{\pi \in \hat{\Pi}} h^\pi \log \left(\frac{h^\pi}{d^{rs}} \right) = \sum_{s \in Z} \left(\sum_{(i,j) \in A} x_{ij}^s \log x_{ij}^s - \sum_{\substack{i \in N \\ i \neq s}} x_i^s \log x_i^s \right) \quad (8.84)$$

Proof. We treat each destination s separately; summing over all destinations gives the result.

Using (8.82) and (8.83), we have

$$\sum_{\pi \in \hat{\Pi}^s} h^\pi \left(\frac{h^\pi}{d^{rs}} \right) = \sum_{\pi \in \hat{\Pi}^s} h^\pi \log \left(\prod_{(i,j) \in A} \left(\frac{x_{ij}^s}{x_i^s} \right)^{\delta_{ij}^\pi} \right) \quad (8.85)$$

$$= \sum_{\pi \in \hat{\Pi}^s} h^\pi \sum_{(i,j) \in A} \delta_{ij}^\pi \log \frac{x_{ij}^s}{x_i^s} \quad (8.86)$$

$$= \sum_{(i,j) \in A} \left(\sum_{\pi \in \hat{\Pi}^s} h^\pi \delta_{ij}^\pi \right) \log \frac{x_{ij}^s}{x_i^s} \quad (8.87)$$

$$= \sum_{(i,j) \in A} x_{ij}^s \log \frac{x_{ij}^s}{x_i^s} \quad (8.88)$$

$$= \sum_{(i,j) \in A} x_{ij}^s \log x_{ij}^s - \sum_{(i,j) \in A} x_{ij}^s \log x_i^s \quad (8.89)$$

$$= \sum_{(i,j) \in A} x_{ij}^s \log x_{ij}^s - \sum_{\substack{i \in N \\ i \neq s}} x_i^s \log x_i^s \quad (8.90)$$

by grouping terms in the last sum by the tail node. \square

Furthermore, the left-hand side of equation (8.84) is very similar to the second term in the objective function (8.74). In fact,

$$\sum_{\pi \in \hat{\Pi}^s} h^\pi \log \left(\frac{h^\pi}{d^{rs}} \right) = \sum_{\pi \in \hat{\Pi}^s} h^\pi \log h^\pi - \sum_{r \in Z} d^{rs} \log d^{rs} \quad (8.91)$$

using properties of logarithms and the fact that $\sum_{\pi \in \Pi^{rs}} = d^{rs}$. They differ only by $\sum_{r \in Z} d^{rs} \log d^{rs}$, which is a constant — and recall from Proposition B.2 that adding a constant does not affect the optimal solution to an optimization problem.

This means that we can replace the objective function (8.74) with

$$\sum_{(i,j) \in A} \int_0^{x_{ij}} t_{ij}(x) dx + \frac{1}{\theta} \sum_{s \in Z} \left(\sum_{(i,j) \in A} x_{ij}^s \log x_{ij}^s - \sum_{\substack{i \in N \\ i \neq s}} x_i^s \log x_i^s \right) \quad (8.92)$$

which does not require path enumeration. We can thus develop the following analogue to the Frank-Wolfe algorithm:

1. Choose an initial, feasible destination-aggregated link assignment \mathbf{x}^s , and the corresponding aggregated link flows \mathbf{x} .
2. Update link travel times based on \mathbf{x} .
3. Calculate target flows:
 - (a) For each destination s , use a method from the previous section to calculate OD-specific flows \mathbf{x}_s^* .
 - (b) Calculate $\mathbf{x}^* \leftarrow \sum_{s \in Z} \mathbf{x}_s^*$
4. Find the value of λ minimizing (8.92) along the line $(1 - \lambda) [\mathbf{x} \ \mathbf{x}^s] + \lambda [\mathbf{x}^* \ \mathbf{x}_s^*]$.
5. Update $\mathbf{x} \leftarrow \lambda \mathbf{x}^* + (1 - \lambda) \mathbf{x}$.
6. If \mathbf{x} and \mathbf{x}^* are sufficiently close, terminate. Otherwise, return to step 2.

8.3.6 Relationships between logit loading and most likely path flows (*)

(This optional section draws a connection between the logit-based stochastic equilibrium model and the concept of entropy maximization in most likely path flows.)

The use of the $h \log h$ terms to represent logit assignment in the previous section may have reminded you of the most likely path flows problem in deterministic user equilibrium, discussed in Section 5.2.2. In fact, these two concepts are related to each other quite closely! This optional section explores this relationship, and describes another algorithm for most likely based path flows.

Recall that the most likely path flows were defined as those maximizing entropy, and solving the following optimization problem:

$$\max_{\mathbf{h}} \quad - \sum_{\pi \in \hat{\Pi}} h_{\pi} \log(h_{\pi}/d) \quad (8.93)$$

$$\text{s.t.} \quad \sum_{\pi \in \Pi} \delta_{ij}^{\pi} h_{\pi} = x_{ij}^* \quad \forall (i, j) \in A \quad (8.94)$$

$$\sum_{\pi \in \hat{\Pi}} h_{\pi} = d \quad (8.95)$$

$$h_{\pi} \geq 0 \quad \forall \pi \in \Pi \quad (8.96)$$

where we have assumed there is a single origin-destination pair for simplicity.

We can transform this into an equivalent optimization that more closely resembles the stochastic user equilibrium optimization problems. First, we replace maximization by minimization by changing the sign of the objective. Second, we can remove d from the objective, because

$$\sum_{\pi \in \hat{\Pi}} h_{\pi} \log(h_{\pi}/d) = \sum_{\pi \in \hat{\Pi}} h_{\pi} \log h_{\pi} - \sum_{\pi \in \hat{\Pi}} h_{\pi} \log d \quad (8.97)$$

$$= \sum_{\pi \in \hat{\Pi}} h_{\pi} \log h_{\pi} - d \log d \quad (8.98)$$

and the constant $d \log d$ can be ignored. This gives

$$\min_{\mathbf{h}} \quad \sum_{\pi \in \hat{\Pi}} h_{\pi} \log h_{\pi} \quad (8.99)$$

$$\text{s.t.} \quad \sum_{\pi \in \Pi} \delta_{ij}^{\pi} h_{\pi} = x_{ij}^* \quad \forall (i, j) \in A \quad (8.100)$$

$$\sum_{\pi \in \hat{\Pi}} h_{\pi} = d \quad (8.101)$$

$$h_{\pi} \geq 0 \quad \forall \pi \in \Pi \quad (8.102)$$

We make one more change: rather than insisting that the path flows match the equilibrium link flows exactly, we change constraint (8.100) to require that the average travel time across all travelers be c^* . That is, we change the constraint to

$$\sum_{\pi \in \hat{\Pi}} h^{\pi} c^{\pi} = c^* d. \quad (8.103)$$

Arguing as in the previous section, we can ignore the non-negativity constraint (8.102), since it will not be binding at optimality.

We then Lagrangianize the other two constraints, introducing multipliers θ and κ for (8.103) and (8.102), respectively. The resulting Lagrangian function is

$$\mathcal{L}(\mathbf{h}, \theta, \kappa) = \sum_{\pi \in \hat{\Pi}} h^{\pi} \log h^{\pi} + \theta \left(\sum_{\pi \in \hat{\Pi}} h^{\pi} c^{\pi} - c^* d \right) + \kappa \left(d - \sum_{\pi \in \hat{\Pi}} h^{\pi} \right) \quad (8.104)$$

without non-negativity constraints. The optimality conditions are thus

$$\frac{\partial \mathcal{L}}{\partial h^\pi} = 1 + \log h^\pi + \theta c^\pi - \kappa = 0 \quad \forall \pi \in \hat{\Pi} \quad (8.105)$$

$$\frac{\partial \mathcal{L}}{\partial \theta} = \sum_{\pi \in \hat{\Pi}} h^\pi c^\pi - c^* d = 0 \quad (8.106)$$

$$\frac{\partial \mathcal{L}}{\partial \kappa} = \sum_{\pi \in \hat{\Pi}} h^\pi - d = 0 \quad (8.107)$$

$$(8.108)$$

The last two of these are simply the constraints (8.100) and (8.101). The first can be solved for the path flows, giving

$$h^\pi = \exp(\kappa - 1 - \theta c^\pi). \quad (8.109)$$

To satisfy the demand constraint (8.101), κ must be chosen so that the sum of (8.109) over all paths gives the total demand d .

Omitting the algebra, we must have

$$\kappa = \log d \left(1 - \log \sum_{\pi' \in \Pi} \exp(-\theta c^{\pi'}) \right) \quad (8.110)$$

or

$$h^\pi = d \frac{\exp(-\theta c^\pi)}{\sum_{\pi' \in \Pi} \exp(-\theta c^{\pi'})}. \quad (8.111)$$

But this is just the logit formula! The Lagrange multiplier θ must be chosen to satisfy the remaining constraint on the average path cost.

This derivation shows that the most likely path flows and stochastic user equilibrium problems have a similar underlying structure. If we relax the requirement that all travelers be on shortest paths, and simply constrain the average cost of travel, the most likely path flows coincide with a logit loading, where the parameter θ is the Lagrange multiplier for this constraint. As θ approaches infinity, the average cost of travel approaches its value at the deterministic user equilibrium solution, and the stochastic user equilibrium path flows approach the most likely path flows in the corresponding deterministic problem. This provides another algorithmic approach for solving for most likely path flows, in addition to those discussed in Section 6.5. In practice, this algorithm is difficult to implement, because of numerical issues that arise as θ grows large.

8.3.7 Alternatives to logit loading

The majority of this section has focused on the logit model for stochastic route choice, because it demonstrates the main ideas simply, and leads to computationally-efficient solution techniques. There are several serious criticisms of logit assignment. For instance, the assumption that the error terms ϵ^π are independent across paths is hard to defend if paths overlap significantly.

Common methods for creating totally acyclic route sets can also create unreasonable artifacts, and allowing all cyclic paths can also be unreasonable if the network topology creates many such paths with low travel times. See Exercise 25 for some concrete examples. The main alternative to the logit model is the *probit* model, in which the ϵ terms have a multivariate normal distribution, with a (possibly nondiagonal) covariance matrix to allow for correlation between these terms.

The framework of stochastic user equilibrium can be generalized to other distributions of the error terms, including correlation. The full development of this general framework is beyond the scope of this book, but we provide an overview and summary. The objective function (8.74) must be replaced with

$$z(\mathbf{x}) = - \sum_{(r,s) \in Z^2} d^{rs} E \left[\min_{\pi \in \Pi^{rs}} (c^\pi + \epsilon) \right] + \sum_{(i,j) \in A} x_{ij} t_{ij} - \sum_{(i,j) \in A} \int_0^{x_{ij}} t_{ij}(x) dx \quad (8.112)$$

where the expectation is taken with respect to the “unobserved” random variables ϵ , and t_{ij} in the last two terms are understood to be functions of x_{ij} . It can be shown that this function is convex, and therefore that the SUE solution is unique. However, evaluating this function is harder. The first term in (8.112) involves an expectation over all paths connecting an OD pair. In discrete choice, this is known as the *satisfaction function*, and expresses the expected perceived travel time on a path chosen by a traveler. In the case of the logit model, this expectation can be computed in closed form; for most distributions it cannot, and must be evaluated through Monte Carlo sampling or another approximation.

The method of successive averages can still be used, even without evaluating the objective function. Step 4a needs to be replaced with a stochastic network loading, using the current travel times and whatever distribution of ϵ is chosen. This often requires Monte Carlo sampling as well: for (multiple) samples of the ϵ terms, the shortest paths can be found using one of the standard algorithms, and the resulting flows averaged together to form an estimate of \mathbf{x}^* .

Because we are using an estimate of \mathbf{x}^* , it is possible that the target direction is not exactly right, and that it is not in a direction in which $z(\mathbf{x})$ is decreasing. Nevertheless, as long as it is correct “on average” (i.e., the sampling is done in an unbiased manner), one can show that the method of successive averages will still converge to the stochastic user equilibrium solution.

8.4 Historical Notes and Further Reading

(These sections are incomplete in this beta version of the text, and will be substantially expanded in the complete first edition.)

The original formulation of the user equilibrium traffic assignment problem in Beckmann et al. (1956) actually modeled demand as elastic. (In this book we chose to focus on the fixed-demand problem in previous chapters, which is a special case.) The transformation of the elastic demand problem to a fixed-demand problem with artificial links was reported in Gartner (1980), based on

earlier work by Murchland (1970). While the original Frank-Wolfe algorithm can be applied to the elastic demand equilibrium problem, the modified version presented in this chapter is faster. It is essentially a specialized version of the double-stage algorithm of Evans (1976) developed for a combined trip distribution and assignment model.

The variational inequality formulation of the traffic assignment problem with link interactions is due to Smith (1979a), Dafermos (1980), and Aashtiani and Magnanti (1981). The diagonalization method was demonstrated in Fisk and Nguyen (1982), and its convergence proved by Dafermos (1982). The simplicial decomposition method is adapted from Smith (1983). The example with multiple equilibria in the simple merge network is taken from Boyles et al. (2015). Other examples of multiple equilibria are given in Netter (1972) and Marcotte and Wynter (2004).

Stochastic network loading using the logit model and totally acyclic path sets was described in Dial (1971). Logit loading with the full cyclic path set was developed by Bell (1995), Akamatsu (1996), and Akamatsu (1997).

The stochastic user equilibrium model was first proposed in Daganzo and Sheffi (1977). Powell and Sheffi (1982) proved the convergence of the method of successive averages for this problem, for all well-behaved distributions of the unobserved utility term. The convex programming case for the logit model was given in Fisk (1980), and the optimization formulation for more general distributions was given in Daganzo (1982).

8.5 Exercises

1. [25] Verify that each of the demand functions D below is strictly decreasing and bounded above (for $\kappa \geq 0$), then find the inverse functions $D^{-1}(d)$.
 - (a) $D(\kappa) = 50 - \kappa$
 - (b) $D(\kappa) = 1000/(\kappa + 1)$
 - (c) $D(\kappa) = 50 - \kappa^2$
 - (d) $D(\kappa) = (\kappa + 2)/(\kappa + 1)$
2. [13] The total misplaced flow reflects consistency of a solution with (in this case, that the OD matrix should be given by the demand functions). Suggest another measure for how close a particular OD matrix and traffic assignment are to satisfying this consistency condition. Your proposed measure should be a nonnegative and continuous function of values related to the solution (e.g., d^{rs} , x_{ij} , κ^{rs} , etc.), which is zero if and only if the OD matrix is completely consistent with the demand functions. Compare your new measure with the total misplaced flow, and comment on any notable differences.
3. [31] Verify that the elastic demand objective function (8.16) is convex, given the assumptions made on the demand functions.

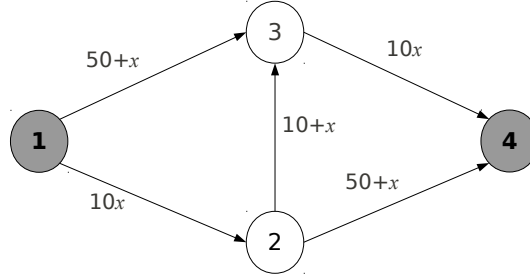


Figure 8.15: Network for Exercise 5.

4. [47] Using the network in Figure 8.15, solve the elastic demand equilibrium problem with demand given by $d^{14} = 15 - \kappa^{14}/30$. Perform three iterations of the Frank-Wolfe method, and report the average excess cost and total misplaced flow for the solution.
5. [48] Using the network in Figure 8.15, solve the elastic demand equilibrium problem with demand given by $d^{14} = 10 - \kappa^{14}/30$.
 - (a) Perform three iterations of the Frank-Wolfe method designed for elastic demand (Section 8.1.5).
 - (b) Transform the problem to an equivalent fixed-demand problem using the Gartner transformation from Section 8.1.2, and perform three iterations of the original Frank-Wolfe method.
 - (c) Compare the performance of these two methods: after three iterations, which is closer to satisfying the equilibrium and demand conditions?
6. [49] Using the network in Figure 8.16, solve the elastic demand equilibrium problem with demand functions $q^{19} = 1000 - 50(u^{19} - 50)$ and $q^{49} = 1000 - 75(u^{49} - 50)$. The cost function on the light links is $3 + (x_a/200)^2$, and the cost function on the thick links is $5 + (x_a/100)^2$. 1000 vehicles are traveling from node 1 to 9, and 1000 vehicles from node 4 to node 9. Perform three iterations of the Frank-Wolfe method and report the link flows and OD matrix.
7. [13] Assume that (\mathbf{x}, \mathbf{d}) and $(\mathbf{x}^*, \mathbf{d}^*)$ are both feasible solutions to an elastic demand equilibrium problem. Show that $(\lambda\mathbf{x} + (1 - \lambda)\mathbf{x}^*, \lambda\mathbf{d} + (1 - \lambda)\mathbf{d}^*)$ is also feasible if $\lambda \in [0, 1]$. This ensures that the Frank-Wolfe solutions are always feasible, assuming we start with \mathbf{x} and \mathbf{d} values which are consistent with each other, and always choose targets in a consistent way.
8. [57] (Calculating derivative formulas.) Let \mathbf{x} and \mathbf{d} be the current, feasible, link flows and OD matrix, and let \mathbf{x}^* and \mathbf{d}^* be any other feasible link flows and OD matrix. Let $\mathbf{x}' = \lambda\mathbf{x}^* + (1 - \lambda)\mathbf{x}$ and $\mathbf{d}' = \lambda\mathbf{d}^* + (1 - \lambda)\mathbf{d}$.

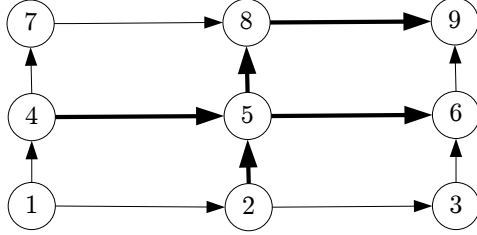


Figure 8.16: Network for Exercise 6.

- (a) Let $f(\mathbf{x}', \mathbf{d}')$ be the objective function for the elastic demand equilibrium problem. Recognizing that \mathbf{x}' and \mathbf{d}' are functions of λ , calculate $df/d\lambda$.
- (b) For the elastic demand problem, show that $\frac{df}{d\lambda} \Big|_{\lambda=0} \leq 0$ if \mathbf{d}^* and \mathbf{x}^* are chosen in the way given in the text. That is, the objective function is nonincreasing in the direction of the “target.” (You can assume that the demand function values are strictly positive if that would simplify your proof.)
9. [24] Consider a two-link network. For each pair of link performance functions shown below, determine whether or not the symmetry condition (8.26) is satisfied.
- $t_1 = 4 + 3x_1 + x_2, t_2 = 2 + x_1 + 4x_2$
 - $t_1 = 7 + 3x_1 + 4x_2, t_2 = 12 + 2x_1 + 4x_2$
 - $t_1 = 4 + x_1 + 3x_2, t_2 = 2 + 3x_1 + 2x_2$
 - $t_1 = 3x_1^2 + x_2, t_2 = 4 + x_1 + 4x_2^3$
 - $t_1 = 3x_1^2 + 2x_2^2, t_2 = 2x_1^2 + 3x_2^3$
 - $t_1 = 50 + x_1, t_2 = 10x_2$
10. [34] Determine which of the pairs of link performance functions in the previous exercise are strictly monotone.
11. [49] Consider the network in Figure 8.16 with a fixed demand of 1000 vehicles from 1 to 9 and 1000 vehicles from 4 to 9. The link performance function on every link a arriving at a “merge node” (that is, nodes 5, 6, 8, and 9) is $4 + (x_a/150)^2 + (x_{a'}/300)^2$ where a' is the other link arriving at the merge. Verify that the link interactions are symmetric, and perform five iterations of the Frank-Wolfe method. Report the link flows and travel times.
12. [49] Consider the network in Figure 8.16 with a fixed demand of 1000 vehicles from 1 to 9 and 1000 vehicles from 4 to 9. The link performance function on every link a arriving at a “merge node” (that is, nodes 5,

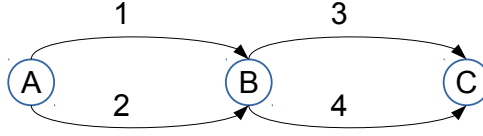


Figure 8.17: Network for Exercise 13.

6, 8, and 9) is $3 + (x_a/200)^2 + (x_{a'}/400)^2$ if the link is light, and $5 + (x_a/100)^2 + (x_{a'}/200)^2$ if the link is thick, where a' is the other link arriving at the merge. Show that not all link interactions are symmetric, and then perform five iterations of the diagonalization method and report the link flows, travel times, and relative gap.

13. [49] See the network in Figure 8.17, where the numbers indicate the label for each link. The link performance functions are:

$$t_1(\mathbf{x}) = 1 + 4x_1 + 2x_2$$

$$t_2(\mathbf{x}) = 2 + x_1 + 2x_2$$

$$t_3(\mathbf{x}) = 3 + 2x_3 + x_4$$

$$t_4(\mathbf{x}) = 2 + 3x_4$$

and the demand from node A to node C is 10 vehicles. For both methods below, start with an initial solution loading all flow on links 1 and 4.

- (a) Use three iterations of the diagonalization method to try to find an equilibrium solution, and report the link flows and average excess cost. (As before, this means finding three \mathbf{x}^* vectors *after* your initial solution.)
 - (b) Use three iterations of simplicial decomposition to try to find an equilibrium solution, and report the link flows and average excess cost. Three iterations means that \mathcal{X} should have three vectors in it when the algorithm terminates (unless it terminates early because the \mathbf{x}^* you find is already in \mathcal{X}). For each subproblem, make the number of improvement steps one less than the size of \mathcal{X} (so when \mathcal{X} has 1 vector, perform 0 steps; when it has 2 vectors, perform 1 step, and so on). For each of these steps, try the sequence of μ values $1/2, 1/4, 1/8, \dots$, choosing the first that reduces the restricted average excess cost.
14. [64] Prove Proposition 8.2.
15. [42] Show that the three path generation methods described in Section 8.3.2 indeed yield totally acyclic path sets, and that they always include the shortest path from the origin to the destination.

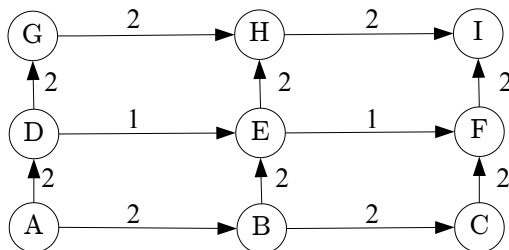


Figure 8.18: Network for Exercises 21 and 22.

16. [32] Show that any set of totally acyclic paths satisfies the segment substitution property. Then show that the set of all paths (including all cyclic paths) also satisfies this property.
17. [53] Complete the example following equation (8.40) by forming the infinite sum for the flow on links (2,3) and (3,2), and showing that they are equal to $1/2$.
18. [62] Reformulate the Markov property, and the results in Section 8.3.3 in terms of conditional probabilities for the link a vehicle used to *arrive* at a given node, rather than the link a vehicle will choose to depart a given node.
19. [34] In the logit formula (8.39), show that the same path choice probabilities are obtained if each link travel time t_{ij} is replaced with $L_i + t_{ij} - L_j$, where \mathbf{L} is a vector of node-specific constants. (In practice, these are usually the shortest path distances from the origin.)
20. [51] What would have to change in the “full cyclic path set” stochastic network loading procedure, if there were multiple links with the same tail and head nodes?
21. [35] Consider the network in Figure 8.18 with 1000 vehicles traveling from node A to node I, where the link labels are the travel times. Use the first criterion in Section 8.3.2 to define the path set, and identify the link flows corresponding to these travel times. Assume $\theta = 1$.
22. [35] Repeat Exercise 21 for the third definition of a path set. Assume $\theta = 1$.
23. [34] Show that the objective function (8.74) is strictly convex.
24. [49] Consider the network in Figure 8.16 with a fixed demand of 1000 vehicles from 1 to 9 and 1000 vehicles from 4 to 9. The light links have delay function $3 + (x_{ij}/200)^2$, and the dark links have delay function $5 +$

$(x_{ij}/100)^2$. Assume that drivers choose paths according to the stochastic user equilibrium principle, with $\theta = 1/5$. Perform three iterations of the method of successive averages, and report the link flows. Assume all paths are allowed.

25. [24] (Limitations of logit assignment). This problem showcases three “problem instances” for the stochastic network loading models described in this chapter (the logit model, and one proposed definition of allowable paths). Throughout, assume that the first definition in Section 8.3.2 is used to define the allowable path set. These instances motivated the development of probit and other, more sophisticated, stochastic equilibrium models.
- Consider the network in Figure 8.19(a), where the numbers by each link represents the travel time and $z \in (0, 1)$. Let p_{\uparrow} represent the proportion of vehicles choosing the top path. In a typical probit model, we have $p_{PROBIT}^{\uparrow} = 1 - \Phi\left(\sqrt{\frac{z}{2\pi-z}}\right)$, where $\Phi(\cdot)$ is the standard cumulative normal distribution function. Calculate p_{LOGIT}^{\uparrow} for the *logit* model as a function of z , and plot p_{PROBIT}^{\uparrow} and p_{LOGIT}^{\uparrow} as z varies from 0 to 1. Which do you think is more realistic, and why?
 - Consider networks of the type shown in Figure 8.19(b), where there is a top path consisting of a single link, and a number of bottom paths. The network is defined by an integer m ; there are $m - 1$ intermediate nodes in the bottom paths, and each consecutive pair of intermediate nodes is connected by two parallel links with travel time $8/m$. In a common probit model, $p_{PROBIT}^{\uparrow} = \Phi(-0.435\sqrt{m})$. What is p_{LOGIT}^{\uparrow} as a function of m ? Again create plots for small values of m (say 1 to 8), indicate which you think is more realistic, and explain why.
 - In the networks in Figure 8.19(c), identify the proportion of travelers choosing each link if $\theta = 1$. The left and right panels show a network before and after construction of a new link. Again identify the proportion of travelers choosing each link if $\theta = 1$. Do your findings seem reasonable?

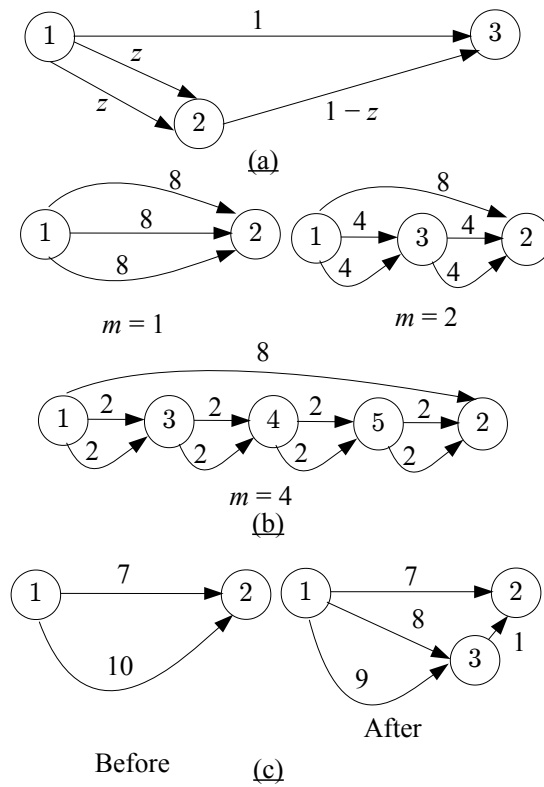


Figure 8.19: Networks for Exercise 25. The label on each link is its *constant* travel time.

Part III

**Dynamic Traffic
Assignment**

Chapter 9

Network Loading

This chapter discusses *network loading*, the process of modeling the state of traffic on a network, given the route and departure time of every vehicle. In static traffic assignment, this is a straightforward process based on evaluating link performance functions. In dynamic traffic assignment, however, network loading becomes much more complicated due to the additional detail in the traffic flow models — but it is exactly this complexity which makes dynamic traffic assignment more realistic than static traffic assignment. Rather than link performance functions, dynamic network loading models generally rely on some concepts of traffic flow theory. There are a great many theories, and an equally great number of dynamic network loading models, so this chapter will focus on those most commonly used.

To give the general flavor of network loading, we start with two simple *link models*, the point queue and spatial queue (Section 9.1), which describe traffic flow on a single link. We next present three simple *node models* describing how traffic streams behave at junctions (Section 9.2). With these building blocks we can perform network loading on simple networks, showing how link and node models interact to represent traffic flow in a modular way (Section 9.3).

However, the point queue and spatial queue models have significant limitations in representing traffic flow. The most common network loading models for dynamic traffic assignment are based on the hydrodynamic model of traffic flow, reviewed in Section 9.4. This theory is based in fluid mechanics, and assumes that the traffic stream can be modeled as the motion of a fluid, but it can be derived from certain car-following models as well, which have a more behavioral basis. The cell transmission model and link transmission model are link models based on this theory, and both of these are discussed in Section 9.5. Section 9.6 concludes the chapter with a discussion of more sophisticated node models that can represent general intersections.

This chapter aims to present several alternative network loading schemes as part of the general dynamic traffic assignment framework in Figure 1.8, so that any of them can be combined with the other steps in a flexible way.

9.1 Link Model Concepts

Link models are efficient ways to represent congestion and traffic flow. Efficiency is key, since link models must be implemented on each link in a large network over the entire analysis period; and furthermore, due to the iterative nature of solving for equilibrium, the network loading must be repeated many times with different path flow values as input. This section presents the basic concepts of link models, and demonstrates them in the point queue and spatial queue models. More sophisticated link models will be presented later in this chapter, in Section 9.5.

So, in this section, we are solely concerned with a single link. Call the length of this link L , so $x = 0$ corresponds to the upstream end of the link, and $x = L$ corresponds to the downstream end. All of the link models we discuss in this book operate in *discrete time*. That is, we divide the analysis period into small time intervals of length Δt , and assume uniform conditions within each time interval. These time intervals are very small relative to the analysis period; at a minimum, no vehicle should be able to traverse more than one link in a single time interval, so Δt can be no greater than the shortest free-flow travel time on a link. In practice, Δt values on the order of 5–10 seconds are a reasonable choice. For convenience in this section, we assume that the unit of time is chosen such that $\Delta t = 1$, so we will only track the state of the network for times $t \in \{0, 1, 2, \dots, \bar{T}\}$, where \bar{T} is the length of the analysis period (in units of Δt).

In any discrete time model, it is important to clarify what exactly we mean when we index a variable with a time interval, such as $N(t)$ or $y(t)$. Does this refer to the value of x at the start of the t -th time interval, the end of the interval, the average of a continuous value through the interval, or something else? This chapter will consistently use the following convention: when referring to a quantity *measured at a single instant in time*, such as the number of vehicles on a link, or the instantaneous speed of a vehicle, we will take such measurements at the *start* of a time interval. When referring to a quantity *measured over time*, such as the number of vehicles passing a fixed point on the link, we take such measurements *over* the time interval. (Figure 9.1) This distinction can also be expressed by referring, say, to the number of vehicles on a link *at time* t (meaning the start of the t -th interval), or the number of vehicles that exit a link *during* time t .

We load the network in increasing order of time. That is, we start with the network state at $t = 0$ (usually assuming an empty condition). Then, with these values known, we compute the network state at $t = 1$, then at $t = 2$, and so forth. That is, for any point in time, any values at an earlier point in time can be treated as “known” values.

The definition of the “network state” at a timestep is intentionally left a bit vague at this point. It contains all information necessary for modeling traffic flow (e.g., the locations of vehicles, traffic signal indications). The collection of network states at all time steps must also have enough information to check consistency with the equilibrium principle (cf. Section 1.3) after the network loading is complete, and for adjusting vehicle flows if equilibrium is not satisfied.

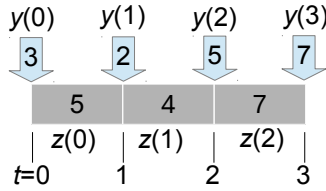


Figure 9.1: Convention for indexing discrete time intervals for two hypothetical variables y (measured at an instant) and z (measured over time).

The exact components of the network state vary from one model to the next (for instance, signals may be modeled in detail, approximately, or not at all), and as you read about the link and node models presented in this chapter, you should think about what information you would need to store in order to perform the computations for each link and node model. At a minimum, it is common to record the cumulative number of vehicles which have entered and left each link at each timestep, since the start of the modeling period. These values are denoted by $N^\uparrow(t)$ and $N^\downarrow(t)$, respectively; the arrows are meant as a mnemonic for “upstream” and “downstream”, since they can be thought of as counters at the ends of the links. So, for instance, $N^\uparrow(3)$ is the number of vehicles which have crossed the upstream end of the link by the start of the 3rd time interval (the cumulative entries), and $N^\downarrow(5)$ is the number of vehicles which have crossed the downstream end of the link by the start of the 5th time interval (cumulative exits). We will assume that the network is empty when $t = 0$ (no vehicles anywhere), and as a result $N^\uparrow(t) - N^\downarrow(t)$ gives the number of vehicles currently on the link at any time t .

It is possible to use different time interval lengths for different links and nodes, and this can potentially reduce the necessary computation time. There are also *continuous time* dynamic network loading models, where flows and other traffic variables are assumed to be functions defined for any real time value, not just a finite number of points. These are not discussed here to keep the focus on the basic network loading ideas, and to avoid technical details associated with infinitesimal calculations. Finally, some formulas may call for the value of a discrete variable at a non-integer point in time, such as $N^\uparrow(3.4)$, in which case a linear interpolation can be used between the neighboring values $N^\uparrow(3)$ and $N^\uparrow(4)$. If possible, the time step should be chosen to minimize or eliminate these interpolation steps, which are time-consuming and which can introduce numerical errors.

9.1.1 Sending and receiving flow

The main outputs of a link model are the *sending flow* and *receiving flow*, calculated for each discrete time interval. The sending flow at time t , denoted $S(t)$, is the number of vehicles which would leave the link during the t -th time

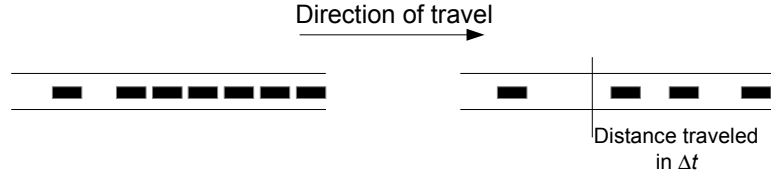


Figure 9.2: Calculating the sending flow when there is a queue on the link (left) and when the link is at free-flow (right).

interval (that is, between times t and $t + 1$) if there was no obstruction from downstream links or nodes (you can imagine that the link is connected to a wide, empty link downstream). You can also think of this as the flow that is ready to leave the link during this time interval. The sending flow is calculated at the downstream end of a link.

To visualize sending flow, Figure 9.2 shows two examples of links. In the left panel, there is a queue at the downstream end of the link. If there is no restriction from downstream, the queue would discharge at the full capacity of the link, and the sending flow would be equal to the capacity of the link multiplied by Δt . In the right panel, the link is uncongested and vehicles are traveling at free-flow speed. The sending flow will be less than the capacity, because relatively few vehicles are close enough to the downstream end of the link to exit within the next time interval. The vertical line in the figure indicates the distance a vehicle would travel at free-flow speed during one time step, so in this case the sending flow would be 3 vehicles. Note that the *actual* number of vehicles which can leave the link during the next time step may be affected by downstream conditions: perhaps the next link is congested, or perhaps there is a traffic signal at the downstream end. These considerations are irrelevant for calculating sending flow, and will be treated with node models, introduced in Section 9.2. In any case, the sending flow is an upper bound on the actual number of vehicles which can depart the link.

The receiving flow during time t , denoted $R(t)$, is the number of vehicles which would enter the link during the t -th time interval if the upstream link could supply a very large (even infinite) number of vehicles: you can imagine that the upstream link is wide, and completely full at jam density. You can also think of this as the maximum amount of flow which can enter the link during this interval, taking into account the available free space. The receiving flow is calculated at the upstream end of the link.

To visualize receiving flow, Figure 9.3 shows two examples of links. In the left panel, the upstream end of the link is empty. This means that vehicles can potentially enter the link at its full capacity, and the receiving flow would equal the link's capacity multiplied by Δt . The actual number of vehicles which will enter the link may be less than this, if there is little demand from upstream — like the sending flow, the receiving flow is simply an upper bound indicating how

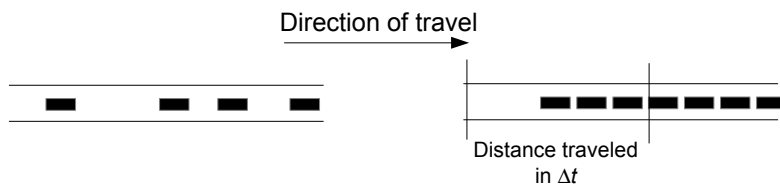


Figure 9.3: Calculating the receiving flow when the link is at free-flow (left) and when there is a queue on the link (right).

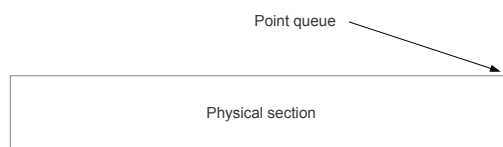


Figure 9.4: Division of a link in point queue models.

many vehicles could potentially enter the link if demand were high enough. In the right panel, there is a stopped queue which nearly fills the entire link. Here the vertical line indicates how far into the link a vehicle would travel at free-flow speed during one time step. Assuming that the stopped vehicles remain stopped throughout the t -th time interval, the receiving flow is the number of vehicles which can physically fit into the link, in this case 2.

Each link model has a slightly different way of calculating the sending and receiving flows, which correspond to different assumptions on traffic behavior with the link, or to different calculation methods. The next two subsections present simple link models. Notice how the different traffic flow assumptions in these models lead to different formulas for sending and receiving flow.

9.1.2 Point queue

Point queue models divide each link into two sections:

- A **physical section** which spans the length of the link and is assumed uncongestible: vehicles will always travel over this section at free-flow speed.
- A **point queue** at the downstream end of the link which occupies no physical space, but conceptually holds vehicles back to represent any congestion delay on the link.

These are shown in Figure 9.4.

Point queues can be thought of in several ways. One can imagine a wide link which necks down at its downstream end — the physical section reflects the

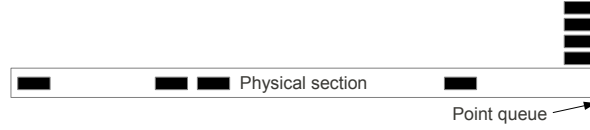


Figure 9.5: Envisioning a point queue as vehicles stacking vertically.

wide portion of the link, and the point queue represents the vehicles which are delayed as the capacity is reduced at the downstream bottleneck. One can also imagine a traffic signal at the downstream end, and magical technology (flying cars?) which allows vehicles to “stack” vertically at the signal — there can be no congestion upstream of this “stack,” since vehicles can always fly to the top. (Figure 9.5) One may even imagine that there is no physical meaning to either of these, and that the physical section and point queue merely represent the delays incurred from traveling the link at free-flow, and the additional travel time due to congestion.

The point queue discharges vehicles at a maximum rate of q_{max}^\downarrow (measured in vehicles per unit time), called the *capacity*. The capacity imposes an upper limit on the sending flow, so we always have

$$S(t) \leq q_{max}^\downarrow \Delta t. \quad (9.1)$$

However, if the queue is empty, or if only a few vehicles are in the queue, the discharge rate may be less than this. Once the queue empties, the only vehicles which can exit the link are ones reaching the downstream end from the uncongested physical section. Since we assume that all vehicles in this section travel at the free-flow speed (which we will denote u_f), this means that only the vehicles that are closer than $u_f \Delta t$ to the downstream end can possibly leave.

We can use the cumulative counts N^\uparrow and N^\downarrow to count the number of vehicles which are close enough to the downstream end to exit in the next time step. Since the entire physical section is traversed at the free-flow speed u_f , a vehicle whose distance from the downstream end of the link is exactly $u_f \Delta t$ distance units must have passed the upstream end of the link exactly $(L - u_f \Delta t)/u_f$ time units ago. We call this a “threshold” vehicle, since any vehicle entering the link after this one has not yet traveled far enough, while any vehicle entering the link before this one is close enough to the downstream end to exit. The number of vehicles between the threshold vehicle and the downstream end of the link can thus be given by

$$N^\uparrow \left(t - \frac{L - u_f \Delta t}{u_f} \right) - N^\downarrow(t) = N^\uparrow \left(t + \Delta t - \frac{L}{u_f} \right) - N^\downarrow(t). \quad (9.2)$$

The sending flow is the smaller of the number of vehicles which are close enough to the downstream end to exit, given by equation (9.2), and the capacity of the queue. Thus

$$S(t) = \min \{ N^\uparrow(t + \Delta t - L/u_f) - N^\downarrow(t), q_{max}^\downarrow \Delta t \}. \quad (9.3)$$

The receiving flow for the point queue model is easy to calculate. Since the physical section is uncongestible, the link capacity is the only limitation on the rate at which vehicles can enter. The capacity of the upstream end of the link may be different than the capacity of the downstream end of the link (perhaps due to a lane drop, or a stop sign at the end of the link), so we denote the capacity of the upstream end by q_{max}^\uparrow . (If we just use q_{max} without an \uparrow or \downarrow superscript, the same capacity applies over the whole link.) The receiving flow is given by

$$R(t) = q_{max}^\uparrow \Delta t. \quad (9.4)$$

In real traffic networks, queues occupy physical space and cannot be confined to a single point. The spatial queue model in the next subsection shows one way to reflect this.

Table 9.1 shows how the point queue model operates, depicting the state of a link over ten time steps. The N^\uparrow and N^\downarrow columns express the number of vehicles which have entered and left the link at each time step, as well as the sending and receiving flows during each timestep. The difference between N^\uparrow and N^\downarrow represents the number of vehicles on the link at any point in time. In this example, we assume that the free-flow speed is $u_f = L/(3\Delta t)$ (so a vehicle takes 3 time steps to traverse the link under free-flow conditions), the upstream capacity is $q_{max}^\uparrow = 10/\Delta t$, and the downstream capacity is $q_{max}^\downarrow = 5/\Delta t$. Initially, the sending flow is zero, because no vehicles have reached the downstream end of the link. The sending flow then increases as flow exits, but eventually reaches the downstream capacity. At this point, a queue forms and vehicles exit at the downstream capacity rate. Eventually, the queue clears, the link is empty, and the sending flow returns to zero. The receiving flow never changes from the upstream capacity, even when a queue is present. In this example, notice that $N^\downarrow(t+1) = N^\downarrow(t) + S(t)$. This happens because we are temporarily ignoring what might be happening from downstream. Depending on downstream congestion, $N^\downarrow(t+1)$ could be less than $N^\downarrow(t) + S(t)$; but it could never be greater, because the sending flow is always a limit on the number of vehicles that can exit. Also notice that for all time steps, $N^\uparrow(t+1) \leq N^\uparrow(t) + R(t)$, because the receiving flow is a limit on the number of vehicles that can enter the link.

In practice, the upstream and downstream capacities are usually assumed the same, in which case we just use the notation q_{max} to refer to capacities at both ends. The network features which would make the capacities different upstream and downstream (such as stop signs or signals) are usually better represented with node models, discussed in the next section.

9.1.3 Spatial queue

The spatial queue model is similar to the point queue model, except that a maximum queue length is now enforced. When the queue reaches this maximum length, no further vehicles are allowed to enter the link. That is, the queue now occupies physical space, and because the link is finite in length, the link can

Table 9.1: Point queue example, with $u_f = L/(3\Delta t)$, $q_{max}^\uparrow = 10/\Delta t$, and $q_{max}^\downarrow = 5/\Delta t$.

t	N^\uparrow	N^\downarrow	R	S
0	0	0	10	0
1	1	0	10	0
2	5	0	10	0
3	10	0	10	1
4	17	1	10	4
5	27	5	10	5
6	30	10	10	5
7	30	15	10	5
8	30	20	10	5
9	30	25	10	5
10	30	30	10	0

become completely blocked. This will result in *queue spillback*, as vehicles on upstream links will be unable to enter the blocked link. Like the point queue model, we assume that the queue is always at the downstream end of the link: there is at most one uncongested physical section at the upstream end of the link, and at most one stopped queue at the downstream end, in that order. This is still a simplification of real traffic (where links can have multiple congested and uncongested sections), but by allowing the length of the physical section to shrink as the queue grows, one can model the queue spillback phenomenon which is common in congested networks.

The sending flow for the spatial queue model is calculated in exactly the same way as for the point queue model: the smaller of the number of vehicles close enough to the downstream end of the link to exit in the next time step, and the capacity of the link:

$$S(t) = \min\{N^\uparrow(t + \Delta t - L/u_f) - N^\downarrow(t), q_{max}^\downarrow \Delta t\}. \quad (9.5)$$

The receiving flow includes an additional term to reflect the finite space on the link for the queue, alongside the link capacity. Since vehicles in the queue are stopped, the space they occupy is given by the jam density k_j , expressed in vehicles per unit length. The maximum number of vehicles the link can hold is $k_j L$, while the number of vehicles currently on the link is $N^\uparrow(t) - N^\downarrow(t)$. The receiving flow cannot exceed the difference between these:

$$R(t) = \min\{k_j L - (N^\uparrow(t) - N^\downarrow(t)), q_{max}^\uparrow \Delta t\}. \quad (9.6)$$

By assuming that the queue is always at the downstream end of the link, the spatial queue model essentially assumes that all vehicles in a queue move together. In reality, there is some delay between when the head of the queue starts moving, and when the vehicle at the tail of the queue starts moving —

Table 9.2: Spatial queue example, with $u_f = L/(3\Delta t)$, $q_{max}^\uparrow = 10/\Delta t$, $q_{max}^\downarrow = 5/\Delta t$, and $k_j L = 20$.

t	N^\uparrow	N^\downarrow	R	S
0	0	0	10	0
1	1	0	10	0
2	5	0	10	0
3	10	0	10	1
4	17	1	4	4
5	21	5	4	5
6	25	10	5	5
7	30	15	5	5
8	30	20	10	5
9	30	25	10	5
10	30	30	10	0

when a traffic light turns green, vehicles start moving one at a time, with a slight delay between when a vehicle starts moving and when the vehicle behind it starts moving. These delays cannot be captured in a spatial queue model. To represent this behavior, we will need a better understanding of traffic flow theory. Section 9.4 will present this information, and we will ultimately build more realistic link models. The point queue and spatial queue models nevertheless illustrate the basic principles of link models, and what the sending and receiving flow represent.

Table 9.2 shows how the spatial queue model operates. It is similar to the point queue example (Table 9.1), but we now introduce a jam density, assuming that the maximum number of vehicles which can fit on the link is $k_j L = 20$. In this example, the receiving flow drops once the queue reaches a certain size. This reflects the finite space available on the link. As a result, it takes longer for the 30 vehicles to enter the link. The queue is still able to completely discharge by the end of the ten time steps. As before, the difference between $N^\downarrow(t)$ and $N^\downarrow(t+1)$ is never more than $S(t)$ (in this example, exactly equal because we are ignoring downstream conditions), and the difference between $N^\uparrow(t)$ and $N^\uparrow(t+1)$ is never more than $R(t)$.

9.2 Node Model Concepts

Node models complement the link models discussed in the previous section, by representing how flows between different links interact with each other. By focusing on sending and receiving flows from each link, one can separate the different processes used to model traffic flow within a single link, and the interactions between links at junctions. Calculations of sending and receiving flows can be done independently, in parallel, for each link. It is even possible to use different link models on different links (say, one link with a spatial queue model

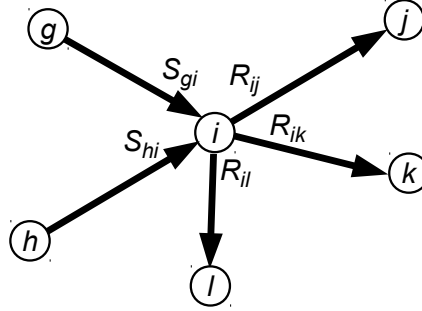


Figure 9.6: Sending and receiving flows interact at nodes to produce turning movement flows.

and another with a point queue).

All that a node model needs is the sending flow of each of the links which enter this node, and the receiving flow for each link leaving this node. (Figure 9.6) That is, when processing node i at time t , we assume that $S_{hi}(t)$ and $R_{ij}(t)$ have respectively been calculated for each incoming link $(h, i) \in \Gamma^{-1}(i)$ and each outgoing link $(i, j) \in \Gamma(i)$. The task is to determine how many vehicles move from each incoming link to each outgoing link during the t -th time interval; denote this value by $y_{hij}(t)$. This is called the *turning movement flow* from (h, i) to (i, j) , or more compactly, the turning movement $[h, i, j]$. Let $\Xi(i)$ denote the set of allowable turning movements at node i ; this provides a natural way to model turn prohibitions, U-turn prohibitions, and so forth.

As an example of this notation, consider node i in Figure 9.6. If all of the turning movements are allowed, then $\Xi(i)$ has six elements: $[g, i, j]$, $[g, i, k]$, $[g, i, l]$, $[h, i, j]$, $[h, i, k]$, and $[h, i, l]$. This set might not include all six elements — for instance, if the left turn from approach (g, i) to (i, j) is prohibited, then $\Xi(i)$ would exclude $[g, i, j]$.

Many different node models can be used. This section discusses three simple node models — links in series, diverges, and merges — to illustrate the general concepts. Later in this chapter, we discuss node models that can be used for signalized intersections, all-way stops, and other types of intersections. There are many variations of all of these, but they share some common principles, which are discussed here.

Any node model must satisfy a number of constraints and principles, for all time intervals t :

1. Vehicles will not voluntarily hold themselves back. That is, it should be impossible to increase any turning movement flow $y_{hij}(t)$ without violating one of the constraints listed below, or one of the other constraints imposed by a specific node model.
2. Each turning movement flow must be nonnegative, that is, $y_{hij}(t) \geq 0$ for

each $[h, i, j] \in \Xi(i)$.

3. Any turning movement which is not allowable has zero flow, that is, $y_{hij}(t) = 0$ whenever $[h, i, j] \notin \Xi(i)$.
4. For each incoming link, the sum of the turning movement flows out of this link cannot exceed the sending flow, since by definition those are the only vehicles which could possibly leave that link in the next time step:

$$\sum_{(i,j) \in \Gamma(i)} y_{hij}(t) \leq S_{hi}(t) \quad \forall (h, i) \in \Gamma^{-1}(i) \quad (9.7)$$

The sum on the left may be less than $S_{hi}(t)$, because it is possible that some vehicles cannot leave (h, i) due to obstructions from a downstream link or from the node itself (such as a red signal).

5. For each outgoing link, the sum of the turning movement flows into this link cannot exceed the receiving flow, since that is the maximum number of vehicles that link can accommodate:

$$\sum_{(h,i) \in \Gamma^{-1}(i)} y_{hij}(t) \leq R_{ij}(t) \quad \forall (i, j) \in \Gamma(i) \quad (9.8)$$

The sum on the left may be less than $R_{ij}(t)$, because there may not be enough vehicles from upstream links to fill all of the available space in the link.

6. Route choices must be respected. That is, the values $y_{hij}(t)$ must be compatible with the directions travelers wish to go based on their chosen paths; we cannot reassign them on the fly in order to increase a y_{hij} value.
7. The *first-in, first-out (FIFO) principle* must be respected. This is closely related to the previous property; we cannot allow vehicles to “jump ahead” in queue to increase a y_{hij} value, unless there is a separate turn lane or other roadway geometry which can separate vehicles on different routes. (Figure 9.7).
8. The *invariance principle* must be respected. If the outflow from a link is less than its sending flow, then increasing the sending flow further could not increase the outflow beyond its current value. Likewise, if the inflow to a link is less than its receiving flow, then increasing the receiving flow could not increase the inflow to the link beyond its current value. In other words, if the sending (or receiving) flow is not “binding,” then its specific value cannot matter for the actual flows.

The invariance principle warrants additional explanation. In the first case, assume that flow into link (i, j) is restricted by its receiving flow R_{ij} . This means that more vehicles wish to enter the link than can be accommodated, so a queue must form on the upstream link. But when there is a queue on the

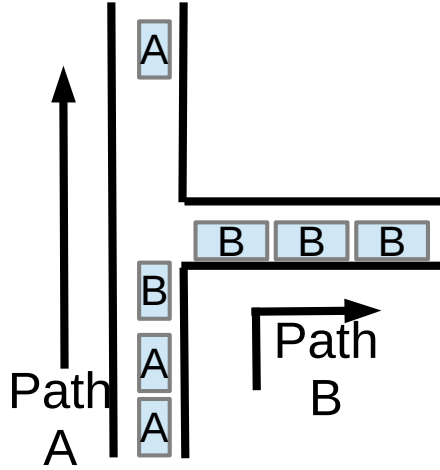


Figure 9.7: The first-in, first-out principle implies that vehicles waiting to turn will obstruct vehicles further upstream.

upstream link, its sending flow will increase to the capacity $q_{max}^{hi}\Delta t$. If the flow y_{hij} increases in response, then the queue might be cleared in the next time interval, the sending flow would drop, a new queue would form, and so on, with the flows oscillating between time intervals. This is unrealistic, and is an artifact introduced by choosing a particular discretization, not a traffic phenomenon one would expect in the field. In the second case, if flow onto link (i, j) is restricted by the sending flow S_{hi} , then there is more space available on link (i, j) than vehicles wish to enter, a short time later the receiving flow will increase to the capacity $q_{max}^{ij}\Delta t$. If y_{hij} would increase because of this increase in the receiving flow, again an unrealistic oscillation would be seen. The exercises ask you to compare some node models which violate the invariance principle to those which do not.

9.2.1 Links in series

The simplest node to model is one with exactly one incoming link, and exactly one outgoing link. (Figure 9.8). This may seem like a trivial node, since there is no real “intersection” here. However, they are often introduced to reflect changes within a link. For instance, if a freeway reduces from three lanes in a direction to two lanes, this reduction in capacity and jam density can be modeled by introducing a node at the point where the lane drops. In this way, each link can have a homogeneous capacity and jam density, and we can simplify the notation — in the link models above, we distinguished between q_{max}^{\uparrow} and q_{max}^{\downarrow} at the two ends of the link. Now we can just use q_{max} for the entire link,

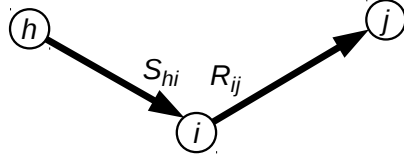


Figure 9.8: Two links in series.

including both ends.

For concreteness, let the incoming link be (h, i) and the outgoing link be (i, j) . In this case, there is only one turning movement, so $\Xi(i) = \{[h, i, j]\}$, and the only turning movement flow we need concern ourselves with is $y_{hij}(t)$. In the case of two links in series, the formula is simple:

$$y_{hij}(t) = \min\{S_{hi}(t), R_{ij}(t)\}, \quad (9.9)$$

that is, the number of vehicles moving from link (h, i) to (i, j) during the t -th time interval is the lesser of the sending flow from the upstream link in that time interval, and the receiving flow of the downstream link. For instance, if the upstream link sending flow is 10 vehicles, while the downstream link receiving flow is 5 vehicles, a total of 5 vehicles will successfully move from the upstream link to the downstream one, because that is all there is space for. If the upstream link sending flow is 3 vehicles and the downstream link receiving flow is 10 vehicles, 3 vehicles will move from the upstream link to the downstream one, because that is all the vehicles that are available.

We can check that this node model satisfies all of the desiderata from Section 9.2. Going through each of these conditions in turn:

1. The flow $y_{hij}(t)$ is chosen to be the minimum of the upstream sending flow, and the downstream receiving flow; any value larger than this would violate either the sending flow constraint or the receiving flow constraint.
2. The sending and receiving flows should both be nonnegative, regardless of the link model, so $y_{hij}(t)$ is as well.
3. There is only one turning movement, so this constraint is trivially satisfied.
4. The formula for $y_{hij}(t)$ ensures it cannot be greater than the upstream sending flow. (This condition simplifies since there is only one incoming and outgoing link, so the summation and “for all” quantifier can be disregarded.)
5. The formula for $y_{hij}(t)$ ensures it cannot be greater than the downstream receiving flow. (This condition simplifies in the same way.)

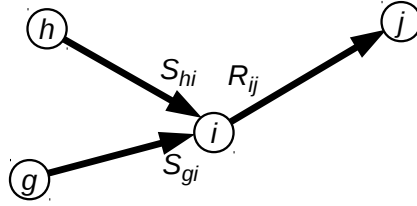


Figure 9.9: Prototype merge node.

6. Route choice is irrelevant when two links meet in series, since all incoming vehicles must exit by the same link.
7. FIFO is also irrelevant, since all vehicles entering the node behave in the same way. (This would not be the case if there was more than one exiting link, and vehicles were on different paths.) So we don't have to worry about FIFO when calculating the turning movement flow.
8. To see that the formula satisfies the invariance principle, we have to check two conditions. If the outflow from (h, i) is less than the sending flow, this means that $y_{hij}(t) = R_{ij}(t)$, and the second term in the minimum of equation (9.9) is binding. Increasing the sending flow (the first term in the minimum) further would not affect its value. Similarly, if the inflow to (i, j) is less than its receiving flow, this means that $y_{hij}(t) = S_{hi}(t)$, and the first term in the minimum is binding. Increasing the receiving flow (the second term) would not affect its value either.

9.2.2 Merges

A merge node has only one outgoing link (i, j) , but more than one incoming link, here labeled (g, i) and (h, i) , as in Figure 9.9. This section only concerns itself with the case of only two upstream links, and generalizing to the case of additional upstream links is left as an exercise. Here $\Xi(i) = \{[g, i, j], [h, i, j]\}$, and we want to calculate the rate of flow from the upstream links to the downstream one, that is, the flow rates $y_{gij}(t)$ and $y_{hij}(t)$. As you might expect, the main quantities of interest are the upstream sending flows $S_{gi}(t)$ and $S_{hi}(t)$, and the downstream receiving flow $R_{ij}(t)$. We assume that these values have already been computed by applying a link model.

For brevity, we will omit the time index in the rest of this section — it is implicit that all calculations are done with the sending and receiving flows at the current time step.

There are three possibilities, one corresponding to free flow conditions at the merge, one corresponding to congestion with queues growing on both upstream links, and one corresponding to congestion on only one upstream link. For the merge to be freely flowing, both upstream links must be able to transmit all of the flow which seeks to leave them, and the downstream link must be able to

accommodate all of this flow. Mathematically, we need $S_{gi} + S_{hi} \leq R_{ij}$, and if this is true then we simply set $y_{gij} = S_{gi}$, and $y_{hij} = S_{hi}$.

In the second case, there is congestion (so $S_{gi} + S_{hi} > R_{ij}$), and furthermore, flow is arriving fast enough on both upstream links for a queue to form at each of them. Empirically, in such cases the flow rate from the upstream links is approximately proportional to the capacity on these links, that is,

$$\frac{y_{gij}}{y_{hij}} = \frac{q_{max}^{gi}}{q_{max}^{hi}} \quad (9.10)$$

A little thought should convince you that this relationship is plausible. Furthermore, in the congested case, all of the available downstream capacity will be used, so

$$y_{gij} + y_{hij} = R_{ij} \quad (9.11)$$

Substituting (9.10) into (9.11) and solving, we obtain

$$y_{gij} = \frac{q_{max}^{gi}}{q_{max}^{gi} + q_{max}^{hi}} R_{ij} \quad (9.12)$$

with a symmetric expression for y_{hij} .

The third case is perhaps a bit unusual. The merge is congested ($S_{gi} + S_{hi} > R_{ij}$), but a queue is only forming on one of the upstream links. This may happen if the flow on one of the upstream links is much less than the flow on the other. In this case, the proportionality rule allows all of the sending flow from one link to enter the downstream link, with room to spare. This “spare capacity” can then be consumed by the other approach. If link (g, i) is the link which cannot send enough flow to meet the proportionality condition, so that

$$S_{gi} < \frac{q_{max}^{gi}}{q_{max}^{gi} + q_{max}^{hi}} R_{ij}, \quad (9.13)$$

then the two flow rates are $y_{gij} = S_{gi}$ and $y_{hij} = R_{ij} - S_{gi}$: one link sends all of the flow it can, and the other link consumes the remaining capacity. The formulas are reversed if it is link (h, i) that cannot send enough flow to meet its proportionality condition.

Exercise 7 asks you to show that the second and third cases can be handled by the single equation

$$y_{gij} = \text{med} \left\{ S_{gi}, R_{ij} - S_{hi}, \frac{q_{max}^{gi}}{q_{max}^{gi} + q_{max}^{hi}} R_{ij} \right\}, \quad (9.14)$$

where med refers to the median of a set of numbers. This formula applies whenever $S_{gi} + S_{hi} > R_{ij}$. An analogous formula holds for the other approach by swapping the g and h indices. Exercise 8 asks you to verify the desiderata of Section 9.2 are satisfied by this equation.

For certain merges, it may not be appropriate to assign flow proportional to the capacity of the incoming links. Rules of the road, signage, or signalization

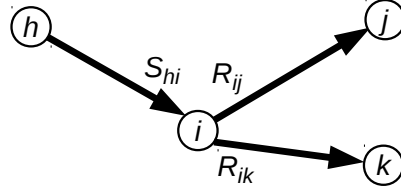


Figure 9.10: Prototype diverge node.

might allocate the capacity of the downstream link differently. In general, the share of the downstream receiving flow that is allocated to approaches (g, i) and (h, i) can be written as ψ_{gi} and ψ_{hi} , respectively, with $\psi_{gi} + \psi_{hi} = 1$ and both of them nonnegative. A more general form of the merge equation can then be written as

$$y_{gij} = \text{med} \{ S_{gi}, R_{ij} - S_{hi}, \psi_{gi} R_{ij} \}, \quad (9.15)$$

with a similar formula for (h, i) .

9.2.3 Diverges

A diverge node is one with only one incoming link (h, i) , but more than one outgoing link, as in Figure 9.10. This section concerns itself with the case of only two downstream links. The exercises ask you to generalize to the case of three downstream links, using the same concepts. Let these two links be called (i, j) and (i, k) , so $\Xi(i) = \{[h, i, j], [h, i, k]\}$. Our interest is calculating the rate of flow from the upstream link to the downstream ones, that is, the flow rates y_{hij} and y_{hik} . We assume that the sending flow S_{hi} and the receiving flows R_{ij} and R_{ik} have already been calculated. Unlike links in series or merges, we also need to represent some model of route choice, since some drivers may choose link (i, j) , and others link (i, k) . Let p_{ij} and p_{ik} be the proportions of drivers choosing these two links during the t -th time interval, respectively. Naturally, p_{ij} and p_{ik} are nonnegative, and $p_{ij} + p_{ik} = 1$. Like the sending and receiving flows, these values can change with time, but to avoid cluttering formulas we will leave the time indices off of p values unless it is unclear which time step we are referring to.

There are two possibilities, one corresponding to free flow conditions at the diverge, and the other corresponding to congestion. What does “free flow” mean? For the diverge to be freely flowing, *both* of the downstream links must be able to accommodate the flow which seeks to enter them. The rates at which vehicles want to enter the two links are $p_{ij}S_{hi}$ and $p_{ik}S_{hi}$, so if both downstream links can accommodate this, we need $p_{ij}S_{hi} \leq R_{ij}$ and $p_{ik}S_{hi} \leq R_{ik}$. In this case we simply have $y_{hij} = p_{ij}S_{hi}$ and $y_{hik} = p_{ik}S_{hi}$: all of the flow which wants to leave the diverge can.

The case of congestion is slightly more interesting, and requires making assumptions about how drivers will behave. One common assumption is that

flow waiting to enter one link at a diverge will obstruct every other vehicle on the link (regardless of which link it is destined for). This most obviously represents the case where the upstream link has only a single lane, so any vehicle which has to wait will block any vehicle behind it; but this model is commonly used even in other cases.¹ When there is congestion, only some fraction ϕ of the upstream sending flow can move. The assumption that any vehicle waiting blocks every vehicle upstream implies that this same fraction applies to both of the downstream links, so $y_{hij} = \phi p_{ij} S_{hi}$ and $y_{hik} = \phi p_{ik} S_{hi}$.

So, how to calculate ϕ ? The inflow rate to a link cannot exceed its receiving flow, so $y_{hij} = \phi p_{ij} S_{hi} \leq R_{ij}$ and $y_{hik} = \phi p_{ik} S_{hi} \leq R_{ik}$, or equivalently $\phi \leq R_{ij}/p_{ij} S_{hi}$ and $\phi \leq R_{ik}/p_{ik} S_{hi}$. Every vehicle which can move will, so

$$\phi = \min \left\{ \frac{R_{ij}}{p_{ij} S_{hi}}, \frac{R_{ik}}{p_{ik} S_{hi}} \right\} \quad (9.16)$$

Furthermore, we can introduce the uncongested case into this equation as well, and state

$$\phi = \min \left\{ \frac{R_{ij}}{p_{ij} S_{hi}}, \frac{R_{ik}}{p_{ik} S_{hi}}, 1 \right\} \quad (9.17)$$

regardless of whether there is congestion at the diverge or not. Why? If the diverge is at free flow, then $\phi = 1$, but $R_{ij}/p_{ij} S_{hi} \geq 1$ and $R_{ik}/p_{ik} S_{hi} \geq 1$. Introducing 1 into the minimum therefore gives the correct answer for free flow. Furthermore, if the diverge is not at free flow, then either $R_{ij}/p_{ij} S_{hi} < 1$ or $R_{ik}/p_{ik} S_{hi} < 1$, so adding 1 does not affect the minimum value. Therefore, this formula is still correct even in the congested case. Exercise 13 asks you to verify the desiderata of Section 9.2 are satisfied by this equation.

9.3 Combining Node and Link Models

This section describes how node and link models are combined, to complete the network loading process. We must also describe what happens at origins and destination (zone) nodes. For the algorithm in this section, we assume that the only links connected to zones are special links called *centroid connectors*, which do not represent a specific roadway so much as a collection of small local streets used by travelers entering or leaving a specific neighborhood. It is common to give centroid connectors between origins and ordinary nodes a very large (even infinite) jam density, and centroid connectors between ordinary nodes and destinations a very large (even infinite) capacity. In both cases the free flow time should be small. These considerations reflect the ideas that centroid connectors should not experience significant congestion (or else they should be modeled as proper links in the network), and simply convey flow from origin nodes and to destination nodes with as little interference as possible. It is common to forbid

¹For instance, this can represent drivers attempting to “queue jump” by cutting into the turn lane at the last moment. Or, if the turn lane is long, it may be appropriate to treat the diverge at the point where the turn lane begins, as opposed to at the physical diverge.

travelers from using centroid connectors except to start and end their trips, that is, to exclude the use of centroid connectors as “shortcuts.” This can be done either by transforming the underlying network, having origins and destinations be distinct nodes adjacent to “one-way” centroid connectors, or by excluding such paths when finding paths for travelers, as discussed in Chapter 10.

If centroid connectors are set up in this way, then flow entering the network can simply be added to the upstream ends of their centroid connectors, and flow leaving the network at destinations can simply vanish, without any constraints in either case. The network loading algorithm can then be stated as follows:

1. Initialize all counts and the time index: $N_{ij}^\uparrow(0) \leftarrow 0$ and $N_{ij}^\downarrow(0) \leftarrow 0$ for all links (i, j) , $t \leftarrow 0$.
2. Use a link model to calculate sending and receiving flows S_{ij} and R_{ij} for all links.
3. Use a node model to calculate transition flows y_{ijk} for all nodes j except for zones.
4. Update cumulative counts: for each non-zone node i , perform

$$N_{hi}^\downarrow(t+1) \leftarrow N_{hi}^\downarrow(t) + \sum_{(i,j) \in \Gamma(i)} y_{hij} \quad (9.18)$$

for each upstream link (h, i) , and

$$N_{ij}^\uparrow(t+1) \leftarrow N_{ij}^\uparrow(t) + \sum_{(h,i) \in \Gamma^{-1}(i)} y_{hij} \quad (9.19)$$

for each downstream link (i, j) .

5. Load trips: for all origins r , let $D_r(t) = \sum_{s \in Z} d^{rs}(t)$ be the total demand starting at this node, and for each centroid connector (r, i) , let p_{ri} be the fraction of demand beginning their trips on that connector. Set $N_{ri}^\uparrow(t+1) \leftarrow N_{ri}^\uparrow(t) + p_{ri}D_r(t)$ for each connector (r, i) .
6. Terminate trips at each destination s : for each centroid connector (i, s) , set $N_{is}^\downarrow(t+1) \leftarrow N_{is}^\downarrow(t) + S_{is}$.
7. Increment the time: $t \leftarrow t + 1$. If t equals the time horizon \bar{T} , then stop. Otherwise, return to step 2.

Figure 9.11 and Table 9.3 illustrate this process on a network with three links in series, using the spatial queue link model. The link parameters are shown in Figure 9.11; notice that link (i, j) has a larger capacity at the upstream end than at the downstream end. The node model at i is the “links in series” model discussed in Section 9.2.1. The node model at j is a modified version of the “links in series” node model: when $5 \leq t < 10$, $y_{ijs} = 0$ regardless of the sending and receiving flows of (i, j) and (j, s) ; at all other times the “links in series” model is

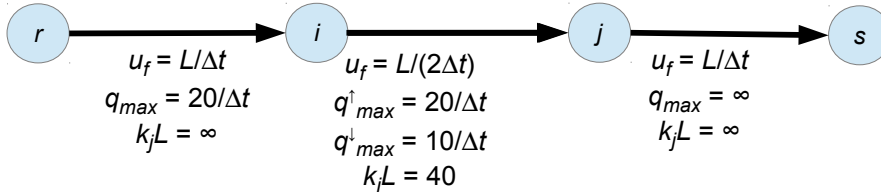


Figure 9.11: Example for network loading algorithm.

used. This might reflect a red traffic signal, or a closed drawbridge between these time intervals, since no flow can move through the node. In this example, the flow downstream is first interrupted when moving from the centroid connector (r, i) to the link (i, j) , whose capacity is lower than the rate at which vehicles are being loaded at origin r . This can be seen by examining the difference between the N^\uparrow and N^\downarrow values for link (r, i) during the initial timesteps. The difference between these values gives the total number of vehicles on the link at that instance in time. Since N^\uparrow is increasing at a faster rate than N^\downarrow , vehicles are accumulating on the link, in a queue at the downstream end. There is no queue at node j , because no bottleneck exists there. Compare $N_{ij}^\uparrow(t)$ and $N_{ij}^\downarrow(t)$ when $3 \leq t \leq 5$. Both the upstream and downstream count values increase at the same rate, which means there is no net accumulation of vehicles.

At $t = 5$, the flow through node j drops to zero, which introduces a further bottleneck. The impacts of the bottleneck are first seen at $t = 6$: 40 vehicles are now on link (i, j) , up from 30. As a result, the receiving flow R_{ij} drops to zero, because the link is full. Therefore, the node model at i restricts any additional inflow to link (i, j) , and the queue on (r, i) grows at an even faster rate than before, even though the number of new vehicles loaded onto the network has dropped. At $t = 10$, the bottleneck at node j is released, and vehicles begin to move again. At $t = 25$, the upstream and downstream counts are equal on all links, which means that the network is empty. All vehicles have reached their destination.

9.4 Elementary Traffic Flow Theory

This section provides an introduction to the hydrodynamic theory of traffic flow, the basis of several widely-used link models which are more realistic than the point or spatial queue models. In this theory, traffic is modeled as a compressible fluid. Its primary advantage is its simplicity: it can capture many important congestion phenomena, while remaining tractable for large networks. Of course, vehicles are not actually molecules of a fluid, but are controlled by drivers with heterogeneous behavior, who drive vehicles of heterogeneous size, power, and so on. By treating vehicles as identical particles, we are ignoring such distinctions; as an example of these limitations, we assume that no overtaking occurs within a link (because the particles are identical, none of them has any reason to move

Table 9.3: Example of network loading algorithm.

t	d_{rs}	R_{ri}	Link (r, i)		S_{ri}	Node i	R_{ij}	Link (i, j)		S_{ij}	Node j	R_{js}	Link (j, s)		S_{js}
			N_{ri}^\uparrow	N_{ri}^\downarrow		q_{rij}		N_{ij}^\uparrow	N_{ij}^\downarrow		q_{ijs}		N_{js}^\uparrow	N_{js}^\downarrow	
0	15	∞	0	0	0	0	20	0	0	0	0	∞	0	0	0
1	15	∞	15	0	15	15	20	0	0	0	0	∞	0	0	0
2	15	∞	30	15	15	15	20	15	0	0	0	∞	0	0	0
3	15	∞	45	30	15	10	10	30	0	10	10	∞	0	0	0
4	15	∞	60	40	20	10	10	40	10	10	10	∞	10	0	10
5	15	∞	75	50	20	10	10	50	20	10	0	∞	20	10	10
6	10	∞	90	60	20	0	0	60	20	10	0	∞	20	20	0
7	10	∞	100	60	20	0	0	60	20	10	0	∞	20	20	0
8	10	∞	110	60	20	0	0	60	20	10	0	∞	20	20	0
9	10	∞	120	60	20	0	0	60	20	10	0	∞	20	20	0
10	10	∞	130	60	20	0	0	60	20	10	10	∞	20	20	0
11	10	∞	140	60	20	10	10	60	30	10	10	∞	30	20	10
12	10	∞	150	70	20	10	10	70	40	10	10	∞	40	30	10
13	0	∞	160	80	20	10	10	80	50	10	10	∞	50	40	10
14	0	∞	160	90	20	10	10	90	60	10	10	∞	60	50	10
15	0	∞	160	100	20	10	10	100	70	10	10	∞	70	60	10
16	0	∞	160	110	20	10	10	110	80	10	10	∞	80	70	10
17	0	∞	160	120	20	10	10	120	90	10	10	∞	90	80	10
18	0	∞	160	130	20	10	10	130	100	10	10	∞	100	90	10
19	0	∞	160	140	20	10	10	140	110	10	10	∞	110	100	10
20	0	∞	160	150	10	10	10	150	120	10	10	∞	120	110	10
21	0	∞	160	160	0	0	10	160	130	10	10	∞	130	120	10
22	0	∞	160	160	0	0	20	160	140	10	10	∞	140	130	10
23	0	∞	160	160	0	0	20	160	150	10	10	∞	150	140	10
24	0	∞	160	160	0	0	20	160	160	0	0	∞	160	150	10
25	0	∞	160	160	0	0	20	160	160	0	0	∞	160	160	0

faster than another). But seen as a first-order approximation, the hydrodynamic theory provides a simple and tractable way to model network traffic. It is also possible to derive certain aspects of the hydrodynamic theory from more behavioral models representing car-following.

In contrast to the link models we will ultimately use in dynamic traffic assignment, fluid-based traffic models are formulated in *continuous* space and time, so it makes sense to talk about the state of traffic at any point in time (not just at the integer timesteps $0, 1, \dots, T$). Link models which are based on fluid models will convert these continuous quantities to discrete ones.

9.4.1 Traffic state variables

We start by modeling a roadway link as a one-dimensional object, using x to index the distance from the upstream end of the link, and using t to index the current time. At any point x , and at any time t , the state of traffic can be described by three fundamental quantities: the *flow* q , the *density* k , and the *speed* u . Each of these can vary over space and time, so $q(x, t)$, $k(x, t)$, and $u(x, t)$ can be seen as functions defined over all x values on the link, and all t values in the analysis period. However, when there is no ambiguity (e.g., only looking at a single point at a single time), we can simply write q , k , and u without providing the space and time coordinates.

Both x and t are treated as continuous variables, as are the vehicles themselves, allowing derivatives of q , k , and u to be meaningfully defined. This assumption is imported from fluid mechanics, where the molecules are so small and numerous that there is essentially no error in approximating the fluid as a continuum. For traffic flow, this assumption is not so trivial, and is one of the drawbacks of hydrodynamic models.

Flow is defined as the rate at which vehicles pass a stationary point, and commonly has units of vehicles per hour. In the field, flow can be measured using point detectors (such as inductive loops) which record the passage of each vehicle at a fixed location. Flow can be thought of as a temporal concentration of vehicles. Density, on the other hand, is defined to be the spatial concentration of vehicles at a given time, and is measured in vehicles per unit length (commonly vehicles per kilometer or vehicles per mile). Density can be obtained from taking a photograph of a link, noting the concentration of vehicles at different locations at a single instant in time. The speed is the instantaneous rate at which the vehicles themselves are traveling, and is measured in units such as kilometers per hour or miles per hour. Speed can be directly measured from radar detectors.

These three quantities are not independent of each other. As a start, there is the basic relationship

$$q = uk \tag{9.20}$$

which must hold at each point and time. (You should check the dimensions of the quantities in this formula to verify their compatibility.) If this equation is not evident to you from the definitions of flow, density, and speed, imagine that we want to know the number of vehicles ΔN which will pass a fixed point over a

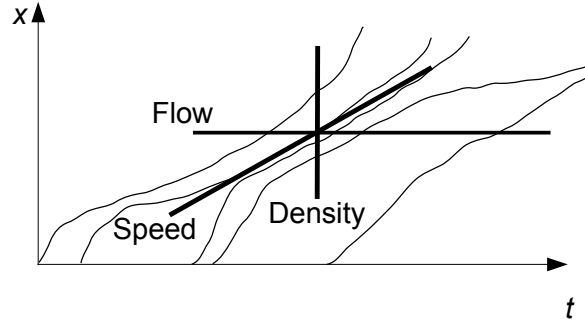


Figure 9.12: Trajectory diagram illustrating speed, flow, and density.

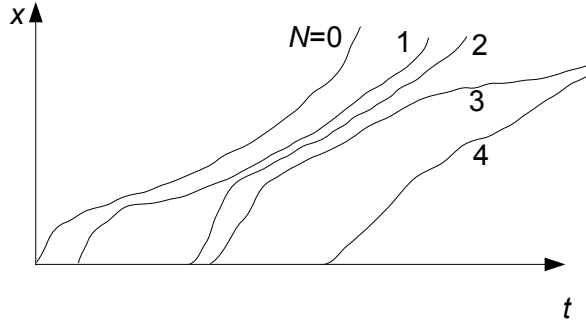
small, finite time interval Δt . If the speed of vehicles is u , any upstream vehicle within a distance of $u\Delta t$ from the fixed point will pass during the next Δt time units, and the number of such vehicles is

$$\Delta N = ku\Delta t. \quad (9.21)$$

The flow rate is approximately $\Delta N/\Delta t$, and the formula (9.20) then follows from taking limits as the time increment Δt shrinks to zero.

Figure 9.12 is a *trajectory diagram* showing the locations of vehicles on the link over time — the horizontal axis denotes time, and the vertical axis denotes space, with the upstream end of the link at the bottom and the downstream end at the top. Speed, flow, and density can all be interpreted in terms of these trajectories. The speed of a vehicle at any point in time corresponds to the slope of its trajectory there. Flow is the rate at which vehicles pass a fixed point: on a trajectory diagram, a fixed point in space is represented by a horizontal line. Time intervals when more trajectories cross this horizontal line have higher flow, and when fewer trajectories cross this line, the flow is lower. Density is the spatial concentration of vehicles at a particular instant in time: on a trajectory diagram, a specific instant is represented by a vertical line. Where more trajectories cross this vertical line, the density is higher, and where fewer trajectories cross, the density is lower.

However, this equation by itself is not enough to describe anything of real interest in traffic flow. Another equation, based on vehicle conservation principles, is described in the next subsection. The Lighthill-Whitham-Richards model, described at the end of this section, makes a further assumption about the relationships of three state variables. These relationships are enough to specify the network loading problem as the solution to a well-defined system of partial differential equations.

Figure 9.13: Vehicle trajectories are contours of $N(x, t)$

9.4.2 Cumulative counts and conservation

The hydrodynamic theory can be simplified by introducing a fourth variable N , again defined at each location x and at each time t . This new variable N is referred to as the *cumulative count*, and has a similar interpretation to the N^\uparrow and N^\downarrow counts defined at the start of Section 9.1, but now applied at any point on the link, not just the upstream and downstream ends. Imagine that each vehicle is labeled with a number; for instance, the vehicle at the upstream end of the link at $t = 0$ may be labeled as zero. The next vehicle that enters the link is then labeled as one, the next vehicle as two, and so forth. The numbering does not have to start at zero. What is important that each entering vehicle be given consecutive numbers. Then $N(x, t)$ gives the number of the vehicle at location x at time t (keeping in mind that we are modeling vehicles as a continuous fluid, so $N(x, t)$ need not be an integer). The contours of $N(x, t)$ then give the trajectories of individual vehicles in the traffic stream. (Figure 9.13)

It is worthwhile to point out a potential source of confusion here. In a trajectory diagram like Figure 9.13, space (indexed by x) is conventionally denoted on the vertical axis, and time (indexed by t) on the horizontal axis. This has the unfortunate side-effect of making the “ x -axis” the vertical one. Second, it is conventional to list the spatial component before the time component, so a point like $(1, 2)$ refers to $x = 1$ and $t = 2$. This means that on a trajectory diagram, the *vertical* component is listed first, as opposed to typical Cartesian coordinates where the *horizontal* component is given first. Unfortunately, both of these conventions are so well-established in the transportation engineering literature that it is best to simply highlight them and become comfortable using them.

The quantity $N(x, t)$ is called a cumulative count for the following reason: at the upstream end of the link ($x = 0$), the quantity $N(0, t)$ gives the cumulative number of vehicles which have entered the link up to time t . Furthermore, at any location x , the difference $N(x, t') - N(x, t)$ gives the number of vehicles which passed point x between times t and t' . Taking the limit as t' approaches

t provides the relationship

$$\frac{\partial N}{\partial t} = q, \quad (9.22)$$

which holds for every x and t where N is differentiable. That is, at any fixed spatial location, the rate at which N increases in time is exactly the flow rate q .

The cumulative counts can also be related to density with a similar argument. At any point in time t , the difference $N(x', t) - N(x, t)$ gives the number of vehicles which lie between locations x and x' at time t . However, we have to be careful regarding the sign convention regarding N described above: since vehicles are numbered in the order they enter the link, in any platoon of vehicles N decreases as we move from the following vehicles to the lead vehicle. So, if $x' > x$, then $N(x', t) \leq N(x, t)$. Thus, taking the limit as x' approaches x , we must have

$$\frac{\partial N}{\partial x} = -k, \quad (9.23)$$

which again applies wherever N is differentiable, and where the negative sign in the formula results from our sign convention.

In this respect, the cumulative counts N can be seen as the most basic description of traffic flow: if we are given $N(x, t)$ at all points x and times t , we can calculate $q(x, t)$ and $k(x, t)$ by using equations (9.22) and (9.23), and therefore $u(x, t)$ everywhere by using (9.20).

Furthermore, wherever the flow and density are themselves continuously differentiable functions, Clairaut's theorem states that the mixed second partial derivatives of N must be equal, that is,

$$\frac{\partial^2 N}{\partial x \partial t} = \frac{\partial^2 N}{\partial t \partial x}, \quad (9.24)$$

so substituting the relationships (9.22) and (9.23) and rearranging, we have

$$\frac{\partial q}{\partial x} + \frac{\partial k}{\partial t} = 0. \quad (9.25)$$

This is an expression of vehicle conservation, that is, vehicles do not appear or disappear at any point. This equation must hold everywhere that these derivatives exist.² Equations (9.22) and (9.23) are useful in another way. If (x_1, t_1) and (x_2, t_2) are any two points in space and time, the difference in cumulative count number between these points is given by the line integral

$$N(x_2, t_2) - N(x_1, t_1) = \int_C q \, dt - k \, dx, \quad (9.26)$$

where C is any curve connecting (x_1, t_1) and (x_2, t_2) . Because vehicles are conserved, this line integral does not depend on the specific path taken. This

²Of course, vehicle conservation must hold even when these derivatives do not exist, it is just that the formula (9.25) is meaningless there. We have to enforce flow conservation in a different way at such points.

is helpful, because we can choose a path which is easy to integrate along. In what follows, we often choose the straight line connecting these two points as the integration path.

Two issues concerning cumulative counts are often confusing, and are worth further explanation. First, the cumulative counts can only be meaningfully compared within the same link. Each link maintains its own counts, and the number associated with a vehicle may change when traveling between links. Instead of thinking of the cumulative count as being a label permanently associated with a vehicle, it is better to think about it as a label given to the vehicle *by a specific link*, and each link maintains its labels independently of all of the other links. Each link simply gives successive numbers to each new vehicle entering the link, without having to coordinate its numbering scheme with other links. This is needed to ensure that link models can function independently, and because in complex networks it is usually impossible to assign “permanent” numbers to vehicles such that any two vehicles entering a link consecutively have consecutive numbers. Second, within any given link, only the *difference* in cumulative counts is meaningful, the absolute numbers do not have specific meaning. For instance, it may be relevant that 10 vehicles entered the link between times 5 and 6, but the specific numbers of these vehicles are not important. A common convention is to have the first vehicle entering the link be assigned the number 0, the next vehicle the number 1, and so forth, but this is not required. In cases where there are already vehicles on the link at the start of the modeling period, the first vehicle entering the link may be assigned a higher number (because the vehicles already on the link must have lower numbers, and a modeler’s aesthetics may prefer nonnegative vehicle counts). But there would be nothing wrong with a negative cumulative count either. In this regard, different choices of the “zero point” are analogous to the different zero points in the Fahrenheit and Celsius temperature scales: either one will give you correct answers as long as you are consistent, and a negative number is not necessarily cause for alarm.

For example, consider a link where $N(x, t) = 1000t - 100x$, with t measured in hours and x measured in miles. The flow at any point and time is $\partial N/\partial t$, which is a constant of 1000 vehicles per hour. Likewise, the density is $-\partial N/\partial x = 100$ vehicles per mile. Using the basic relationship (9.20), the speed must be 10 miles per hour uniformly on the link.

As a more involved example, consider a link which is 1 mile long, with all times measured in minutes and distances in miles. If we are given that

$$N(x, t) = 60t - 120x + \frac{60x^2}{t+1}, \quad (9.27)$$

we can calculate the densities and flows everywhere:

$$k(x, t) = -\frac{\partial N}{\partial x} = 120 \left(1 - \frac{x}{t+1} \right) \quad (9.28)$$

$$q(x, t) = \frac{\partial N}{\partial t} = 60 \left(1 - \left(\frac{x}{t+1} \right)^2 \right). \quad (9.29)$$

Exercise 24 asks you to verify that the conservation relationship (9.25) is satisfied by explicit computation.

Ordinarily, the $N(x, t)$ map is not given — indeed, the goal of network loading is to calculate it. For if we know $N(x, t)$ everywhere, we can calculate flow, density, and speed everywhere, using equations (9.22), (9.23), and (9.20). So let's assume that we *only* know the density and flow maps (9.28) and (9.29), and try to recover information about the cumulative counts. For the given N map, the vehicle at $x = 1/2$ at $t = 0$ has the number 0. (As discussed above, we do not necessarily have to set the zero point at $N(0, 0)$.) To calculate the number of the vehicle at $x = 1$ and $t = 1$ (the downstream end of the link, one minute later), we can use equation (9.26).

As this equation involves a line integral, we must choose a path between $(x, t) = (1/2, 0)$ and $(1, 1)$. Because of the conservation relationship (9.25), we can choose any path we wish. For the purposes of an example, we will calculate this integral along three different paths, and verify that they give the same answer. Figure 9.14 shows the three paths of integration.

Path A : This path consists of the line segment from $(1/2, 0)$ to $(1, 0)$, followed by the segment from $(1, 0)$ to $(1, 1)$. Because these line segments are parallel to the axes, this reduces the line integral to two integrals, one over x alone, and the other over t alone. We thus have

$$\begin{aligned} N(1, 1) &= \int_A q \, dt - k \, dx = - \int_{1/2}^1 k(x, 0) \, dx + \int_0^1 q(1, t) \, dt \\ &= - \int_{1/2}^1 120(1 - x) \, dx + \int_0^1 60 \left(1 - \left(\frac{1}{t+1} \right)^2 \right) \, dt \\ &= -15 + 30 = 15, \end{aligned}$$

and the vehicle at the downstream end of the link at $t = 1$ has the number 15.

Path B : This path consists of the line segment from $(1/2, 0)$ to $(1/2, 1)$, followed by the segment from $(1/2, 1)$ to $(1, 1)$. As before, we have

$$\begin{aligned} N(1, 1) &= \int_B q \, dt - k \, dx = \int_0^1 q(1/2, t) \, dt - \int_{1/2}^1 k(x, 1) \, dx \\ &= \int_0^1 60 \left(1 - \left(\frac{1/2}{t+1} \right)^2 \right) \, dt - \int_{1/2}^1 120 \left(1 - \frac{x}{2} \right) \, dx \\ &= 52.5 - 37.5 = 15. \end{aligned}$$

Path C : This path is the line segment directly connecting $(1/2, 0)$ to $(1, 1)$. Although this line is not parallel to either axis, the integral actually ends up being the easiest to evaluate, because $x/(t+1)$ is constant along this

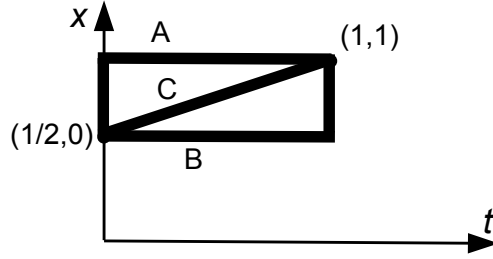


Figure 9.14: Three possible paths for the line integral between $(1/2, 0)$ and $(1, 1)$.

line, and equal to $1/2$. Therefore $k(x, t) = 60$ at all points along this line, and $q(x, t) = 45$. Since $dx = (1/2)dt$ on this line segment, we have

$$\begin{aligned} N(1, 1) &= \int_C q \, dt - k \, dx = \int_0^1 (45dt - 60(1/2)dt) = \int_0^1 15 \, dt \\ &= 15. \end{aligned}$$

All three integrals gave the same answer (as they must), which we can verify by checking $N(1, 1)$ with equation (9.27). So, we can choose whichever integration path is easiest. In this example, the integrals in Path B involved the most work. The integral in Path C required a bit more setup, but the actual integral ended up being very easy, since q and k were constants along the integration path. Such a path is called a *characteristic*, and will be described in more detail later in this chapter.

9.4.3 The Lighthill-Whitham-Richards model

At this point, we have two relationships between the flow, density, and speed variables: the basic relationship (9.20) and the conservation relationship (9.25). These first two relationships can be derived directly from the definitions of these variables, and can describe a wide range of fluid phenomena — at this point nothing yet has been specific to vehicle flow. The Lighthill-Whitham-Richards (LWR) model provides a third relationship, completing the hydrodynamic theory.³

Specifically, the LWR model postulates that the flow at any point is a function of the density at that point, that is,

$$q(x, t) = Q(k(x, t)) \quad (9.30)$$

for some function Q . Equivalently, by the relationship (9.20), we can assume that the speed at any point depends only on the density at that point. It must

³Lighthill and Whitham published this model in 1955, as the sequel to a paper on flow in rivers. Richards independently proposed an equivalent model in 1956. All three are now given credit for this model.

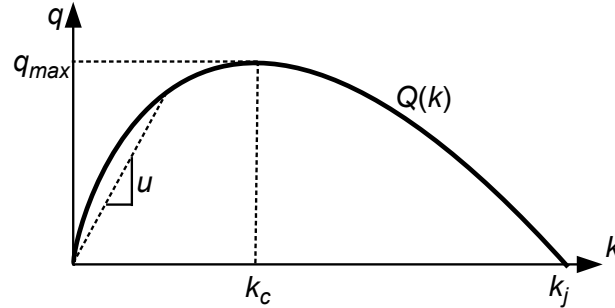


Figure 9.15: Fundamental diagram for the LWR model.

be emphasized that this relationship, unlike (9.20) and (9.25), is an assumed behavior and does not follow from basic principles. In dynamic network loading, we typically assume that this function Q is uniform over space and time on a link, an assumption we adopt in this section for simplicity. It is possible to generalize the results in this section when Q varies over space and time.

The function Q is commonly called the *fundamental diagram*; an example of such a diagram is shown in Figure 9.15. A few features of fundamental diagrams are worth noting: they are concave functions⁴, and typically assumed to be continuous and piecewise differentiable. They also have two zeros: one at $k = 0$ (zero density means zero flow, because no vehicles are present), and another at the *jam density* k_j , corresponding to a maximum density where there is no flow because all vehicles are stopped. At intermediate values of density, the flow is positive, although for a given flow value q , there can be two possible density values corresponding to this flow, one corresponding to uncongested conditions and the other to congested conditions.

If Q is concave, and has two zeros at 0 and k_j , then there is some intermediate point where $Q(k)$ is maximal. This maximal value of Q is called the *capacity* of the link, denoted q_{max} , and the *critical density* k_c is defined to be a value such that $Q(k_c) = q_{max}$. The k values for which $Q(k) < q_{max}$ and $k < k_c$ are referred to as *subcritical*, and reflect uncongested traffic flow; the k values for which $Q(k) < q_{max}$ and $k > k_c$ are *supercritical*, and reflected congested flow.⁵

Using $q = uk$, the speed at any point can be seen as the slope of the secant line connecting the origin to the point on the fundamental diagram correspond-

⁴A function f is concave if $-f$ is convex.

⁵The definitions of subcritical and supercritical in the transportation field are exactly opposite to how these terms are used in fluid mechanics. This is a bit annoying, but this difference in convention reflects differences in the “default state” of flow. Traffic engineers view the default state of traffic flow as being uncongested, when vehicles can move freely and (as we show later) shockwaves only travel downstream. Most fluids have to be moving rather quickly for the same state to occur, and the default state of rivers and many other fluid systems is a slower rate of travel, where waves can move both upstream and downstream — what traffic engineers would describe as a “congested” state.

ing to the density at that point. That is, in the LWR model, the density k at a point completely determines the traffic state. The flow q at that point is obtained from the fundamental diagram Q , and the speed u can then be obtained from equation (9.31).

To summarize, the three equations relating flow, density, and speed are:

$$q(x, t) - k(x, t)u(x, t) = 0 \quad (9.31)$$

$$q(x, t) - Q(k(x, t)) = 0 \quad (9.32)$$

$$\frac{\partial q}{\partial x} + \frac{\partial k}{\partial t} = 0 \quad (9.33)$$

and these equations must hold everywhere (with the exception that (9.33) may not be defined if q or k is not differentiable at a point).

Together with initial conditions (such as the values of k along the link at $t = 0$) and boundary conditions (such as the “inflow rates” q at the upstream end $x = 0$ throughout the analysis period, or restrictions on q at the downstream end from a traffic signal), this system of equations can in principle be solved to yield $k(x, t)$ everywhere. Exercise 24 asks you to verify that the $N(x, t)$ map used in the example in the previous section is consistent with the fundamental diagram $Q(k) = \frac{1}{4}k(240 - k)$.

The points where k is not differentiable are known as *shockwaves*, and often correspond to abrupt changes in the density. Figure 9.16 shows an example of several shockwaves associated with the changing of a traffic light. Notice that in region A, the density is subcritical (uncongested); in region B, traffic is at jam density; and in region C, traffic is at critical density and flow is at capacity. The speed of a shockwave can still be determined from conservation principles, even though the conservation equation (9.33) does not apply because the density and flow derivatives do not exist at a shock.

Assume that k_A and k_B are the densities immediately upstream and immediately downstream of the shockwave (Figure 9.17). The corresponding flow rates $q_A = Q(k_A)$ and $q_B = Q(k_B)$ can be calculated from the fundamental diagram, and finally the speeds are obtained as $u_A = q_A/k_A$ and $u_B = q_B/k_B$. Furthermore, let u_{AB} denote the speed of the shockwave. Then the speed of vehicles in region A *relative to the shockwave* is $u_A - u_{AB}$, and the rate at which vehicles cross the shockwave from region A is $(u_A - u_{AB})k_A$; this is nothing more than equation (9.20) as viewed from the perspective of an observer moving with the shockwave.

Likewise, the relative speed of the vehicles in region B is $u_B - u_{AB}$, and the rate at which vehicles cross the shockwave and enter region B is $(u_B - u_{AB})k_B$. Obviously these two quantities must be equal, since vehicles do not appear or disappear at the shock. Equating these flow rates from the left and right sides of the shockwave, we can solve for the shockwave speed:

$$u_{AB} = \frac{q_A - q_B}{k_A - k_B}. \quad (9.34)$$

Notice that this calculated speed is the same regardless of whether A is the upstream region and B the downstream region, or vice versa. This equation also

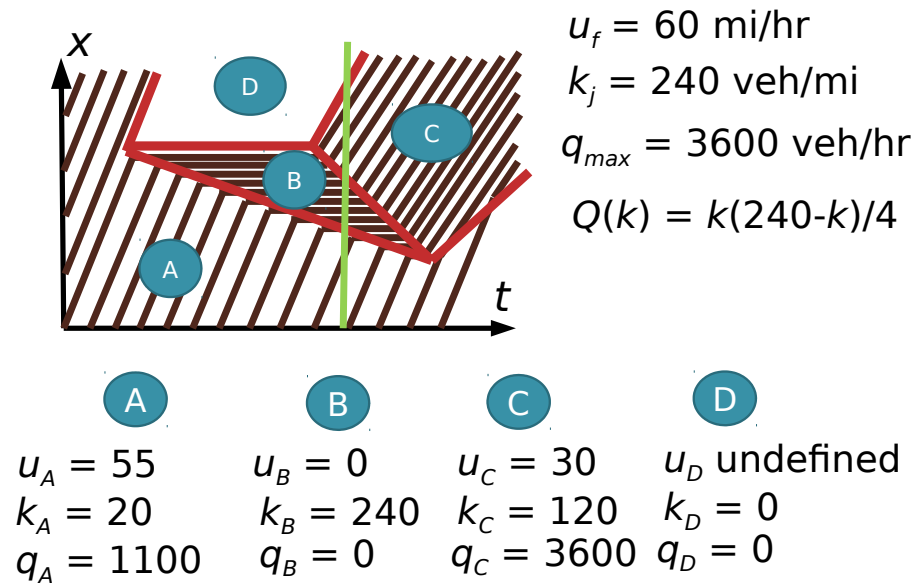


Figure 9.16: Shockwaves associated with a traffic signal; vehicle trajectories indicated in brown and shockwaves in red.

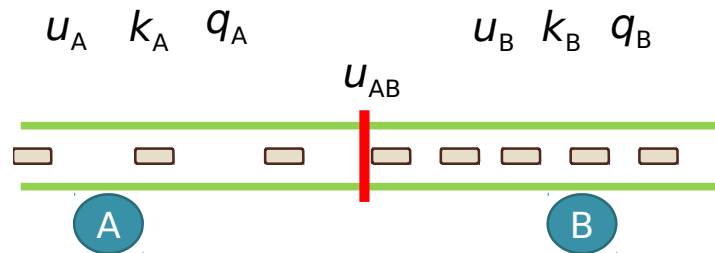


Figure 9.17: Flow conservation produces an equation for shockwave speed.

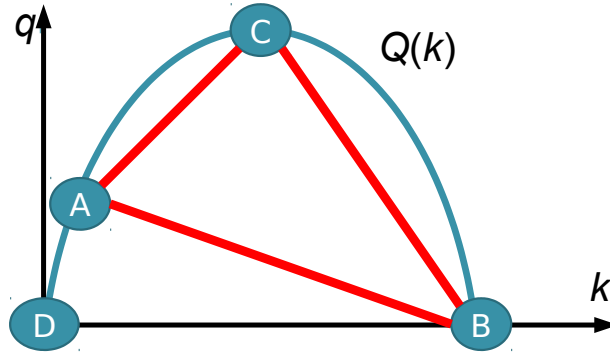


Figure 9.18: Identifying shockwave speed on the fundamental diagram.

has a nice geometric interpretation: the speed of the shockwave is the slope of the line connecting regions A and B on the fundamental diagram (Figure 9.18).

For instance, in Figure 9.16, in region A the flow and density are 1100 vehicles per hour and 20 vehicles per mile, and in region B the flow and density are 0 vehicles per hour and 240 vehicles per mile. Therefore, using (9.34), the shockwave between regions A and B has a speed of $(1100 - 0)/(20 - 240) = -5$ miles per hour. The negative sign indicates that the shockwave is moving *upstream*. Since A represents uncongested traffic, and region B represents the stopped queue at the traffic signal, the interpretation is that the queue is growing at 5 miles per hour. Tracing the derivation of equation (9.34), the rate at which vehicles enter the shockwave is $(55 + 5) \times 20$ from the perspective of region A, or 1200 vehicles per hour. You should check that the same figure is obtained from the perspective of region B. Vehicles are entering the queue faster than the upstream flow rate (1200 vs. 1100 vph) because the queue is growing upstream, moving to meet vehicles as they arrive.

The system of equations (9.31)–(9.33) can then be solved, introducing shockwaves as necessary to accommodate the initial and boundary conditions, using (9.34) to determine their speed. This is the LWR model. However, this theory does not immediately suggest a technique for actually solving the system of partial differential equations, which is the topic of the next subsection.

Notice also that the fundamental diagram determines the maximum speed at which a shockwave can move. Because the fundamental diagram is concave, its slope at any point can never be greater than the free-flow speed $u_f = Q'(0)$, nor less than the slope at jam density $Q'(k_j)$. The absolute values of these slopes give the fastest speeds shockwaves can move in the downstream and upstream directions, respectively, because shockwave speeds are the slopes of lines connecting points on the fundamental diagram. This leads to an important notion, the *domain of dependence*. Consider a point (x, t) in space and time. Through this point, draw lines with slopes $Q'(0)$ and $Q'(k_j)$. The area between

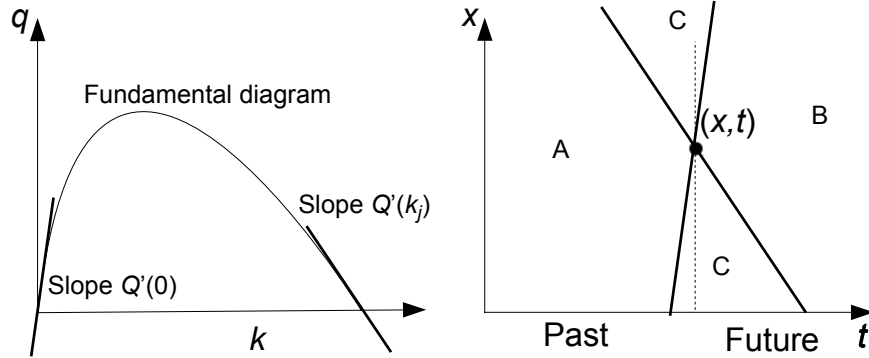


Figure 9.19: The fundamental diagram (left) and the domain of dependence (right).

these lines to the left of (x, t) represents all the points which can potentially influence the traffic state at (x, t) (labeled as region A in Figure 9.19), and the area between the lines to the right of (x, t) represents all of the points the traffic state at (x, t) can potentially influence (labeled as region B). In the LWR model, points outside of these regions (the regions labeled C) are independent of what happens at (x, t) (say, a signal turning red or green): in the past, they are either too recent or too distant to affect what is happening at (x, t) . In the future, they are too soon or too distant to be affected by an event at (x, t) . *This is a crucial fact for dynamic network loading.* If you want to know what is happening at (x, t) , it is sufficient to know what has happened in the past, in the domain of dependence. We do not need to know what is happening simultaneously at other points in the network, and as a result we can perform network loading in an decentralized fashion, performing calculations in parallel since they do not depend on each other.⁶

9.4.4 Characteristics and the Newell-Daganzo method

By substituting equation (9.32) into (9.33), and setting aside (9.31) for the moment, we can obtain a partial differential equation in k alone:

$$\frac{dQ}{dk} \frac{\partial k}{\partial x} + \frac{\partial k}{\partial t} = 0 \quad (9.35)$$

It is in this form that the LWR model is most frequently solved, obtaining $k(x, t)$ values everywhere. This type of partial differential equation can be approached using the *method of characteristics*, which is briefly described below.

A *characteristic* is a curve in (x, t) space, along which the density $k(x, t)$ varies in a predictable way. If the fundamental diagram Q does not vary in space

⁶If you have studied relativistic physics, there are many similarities with the notion of light cones and causality.

and time, then straight lines form characteristics of (9.35). To demonstrate, assume that we know the density $k(x_1, t_1)$ at some point, and consider a straight line C through (x_1, t_1) with some slope v . Then the directional derivative of k along this line is given by

$$k' = \nabla k \cdot [v \ 1] = \frac{\partial k}{\partial x} v + \frac{\partial k}{\partial t}. \quad (9.36)$$

Substituting the conservation equation (9.35) and rearranging, we have

$$k' = \left(v - \frac{dQ}{dk} \right) \frac{\partial k}{\partial x}. \quad (9.37)$$

This gives us a formula for the change in density along the line. But what is really useful is noticing that if $v = \frac{dQ}{dk}$, then (9.37) vanishes and *the density is constant along the line*. Put another way, the density is constant along any line whose slope is equal to the slope of the *tangent* of the fundamental diagram at that density value.⁷

To see this, return to the example we have been using where $Q(k) = \frac{1}{4}k(240 - k)$ and $N(x, t) = 60t - 120x + \frac{60x^2}{t+1}$. We have calculated the density and flow at all points and times in equations (9.28) and (9.29). In our calculation, we observed that the line integral along path C was the simplest to evaluate, because $k(x, t)$ was a constant 60 vehicles per mile along the line $x/(t+1) = 1/2$. This path is in fact a characteristic: the derivative of the fundamental diagram at this point is $Q'(60) = 60 - k/2 = 30$ miles per hour. Rearranging the equation of the line, we have $x = t/2 + 1/2$; in other words, this is a line whose location moves half a mile in one minute: 30 miles per hour, the same as the derivative of the fundamental diagram.

These slopes will be different at points where the density is different, which means that characteristic lines can potentially intersect. This indicates the presence of a shockwave separating regions of different density. One interpretation of these characteristic lines is as “directions of influence,” since the density at a point will determine the density at all later points along this line. In uncongested regions, where k is subcritical, dQ/dk is positive, meaning that the characteristics have positive slope: uncongested states will propagate downstream. In congested regions with supercritical density, dQ/dk is negative, and the characteristics have negative slope: congested states will propagate upstream.

Furthermore, by combining knowledge of characteristics with equation (9.26), we can determine the cumulative count and the density at any point, given sufficient initial and boundary data. This method was first suggested by G. F. Newell, and developed further by Carlos Daganzo. Suppose we wish to calculate the density at a point (x, t) . If we knew the density at this point, then we would know the slope of the characteristic through this point. This characteristic could be traced back until it intersected a point (x_0, t_0) where the cumulative count

⁷Note that the speed of the characteristic is different from the speed of the vehicles themselves, or the speed of shockwaves (which are given by slopes of secant lines on the fundamental diagram).

N was known, either because it corresponds to an initial or boundary point, or a point where N has already been calculated. Then, applying (9.26), we would have

$$N(x, t) = N(x_0, t_0) + \int_C q \, dt - k \, dx \quad (9.38)$$

where C is the straight line between (x_0, t_0) and (x, t) . Since this line is a characteristic, we have $dx/dt = dQ/dk$, and so

$$N(x, t) = N(x_0, t_0) + \int_C \left(q - k \frac{dQ}{dk} \right) dt. \quad (9.39)$$

Furthermore, as a characteristic, k (and therefore q) are constant. So the integral is easy to evaluate, and we have

$$N(x, t) = N(x_0, t_0) + \left(q - k \frac{dQ}{dk} \right) (t - t_0). \quad (9.40)$$

The only trouble is that we do not actually know the density at (x, t) . Each possible value of density corresponds to a slightly different cumulative count, based on equation (9.40). The insight of the Newell-Daganzo method is that *the correct value of the cumulative count is the lowest possible value*. That is, imagine that the density at (x, t) is k , and let (x_k, t_k) be the known point corresponding to the characteristic slope of density k . Then

$$N(x, t) = \inf_{k \in [0, k_j]} \left\{ N(x_k, t_k) + \left(q - k \frac{dQ}{dk} \right) (t - t_k) \right\}. \quad (9.41)$$

Rigorously validating this insight requires knowledge of the calculus of variations, which is beyond the scope of this text. An intuitive justification is that (9.40) represents an upper bound on the cumulative count imposed by a boundary or initial point — we know the number of the vehicle passing that boundary or initial point, and (9.40) expresses one possible value of the cumulative count at the point in question, if the density took a particular value and no shockwave intervened. However, due to the possible presence of shockwaves, there may be another, more restrictive constraint imposed on the cumulative count. The equation (9.41) thus finds the “most restrictive” boundary or initial condition, which gives the correct cumulative count.

Better yet, this method becomes exceptionally easy if we assume that the fundamental diagram takes a simple form, such as a triangular shape (Figure 9.20), where the equation is given by

$$Q(k) = \min \{ u_f k, w(k_j - k) \}. \quad (9.42)$$

In this case, there are only two possible characteristic speeds: one ($+u_f$) corresponding to uncongested conditions, and the other ($-w$) corresponding to congested conditions. The uncongested speed u_f is the free-flow speed, and w is known as the *backward wave speed*. In the case where the characteristic speed

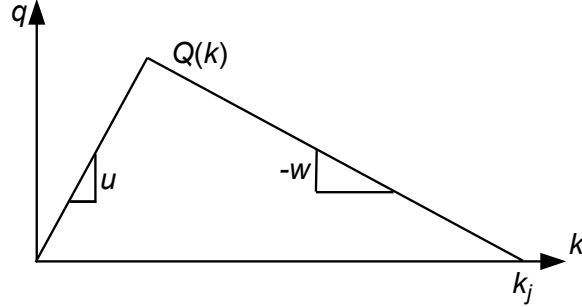


Figure 9.20: A triangular fundamental diagram, defined by three parameters u_f , w , and k_j .

is u_f , the vehicle speed u equals the characteristic speed u_f , and both are equal to q/k . Therefore, the line integral along the characteristic in (9.40) is

$$(q - ku_f)(t - t_0) = (q - uk)(t - t_0) = 0 \quad (9.43)$$

because $q = uk$.

In other words, *the vehicle number is constant along characteristics at free-flow speed*. For the congested characteristic with slope $-w$, we have

$$(q - k(-w))(t - t_0) = w \left(\frac{q}{w} + k \right) (t - t_0) = k_j w (t - t_0) \quad (9.44)$$

since $k + \frac{q}{w} = k_j$, as can be seen in Figure 9.20. This expression can also be written as $k_j(x_0 - x)$. Since these are the only two characteristics which can prevail at any point, equation (9.41) gives

$$N(x, t) = \min \{N(x_U, t_U), N(x_C, t_C) + k_j(x_C - x)\} \quad (9.45)$$

where (x_U, t_U) is the known point intersected by the uncongested characteristic, and (x_C, t_C) is the known point intersected by the congested characteristic.

Trapezoidal fundamental diagrams, such as that in Figure 9.21, are also commonly used, given by the equation

$$Q(k) = \min \{u_f k, q_{max}, w(k_j - k)\} . \quad (9.46)$$

In this case, there is a third possible characteristic speed, corresponding to the flat region where flow is at capacity. Since the derivative of the fundamental diagram at this point is zero, this characteristic has zero speed, represented by a horizontal line on a space-time diagram. Tracing this characteristic back to a known point (x, t_R) , the change in vehicle number between this known point and the unknown point (x, t) is just

$$q_{max}(t - t_R) \quad (9.47)$$

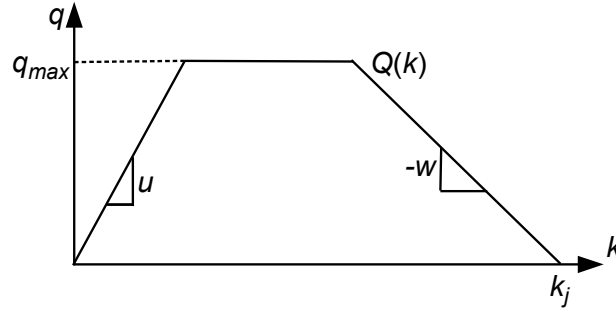


Figure 9.21: A trapezoidal fundamental diagram, defined by four parameters u_f , q_{max} , w , and k_j .

since $dx = 0$ in the line integral (9.26). (Note that the location of this third known point is the same as the location of the point we are solving for, since the characteristic is stationary.) This adds a third term to the minimum in (9.45), giving

$$N(x, t) = \min \{N(x_U, t_U), N(x, t_R) + q_{max}(t - t_R), N(x_C, t_C) + k_j(x_C - x)\} \quad (9.48)$$

for trapezoidal fundamental diagrams.

The trapezoidal fundamental diagram requires four parameters to calibrate: the free-flow speed u_f , the capacity q_{max} , the jam density k_j , and the backward wave speed $-w$. The first three of these are fairly straightforward to estimate from traffic engineering principles. The backward wave speed is a bit trickier; empirically it is often a third to a half of the free-flow speed.

As a demonstration of this method, consider a link which is 1 mile long. The fundamental diagram is shown in the left panel of Figure 9.22, and has the equation

$$Q(k) = \min \left\{ k, 120 - \frac{1}{2}k \right\} \quad (9.49)$$

when flow is measured in vehicles per minute, and density in vehicles per mile. Initially, vehicles on the link flow at an uncongested 48 veh/min — this state has existed for a long time in the past, and vehicles continue entering the link at this rate. However, at the downstream end of the link there is an obstruction which prevents any vehicles from passing, such as a red light or an incident blocking all lanes. This will cause a queue of stopped vehicles to form. Assume that we want to know how many vehicles lie between the obstruction and three given points: half a mile upstream of the obstruction, 30 seconds after it begins; an eighth of a mile upstream of the obstruction, 30 seconds after it begins; and an eighth of a mile upstream of the obstruction, one minute after it begins. We do not know, *a priori*, whether these points lie within the queue or in the portion of the link which is still uncongested.

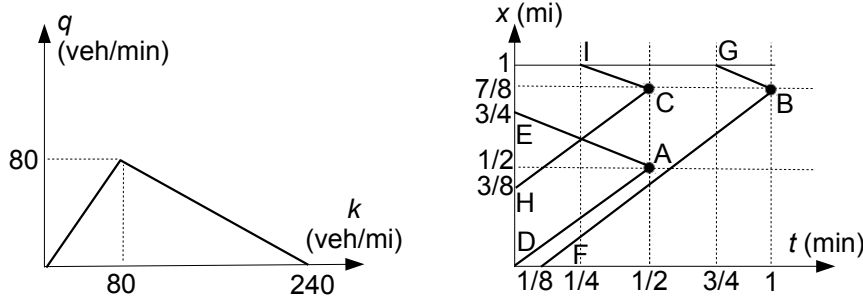


Figure 9.22: Example of the Newell-Daganzo method.

The first step is to establish a coordinate system. As always, we set $x = 0$ at the upstream end of the link. For this problem, it will be convenient to set $t = 0$ at the time when the obstruction begins, and to count vehicles starting from the first vehicle stopped at the obstruction, that is, $N(1, 0) = 0$. This way, $N(x, t)$ will immediately give the number of vehicles between point x and the obstruction at time t . Next, we use the given data from the problem to construct initial and boundary conditions where we already know $N(x, t)$. Since no vehicles can pass the obstruction, we know that $N(1, t) = 0$ for all t . Since the link is initially uncongested at a flow rate of 48 veh/min, the fundamental diagram (9.49) gives the initial density to be 48 veh/mi. Therefore, the initial condition is $N(x, 0) = 48 - 48x$, and the vehicle number at the origin of the coordinate system is 48. Since vehicles continue to enter the link at a rate of 48 veh/min, we have $N(0, t) = 48 + 48t$ along the upstream boundary of the link. The three points where we must calculate $N(x, t)$ are labeled as A, B, and C in the right panel of Figure 9.22.

We start with point A, half a mile upstream of the obstruction and 30 seconds after it begins. There are two possible characteristics at this point, one with slope $+1$ (corresponding to uncongested conditions) and one with slope $-\frac{1}{2}$ (corresponding to congested conditions). We can trace back these characteristics until they reach a point where $N(x, t)$ is known, in this case an initial or boundary condition. These points of intersection are labeled D and E in Figure 9.22. From the initial condition, we know that $N(D) = 48$ and $N(E) = 12$. Along the uncongested characteristic, there is no change in the cumulative count, while along the congested characteristic the cumulative count increases at a rate of $k_j = 240$ veh/mi for each mile traveled. Therefore, equation (9.43) tells us that $N(A) = 48 + 0 = 48$ if point A is uncongested, while equation (9.44) tells us that $N(A) = 12 + \frac{1}{4}240 = 72$ if point A is congested. The correct value is the smaller of the two: $N(A) = 48$, and this point is uncongested (the queue has not yet reached this point).

We next move to point B, an eighth of a mile upstream of the obstruction and 60 seconds after it begins. Tracing back the two possible characteristics from point B leads us to the points labeled F and G, and from the boundary

conditions we know $N(F) = 54$ and $N(G) = 0$. So $N(B)$ is the lesser of $N(F) + 0 = 54$ and $N(G) + \frac{1}{8}240 = 30$. This means that there are 30 vehicles between point B and the obstruction, and since the congested characteristic produced the lower value, point B lies within the queue.

At point C, the two characteristics lead to the points labeled H and I, where $N(H) = 30$ from an initial condition and $N(I) = 0$ from a boundary condition. So $N(C) = \min\{30 + 0, 0 + \frac{1}{8}240\} = 30$. In this case, both characteristics led to the *same* value of $N(C)$. This indicates that the shockwave passes exactly through point C.

An alternative approach for this problem would be to explicitly calculate the location of the shockwave, determine the densities in each region, and apply (9.26) directly. For this problem, this approach would be simpler than the Newell-Daganzo method. However, if there were multiple shockwaves introduced into the problem (say, from multiple red/green cycles), it would become very tedious to track the locations of all of the shockwaves and determine which region the points lie in. The Newell-Daganzo method can be applied just as easily in such a case, once the boundary conditions are determined.

9.5 LWR-based Link Models

The hydrodynamic traffic flow model developed by Lighthill, Whitham, and Richards forms the basis for several popular link models. The LWR model is simple enough to be usable in large-scale dynamic network loading, while capturing enough key properties of traffic flow for its results to be meaningful. Through shockwaves, we can capture how congestion grows and shrinks over time. These shockwaves allow us to model queue spillback (when a congestion shockwave reaches the upstream end of a link) and to account for delays in queue startup (unlike the spatial queue model). This section describes two link models based on the LWR model — the cell transmission model, which is essentially an explicit solution scheme for the LWR system of partial differential equations, and the link transmission model, which uses the Newell-Daganzo method to directly calculate sending and receiving flows. Lastly, we show how the point and spatial queue models can be seen as special cases of LWR-based link models, with a suitable choice of the fundamental diagram.

9.5.1 Cell transmission model

In the cell transmission model, in addition to discretizing time into intervals of length Δt , we also discretize space, dividing the link into cells of length Δx . These two discretizations are not chosen independently. Rather, they are related by

$$\Delta x = u_f \Delta t, \quad (9.50)$$

that is, the length of each cell is the distance a vehicle would travel in time Δt at free flow. The reasons for this choice are discussed at the end of this section.

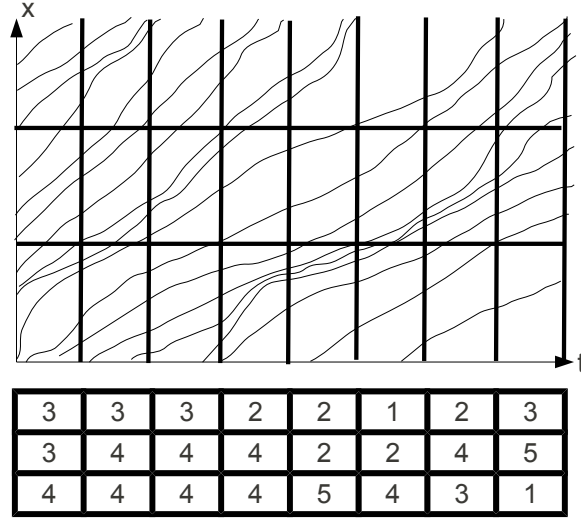


Figure 9.23: Discretizing space and time into cells, $n(x, t)$ values below.

With this discretization in mind, we will use the notation $n(x, t)$ to describe the number of vehicles in cell x at time t , where x and t are both integers — we must convert the continuous LWR variables k and q into discrete variables for dynamic network loading, which we will call n and y .

If the cell size is small, we can make the approximation that

$$n(x, t) \approx k(x, t)\Delta x, \tag{9.51}$$

essentially assuming that the density within the cell is constant. Further define $y(x, t)$ to be the number of vehicles which *enter* cell x during the t -th time interval. Making a similar assumption, we can make the approximation

$$y(x, t) \approx q(x, t)\Delta t. \tag{9.52}$$

In a space-time diagram showing vehicle trajectories, such as Figure 9.23, $n(x, t)$ and $y(x, t)$ respectively correspond to the number of trajectories crossing the vertical line at t between locations x and $x + \Delta x$, and the number of trajectories crossing the horizontal line at x between times t and $t + \Delta t$.

The cell transmission model provides methods for solving for $n(x, t)$ for all integer values of x and t ; these can then be converted to density values $k(x, t)$ through (9.51). These density values can then be used to calculate flows $q(x, t)$ through the fundamental diagram, and $u(x, t)$ values through equation (9.20), finally providing an approximate solution to the system of partial differential equations (9.31)–(9.33). Recall that the goal of a link model is to determine

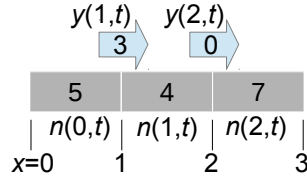


Figure 9.24: Where discrete values are calculated in the cell transmission model.

sending and receiving flow at each time step. For this reason, we will be content with determining how the $n(x, t + \Delta t)$ values can be calculated, given the $n(x, t)$ values (which are already known), and the $y(x, t)$ values, which must be calculated.

Since $q(x, t) = Q(k(x, t))$, substitution into equations (9.51) and (9.52) give

$$y(x, t) \approx Q\left(\frac{n(x, t)}{\Delta x}\right) \Delta t. \quad (9.53)$$

Substituting the particular form of the fundamental diagram gives an equation for $y(x, t)$ in terms of $n(x, t)$. Using the trapezoidal diagram of Figure 9.21, we have

$$y(x, t) \approx \min \left\{ u_f n(x, t) \frac{\Delta t}{\Delta x}, q_{max} \Delta t, w \Delta t \left(k_j - \frac{n(x, t)}{\Delta x} \right) \right\}. \quad (9.54)$$

Using the fact that $\Delta x / \Delta t = u_f$, the first term in the minimum is simply $n(x, t)$. The third term can be simplified by defining $\bar{n}(x) = k_j \Delta x$ to be the maximum number of vehicles which can fit into a cell and $\delta = w / u_f$ to be the ratio between the backward wave speed and free-flow speed. Then, factoring out $1 / \Delta x$ from the term in parentheses and again using $\Delta x / \Delta t = u_f$, the third term simplifies to $\delta(\bar{n}(x) - n(x, t))$.

There is one more point which is subtle, yet incredibly important. Being a “flow” variable, $y(x, t)$ is calculated at a single point (over a time interval), while $n(x, t)$ is calculated at a single time (over a longer spatial interval). As shown in Figure 9.24, the x in $y(x, t)$ refers to a single location, while the x in $n(x, t)$ refers to an entire cell. So, when we are calculating the flow across the (single) point x , which is the boundary between two cells, do we look at the adjacent cell upstream $n(x - 1, t)$, or the adjacent cell downstream $n(x, t)$?

The correct answer depends on the fundamental diagram, and the meaning of characteristics. In uncongested conditions, corresponding to the increasing part of the fundamental diagram and the first term in the minimum, the traffic state moves from upstream to downstream (because the characteristic has positive speed). In congested conditions, corresponding to the decreasing part of the fundamental diagram and the third term in the minimum, the traffic state moves from downstream to upstream (because the characteristic has negative speed.)

So, if traffic is uncongested at the (single) point x , we need to refer to the upstream cell, while if traffic is congested at the (single) point x , we must refer to the downstream cell. So, the final expression for the cell transmission model flows is

$$y(x, t) = \min\{n(x - \Delta x, t), q_{max}\Delta t, \delta(\bar{n}(x) - n(x, t))\}. \quad (9.55)$$

This expression also has a nice intuitive interpretation. The number of vehicles moving from cell $x - 1$ to cell x is limited either by the number of vehicles in the upstream cell (the first term), the capacity of the roadway (the second term), or by the available space in the downstream cell (the third term).

Expression (9.55) is the discrete equivalent of the differential equation (9.32). We now derive the discrete form of the partial differential equation giving the conservation law (9.33). The derivative $\frac{\partial q}{\partial x}$ can be approximated as

$$\frac{\partial q}{\partial x}(x, t) \approx \frac{1}{\Delta t \Delta x} (y(x + \Delta x, t) - y(x, t)) \quad (9.56)$$

and the derivative $\frac{\partial k}{\partial t}$ can be approximated as

$$\frac{\partial k}{\partial t}(x, t) \approx \frac{1}{\Delta t \Delta x} (n(x, t + \Delta t) - n(x, t)). \quad (9.57)$$

Substituting into (9.33), we have

$$\frac{1}{\Delta t \Delta x} (y(x + \Delta x, t) - y(x, t) + n(x, t + \Delta t) - n(x, t)) = 0 \quad (9.58)$$

or, in a more convenient form,

$$n(x, t + \Delta t) = n(x, t) + y(x, t) - y(x + \Delta x, t) \quad (9.59)$$

This also has a simple intuitive interpretation: the number of vehicles in cell x at time $t + 1$ is simply the number of vehicles in cell x at the previous time t , plus the number of vehicles which flowed into the cell during the t -th time interval, minus the number of vehicles which left.

Together, the equations (9.55) and (9.59) define the cell transmission model for trapezoidal fundamental diagrams. There are only two pieces of “missing” information, at the boundaries of the link. Refer again to Figure 9.24. How should $y(0, t)$ and $y(L, t)$ be calculated? For $y(0, t)$, the first term in formula (9.55) involves $n(-\Delta x, t)$, while for $y(L, t)$, the third term in the formula involves $n(L + \Delta x, t)$, and both of these cells are “out of range.” The answer is that these boundary flows are used to calculate the sending and receiving flows for the link, and a node model will then give the actual values link inflows $y(0, t)$ and link outflows $y(L, t)$.

Remember that the sending flow is the maximum number of vehicles which could leave the link if there was no obstruction from downstream. In terms of (9.55), this means that the third term in the minimum (which corresponds to downstream congestion) is ignored. Then, the first two terms in the minimum

(which only refer to cells on the link) are the possible restrictions on the flow leaving the link, so

$$S(t) = \min\{n(C, t), q_{max}\Delta t\}. \quad (9.60)$$

Likewise, the receiving flow is the maximum number of vehicles which could enter the link if there were a large number of vehicles wanting to enter from upstream. In terms of (9.55), this means that the first term in the minimum (which corresponds to the number of vehicles wanting to enter) is ignored. The second two terms in the minimum refer to cells on the link, and

$$R(t) = \min\{q_{max}\Delta t, \delta(\bar{n}(0) - n(0, t))\}. \quad (9.61)$$

It remains to explain the choice of the discretization (9.50). An intuitive explanation for linking the cell length and time discretizations in this way is that this choice limits vehicles to moving at most one cell between time steps. In fact, in uncongested conditions, *all* vehicles in a cell will move to the next cell downstream in the next time interval, simplifying calculations — for instance, the simplification in (9.54) only works because of the choice made in (9.50). The underlying mathematical reason has to do with the speed of characteristics, which for the trapezoidal fundamental diagram lie between $-w$ and $+u_f$. Empirically, $w < u_f$, so the fastest moving characteristic (in either direction) is one with speed u_f . More important than vehicles moving at most one cell between time steps is that *characteristics cannot move more than one cell between time steps*, in either the upstream or downstream directions. This condition is needed for stability of finite-difference approximations to partial differential equations, and corresponds to the *Courant-Friedrich-Lewy (CFL) condition* for explicit solution methods such as the cell transmission model.

Table 9.4 provides a demonstration of how the cell transmission model works. In this example, a link is divided into three cells, implying that it takes three time steps for a vehicle to traverse the link at free-flow. The fundamental diagram is such that at most 10 vehicles can move between cells in one time step, at most 30 vehicles can fit into one cell at jam density, and the ratio between the backward and forward characteristic speeds is $\delta = 2/3$. There is a red light at the downstream end of the link which turns green at $t = 10$, and remains green thereafter. In this table, $d(t)$ represents the number of vehicles that wish to enter the link during the t -th timestep (perhaps the sending flow from an upstream link), while $R(t)$ is the receiving flow for the link, calculated from (9.61). The middle columns of the table show the main cell transmission model calculations: the number of vehicles in each cell at the start of each timestep, $N(x, t)$, and the number of vehicles moving into each cell during the t -th time step. These values are calculated from (9.59) and (9.55), respectively, along with the initial condition that the link is empty, that is, $N(x, 0) = 0$ for all x . The rightmost columns of the table show the link's sending flow, calculated from (9.60), and the actual flow which leaves the link, denoted $y(3, t)$. This latter value is constrained to be zero as long as the light at the downstream end of the link is red.

Notice that the table has non-integer values: we do not need to round cell occupancies and flows to whole values, since the LWR model assumes vehicles are a continuously-divisible fluid. Preserving non-integer values also ensures that the cell transmission model remains accurate no matter how small the timestep Δt is (in fact, its accuracy should increase as this happens). Insisting that flows and occupancies be rounded to whole numbers can introduce significant error if the timestep is small, unless one is careful with implementation.

Table 9.5 shows only the cell occupancies at each timestep, color-coded according to the density in the cells. In this example, the link is initially at free-flow, until the first vehicles encounter the red light and must stop. A queue forms, and a shockwave begins moving backward. When this shockwave reaches the cell at the upstream end of the link, the receiving flow of the link decreases, and the inflow to the link is limited. When the light turns green, a second shockwave begins moving backward as the queue clears. Once this shockwave overtakes the first, vehicles can begin entering the link again. For a few time steps, the inflow is greater than $d(t)$, representing demand which was blocked when the receiving flow was restricted by the queue and which was itself queued on an upstream link (the “queue spillback” phenomenon). Unlike the point queue and spatial queue link models, the cell transmission model tells us what is happening in the interior of a link, not just at the endpoints. This is both a blessing and a curse: sometimes this additional information is helpful, while other times we may not be concerned with such details. The link transmission model, described next, can simplify computations if we do not need information on the internal state of a link.

9.5.2 Link transmission model

The link transmission model allows us to calculate sending and receiving flows for links with any trapezoidal fundamental diagram.⁸ In contrast to the cell transmission model, it only involves calculations at the ends of the links — details of what happen in the middle of the link are ignored. As a consequence, the link transmission model does not require us to keep track of information within the link. However, it does require us to keep track of information on the past state of the link, whereas the cell transmission model calculations only involve quantities at the current time step, and the previous time step. The link transmission model can also overcome the “shock spreading” phenomenon, where backward-moving shockwaves in the cell transmission model can diffuse across multiple cells, even though they are crisp in the LWR model. (See Exercise 29.)

Assume that the trapezoidal fundamental diagram is parameterized as in Figure 9.21. There are three characteristic speeds, $+u_f$ at free-flow, 0 at capacity flow, and $-w$ at congested flow. As with the point queue and spatial queue models, we apply the Newell-Daganzo method to calculate sending and receiving flows. We will only need to refer to cumulative counts N at the upstream

⁸In fact, it can be generalized to any piecewise linear fundamental diagram without too much difficulty; see Exercise 34.

Table 9.5: Cell occupancies from the example in Table 9.4, green is zero density and red is jam density.

t	$N(0,t)$	$N(1,t)$	$N(2,t)$
0	0	0	0
1	10	0	0
2	10	10	0
3	10	10	10
4	10	10	20
5	10	13.3	26.7
6	9	21.1	28.9
7	11.1	26.3	29.6
8	15.6	28.5	29.9
9	20.6	29.4	30
10	25.2	29.8	30
11	28.3	29.9	20
12	29.4	23.3	16.7
13	25.3	18.9	15.6
14	21	16.7	15.2
15	13.4	15.7	15.1
16	3.9	15.3	15
17	0	9.2	15
18	0	0	14.2
19	0	0	4.2
20	0	0	0

and downstream ends of the link, that is, at $N(0, \cdot)$ and $N(L, \cdot)$, respectively. In keeping with the notation introduced in Section 9.1, we will refer to these as N^\uparrow and N^\downarrow . If there is no obstruction from a downstream link or node, then the end of the link will be uncongested, and the characteristic at this end will either have slope $+u_f$ or slope zero. The Newell-Daganzo method thus gives

$$N^\downarrow(t + \Delta t) = \min\{N^\uparrow((t + \Delta t) - L/u_f), N^\downarrow(t) + q_{max}\Delta t\} \quad (9.62)$$

and the equation for the sending flow is obtained as the difference between $N^\downarrow(t + 1)$ and $N^\downarrow(t)$:

$$S(t) = \min\{N^\uparrow((t + \Delta t) - L/u_f) - N^\downarrow(t), q_{max}\Delta t\}. \quad (9.63)$$

For the receiving flow, we have to take into account the two relevant characteristic speeds of 0 and $-w$, since the receiving flow is calculated assuming an inflow large enough that the upstream end of the link is congested (or at least at capacity). The stationary characteristic corresponds to the known point $N(0, t)$, while the backward-moving characteristic corresponds to the known point $N(L, t - L/w)$. Thus, applying the last two terms of equation (9.46) would give

$$N^\uparrow(t + \Delta t) = \min\{N^\uparrow(t) + q_{max}\Delta t, N^\downarrow((t + \Delta t) - L/w) + k_j L\} \quad (9.64)$$

and

$$R(t) = \min\{q_{max}\Delta t, (N^\downarrow((t + \Delta t) - L/w) + k_j L) - N^\uparrow(t)\}. \quad (9.65)$$

It is possible to show that equations (9.63) and (9.65) ensure that the number of vehicles on the link is always nonnegative, and less than $k_j L$.

The link transmission model is demonstrated on an example similar to the one used for the cell transmission model; the only difference is that the ratio of backward-to-forward characteristics has been adjusted from 2/3 to 3/4. In particular, $L/u_f = 3\Delta t$, and $L/w = 4\Delta t$, so forward-moving characteristics require three time steps to cross the link, and backward-moving characteristics require four time steps. The total number of vehicles which can fit on the link is $k_j L = 90$. Otherwise, the example is the same: the demand profile is identical, and a red light prevents outflow from the link until $t = 10$. The results of the calculations are shown in Table 9.6. The rightmost column shows the number of vehicles on the link, which is the difference between the upstream and downstream cumulative counts at any point in time. Notice that inflow to the link is completely blocked during the 12th and 13th time intervals, even though the number of vehicles on the link is less than the jam density of 90. This happens because the queue has started to clear at the downstream end, but the clearing shockwave has not yet reached the upstream end of the link. The vehicles at the upstream end are still stopped, and no more vehicles can enter. In contrast, the spatial queue model would allow vehicles to start entering the link as soon as they began to leave.

Table 9.6: Link transmission model example, with $L/u_f = 3\Delta t$, $L/w = 4\Delta t$, and $k_j L = 90$.

t	$d(t)$	$R(t)$	Inflow	$N^\uparrow(t)$	$N^\downarrow(t)$	$S(t)$	Outflow	Vehicles on link
0	10	10	10	0	0	0	0	0
1	10	10	10	10	0	0	0	10
2	10	10	10	20	0	0	0	20
3	10	10	10	30	0	10	0	30
4	10	10	10	40	0	10	0	40
5	9	10	9	50	0	10	0	50
6	8	10	8	59	0	10	0	59
7	7	10	7	67	0	10	0	67
8	6	10	6	74	0	10	0	74
9	5	10	5	80	0	10	0	80
10	4	5	4	85	0	10	10	85
11	3	1	1	89	10	10	10	79
12	2	0	0	90	20	10	10	70
13	1	0	0	90	30	10	10	60
14	0	10	5	90	40	10	10	50
15	0	10	0	95	50	10	10	45
16	0	10	0	95	60	10	10	35
17	0	10	0	95	70	10	10	25
18	0	10	0	95	80	10	10	15
19	0	10	0	95	90	5	5	5
20	0	10	0	95	95	0	0	0

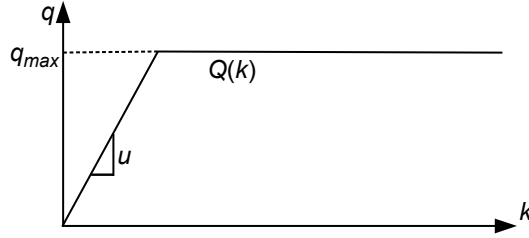


Figure 9.25: Fundamental diagram in the point queue model.

9.5.3 Point and spatial queues and the LWR model (*)

(This optional section shows how the previously-introduced point and spatial queue models can be seen as special cases of the LWR model.)

The first link models introduced in this chapter were the point and spatial queue models, in Section 9.1. These were presented as simple link models to illustrate concepts like the sending and receiving flow, rather than realistic depictions of traffic flow. Nevertheless, it is possible to view the point and spatial queue models as special cases of the LWR model by making an appropriate choice of the fundamental diagram, as shown in this section. Applying the Newell-Daganzo method with this fundamental diagram gives us a second way to derive the expressions for sending and receiving flow for these models.

The point queue model is equivalent to assuming that the flow-density relationship is as shown in Figure 9.25. This diagram is unlike others we've seen, because there is no jam density. This represents the idea that the point queue occupies no physical space: no matter how many vehicles are in queue, there is nothing to prevent additional vehicles from entering the link and joining the queue. It is also the simplest diagram which we have seen so far, and is defined by only two parameters: the free-flow speed u_f and the capacity q_{max} . (Even the triangular fundamental diagram in Figure 9.20 required a third parameter, either $-w$ or k_j .)

The Newell-Daganzo method leads to a simple expression for the sending and receiving flows in a point queue model. To calculate the sending flow $S(t)$, we need to examine the downstream end of the link, so $x = L$. Since we are solving in increasing order of time, we already know $N^\downarrow(0), N^\downarrow(\Delta t), \dots, N^\downarrow(t)$ (the number of vehicles which have left the link at each time interval). Likewise, we know how many vehicles have entered the link at earlier points in time, so we know $N^\uparrow(0), N^\uparrow(\Delta t), \dots, N^\uparrow(t)$. For the sending flow, we are assuming that there are no obstructions from downstream. If this were the case, then we can calculate $N^\downarrow(t + \Delta t)$ using the Newell-Daganzo method, and

$$S(t) = N^\downarrow(t + \Delta t) - N^\downarrow(t). \quad (9.66)$$

In the point queue model, there are two possible wave speeds, u_f (corre-

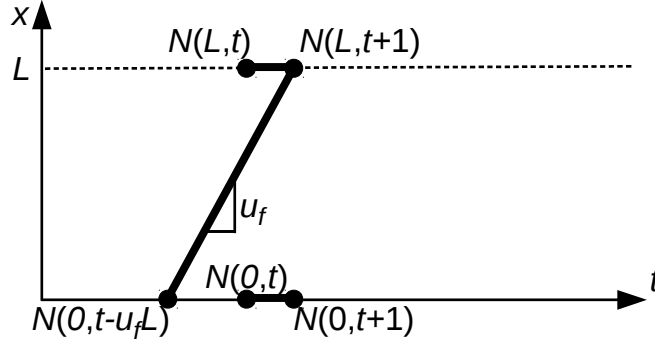


Figure 9.26: Point queue model characteristics for sending and receiving flow.

sponding to free-flow conditions) and zero (corresponding to flow moving at capacity). Figure 9.26 shows how these characteristics can be traced back to known data, either at the upstream end $x = 0$, or at the downstream end $x = L$. Therefore, $t_R = t$, $x_U = 0$, and $t_U = (t + \Delta t) - L/u_f$. We can think of this as a special case of a “trapezoidal” diagram where the jam density k_j is infinite. Applying equation (9.48), we see that if $k_j = \infty$ then the minimum must occur in one of the first two terms, so

$$N^\downarrow(t + \Delta t) = \min\{N^\uparrow((t + \Delta t) - L/u_f), N^\downarrow(t) + q_{max}\Delta t\} \quad (9.67)$$

and

$$S(t) = \min\{N^\uparrow((t + \Delta t) - L/u_f) - N^\downarrow(t), q_{max}\Delta t\}. \quad (9.68)$$

The two terms in the minimum in (9.68) correspond to the case when the queue is empty, and when there are vehicles in queue. In the first term, since there is no queue, we just need to know how many vehicles will finish traversing the physical section of the link between t and $t + \Delta t$; this is exactly the difference between the total number of vehicles which have entered by time $t + \Delta t - L/u_f$ and the total number that have left by time t . When there is a queue, the vehicles exit the link at the full capacity rate.

In these expressions, it is possible that $(t + \Delta t) - L/u_f$ is not an integer, that is, it does not line up with one of the discretization points exactly. In this case the most accurate choice is to interpolate between the known time points on either side (remember that we chose Δt so that $L/u_f \geq 1$). If you are willing to sacrifice some accuracy for efficiency, you can choose to round to the nearest integer, or to adjust the length of the link so that L/u_f is an integer.

For the receiving flow, we look at the upstream end of the link. We can treat the same points as known — $N^\uparrow(0)$, $N^\uparrow(\Delta t)$, \dots , $N^\uparrow(t)$ and $N^\downarrow(0)$, $N^\downarrow(\Delta t)$, \dots , $N^\downarrow(t)$. *Since the fundamental diagram for the point queue model has no decreasing portions, the known data at the downstream end can never be relevant.* (A line connecting one of these points to the unknown point $N^\uparrow(t + \Delta t)$ must have

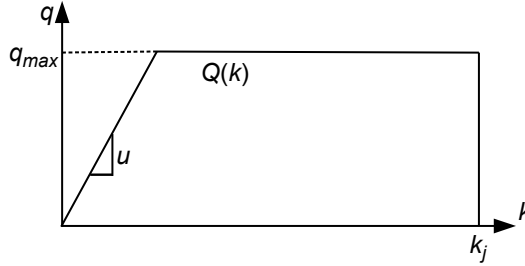


Figure 9.27: Fundamental diagram in the spatial queue model.

negative slope, see Figure 9.26.) Furthermore, for the receiving flow, the characteristic with positive slope $+u_f$, corresponding to upstream conditions, is irrelevant because we are assuming an unlimited number of vehicles are available to move from upstream — and therefore its term in (9.48) will never be the minimum. We are only left with the middle term, corresponding to capacity, so

$$N^\uparrow(t + \Delta t) = N^\uparrow(t) + q_{max}\Delta t \quad (9.69)$$

and

$$R(t) = q_{max}\Delta t. \quad (9.70)$$

In terms of the fundamental diagram, the spatial queue model takes the form in Figure 9.27. This diagram requires three parameters to calibrate: the free-flow speed u_f , the capacity q_{max} , and the jam density k_j . Notice, however, that the fundamental diagram is discontinuous, and immediately drops from q_{max} to zero once jam density is reached. This implies that backward-moving shockwaves can have infinite speed in the spatial queue model — a physical interpretation is that when vehicles at the front of the queue begin moving, vehicles at the rear of the queue immediately start moving as well. In reality, there is a delay before vehicles at the rear of the queue begin moving, and this can be treated as an artifact arising from simplifying assumptions made in the spatial queue model.⁹

There are thus three possible characteristic speeds: $+u_f$ at free-flow, 0 at capacity flow, and $-\infty$ when the queue reaches jam density. The Newell-Daganzo method is applied in much the same way as was done for the point queue model. In particular, the sending flow expression is exactly the same, because the two characteristics with nonnegative velocity are the same. We thus have

$$N(L, t + 1) = \min\{N^\uparrow((t + \Delta t) - L/u_f), N^\downarrow(t) + q_{max}\Delta t\} \quad (9.71)$$

and

$$S(t) = \min\{N^\uparrow((t + \Delta t) - L/u_f) - N^\downarrow(t), q_{max}\Delta t\}. \quad (9.72)$$

⁹Alternatively, connected and autonomous vehicles may be able to exhibit such behavior if an entire platoon of vehicles coordinates its acceleration.

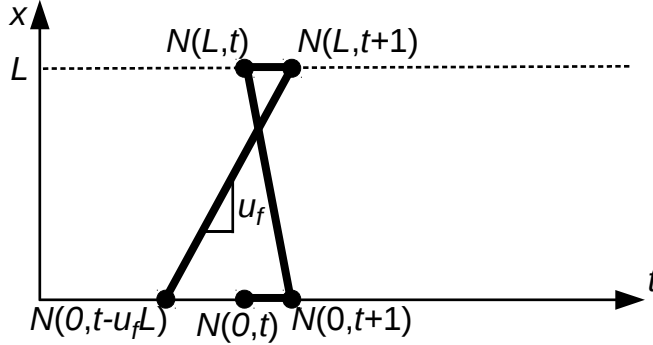


Figure 9.28: Spatial queue model characteristics for sending and receiving flow; the upstream-moving wave is an approximation of the vertical component of the fundamental diagram.

For the receiving flow, we have to take into account the new shockwave speed. Dealing with an infinite speed can be tricky, since, taken literally, would mean that the upstream cumulative count $N^\uparrow(t + \Delta t)$ could depend on the downstream cumulative count at the same time $N^\downarrow(t + \Delta t)$. Since we are solving the model in forward order of time, however, we do not know the value $N^\downarrow(t + \Delta t)$ when calculating $N^\uparrow(t + \Delta t)$. In an acyclic network, we could simply do the calculations such that $N^\downarrow(t + \Delta t)$ is calculated first before $N^\uparrow(t + \Delta t)$, using the concept of a topological order. In networks with cycles — virtually all realistic traffic networks — this will not work. Instead, what is best is to approximate the “infinite” backward wave speed with one which is as large as possible, basing the calculation on the most recent known point $N^\downarrow(t)$ (Figure 9.28). Effectively, this replaces the infinite backward wave speed with one of speed $L/\Delta t$. Equation (9.48) thus gives

$$N^\uparrow(t + \Delta t) = \min\{N^\uparrow(t) + q_{max}\Delta t, N^\downarrow(t) + k_j L\} \tag{9.73}$$

and

$$R(t) = \min\{q_{max}\Delta t, (N^\downarrow(t) + k_j L) - N^\uparrow(t)\}. \tag{9.74}$$

Equation (9.74) will ensure that the number of vehicles on the link will never exceed $k_j L$, assuming that this is true at time zero, as you are asked to show in Exercise 4.

9.5.4 Discussion

This chapter has presented four different link models: point queues, spatial queues, the cell transmission model, and the link transmission model. Although not initially presented this way, all four can be seen as special cases of the Lighthill-Whitham-Richards model. The point and spatial queue models can

be derived from particularly simple forms of the fundamental diagram (as well as from physical first principles, as in Section 9.1), while the cell transmission model and link transmission model are more general methods which can handle more sophisticated fundamental diagrams (typically triangular or trapezoidal in practice). The cell transmission model directly solves the LWR system of partial differential equations by discretizing in space and time, and applying a finite-difference approximation. The link transmission model is based on the Newell-Daganzo method. The primary distinction between these methods is that the Newell-Daganzo method only requires tracking the cumulative counts N at the upstream and downstream ends of each link in time, while the cell transmission model also requires tracking the number of vehicles n at intermediate cells within the link. However, the cell transmission model does not require storing any values from previous time steps, and can function entirely using the number of vehicles in each cell at the current time. The Newell-Daganzo method requires that some past cumulative counts be stored, for the amount of time needed for a wave to travel from one end of the link to the other. Which is more desirable depends on implementation details, and on the specific application context — at times it may be useful to know the distribution of vehicles within a link (as the cell transmission model gives), while for other applications this may be an irrelevant detail. One final advantage of the Newell-Daganzo method is that the values it gives are exact. In the cell transmission model, backward-moving shockwaves will tend to “spread out” as a numerical issue involved in the discretization; this will not happen when applying the Newell-Daganzo method. The exercises explore this issue in more detail. On the other hand, the cell transmission model is easier to explain to decision-makers, and its equations have intuitive explanations in terms of vehicles moving within a link and the amount of available space. The Newell-Daganzo method is a “deeper” method requiring knowledge of partial differential equations, and seems more difficult to convey to nontechnical audiences.

The point queue and spatial queue models have their places as well, despite their strict assumptions. The major flaw in the point queue model, from the standpoint of realism, is its inability to model queue spillbacks which occur when links are full. On the other hand, by ignoring this phenomenon, the point queue model is much more tractable, and is amenable even to closed-form expressions of delay and sensitivity to flows. It is also more robust to errors in input data, because queue spillback can introduce discontinuities in the network loading. There are cases where this simplicity and robustness may outweigh the (significant) loss in realism induced by ignoring spillbacks. The spatial queue model can represent spillbacks, but will tend to underestimate its effect due to its assumption of infinitely-fast backward moving shockwaves. Nevertheless, it can also lead to simpler analyses than the link transmission model.

9.6 Fancier Node Models

Section 9.2 introduced node modeling concepts, and how the sending and receiving flows from adjacent links are mapped to transition flows showing how many vehicles move from each incoming link to each outgoing link in a single time step. In analogy with Section 9.5, we now expand our discussion of node models beyond the simple intersection types presented thus far. A general intersection can have any number of incoming and outgoing links. General intersections can represent signal-controlled intersections, stop-controlled intersections, roundabouts, and so forth. Node models for general intersections are less standardized, and less well-understood, than node models for merges and diverges, and representations of these can vary widely in dynamic traffic assignment implementations. This section presents four alternative models for general intersections: the first for simple traffic signals, the second for intersections where drivers strictly take turns or have equal priority for all turning movements (such as an all-way stop), and the third where approaches have different priority levels and crossing conflicts must be considered, such as a two-way stop or signal with permitted phasing. A fourth node model is presented which allows one to use arbitrary models for turning movement capacity and intersection delay. As you read through this section, it would be helpful to think about how these models can be adapted or extended to represent more sophisticated signal types, roundabouts, and other types of junctions. Many of the ideas in these models are based on the ideas described for merges and diverges.

9.6.1 Basic signals

For the purposes of this section, a “basic signal” is one which (1) only has protected phases (no permitted turns which must yield to oncoming traffic) and (2) all turn movements corresponding to the same approach move simultaneously (for instance, where there are no turn lanes). See Figure 9.29 for an illustration. In this case, the node can be modeled as a diverge intersection, where the “upstream” link varies over time, depending on which approach has the green indication. Flows from other approaches (which have red indications) are set to zero. Following the notation for diverges, we use p_{hij} to reflect the proportion of the sending flow from approach (h, i) which wishes to leave via link (i, j) : these values must be nonnegative, and $\sum_{(i,j):(h,i,j) \in \Xi(i)} p_{hij} = 1$ for all approaches (h, i) . The algorithm is as follows:

1. Let $(h^*, i) \in \Gamma^{-1}(i)$ be the approach which has the green indication at the current time.
2. Calculate the fraction of flow which can move:

$$\phi = \min_{(i,j):(h^*,i,j) \in \Xi(i)} \left\{ \frac{R_{ij}}{p_{h^*ij} S_{h^*i}}, 1 \right\}. \quad (9.75)$$

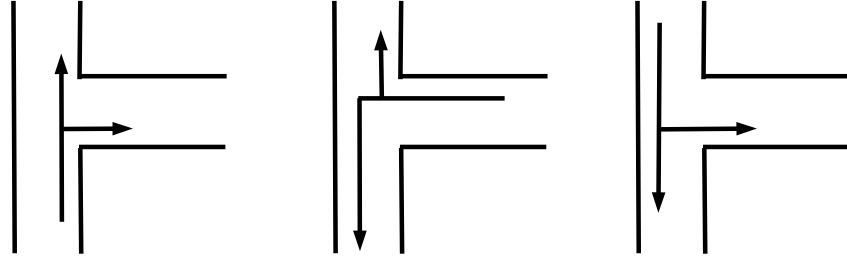


Figure 9.29: Phasing plan for a basic signal.

3. Calculate the transition flows for each turning movement:

$$y_{hij} = \begin{cases} \phi S_{hi} p_{hij} & \text{if } h = h^* \text{ and } (h, i, j) \in \Xi(i) \\ 0 & \text{otherwise} \end{cases} \quad (9.76)$$

In this implementation, one must be a little bit careful if the green times in the signal are not multiples of the time step Δt . It is possible to round the green times so that they are multiples of Δt , but this approach can introduce considerable error over the analysis period: for instance, assume that Δt is equal to six seconds, and a two-phase intersection has green times of 10 seconds and 14 seconds, respectively. Rounding to multiples of the time step would give both phases twelve seconds each, which seems reasonable enough; but over a three-hour analysis period, the phases would receive 75 and 105 minutes of green time in reality, as compared to 90 minutes each in simulation. In highly congested situations, this can introduce considerable error. This issue can be avoided if, instead of rounding, one gives the green indication to the approach which would have green in reality at that time. In the example above, the intersection has a cycle length of 24 seconds. So, when $t = 60$ seconds, we are 12 seconds into the third cycle; and at this point the green indication should be given to the second phase. In this way, there is no systematic bias introduced into the total green time each approach receives.

9.6.2 Equal priority movements

An all-way stop intersection is characterized by turn-taking: that is, vehicles have the opportunity to depart the intersection in the order in which they arrive. No turning movement has priority over any other, but the turning movements from different approaches interact with each other and may compete for space on the same outgoing link. This is different from the basic signal model, where the phasing scheme ensures that at most one approach is attempting to use an outgoing link at any given point of time.

Intersections with equal priority movements have characteristics of both diverges and merges. Like a diverge, if a vehicle is unable to turn into a down-

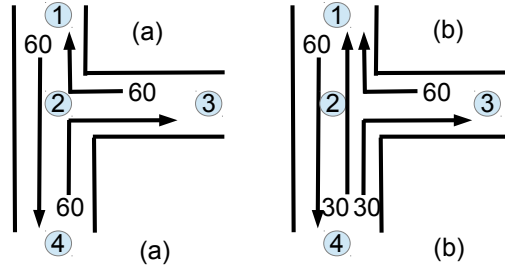


Figure 9.30: Oriented capacities for a three-leg intersection, where all approaches have $q_{max} = 60$. (a) All approaches map to a unique outgoing link. (b) The flow on approach (4,2) is split between two outgoing links.

stream link because of an obstruction, we assume that the vehicle obstructs all other vehicles from the same approach, respecting the FIFO principle. This means that the outflows for all of the turning movements corresponding to any approach must follow the same proportions as the number of drivers *wishing* to use all of these movements. Similar to a merge, we assume that if there are high sending flows from all the approaches, the fraction of the receiving flow allocated to each approach is divided up proportionally. However, instead of allocating the receiving flow R_{ij} to approach (h, i) based on the full capacity q_{max}^{hi} , we instead divide up the receiving flow based on the *oriented capacity*

$$q_{max}^{hij} = q_{max}^{hi} p_{hij}, \tag{9.77}$$

where p_{hij} is the proportion of the flow from approach (h, i) which wishes to exit on link (i, j) . (If turn movement $[h, i, j]$ is not in the allowable set $\Xi(i)$, then $p_{hij} = 0$.)

Multiplying the capacity by this proportion reflects the fact that an upstream approach can only make use of an available space on a downstream link if there is a vehicle wishing to turn. In Figure 9.30(a), each incoming link uses a unique exiting link, and thus can claim its full capacity. In Figure 9.30(b), half the vehicles on link (4,2) want to turn right and half wish to go straight, whereas all the vehicles on link (3,2) wish to turn right. Link (3,2) therefore has twice as many opportunities to fill available space on link (2,1), and thus its rightful share is twice that of link (4,2). For any two approaches $[h, i, j]$ and $[h', i, j]$ using the same outgoing link, we thus require that

$$\frac{y_{hij}}{y_{h'ij}} = \frac{q_{max}^{hij}}{q_{max}^{h'ij}}. \tag{9.78}$$

assuming that both approaches are fully competing for the link (i, j) . If an approach has a small sending flow, it may use less of its assigned receiving flow than equation (9.78) allocates, and this unused receiving flow may be used by other approaches.

We also define the *oriented sending flow*

$$S_{hij} = S_{hi}p_{hij} \quad (9.79)$$

to reflect the demand for the turning movement $[h, i, j]$. Following the same principles as the merge model, if the oriented sending flow from an approach is less than its proportionate share of a downstream link's receiving flow, its unused share will be divided among the other approaches with unserved demand still remaining, in proportion to their oriented capacities. If the oriented sending flow for a turn movement is greater than the oriented receiving flow for that movement, then by the FIFO principle applied to diverges, it will restrict flow to all other downstream links by the same proportion, and for any two turning movements (h, i, j) and (h, i, j') from the same approach we must have

$$\frac{y_{hij}}{y_{hij'}} = \frac{S_{hij}}{S_{hij'}} = \frac{p_{hij}}{p_{hij'}}. \quad (9.80)$$

We can rearrange this equation to show that the ratio y_{hij}/S_{hij} (the ratio of actual flow and desired flow for any turning movement) is uniform for all the turning movements approaching from link (h, i) — this ratio plays the same role as ϕ in a diverge.

The presence of multiple incoming and outgoing links causes another complication, in that the flows between approaches are all linked together. If an approach is restricted by the receiving flow of a downstream link, flow from that approach is restricted to *all* other downstream links. This means that the approach may not fully consume its “rightful share” of another downstream link, thereby freeing up additional capacity for a different approach. Therefore, we cannot treat the approaches or downstream links separately or even sequentially in a fixed order, because we do not know *a priori* how these will be linked together.

However, there is an algorithm which generates a consistent solution despite these mutual dependencies. In this algorithm, each approach link can be *demand-constrained*, or *supply-constrained by a downstream link*. If an approach is demand-constrained, its oriented sending flow to all downstream links is less than its rightful share, and therefore all of the sending flow can move. If an approach is supply-constrained by link (i, j) , then the approach is unable to move all of its sending flow, and the fraction which can move is dictated by link (i, j) . (That is, receiving flow on (i, j) is the most restrictive constraint for the approach). The algorithm must determine which links are demand-constrained, and which are supply-constrained by a downstream link.

To find such a solution, we define two sets of auxiliary variables, \tilde{S}_{hij} to reflect the amount of unallocated sending flow for movement $[h, i, j]$, and \tilde{R}_{ij} to reflect the amount of unallocated receiving flow for outgoing link (i, j) . These are initialized to the oriented sending flows and link receiving flows, and reduced iteratively as flows are assigned and the available sending and receiving flows are used up. The algorithm also uses the notion of *active* turning movements; these are turning movements whose flows can still be increased. A turning movement

$[h, i, j]$ becomes inactive either when \tilde{S}_{hij} drops to zero (all vehicles that wish to turn have been assigned), or when \tilde{R}_{ij} drops to zero for *any* outgoing link (i, j') that approach (h, i) is using (that is, for which $p_{hij'} > 0$). Allocating all of the receiving flow for one outgoing link can thus impact flow on turning movements which use other outgoing links, because of the principle that vehicles wishing to turn will block others, as expressed in equation (9.80). The set of active turning movements will be denoted by A ; a turning movement remains active until we have determined whether it is supply-constrained or demand-constrained.

At each stage of the algorithm, we will increase the flows for all active turning movements. We must increase these flows in a way which is consistent both with the turning fractions (9.80), and with the division of receiving flow for outgoing links given by equation (9.78), and we will use α_{hij} to reflect the rate of increase for turning movement (h, i, j) . The absolute values of these α values do not matter, only their proportions, so you can scale them in whatever way is most convenient to you. Often it is easiest to pick one turning movement (h, i, j) and fix its α value either to one, or to its oriented sending flow. The turning proportions from (h, i) then fix the α values for all other turning movements from the same approach. You can then use equation (9.78) to determine α values for turning movements competing for the same outgoing link, then use the turning fractions for the upstream link on those turning movements, and so on until a consistent set of α values has been determined.

The algorithm then increases the active turning movement flows in these proportions until some movement becomes inactive, because its sending flow or the receiving flow on its outgoing link becomes exhausted. The process is then repeated with the smaller set of turning movements which remain active, and continues until all possible flows have been assigned. This algorithm is a bit more involved than the node models seen thus far, and you may find it helpful to follow the example below as you read through the algorithm steps.

1. Initialize by calculating oriented capacities and sending flows using equations (9.77) and (9.79); by setting $y_{hij} \leftarrow 0$ for all $[h, i, j] \in \Xi(i)$; by setting $\tilde{S}_{hij} \leftarrow S_{hij}$ for all $[h, i, j] \in \Xi(i)$ and $\tilde{R}_{ij} \leftarrow R_{ij}$ for all $(i, j) \in \Gamma(i)$; and by declaring active all turning movements with positive sending flow: $A \leftarrow \{[h, i, j] \in \Xi(i) : S_{hij} > 0\}$.
2. Identify a set of α_{hij} values for all active turning movements which is consistent with the turning fractions ($\alpha_{hij}/\alpha_{hij'} = p_{hij}/p_{hij'}$ for all turning movements from the same incoming link) and oriented capacities ($\alpha_{hij}/\alpha_{h'ij} = q_{max}^{hij}/q_{max}^{h'ij}$ for all turning movements to the same outgoing link).
3. For each outgoing link $(i, j) \in \Gamma(i)$, identify the rate α_{ij} at which its receiving flow will be reduced, by adding α_{hij} for all active turn movements whose outgoing link is (i, j) : $\alpha_{ij} \leftarrow \sum_{[h, i, j] \in A} \alpha_{hij}$.
4. Determine the point at which some turning movement will become inac-

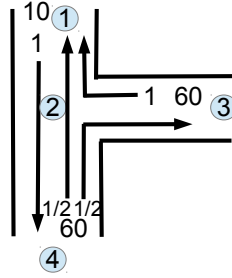


Figure 9.31: Example of equal priorities algorithm, showing sending flows and turning proportions. All links have capacity of 60.

tive, by calculating

$$\theta = \min \left\{ \min_{[h,i,j] \in A} \left\{ \frac{\tilde{S}_{hij}}{\alpha_{hij}} \right\}, \min_{(i,j) \in \Gamma(i): \alpha_{ij} > 0} \left\{ \frac{\tilde{R}_{ij}}{\alpha_{ij}} \right\} \right\}. \quad (9.81)$$

5. Increase flows for active turning movements, and update unallocated sending and receiving flows: for all $[h, i, j] \in A$ update $y_{hij} \leftarrow y_{hij} + \theta \alpha_{hij}$, $\tilde{S}_{hij} \leftarrow \tilde{S}_{hij} - \theta \alpha_{hij}$, and $\tilde{R}_{ij} \leftarrow \tilde{R}_{ij} - \theta \alpha_{hij}$.
6. Update the set of active turning movements, by removing from A any turning movement $[h, i, j]$ for which $\tilde{S}_{hij} = 0$ or for which $\tilde{R}_{ij} = 0$ for any $(i, j') \in \Gamma(i)$ which is being used ($p_{hij'} > 0$).
7. If there are any turning movements which are still active ($A \neq \emptyset$), return to step 3. Otherwise, stop.

As a demonstration, consider the intersection in Figure 9.31, where all links (incoming and outgoing) have the same capacity of 60 vehicles per time step, and the sending flows S_{hi} and turning proportions p_{hij} are shown. None of the downstream links is congested, so their receiving flows are equal to the capacity. (In the figure, two sets of numbers are shown for each approach; the “upstream” number is the sending flow and the “downstream” number(s) are the proportions.) The oriented capacities can be seen in Figure 9.30(b). For this example, the six turning movements will be indexed in the following order:

$$\Xi(2) = \{[1, 2, 3], [1, 2, 4], [3, 2, 1], [3, 2, 4], [4, 2, 1], [4, 2, 3]\} \quad (9.82)$$

All vectors referring to turning movements will use this ordering for their components.

Step 1 of the algorithm initializes the oriented sending flows using equation (9.79),

$$\mathbf{S} = [S_{hij}] = [0 \quad 10 \quad 60 \quad 0 \quad 30 \quad 30] \quad (9.83)$$

and the oriented capacities using equation (9.77)

$$\mathbf{q}_{\max} = [q_{max}^{hij}] = [0 \quad 60 \quad 60 \quad 0 \quad 30 \quad 30]. \quad (9.84)$$

The step also initializes the turning movement flows and auxiliary variables:

$$\mathbf{y} \leftarrow [0 \ 0 \ 0 \ 0 \ 0 \ 0], \quad (9.85)$$

$$\tilde{\mathbf{S}} \leftarrow [S_{hij}] = [0 \ 10 \ 60 \ 0 \ 30 \ 30], \quad (9.86)$$

and

$$\tilde{\mathbf{R}} = [\tilde{R}_{21} \ \tilde{R}_{23} \ \tilde{R}_{24}] = [60 \ 60 \ 60]. \quad (9.87)$$

The set of active turn movements is $A = \{[1, 2, 4], [3, 2, 1], [4, 2, 1], [4, 2, 3]\}$.

Step 2 of the algorithm involves calculation of a set of consistent α_{hij} values. One way of doing this is to start by setting $\alpha_{423} \leftarrow S_{423} = 30$. The turning fractions from (4,2) then require that $\alpha_{421} = \alpha_{423} = 30$. The allocation rule for outgoing link (2,1) then forces $S_{321} = 60$: the oriented capacity for $[3, 2, 1]$ is twice that of $[4, 2, 1]$, and the α values must follow the same proportion. Turning movement $[1, 2, 4]$ is independent of all of the other turning movements considered thus far, so we can choose its α value arbitrarily; say, $\alpha_{124} \leftarrow S_{124} = 10$. (You should experiment around with different ways of calculating these α values, and convince yourself that the final flows are the same as long as the proportions of α values for interdependent turning movements are the same.) We thus have

$$\boldsymbol{\alpha} = [\alpha_{hij}] = [0 \ 10 \ 60 \ 0 \ 30 \ 30]. \quad (9.88)$$

The α values for inactive turning movements have been set to zero for clarity; their actual value is irrelevant because they will not be used in any of the steps that follow.

With these flow increments, flow on outgoing links (2,1), (2,3), and (2,4) will be $\alpha_{21} = 90$, $\alpha_{23} = 30$, and $\alpha_{24} = 10$, as dictated by Step 3.

In Step 4, we determine how much we can increase the flow at the rates given by $\boldsymbol{\alpha}$ until some movement becomes inactive. We have

$$\theta = \min \left\{ \begin{array}{ccccccc} \tilde{S}_{124} & \tilde{S}_{321} & \tilde{S}_{421} & \tilde{S}_{423} & \tilde{R}_{21} & \tilde{R}_{23} & \tilde{R}_{24} \\ \frac{10}{10}, & \frac{60}{60}, & \frac{30}{30}, & \frac{30}{30}, & \frac{60}{90}, & \frac{60}{30}, & \frac{60}{10} \end{array} \right\} \quad (9.89)$$

or $\theta = 2/3$.

We can now adjust the flows, as in Step 5. We increase the flow on each active turning movement by $\frac{2}{3}\alpha_{hij}$, giving

$$\mathbf{y} \leftarrow [0 \ 6\frac{2}{3} \ 40 \ 0 \ 20 \ 20]. \quad (9.90)$$

We subtract these flow increments from the auxiliary sending and receiving flows, giving

$$\tilde{\mathbf{S}} \leftarrow [0 \ 3\frac{1}{3} \ 20 \ 0 \ 10 \ 10] \quad (9.91)$$

and

$$\tilde{\mathbf{R}} \leftarrow [0 \ 40 \ 53\frac{1}{3}]. \quad (9.92)$$

Step 6 updates the set of active turning movements. With the new $\tilde{\mathbf{S}}$ and $\tilde{\mathbf{R}}$ values, we see that $[3, 2, 1]$ and $[4, 2, 1]$ have become inactive, since there is no

remaining receiving flow on link (2, 1). Furthermore, this inactivates movement [4, 2, 3]: even though there are still travelers that wish to turn in this direction, and space on the downstream link (\tilde{S}_{423} and \tilde{R}_{23} are still positive), they are blocked by travelers waiting to use movement [4, 2, 1]. So, there is only one active movement remaining, $A \leftarrow \{[1, 2, 4]\}$, and we must return to step 3.

In Step 3, we must recalculate the α_{ij} values because some of the turning movements are inactive. With the new set A , we have $\alpha_{21} = \alpha_{23} = 0$ and $\alpha_{24} = 10$. The new step size is

$$\theta = \min \left\{ \frac{\tilde{S}_{124}}{\frac{3\frac{1}{3}}{10}}, \frac{\tilde{R}_{24}}{\frac{53\frac{1}{3}}{10}} \right\} = \frac{1}{3}. \quad (9.93)$$

We then increase the flows, increasing y_{124} by $10 \times \frac{1}{3}$ to 10, decreasing \tilde{S}_{124} to zero, and decreasing \tilde{R}_{24} to 50. This change inactivates movement [1, 2, 4]. Since there are no more active turning movements, the algorithm terminates, and the final vector of flows is

$$\mathbf{y} = [0 \quad 10 \quad 40 \quad 0 \quad 20 \quad 20]. \quad (9.94)$$

9.6.3 Intersections with priority

Intersections which allow crossing conflicts are more complex to model than the intersection types described above. These include intersections with stop control only on some of the approaches, but not all, or signalized intersections with permitted movements that must yield to another traffic stream. One way to model this type of intersection is to introduce a set C of conflict points. Like outgoing links, we associate a receiving flow R_c indicating the maximum number of vehicles which can pass through this conflict point during a single time step Δt . For each conflict point $c \in C$, let $\Xi(c)$ denote the turning movements which make use of this conflict point. Then, for each turning movement $[h, i, j] \in \Xi(c)$, we define a (strictly positive) priority parameter β_{hij}^c . These priority parameters are interpreted through their ratios: the share of the receiving flow allocated to turning movement $[h, i, j]$ relative to that allocated to turning movement $[h', i, j']$ is in the same proportion as the ratio

$$\beta_{hij}^c p_{hij} / \beta_{h'i'j'}^c p_{h'i'j'}. \quad (9.95)$$

These play an analogous role to the oriented capacities defined in the previous subsection, in suggesting how the conflict point receiving flow should be divided among the competing approaches. They are used more generally, however, to reflect priority rules. For instance, a through movement often has priority over a turning movement which crosses it. Even if the capacity and sending flows of the through lane and turn lane are the same, the through lane should have access to a greater share of the crossing point's receiving flow. If, at full saturation, ten through vehicles move for every turning vehicle, then the β value for the through movement should be ten times the β value for the turning movement.

The node model is simplified if we assume that the β values are all strictly positive. You might find this unrealistic: in the example above, if the turning movement must yield to the through movement, then at full saturation perhaps *no* turning vehicles could move. In practice, however, priority rules are not strictly obeyed as traffic flows near saturation. Polite through drivers may stop to let turning drivers move, or aggressive turning drivers might force their way into the through stream. (Think of what would happen at a congested freeway if vehicles merging from an onramp took the “yield” sign literally!) The requirement of strictly positive β values thus has some practical merits, as well as mathematical ones. The exercises explore ways to generalize this node model, including strict priority, and cases where different turning movements may consume different amounts of the receiving flow (for instance, if they are moving at different speeds).

We can now adapt the algorithm for equal priority for the case of intersections with different priorities. The algorithm is augmented by adding receiving flows R_c and auxiliary receiving flows \tilde{R}_c for each conflict point, and we extend some of the computations to include the set of conflict points. First, we require that the ratio of α values for two turning movements using the same crossing point follow the ratio of the β values, assuming saturated conditions:

$$\frac{\alpha_{hij}}{\alpha_{h'ij'}} = \frac{\beta_{hij}^c p_{hij}}{\beta_{h'ij'}^c p_{h'ij'}} \quad \forall c \in C; [h, i, j], [h', i, j'] \in \Xi(c). \quad (9.96)$$

Note that the β values are multiplied by the relevant turning fraction for the movements. As with the oriented capacity, this respects the fact that the more flow is attempting to turn in a particular direction, the more opportunities or gaps will be available for it to claim.

Second, we must calculate the inflow rates to conflict points given the α values from active turning movements:

$$\alpha_c = \sum_{[h,i,j] \in \Xi(c) \cap A} \alpha_{hij}. \quad (9.97)$$

Third, the calculation of the step size θ must now include obstructions from conflict points:

$$\theta = \min \left\{ \min_{[h,i,j] \in A} \left\{ \frac{\tilde{S}_{hij}}{\alpha_{hij}} \right\}, \min_{(i,j) \in \Gamma(i): \alpha_{ij} > 0} \left\{ \frac{\tilde{R}_{ij}}{\alpha_{ij}} \right\}, \min_{c \in C: \alpha_c > 0} \left\{ \frac{\tilde{R}_c}{\alpha_c} \right\} \right\}. \quad (9.98)$$

With these modifications, the algorithm proceeds in the same way as before. Specifically,

1. Initialize by calculating oriented capacities and sending flows using equations (9.77) and (9.79); by setting $y_{hij} \leftarrow 0$ for all $[h, i, j] \in \Xi(i)$; by setting $\tilde{S}_{hij} \leftarrow S_{hij}$ for all $[h, i, j] \in \Xi(i)$, $\tilde{R}_{ij} \leftarrow R_{ij}$ for all $(i, j) \in \Gamma(i)$, and $\tilde{R}_c \leftarrow R_c$ for each $c \in C$; and by declaring active all turning movements with positive sending flow: $A \leftarrow \{[h, i, j] \in X(i) : S_{hij} > 0\}$.

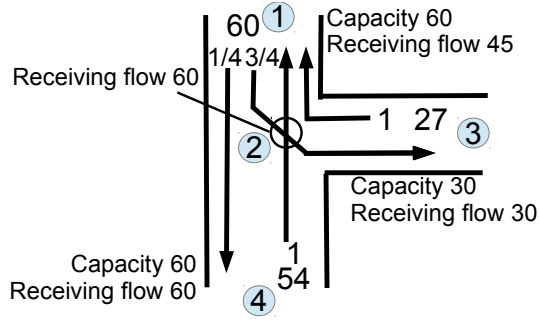


Figure 9.32: Example of intersection priority algorithm, showing sending flows and turning proportions. Link capacity and receiving flows shown.

2. Identify a set of α_{hij} values for all active turning movements which is consistent with the turning fractions ($\alpha_{hij}/\alpha_{hij'} = p_{hij}/p_{hij'}$ for all turning movements from the same incoming link), oriented capacities ($\alpha_{hij}/\alpha_{h'ij} = q_{max}^{hij}/q_{max}^{h'ij}$ for all turning movements to the same outgoing link), and conflict points based on equation (9.96).
3. For each outgoing link $(i, j) \in \Gamma(i)$ and conflict point $c \in C$, identify the rate α_{ij} at which its receiving flow will be reduced, by adding α_{hij} for all active turn movements whose outgoing link is (i, j) : $\alpha_{ij} \leftarrow \sum_{[h,i,j] \in A} \alpha_{hij}$ for links, and equation (9.97) for conflict points.
4. Determine the point at which some turning movement will become inactive, by calculating θ using equation (9.98).
5. Increase flows for active turning movements, and update unallocated sending and receiving flows: for all $[h, i, j] \in A$ update $y_{hij} \leftarrow y_{hij} + \theta\alpha_{hij}$, $\tilde{S}_{hij} \leftarrow \tilde{S}_{hij} - \theta\alpha_{hij}$; for all $(i, j) \in \Gamma(i)$ update $\tilde{R}_{ij} \leftarrow \tilde{R}_{ij} - \theta\alpha_{ij}$, and for all $c \in C$ update $\tilde{R}_c \leftarrow \tilde{R}_c - \theta\alpha_c$.
6. Update the set of active turning movements, by removing from A any turning movement $[h, i, j]$ for which $\tilde{S}_{hij} = 0$, for which $\tilde{R}_{ij'} = 0$ for any $(i, j') \in \Gamma(i)$ which is being used ($p_{hij'} > 0$), or for which $\tilde{R}_c = 0$ for any movement $[h, i, j'] \in \Xi(c)$ and conflict point c .
7. If there are any turning movements which are still active ($A \neq \emptyset$), return to step 3. Otherwise, stop.

As a demonstration, consider the intersection in Figure 9.32, where all links (incoming and outgoing) have the same capacity of 60 vehicles per time step, and the sending flows S_{hi} and turning proportions p_{hij} are shown. (In the figure, two sets of numbers are shown for each approach; the “upstream” number is the sending flow and the “downstream” number(s) are the proportions.) Links (2, 4)

and (2, 3) have receiving flows of 60 vehicles, while link (2, 1) has a receiving flow of only 45 vehicles. There is one conflict point, indexed c , which is marked with a circle in Figure 9.32). Conflict point c has a receiving flow of 60, and the turning movement $[1, 2, 3]$ must yield to the through movement $[4, 2, 1]$, as reflected by the ratio $\beta_{421}/\beta_{123} = 5$. For this example, the six turning movements will be indexed in the following order:

$$\Xi(2) = \{[1, 2, 3], [1, 2, 4], [3, 2, 1], [3, 2, 4], [4, 2, 1], [4, 2, 3]\} \quad (9.99)$$

All vectors referring to turning movements will use this ordering for their components.

Step 1 of the algorithm initializes the oriented sending flows using equation (9.79),

$$\mathbf{S} = [S_{hij}] = [45 \quad 15 \quad 27 \quad 0 \quad 54 \quad 0] \quad (9.100)$$

and the oriented capacities using equation (9.77)

$$\mathbf{q}_{\max} = [q_{max}^{hij}] = [30 \quad 30 \quad 30 \quad 0 \quad 60 \quad 0]. \quad (9.101)$$

The step also initializes the turning movement flows and auxiliary variables:

$$\mathbf{y} \leftarrow [0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0], \quad (9.102)$$

$$\tilde{\mathbf{S}} \leftarrow [S_{hij}] = [45 \quad 15 \quad 27 \quad 0 \quad 54 \quad 0], \quad (9.103)$$

and

$$\tilde{\mathbf{R}} = [\tilde{R}_{21} \quad \tilde{R}_{23} \quad \tilde{R}_{24} \quad \tilde{R}_c] = [45 \quad 30 \quad 60 \quad 60]. \quad (9.104)$$

The set of active turn movements is $A = \{[1, 2, 3], [1, 2, 4], [3, 2, 1], [4, 2, 1]\}$.

Step 2 of the algorithm involves calculation of a set of consistent α_{hij} values. One way of doing this is to start by setting $\alpha_{123} \leftarrow S_{123} = 15$. The turning fractions from (1, 2) then require that $\alpha_{124} = \alpha_{123} = 45$. The allocation rule (9.96) for conflict point c forces $S_{421} = 300$: the β value for $[4, 2, 1]$ is five times that of the β value for $[1, 2, 3]$, and the turning fraction for $[4, 2, 1]$ is a third higher. Since $S_{421} = 300$, we must have $S_{321} = 150$ because the oriented capacity of $[3, 2, 1]$ is half that of $[4, 2, 1]$. This gives the flow increments

$$\boldsymbol{\alpha} = [\alpha_{hij}] = [45 \quad 15 \quad 150 \quad 0 \quad 300 \quad 0]. \quad (9.105)$$

With these flow increments, flow on outgoing links (2, 1), (2, 3), (2, 4), and the conflict point c will be $\alpha_{21} = 450$, $\alpha_{23} = 45$, $\alpha_{24} = 15$, and $\alpha_c = 345$, as dictated by Step 3.

In Step 4, we determine how much we can increase the flow at the rates given by $\boldsymbol{\alpha}$ until some movement becomes inactive. We have

$$\theta = \min \left\{ \frac{\tilde{S}_{123}}{15}, \frac{\tilde{S}_{124}}{45}, \frac{\tilde{S}_{321}}{150}, \frac{\tilde{S}_{421}}{300}, \frac{\tilde{R}_{21}}{45}, \frac{\tilde{R}_{23}}{45}, \frac{\tilde{R}_{24}}{15}, \frac{\tilde{R}_c}{345} \right\} \quad (9.106)$$

or $\theta = \frac{1}{10}$.

We can now adjust the flows, as in Step 5. We increase the flow on each active turning movement by $\frac{1}{10}\alpha_{hij}$, giving

$$\mathbf{y} \leftarrow [4.5 \quad 1.5 \quad 15 \quad 0 \quad 30 \quad 0] . \quad (9.107)$$

We subtract these flow increments from the auxiliary sending and receiving flows, giving

$$\tilde{\mathbf{S}} \leftarrow [40.5 \quad 13.5 \quad 12 \quad 0 \quad 24 \quad 0] \quad (9.108)$$

and

$$\tilde{\mathbf{R}} \leftarrow [0 \quad 25.5 \quad 58.5 \quad 25.5] . \quad (9.109)$$

Step 6 updates the set of active turning movements. With the new $\tilde{\mathbf{S}}$ and $\tilde{\mathbf{R}}$ values, we see that $[3, 2, 1]$ and $[4, 2, 1]$ have become inactive, since there is no remaining receiving flow on link $(2, 1)$. So, there are two active movements remaining, $A \leftarrow \{[1, 2, 3], [1, 2, 4]\}$, and we must return to step 3.

In Step 3, we must recalculate the α_{ij} values because some of the turning movements are inactive. With the new set A , we have $\alpha_{21} = 0$, $\alpha_{23} = 45$, $\alpha_{24} = 15$, and $\alpha_c = 45$. The new step size is

$$\theta = \min \left\{ \frac{\tilde{S}_{123}}{45}, \frac{\tilde{S}_{124}}{15}, \frac{\tilde{R}_{23}}{45}, \frac{\tilde{R}_{24}}{15}, \frac{\tilde{R}_c}{45} \right\} = \frac{17}{30} . \quad (9.110)$$

We then increase the flows, increasing y_{123} by $45 \times \frac{17}{30}$ to 30 and y_{124} by $15 \times \frac{17}{30}$ to 10, decreasing \tilde{S}_{123} to 15, \tilde{S}_{124} to 5, \tilde{R}_{23} to 0, \tilde{R}_{24} to 50, and \tilde{R}_c to 0. This change inactivates movement $[1, 2, 3]$; and movement $[1, 2, 4]$ is then inactivated because these movement's flows are blocked by vehicles waiting to take $[1, 2, 3]$. Since there are no more active turning movements, the algorithm terminates, and the final vector of flows is

$$\mathbf{y} = [30 \quad 10 \quad 15 \quad 0 \quad 30 \quad 0] . \quad (9.111)$$

9.7 Historical Notes and Further Reading

(These sections are incomplete in this beta version of the text, and will be substantially expanded in the complete first edition.)

The idea of dividing networks into link models and node models (which operate independently of each other, possibly with different models) was suggested by Nie et al. (2008) and Yperman (2007). The sending flow and receiving flow concepts date to Daganzo (1995b) and Lebacque (1996) (who used the terms “demand” and “supply”), although the presentation in this chapter more closely follows that of Yperman (2007). The point queue and spatial queue models were described in Vickrey (1969) and Zhang and Nie (2006).

The list of node model desiderata in this chapter is that of Tampère et al. (2011); the invariance principle specifically is discussed more in Lebacque and Khoshyaran (2005). The diverge and merge equations are that of Daganzo

(1995a); for an alternative merge model (which does *not* satisfy the invariance principle), see Jin and Zhang (2003).

The hydrodynamic traffic flow theory described in Section 9.4.3 was independently developed by Lighthill and Whitham (1955) and Richards (1956). Newell (1993a), Newell (1993b), and Newell (1993c) recognized that the cumulative vehicle counts N , and the analysis of characteristics resulting from a triangular fundamental diagram, greatly simplify the solution of the model, a theory completed by Daganzo (2005a) and Daganzo (2005b). Interestingly, an equivalent model (used for soil erosion) was separately developed by Luke (1972). There are alternative means of solving the LWR model not presented in this book, through recognizing it as a Hamilton-Jacobi system of partial differential equations (LeVeque, 1992; Evans, 1998) which can be solved using the Lax-Hopf formula or viability theory (Lax, 1957; Hopf, 1970; Claudel and Bayen, 2010); or for the purposes of a link model, representing sending and receiving flows using a “double queue,” one at each end of the link (Osorio and Bierlaire, 2009; Osorio et al., 2011), a representation suitable for a stochastic version of the LWR model.

The cell transmission model was reported in Daganzo (1994) and Daganzo (1995a), essentially a Gudanov scheme for solving the LWR system (Godunov, 1959; Lebacque, 1996). The link transmission model was developed by Yperman (2007); see also Gentile (2010).

The more sophisticated node models reported later in the chapter are adapted from Tampère et al. (2011) and Corthout et al. (2012).

Finally, network loading can be accomplished by entirely different means than that reported in this chapter, without the use of explicit link and node models discretized in space and time. For instance, the discretization can be done in the space of *vehicle trajectories* (Bar-Gera, 2005). In mathematical terms, this involves converting from Eulerian coordinates (x and t) to Lagrangian coordinates (with the cumulative count N in place of either x or t). For more on this alternative, and reformulations of the LWR model with this change of variables, see Laval and Leclercq (2013).

Another common alternative is to use traffic simulation to perform the network loading. Examples include the software packages VISSIM (Fellendorf, 1994), AIMSUN (Barcelo, 1998), DynaMIT (Ben-Akiva et al., 1998), VISTA (Ziliaskopoulos and Waller, 2000), DYNASMART (Mahmassani, 2000), Dynameq (Mahut et al., 2003), and DynusT.

9.8 Exercises

1. [14] Table 9.7 shows cumulative inflows and outflows to a link with a capacity of 10 vehicles per time step, and a free-flow time of 2 time steps. Use the point queue model to calculate the sending and receiving flow for each time step.
2. [14] Repeat Exercise 1 with the spatial queue model. Assume that the jam

Table 9.7: Upstream and downstream counts for Exercises 1 and 2.

t	$N^\uparrow(t)$	$N^\downarrow(t)$
0	0	0
1	5	0
2	10	0
3	15	2
4	20	4
5	20	6
6	20	10
7	20	15
8	20	20
9	20	20

density is such that at most 20 vehicles can fit on the link simultaneously.

3. [33] In the point queue model, if the inflow and outflow rates q_{in} and q_{out} are constants with $q_{in} \geq q_{out}$, show that the travel time experienced by the n -th vehicle is $L/u_f + \frac{n}{q_{in}} \left(\frac{q_{in}}{q_{out}} - 1 \right)$. The same result holds for the spatial queue model, if there is no spillback.
4. [24] In the spatial queue model, show that if the total number of vehicles on a link is at most $k_j L$ at time t , then the total number of vehicles on the link at time $t + 1$ is also at most $k_j L$.
5. [13] In a merge, approach 1 has a sending flow of 50 and a capacity of 100. Approach 2 has a sending flow of 100 and a capacity of 100. Report the number of vehicles moving from each approach if (a) the outgoing link has a receiving flow of 150; (b) the outgoing link has a receiving flow of 120; and (c) the outgoing link has a receiving flow of 100.
6. [32] Extend the merge model of Section 9.2.2 to a merge node with three incoming links.
7. [43] Show that the formula (9.14) indeed captures both case II and case III of the merge model presented in Section 9.2.2.
8. [32] Show that the merge model of Section 9.2.2 satisfies all the desiderata of Section 9.2.
9. [41] Consider an alternative merge model for the congested case, which allocates the receiving flow proportional to the *sending flows* of the incoming links, rather than proportional to the *capacities* of the incoming links as was done in Section 9.2.2. Show that this model does *not* satisfy the invariance principle.

10. [13] In a diverge, the incoming link has a sending flow of 120, 25% of the vehicles want to turn onto outgoing link 1, and the remainder want to turn onto outgoing link 2. Report the number of vehicles moving to outgoing links 1 and 2 if their respective receiving flows are (a) 80 and 100; (b) 80 and 60; (c) 10 and 40.
11. [10] Extend the diverge model of Section 9.2.3 to a diverge node with three outgoing links.
12. [21] Develop a model for a diverge node with two outgoing links, in which flows waiting to enter one link do *not* block flows entering the other link. When might this model be more appropriate?
13. [25] Show that the diverge model of Section 9.2.3 satisfies all the desiderata of Section 9.2.
14. [10] In the network loading procedure in Section 9.3, we specified that centroid connectors starting at origins should have high jam density, and those ending at destinations should have high capacity. Would anything go wrong if centroid connectors starting at origins also had high capacity? What if centroid connectors ending at destinations had high jam density?
15. [12] Draw trajectory diagrams which reflect the following situations: (a) steady-state traffic flow, no vehicles speeding up or slowing down; (b) vehicles approaching a stop sign, then continuing; (c) a slow semi truck merges onto the roadway at some point in time, then exits at a later point in time. Draw at least five vehicle trajectories for each scenario.
16. [33] The relationship $q = uk$ can be used to transform the fundamental diagram (which relates density and flow) into a relationship between density and speed, and vice versa. For each of these three speed-density relationships, derive the corresponding fundamental diagram.
 - (a) The Greenshields (linear) model: $u = u_f(1 - k/k_j)$.
 - (b) The Greenberg (logarithmic) model: $u = C \log(k/k_j)$ where C is a constant.
 - (c) The Underwood (exponential) model: $u = u_f \exp(-k/k_c)$.
 - (d) The Pipes model: $u = u_f(1 - (k/k_j)^n)$ where n is a constant. (The Greenshields model is a special case when $n = 1$.)
 - (e) What features of the Greenberg and Underwood models make them less suitable for dynamic network loading? (Hint: Draw plots of these speed-density diagrams.)
17. [51] Which of the following statements are true with the LWR model and a concave fundamental diagram?
 - (a) Speed uniquely defines the values of flow and density.

- (b) It is possible for higher density to be associated with higher speed.
 (c) No shockwave can move downstream faster than the free-flow speed.
 (d) Flow uniquely defines the values of speed and density.
 (e) With a triangular fundamental diagram, traffic speed is constant for subcritical densities.
 (f) Density uniquely defines the values of speed and flow.
18. [44] The relationship $q = uk$ can transform the fundamental diagram (which relates density and flow) into a relationship between speed and flow. The speed-flow relationship is traditionally plotted with the speed on the vertical axis and flow on the horizontal axis.

- (a) Show that for any concave fundamental diagram and any flow value q less than the capacity, there are exactly two possible speeds u_1 and u_2 producing the flow q , one corresponding to subcritical (uncongested) conditions and the other corresponding to supercritical (congested) conditions.
- (b) Derive and plot the speed-flow relationship corresponding to the Greenshields model of Exercise 16. Express this relationship with two functions $u_1(q)$ and $u_2(q)$ corresponding to uncongested and congested conditions, respectively; these functions should have a domain of $[0, q_{max}]$ and intersect at capacity.
- (c) Derive and plot the fundamental diagram corresponding to the Highway Capacity Manual speed-flow relation for basic freeway segments. The capacity of such a segment is given by $q_{max} = 1800 + 5u_f$. In this equation, speeds are measured in km/hr and flows in veh/hr/lane:

$$u_1(q) = \begin{cases} u_f & \text{if } q \leq 3100 - 15u_0 \\ u_f - \frac{23u_0 - 1800}{28} \left(\frac{q + 15u_f - 3100}{20u_f - 1300} \right)^{2.6} & \text{if } 3100 - 15u_0 \leq q \leq q_{max} \end{cases} \quad (9.112)$$

$$u_2(q) = 28q \quad (9.113)$$

19. [22] For each of these fundamental diagrams, derive the speed-density function (that is, the travel speed for any given density value), and provide a sketch.
- (a) $Q(k) = Ck(k_j - k)$, where C is a constant and k_j is the jam density.
 (b) $Q(k) = \min\{u_f k, w(k_j - k)\}$
 (c) $Q(k) = \min\{u_f k, q_{max}, w(k_j - k)\}$
20. [44] Consider a long, uninterrupted freeway with a capacity of 4400 vehicles per hour, a jam density of 200 vehicles per mile, and a free-flow speed of 75 miles per hour. Initially, freeway conditions are uniform and steady

with a subcritical flow of 2000 vehicles per hour. An accident reduces the roadway capacity to 1000 veh/hr for thirty minutes. Draw a shockwave diagram to show the effects of this accident, reporting the space-mean speed, volume, and density in each region of your diagram, and the speed and direction of each shockwave. Assume that the fundamental diagram takes the shape of the Greenshields model (Exercise 16), and that a stopped queue discharges at capacity.

21. [46] Consider a roadway with a linear-speed density relationship (cf. Exercise 16) whose capacity is 2000 veh/hr and free-flow speed is 40 mi/hr. Initially, the flow is 1000 veh/hr and uncongested. A traffic signal is red for 45 seconds, causing several shockwaves. When the light turns green, the queue discharges at capacity.
 - (a) Sketch a time-space diagram, indicating all of the shockwaves which are formed.
 - (b) Calculate the speed and direction of each shockwave from your diagram.
 - (c) What is the minimum green time needed to ensure that no vehicle has to stop more than once before passing the intersection? (Neglect any yellow time, reaction time, etc. Assume that when the signal is green, people move immediately, and that when it is red, people stop immediately.)
22. [56] Consider a single-lane roadway with a triangular fundamental diagram, a free-flow speed of 60 mi/hr, a backward wave speed of 30 mi/hr, and a jam density of 200 veh/mi. Initially, traffic flow is uncongested, and the volume is half of capacity. A slow-moving truck enters the roadway at time $t = 0$, and travels at 20 mi/hr. This vehicle turns off of the roadway one mile later.
 - (a) What is the capacity of the roadway?
 - (b) At time $t = 2$ minutes, you are a quarter of a mile behind the truck. Use the Newell-Daganzo method to determine how many vehicles are between you and the truck.
 - (c) In total, how many shockwaves are generated by the slow-moving truck? Sketch them on a trajectory diagram.
23. [11] Show that a shockwave connecting two uncongested (subcritical) traffic states always moves downstream, while a shockwave connecting two congested (supercritical) traffic states always moves upstream. This is related to the observation in the chapter that “uncongested states propagate downstream, and congested states propagate upstream.”
24. [36] This exercise asks you to fill in some details of the example in Section 9.4 where the fundamental diagram was $Q(k) = \frac{1}{240}k(240 - k)$ and

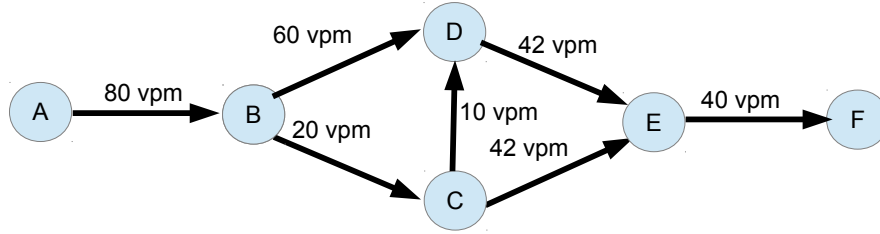


Figure 9.33: Network for Exercise 25

the cumulative count map was $N(x, t) = 60t - 120x + \frac{60x^2}{t+1}$. Times are measured in minutes, and distances in miles.

- (a) Calculate the capacity, jam density, and free-flow speed associated with this fundamental diagram.
 - (b) Verify that the conservation relationship (9.25) is satisfied by the flow and density maps $q(x, t)$ and $k(x, t)$.
 - (c) Verify that the density and flow maps are consistent with the given fundamental diagram.
 - (d) Calculate the speed $u(x, t)$ at each point and time. Are vehicles accelerating, decelerating, or maintaining a constant speed?
25. [45] Consider the network in Figure 9.33, where each link has a free-flow time of 5 minutes and a capacity shown on the figure, and vehicles split equally at each diverge (that is, $p_{ABC} = p_{ABD} = p_{BCD} = p_{BCE} = 1/2$ at all times). Vehicles enter the network at a rate of 80 veh/min for 20 minutes, and then the inflow rate drops to zero. Perform dynamic network loading, using point queues for the link models. For each link in the network, plot the cumulative counts N^\uparrow and N^\downarrow over time, as well as the sending flow and receiving flow over time. At what time does the last vehicle leave the network?
 26. [25] Write the formula for the fundamental diagram $Q(k)$ in the cell transmission model example depicted in Table 9.4.
 27. [13] A link is divided into four cells; on this link the capacity is 10 vehicles per time step, each cell can hold at most 40 vehicles, and the ratio of backward wave speed to free-flow speed is 0.5. Currently, the number of vehicles in each cell is as in Table 9.8 (Cell 1 is at the upstream end of the link, Cell 4 at the downstream end.) Calculate the number of vehicles that will move between each pair of cells in the current time interval (that is, the y_{12} , y_{23} , and y_{34} values.), and the number of vehicles in each cell at the start of the next time interval. Assume no vehicles enter or exit the link. .

Table 9.8: Current cell occupancies for Exercise 27.

Cell 1	Cell 2	Cell 3	Cell 4
8	10	30	5

Table 9.9: Upstream and downstream counts for Exercise 28.

t	$N^\uparrow(t)$	$N^\downarrow(t)$
0	0	0
1	5	0
2	10	0
3	15	2
4	16	4
5	17	6
6	20	10
7	20	15

28. [23] Table 9.9 shows cumulative inflows and outflows to a link with a capacity of 10 vehicles per time step, a free-flow time of 2 time steps, and a backward wave time of 4 time steps. At jam density, there are 20 vehicles on the link. Use the link transmission model to calculate the sending flow $S(7)$ and the receiving flow $R(7)$.
29. [44] (*Exploring shock spreading.*) A link is seven cells long; at most 15 vehicles can fit into each cell, the capacity is 5 vehicles per timestep, and $w/u_f = 1/2$. Each time step, 2 vehicles wish to enter the link, and will do so if the receiving flow can accommodate. There is a traffic signal at the downstream end of the link. During time steps 0–9, and from time step 50 onward, the light is green and all of the link’s sending flow can leave. For the other time steps, the light is red, and the sending flow of the link is zero.
- (a) Use the cell transmission model to propagate flow for 80 time steps, portraying the resulting cell occupancies in a time-space diagram (time on the horizontal axis, space on the vertical axis). At what time interval does the receiving flow first begin to drop; at what point does it reach its minimum value; and what is that minimum value? Is there any point at which the entire link is at jam density?
- (b) Repeat, but with $w/u_f = 1$.
- (c) Repeat, but instead use the link transmission model (with the same time step) to determine how much flow can enter or leave the link.
30. [68] Consider the network in Figure 9.34. The figure shows each link’s length, capacity, jam density, free-flow speed, and backward wave speed.

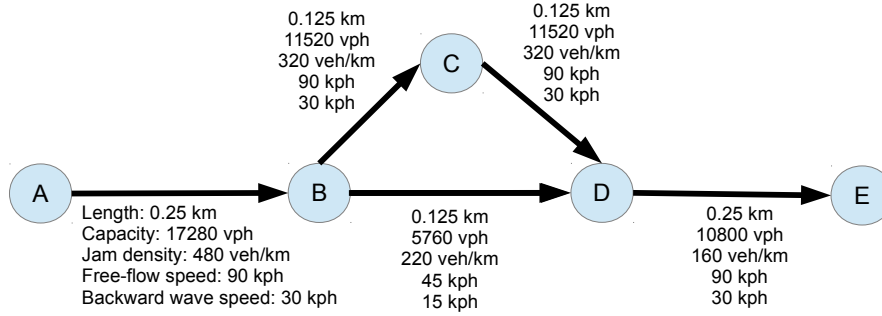


Figure 9.34: Network for Exercise 30

The inflow rate at node A is 4320 veh/hr for $0 \leq t < 30$ (t measured in seconds), 8640 veh/hr for $25 \leq t < 120$, and 0 veh/hr thereafter. The splitting proportion towards node C is $\frac{2}{3}$ for $0 \leq t < 50$, $\frac{1}{6}$ for $50 \leq t < 90$, and $\frac{1}{2}$ for $t \geq 90$.

- Use a point queue model to propagate the vehicle flow with the time step $\Delta t = 5$ s. Plot the turning movement flows y_{ABC} , y_{ABD} , y_{BDE} , and y_{BCE} from $t = 0$ until the last vehicle has left the network. (q_{12} is the rate at which flow leaves the downstream end of link 1 to enter the upstream end of link 2).
 - Use the cell transmission model to propagate the vehicle flow with the time step $\Delta t = 5$ s. Plot the same flow rates as in the previous part.
 - Use the link transmission model to propagate the vehicle flow with the time step $\Delta t = 10$ s. Plot the same flow rates as in the previous part.
 - Comment on any differences you see in these plots for the three flow models.
- [21] Assuming that a cell initially has between 0 and \bar{n} vehicles, show the cell transmission model formula (9.55) ensures that it will have between 0 and \bar{n} vehicles at all future time steps, regardless of upstream or downstream conditions.
 - [21] On a link, we must have $N^\uparrow(t) \geq N^\downarrow(t)$ at all time steps. Assuming this is true for all time steps before t , show that the link transmission model formulas (9.63) and (9.65) ensure this condition holds at t as well.
 - [42] Generalize the cell transmission model formula (9.55) to handle an arbitrary piecewise-linear fundamental diagram (not necessarily triangular or trapezoidal).

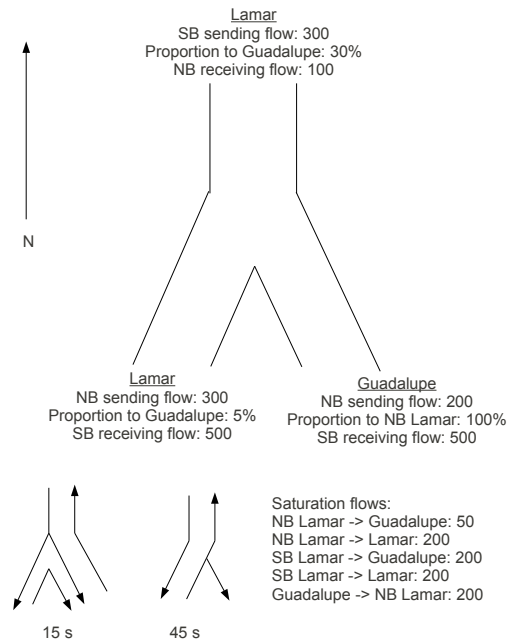


Figure 9.35: Intersection for Exercise 35

34. [65] Generalize the link transmission model formulas (9.63) and (9.65) to handle an arbitrary piecewise-linear fundamental diagram.
35. [35] Figure 9.35 represents the intersection of Lamar and Guadalupe, showing sending and receiving flows, saturation flows, turning movement proportions, and the signal timing plan (assume no lost time due to clearance intervals or startup delay). Note that the receiving flow on northbound Lamar is quite low, because of congestion spilling back from a nearby signal just downstream. No U-turns are allowed, and drivers may not turn left from Guadalupe onto southbound Lamar.
 - (a) Find the transition flows y_{ijk} for all five turning movements at the current time step, using the “smoothed signal” node model.
 - (b) The southbound receiving flow on Guadalupe is now reduced to 50 due to congestion further downstream. Find the updated transition flow rates for all turning movements.
36. [51] Extend the “basic signal” node model to account for turns on red, where a vehicle facing a red indication may make a turn in the direction nearest to them (usually right-on-red in countries that drive on the right,

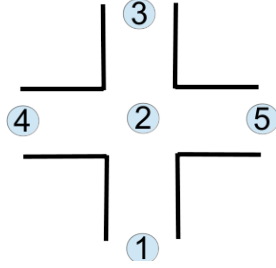


Figure 9.36: Intersection for Exercise 39.

left-on-red in countries that drive on the left). Vehicles turning on red must yield to traffic which has a green indication.

37. [53] Extend the “basic signal” node model to the case where there are turn lanes, and not all turn lanes from an approach have the same green time.
38. [61] Extend the “basic signal” node model to account for permitted turns on green (a turning movement which has a green indication, but must yield to oncoming traffic).
39. [36] Consider the four-legged intersection shown in Figure 9.36. The set of turning movements is $\Xi = \{[1, 2, 3], [1, 2, 4], [3, 2, 1], [3, 2, 4], [5, 2, 4]\}$. For the current time step, the sending and receiving flow values are $S_{12} = 6$, $S_{32} = 6$, $S_{52} = 1$, $R_{12} = 14$, $R_{23} = 14$, and $R_{24} = 14$. Half of the drivers approaching from link (1,2) want to turn left, and half want to go straight. Half of the drivers approaching from link (3,2) want to turn right, and half want to go straight. Finally, the intersection geometry and signal timing are such that the capacities of the incoming links are $q_{12}^{max} = 200$, $q_{32}^{max} = 2000$, and $q_{52}^{max} = 20$.
- Apply the “equal priorities” algorithm, and report the transition flows (y values) for each turning movement.
40. [36] Repeat Exercise 39, but with these values of the sending and receiving flows: $S_{12} = 18$, $S_{32} = 18$, $S_{52} = 3$, $R_{12} = 42$, $R_{23} = 42$, and $R_{24} = 12$.
41. [68] Modify the “partial stop control” node model to allow cases of absolute priority (if movement $[h, i, j]$ has absolute priority over $[h', i, j']$ at conflict point c , then $\alpha_{h'ij'}^c / \alpha_{hij}^c$ would be zero.) You will need to decide how priority will be granted for any combination of sending flows wishing to use a conflict point (including flows of equal absolute priority, and when not all sending flows are present), and your formulas can never divide by zero.
42. [89] Show that all of the node models in Section 9.6 satisfy all of the desiderata from Section 9.2.

Chapter 10

Time-Dependent Shortest Paths

This chapter discusses how travelers make choices when traveling in networks whose state varies over time. Two specific choices are discussed: how drivers choose a route when link costs are time-varying (Sections 10.1 and 10.2), and how drivers choose a departure time (Section 10.3). This chapter is the complement of the previous one. In network loading, we assumed that the travelers' choices were known, and we then determined the (time-varying) flow rates and congestion pattern throughout the network. In this chapter, we take this congestion pattern as known, and predict the choices that travelers would make given this congestion pattern. In particular, by taking the congestion level as fixed, we can focus the question on an *individual* traveler and do not need to worry about changes in congestion based on these choices just yet.

10.1 Time-Dependent Shortest Path Concepts

The *time-dependent shortest path* problem involves finding a path through a network of minimum cost, when the cost of links varies with time. As with the static shortest path problem, the “cost” of a link can include travel time, monetary costs, a combination of these, or any other disutility which can be added across links; and by a “shortest” path we mean one with least cost. If a traveler enters link (i, j) at time t , they will experience a cost of $c_{ij}(t)$. Unlike the static shortest path problem, however, we must always keep track of the travel time on a link even if the cost refers to a separate quantity: because the network state is dynamic, we must always know the time at which a traveler enters a link. The travel time experienced by a traveler entering link (i, j) at time t is denoted by $\tau_{ij}(t)$. This problem can be formulated either in discrete time (where t and $\tau_{ij}(t)$ are limited to being integer multiples of the timestep Δt) or in continuous time (where t can take any real value within a stated range).

There are several different variations of the time-dependent shortest path problem, all stemming from dynamic link costs and travel times. One can consider time-dependent shortest paths where *waiting* at intermediate nodes is allowed, or one can forbid it. In public transit networks, waiting at intermediate nodes is logical, but in road networks, one would not expect drivers to voluntarily stop and wait in the middle of a route. One can also restrict attention to problems where the link travel times satisfy the *first-in, first-out* (FIFO) property, where it is impossible for one to leave a link earlier by entering later, that is, for any link (i, j) and any distinct $t_1 < t_2$, we have

$$t_1 + \tau_{ij}(t_1) \leq t_2 + \tau_{ij}(t_2). \quad (10.1)$$

If this is true, more efficient algorithms can be developed. As an example, if the cost of a link is its travel time ($c_{ij}(t) = \tau_{ij}(t)$), there is no benefit to waiting in a FIFO network. In discrete time, equation (10.1) is equivalent to requiring

$$\tau_{ij}(t + \Delta t) - \tau_{ij}(t) \leq \Delta t, \quad (10.2)$$

that is, a link's travel time cannot decrease by more than Δt in one time step. In continuous time, if $\tau_{ij}(t)$ is a piecewise differentiable function, we must have

$$-\frac{d}{dt}\tau_{ij}(t) \leq -1, \quad (10.3)$$

everywhere that $\tau_{ij}(t)$ has a derivative, which expresses the same idea.

One can also distinguish time-dependent shortest path problems by whether the departure time is fixed, whether the arrival time is fixed, or neither. In the first case, the driver has already decided when they will leave, and want to find the route to the destination with minimum cost when leaving at that time, regardless of the arrival time at the destination. In the second case, the arrival time at the destination is known (perhaps the start of work), and the traveler wants to find the route with minimum cost arriving at the destination at that time (regardless of departure time). In the third case, both the departure and arrival times are flexible (as with many shopping trips), and the traveler wants to find a minimum cost route without regard to when they leave or arrive. In this way, we can model the departure time choice decision simultaneously with the route choice decision. Section 10.3 develops this approach further, showing how we can incorporate penalty “costs” associated with different departure and arrival times. Strictly speaking, one can imagine a fourth variant where *both* departure and arrival time are fixed, but this problem is not always well-posed; there may be no path from the origin to the destination with those exact departure and arrival times. In such cases, we can allow free departure and/or arrival times, but severely penalize departure/arrival times that differ greatly from the desired times.

In comparison with the static shortest path problem, the fixed departure time variant is like the one origin-to-all destinations shortest path problem, and the fixed arrival time variant is like the all origins-to-one destination shortest path problem.

To be specific, in this chapter, we will first focus on time-dependent shortest paths with fixed departure times and free arrival times, in Section 10.2, but all of these approaches can be adapted to solve the fixed arrival time/free departure time version without much difficulty. The origin r and departure time t_0 will be specified, and we will find the paths to all other nodes along which the sum of the costs c_{ij} is minimal, given the changes in the costs which will occur during travel. We will study two specific variants of the problem. In the first, time is continuous, but the network is assumed to follow the FIFO principle and the link costs must be the travel times. In the other variants, we do not require the FIFO assumption, and the link costs need not be the same as the travel times; but in exchange we will restrict ourselves to discrete time. Section 10.3 will then treat the case of free departure and arrival times, where only the origin r will be specified.

10.1.1 Time-expanded networks

For discrete-time shortest path problems, we can form what is known as the *time-expanded network*. This technique transforms a time-dependent shortest path problem into a static shortest path problem, that can be solved using algorithms for this simpler problem (see Section 2.4). However, the time-expanded network contains many more links and nodes than the original network, and can impose computational burdens.

If the time step is Δt , then the possible arrival times at any node are $0, \Delta t, 2\Delta t, \dots, T\Delta t$ where T is the time horizon under consideration.¹ In the time-expanded network, we replicate each node $T + 1$ times, one for each possible arrival time. So, if the original network has n nodes, the time-expanded network has nT nodes. Each node in the time-expanded network is written in the form $i : t$, denoting the physical node i at the time interval $t\Delta t$. This labeling is shown in Figure 10.1. The original network is called the *physical network* when we need to distinguish it from the corresponding time-expanded network.

Links in the time-expanded network represent both the physical connection between nodes, as well as the time required to traverse the link. In particular, for each link (i, j) in the original network, and for each time interval $t\Delta t$, if the travel time is $\tau_{ij}(t\Delta t) = k\Delta t$, we create a link $(i : t, j : (t + k))$ in the time expanded network, assuming that $t + k \leq T$, and set the cost of this link equal to $c_{ij}(t)$. A little bit of care must be taken when links would arrive at a node later than the time horizon. If the time horizon is large enough, this should not be a significant issue. In this chapter, we will not create a time-expanded link if its head node falls outside of the time horizon. Exercises 4 and 5 introduce two other ways to treat boundary issues associated with the time horizon.

The advantage of the time-expanded network is that it reduces the time-dependent shortest path problem to the static one: solving the one-to-all static

¹In transportation systems, it is often reasonable to assume that after a large enough amount of time, all congestion will dissipate and travel times can be treated as constants equal to free-flow time. All routing after this point can be done with a static shortest path algorithm.

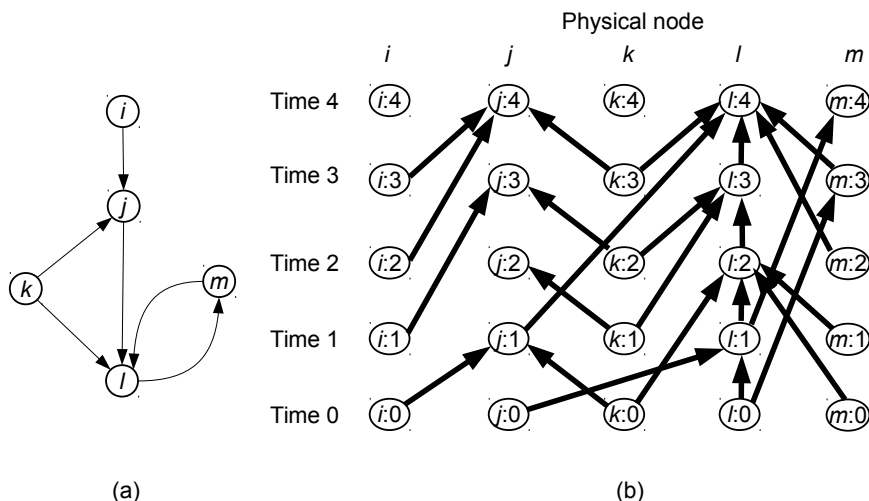


Figure 10.1: (a) Original network (b) Corresponding time-expanded network with five physical nodes and five time intervals.

shortest path problem from node i at time t_0 corresponds exactly to solving the fixed-departure time-dependent shortest path problem. Furthermore, if all link travel times are positive, the time-expanded network is acyclic, with the time labels forming a natural topological order. Shortest paths on acyclic networks can be solved rather quickly, so this is a significant advantage. Even if some link travel times are zero (as may occur with some artificial links or centroid connectors), the time-expanded network remains acyclic unless there is a *cycle* of zero-travel time links in the network; and if that is the case, it is often possible to collapse the zero-travel time cycle into a single node. Time-expanded networks can also unify some of the variants of the time-dependent shortest path problem. If waiting is allowed at a node i , we can represent that with links of the form $(i : t, i : (t + 1))$ connecting the same physical node to itself, a time step later. The FIFO principle in a time-dependent network means that two links connecting the same physical nodes will never cross (although they may terminate at the same node).

A disadvantage is that the number of time intervals T may be quite large. In dynamic traffic assignment, network loading often uses a time step on the order of a few seconds. If this same time step is used for time-dependent shortest paths, a typical planning period of a few hours means that T is in the thousands. Given a physical network of thousands of nodes, the time-expanded network can easily exceed a million nodes. Even though the time-expanded network is acyclic, this is a substantial increase in the underlying network size, increasing both computation time and amount of computer memory required. With clever implementations, it is often possible to avoid explicitly generating the entire

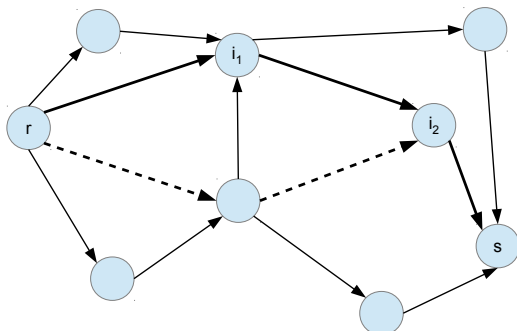


Figure 10.2: Bellman's principle illustrated. If the dashed path were a shorter route from s to i_2 , then it would also form a "shortcut" for the route from s to d .

time-expanded network, and only generate links and nodes as needed.

10.1.2 Bellman's principle in time-dependent networks

The number of paths between any two points in a network can be very large, growing exponentially with the network size. Therefore, any approach based on enumerating all paths and comparing their cost will not scale well to realistic networks. So, efficient algorithms for finding shortest paths, whether time-dependent or not, must be more clever. The key insight is called *Bellman's principle*. In the static shortest path problem, Bellman's principle states that any segment of a shortest path between two nodes must itself be a shortest path between the endpoints of that segment. Otherwise, the shortest path between the endpoints of the segment could be spliced into the original path, reducing its cost (Figure 10.2).

In a time-dependent shortest path problem, the same general idea applies but we must be slightly more careful about how the principle is defined. In a FIFO network where link costs are equal to the travel time, the principle holds identically: since it is always better to arrive at nodes as soon as possible, any "shortcut" between two nodes in a path can be spliced into the full path, thereby reducing its total travel time. When the FIFO principle does not hold, or if link costs are different than travel time, this may not be true. Figure 10.3 shows two counterexamples. In the first, because the FIFO principle is violated and waiting is not allowed, the path segment $[1, 2, 3]$ has a shorter travel time between nodes 1 and 3 than path $[1, 3]$. However, when the link $(3, 4)$ is added to these paths, the path $[1, 2, 3, 4]$ has a higher travel time than path $[1, 3, 4]$ because link $(3, 4)$ is entered at different times, and so its travel time is different. In the second, path $[1, 3]$ has a lower cost than path $[1, 2, 3]$, even though path $[1, 3, 4]$ has higher cost than path $[1, 2, 3, 4]$. In the second case, this is true even though arriving at any node later can only increase cost, and the same result

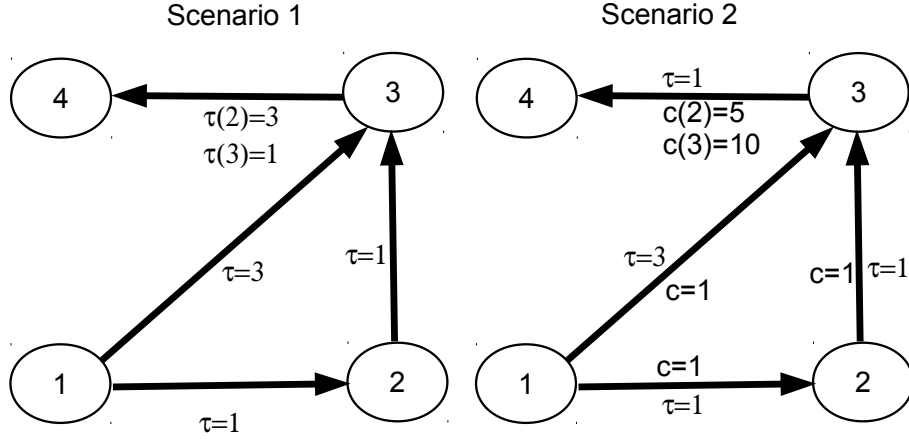


Figure 10.3: Two counterexamples to the naive application of Bellman's principle in time-expanded networks.

holds even if waiting is allowed.

In these cases, the correct approach is to apply Bellman's principle to the time-expanded network, an approach which also works in FIFO networks. Bellman's principle *does* apply to the time-expanded network, corresponding to the following principle in the physical network: **Let π be a shortest path between nodes r and s when departing at time t_0 . If i and h are two nodes in this path, and if the arrival times at these nodes are t_i and t_h , respectively, then the segment of π between i and h must be a shortest path between these nodes when departing at t_i and arriving at t_h .** All of the algorithms in this chapter implicitly use this principle, by allowing us to construct shortest paths a single link at a time: if we already know the shortest path π_i from the origin at t_0 to some other node i at time t_i , then any other shortest path passing through node i at time t_i can be assumed to start with π_i .

10.2 Time-Dependent Shortest Path Algorithms

This section provides two algorithms for the time-dependent shortest path problem with fixed departure times. In the first, time can be modeled as either discrete or continuous, but the network must satisfy the FIFO principle and the cost of a link must be its travel time. In the second and third, time must be discrete, but FIFO need not hold and the cost of a link may take any value. The

second algorithm finds the time-dependent shortest path from a single origin and departure time to all destinations, while the third algorithm finds the time-dependent shortest paths from a single origin and *all* departure times to a single destination. There are many other possible variants of time-dependent shortest path algorithms, some of which are explored in the exercises — in particular, Exercise 6 asks you to develop a time-dependent shortest path algorithm for all origins and departure times simultaneously, which often arises in dynamic traffic assignment software. Nevertheless, the algorithms here should give the general flavor of how they function. Which one of these algorithms is best, or whether another variant is better, depends on the particular dynamic traffic assignment implementation. All of these algorithms use labels with similar (or even identical) names, but the meanings of these labels are slightly different in each.

10.2.1 FIFO networks

For this algorithm to apply, assume that the FIFO principle holds, in either its discrete or continuous form. Also assume that the cost of a link is simply its travel time, so $c_{ij}(t) = \tau_{ij}(t)$ for all links (i, j) and times t . We are also given the origin r and departure time t_0 . Because the FIFO principle holds, waiting at an intermediate node is never beneficial, and so it suffices to find the earliest time we can reach a node and ignore all later times. This allows us to adapt Dijkstra's algorithm (Section 2.4.3) for the static shortest path problem, a label-setting approach. Two labels are defined for each node: L_i gives the earliest possible arrival time to node i found so far, when departing origin r at time t_0 . The backnode label q_i gives the previous node in a path corresponding to this earliest known arrival time. By the dynamic version of Bellman's principle in FIFO networks, this suffices for reconstructing the shortest path from r to i . For nodes i where we have not yet calculated the earliest possible arrival time, we will set L_i to ∞ , and for nodes i where the backnode is meaningless (either because we have not yet found a path there, or because i is the origin), we set q_i to -1 . We also maintain a set of *finalized nodes* F . If node i belongs to F , this means that we are sure that we have found the earliest possible arrival time to i over all possible paths.

The algorithm functions as follows:

1. Initialize by setting $L_i \leftarrow \infty$ for all nodes i , except for the origin, where $L_r \leftarrow t_0$.
2. Initialize the set of finalized nodes $F \leftarrow \emptyset$, and the backnode vector $\mathbf{q} \leftarrow -\mathbf{1}$.
3. Choose an unfinalized node i with the lowest L_i value. (At the first iteration, this will be the origin r .)
4. Finalize node i : $F \leftarrow F \cup \{i\}$.

5. For each link $(i, j) \in \Gamma(i)$ such that $L_i + \tau_{ij}(L_i)$ is within the time horizon, perform the following steps:
 - (a) Update $L_j \leftarrow \min \{L_j, L_i + \tau_{ij}(L_i)\}$.
 - (b) If L_j changed in the previous step, update $q_j \leftarrow i$.
6. If all nodes are finalized ($F = N$), terminate. Otherwise, return to step 3.

As an example of this algorithm, consider the network in Figure 10.4. The time-dependent travel times are shown in this figure. The FIFO assumption is satisfied: for links (1, 3) and (2, 4) the travel times are constant; for links (1, 2) and (3, 4) the travel times are increasing (so arriving earlier always means leaving earlier); and for link (2, 3) the travel time is decreasing, but at a slow enough rate that you cannot leave earlier by arriving later, cf. (10.3). Assume that the initial departure time from node 1 is at $t_0 = 2$, and that the time horizon is large enough that $L_i + \tau_{ij}(L_i)$ is always within the time horizon whenever step 5 is encountered. The steps of the algorithm are explained below, and summarized in Table 10.1. This table shows the state of the algorithm *just before* step 3 is performed, counting the first time through as iteration zero.

Initially, no nodes are finalized, all cost labels are initialized to ∞ (except for the origin, which is assigned 2, the departure time), and all backnode labels are initialized to -1 . The unfinalized node with the least L value is node 1, which is selected as the node i to scan. At the current time of 2, link (1, 2) has a travel time of 6, and link (1, 3) has a travel time of 10. Following these links would result in arrival at nodes 2 and 3 at times 8 and 12, respectively. Each of these is less than their current values of ∞ , so the cost and backnode labels are adjusted accordingly. At the next iteration, node 2 is the unfinalized node with the least L value, so $i = 2$. At time 8, link (2, 3) has a travel time of 1, and link (2, 4) has a travel time of 5. Following these links, one would arrive at nodes 3 and 4 at times 9 and 13, respectively. Both of these values are less than the current L values for these nodes, so their L and q labels are changed. At the next iteration, node 3 is the unfinalized node with the least L value, so $i = 3$. At this time, link (3, 4) would have a travel time of $4\frac{1}{2}$, and choosing it means arriving at node 4 at time $13\frac{1}{2}$. This is greater than the current value ($L_4 = 13$), so no labels are adjusted. Finally, node 4 is chosen as the only unfinalized node. Since it has no outgoing links ($\Gamma(4)$ is empty), there is nothing to do in step 5, and since all nodes are finalized the algorithm terminates.

At this point, we can trace back the shortest paths using the backnode labels: the shortest paths to nodes 2, 3, and 4 are [1, 2], [1, 2, 3], and [1, 2, 4], respectively; and following these paths one arrives at the nodes at times 8, 9, and 13.

10.2.2 Discrete-time networks, one departure time

This algorithm applies in any discrete-time network, regardless of whether or not the FIFO principle holds, and regardless of whether $c_{ij}(t) = \tau_{ij}(t)$ or not. We are given the origin r and departure time t_0 , and work in the time-expanded

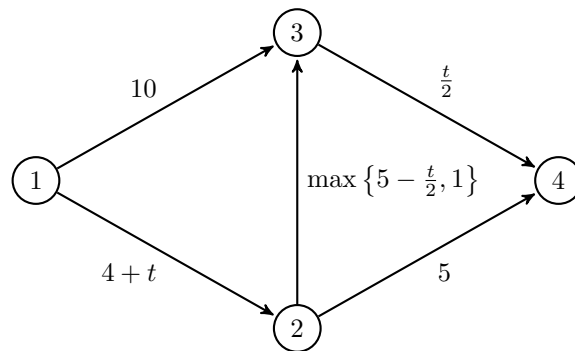


Figure 10.4: Network and travel time functions for demonstrating the FIFO time-dependent shortest path algorithm.

Table 10.1: FIFO time-dependent shortest path algorithm for the network in Figure 10.4, departing node 1 at $t_0 = 2$.

Iteration	F	i	L_1	L_2	L_3	L_4	q_1	q_2	q_3	q_4
0	\emptyset	—	2	∞	∞	∞	-1	-1	-1	-1
1	{1}	1	2	8	12	∞	-1	1	1	-1
2	{1, 2}	2	2	8	9	13	-1	1	2	2
3	{1, 2, 3}	3	2	8	9	13	-1	1	2	2
4	{1, 2, 3, 4}	4	2	8	9	13	-1	1	2	2

network. Each node $i : t$ in the time-expanded network is associated with two labels. The label L_i^t denotes the cost of the shortest path from r to i known so far, when departing at time t_0 and arriving at time t . The backnode label q_i^t provides the previous node on a path corresponding to cost L_i^t . As before, $L_i^t = \infty$ signifies that no path to i at time t is yet known, and $q_i^t = -1$ signifies that the backnode is meaningless. Since the FIFO principle may not hold, it is not always advantageous to arrive at a node as early as possible. To reflect this, we work in the time-expanded network, where we can naturally identify the best time to arrive at nodes. We presume that all links have strictly positive travel time, so that the time-expanded graph is acyclic and each link connects a node of earlier time to a node of later time. If there are zero-travel time links in the physical network, but no cycles of zero-travel time links, then step 4 is assumed to proceed in topological order by physical node. In this case, we can adapt an algorithm for finding static shortest paths (e.g., from Section 2.4), in acyclic networks by applying it to the time-expanded network in the following way:

1. Initialize $\mathbf{q} \leftarrow -\mathbf{1}$ and $\mathbf{L} \leftarrow \infty$.
2. Set $L_r^{t_0} \leftarrow 0$.
3. Initialize the current time to the departure time, $t \leftarrow t_0$
4. For each time-expanded node $i : t$ for which $L_i^t < \infty$, and for each time-expanded link $(i : t, j : t')$, perform the following steps:
 - (a) Set $L_j^{t'} = \min \{L_j^{t'}, L_i^t + c_{ij}(t)\}$.
 - (b) If $L_j^{t'}$ changed in the previous step, update $q_j^{t'} \leftarrow i : t$.
5. If $t = T$, then terminate. Otherwise, move to the next time step ($t \leftarrow t+1$) and return to step 4.

At the conclusion of this algorithm, we have the least-cost paths for each possible arrival time at each destination. To find the least-cost path to a particular destination s (at any arrival time), you can consult the L_s^t labels at all times t , and trace back the path for the arrival time t with the least L_s^t value.

This algorithm is demonstrated on the network in the *right* panel of Figure 10.3, and its progress is summarized in Table 10.2. The table shows the state of the algorithm *just before* step 4 is executed. For brevity, this table only reports L_i^t and q_i^t labels for nodes and arrival times which are reachable in the network (that is, i and t values for which $L_i^t < \infty$ at the end of the algorithm). All other cost and backnode labels are at ∞ and -1 throughout the entire duration of the algorithm.

Initially, all cost labels are set to ∞ and all backnode labels to -1 , except for the origin and departure time: $L_1^0 = 0$. The algorithm then sets $t = 0$, and

Table 10.2: Discrete time-dependent shortest path algorithm for the network in the right panel of Figure 10.3, departing node 1 at $t_0 = 0$.

t	L_1^0	L_2^1	L_3^2	L_3^3	L_4^3	L_4^4	q_1^0	q_2^1	q_3^2	q_3^3	q_4^3	q_4^4
0	0	∞	∞	∞	∞	∞	-1	-1	-1	-1	-1	-1
1	0	1	∞	1	∞	∞	-1	1	-1	1	-1	-1
2	0	1	2	1	∞	∞	-1	1	2	1	-1	-1
3	0	1	2	1	7	∞	-1	1	2	1	3	-1
4	0	1	2	1	7	11	-1	1	2	1	3	3

scans over all physical nodes which are reachable at this time² Only node 1 can be reached at this time, and the possible links are $(1 : 0, 2 : 1)$ and $(1 : 0, 3 : 3)$. Following either link incurs a cost of 1, which is lower than the (infinite) values of L_2^1 and L_3^3 , so the cost and backnode labels are updated.

The algorithm then sets $t = 1$. Only node 2 is reachable at this time, and the only link is $(2 : 1, 3 : 2)$. Following this link incurs a cost of 1; in addition to the cost of 1 already involved in reaching node 2, this gives a cost of 2 for arriving at node 3 at time 2. The cost and time labels for $3 : 2$ are updated. Since the costs and times are different, notice that arriving at node 3 at a later time (3 vs. 2) incurs a lower cost (1 vs. 2). This is why we need to track labels for different arrival times, unlike the algorithm in the previous section.

The next time step is $t = 2$. Only node 3 is reachable at this time (from the path $[1, 2, 3]$), and the only link is $(3 : 2, 4 : 3)$. Following this link incurs a cost of 5, resulting in a total cost of 7, and the labels for $4 : 3$ are updated. Time $t = 3$ is next, and again only node 3 is reachable at this time — but from the path $[1, 3]$. The only link is $(3 : 3, 4 : 4)$, and following this link incurs a cost of 10, for a total cost of 11. Labels are updated for $4 : 4$. There are no further label changes in the algorithm (all nodes have already been scanned at all reachable times), and it terminates as soon as t is increased to the time horizon.

After termination, the L_3^t labels show that we can reach node 3 either with a cost of 1 (arriving at time 3) or a cost of 2 (arriving at time 2). The least-cost path thus arrives at time 3, and it is $[1, 3]$. The L_4^t labels show that we can reach node 4 either with a cost of 7 (arriving at time 3) or a cost of 11 (arriving at time 4). The least-cost path arrives at time 3, and it is $[1, 2, 3, 4]$. (The least-cost path to node 2 is $[1, 2]$, since there is only one possible arrival time there). Notice that the naïve form of Bellman's principle is not satisfied: the least-cost path to node 2 is *not* a subset of the least-cost path to node 3. This was why we needed to keep track of different possible arrival times to nodes — the least-cost path to node 3 *arriving at time 2* is indeed a subset of the least-cost path to node 4 *arriving at time 3*.

²Reachability is expressed by the condition $L_i^t < \infty$; a node which cannot be reached at this time will still have its label set to the initial value. Any node which *can* be reached at this time will have a finite L_i^t value, since step 4a will always reduce an infinite value.

10.3 Departure Time Choice

Travelers often have some flexibility when choosing their departure or arrival times, and may choose to leave earlier or later in order to minimize cost. For example, commuters with flexible work hours may want to time their commutes to avoid congestion, or when dynamic congestion charges are lower. Both flexible departures and arrivals can be incorporated into the shortest path algorithms described in the previous section — in fact, you may have already noticed that the algorithms in Sections 10.2.1 and 10.2.2 allow flexible arrival times. This section explores these choices more systematically, showing how the departure time can also be made flexible, and ways to represent different departure time behaviors.

10.3.1 Artificial origins and destinations

In the time-expanded network, departure and arrival time choice can be modeled by adding artificial “super-origin” and “super-destination” nodes which reflect the start and end of a trip without regard to the time. A super-origin is connected to time-expanded nodes which correspond to allowable departure times, and a super-destination is connected to time-expanded nodes which correspond to allowable arrival times. Initially, we will assign these links a cost of zero, which means the traveler is indifferent among any of these departure or arrival times. Section 10.3.2 describes how arrival and departure time preferences can be modeled.

Figure 10.5 shows how super-origins and super-destinations can be added to the physical network of Figure 10.1(a), assuming that nodes i and k are origins, and nodes j and l are destinations. The figure is drawn as if departures were allowed only for times 0, 1, and 2, but arrivals are allowed at any time. The network is similar to the time-expanded network in Figure 10.1(b), but now includes four artificial nodes, and artificial links corresponding to allowable departure and arrival times.

Once these artificial links and nodes are added to the network, the algorithm from Section 10.2.2 can be applied directly, with only very minor changes. In what follows, r is the origin and the algorithm finds least-cost paths to all destinations, for any allowable departure and arrival time:

1. Initialize $\mathbf{q} \leftarrow -\mathbf{1}$ and $\mathbf{L} \leftarrow \infty$ for all nodes in the time-expanded network.
2. Set $L_r \leftarrow 0$ for the super-origin r , and for each artificial link $(r, r : t)$ set $L_r^t \leftarrow c_{r,depart}(t)$ and $q_r^t \leftarrow r$, where $c_{r,depart}(t)$ is the cost on the artificial link for departing node r at time t .
3. Initialize the current time t to the earliest possible departure time (the lowest t index for which an artificial link $(r, r : t)$ exists).
4. For each time-expanded node $i : t$ for which $L_i^t < \infty$, and for each time-expanded link $(i : t, j : t')$, perform the following steps:

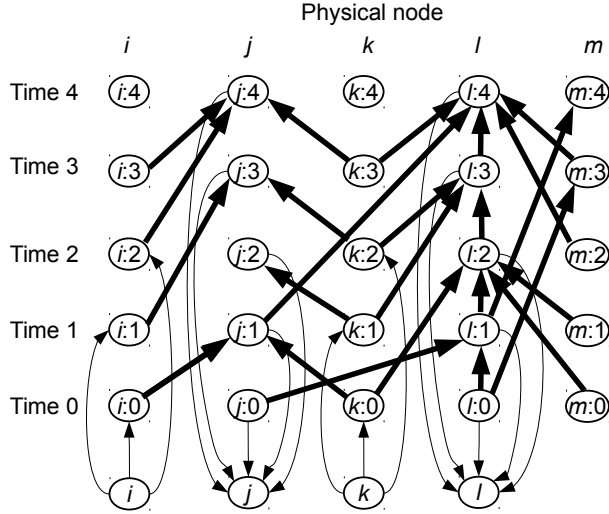


Figure 10.5: Time-expanded network with artificial origins and destinations. Artificial links shown with light arrows for clarity.

- (a) Set $L_j^{t'} = \min \{L_j^{t'}, L_i^t + c_{ij}(t)\}$.
- (b) If $L_j^{t'}$ changed in the previous step, update $q_j^{t'} \leftarrow i : t$.

Likewise, for each artificial link $(i : t, i)$ reaching a super-destination node i with cost $c_{i,arrive}(t)$, perform the following steps:

- (a) Set $L_i = \min \{L_i, L_i^t + c_{i,arrive}(t)\}$.
- (b) If L_i changed in the previous step, update $q_i \leftarrow i : t$.

5. If $t = T$, then terminate. Otherwise, move to the next time step ($t \leftarrow t+1$) and return to step 4.

The algorithm initializes labels differently in step 2; step 3 starts at the earliest possible departure time rather than the fixed time t_0 ; and step 4 is expanded to update labels both at adjacent time-expanded nodes and super-destinations. All other steps work in the same way.

To demonstrate this algorithm, consider the network in Figure 10.6, where the time horizon is $T = 20$ and the destination is node 4, and where the cost of a link is equal to its travel time. Table 10.3 shows the cost and forwardnode labels at the conclusion of the algorithm. Each iteration of the algorithm generates one row of this table, starting with $T = 1$ and working up to $T = 20$. Whenever $L_i^t = \infty$ is seen in Table 10.3, there is no way to reach the destination node within the time horizon, if leaving node i at time t .

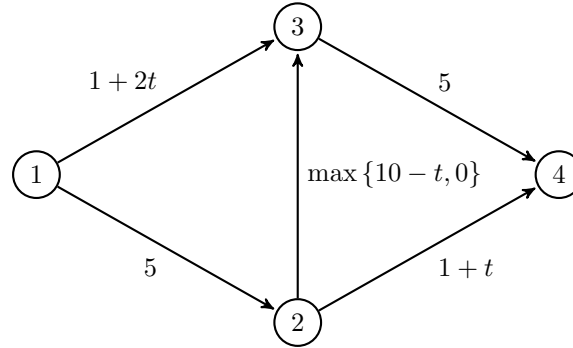


Figure 10.6: Network and travel time functions for demonstrating the all departure times time-dependent shortest path algorithm.

Table 10.3 also shows the labels for the super-origin and super-destination, below the labels for the time-expanded nodes. The backnode label for the super-destination tells us that the least-cost path arrives at node 4 at time 6; the q_4^6 label tells us the least-cost path there comes through node 3 at time 1; q_3^1 tells us the least-cost path there comes through node 1 at time 0, and q_1^0 brings us to the super-origin. Therefore, we should depart the origin at time 0, and follow the path $[1, 3, 4]$ to arrive at the destination at time 6, with a total cost of $c_4 = 6$.

10.3.2 Arrival and departure time preferences

While there is often flexibility in departure or arrival time, usually travelers are not completely indifferent about when they depart or arrive. For instance, there may be a well-defined arrival deadline (start of work, or check-in time before a flight), and a strong desire to arrive before this deadline. Departing extremely early would ensure arriving before the deadline, but carries opportunity costs (by departing later, the traveler would have more time to do other things). Both situations can be modeled by attaching a cost to when a traveler departs the origin, and to when they arrive at the destination.

A common way to model arrival costs is with the *schedule delay* concept. In this model, travelers have a preferred arrival time t^* at the destination, and arriving either earlier or later than t^* is undesirable. The most general formulation involves a function $f(t)$ denoting the cost (or disutility) of arriving at the destination at time t . This function is typically convex and has a minimum at t^* . One example of such a function is

$$f(t) = \alpha[t^* - t]^+ + \beta[t - t^*]^+, \quad (10.4)$$

where $[\cdot]^+ = \max\{\cdot, 0\}$ expresses the positive part of the quantity in brackets. In (10.4), the first bracketed term then represents the amount by which the

Table 10.3: Labels at conclusion of time-dependent shortest path algorithm with departure time choice, to node 4 in Figure 10.6.

t	L_1^t	L_2^t	L_3^t	L_4^t	q_1^t	q_2^t	q_3^t	q_4^t
20	0	5	5	10	1	1 : 15	2 : 20	3 : 15
19	0	5	5	10	1	1 : 14	2 : 19	3 : 14
18	0	5	5	10	1	1 : 13	2 : 18	3 : 13
17	0	5	5	10	1	1 : 12	2 : 17	3 : 12
16	0	5	5	10	1	1 : 11	2 : 16	3 : 11
15	0	5	5	10	1	1 : 10	2 : 15	3 : 10
14	0	5	5	∞	1	1 : 9	2 : 14	-1
13	0	5	5	12	1	1 : 8	2 : 13	2 : 6
12	0	5	5	10	1	1 : 7	2 : 12	3 : 7
11	0	5	5	11	1	1 : 6	2 : 11	2 : 5
10	0	5	5	∞	1	1 : 5	2 : 10	-1
9	0	5	∞	8	1	1 : 4	-1	3 : 4
8	0	5	∞	∞	1	1 : 3	-1	-1
7	0	5	5	∞	1	1 : 2	1 : 2	-1
6	0	5	∞	6	1	1 : 1	-1	3 : 1
5	0	5	∞	∞	1	1 : 0	-1	-1
4	0	∞	3	∞	1	-1	1 : 1	-1
3	0	∞	∞	∞	1	-1	-1	-1
2	0	∞	∞	∞	1	-1	-1	-1
1	0	∞	1	∞	1	-1	1 : 0	-1
0	0	∞	∞	∞	1	-1	-1	-1

(L, q) labels at super-origin 1: $(0, -1)$

(L, q) labels at super-destination 4: $(6, 4 : 6)$

traveler arrived early, compared to the preferred time, and the second bracketed term represents the amount by which the traveler arrived late. The coefficients α and β then weight these terms and convert them to cost units; generally $\alpha < \beta$ to reflect the fact that arriving early by a certain amount of time, while undesirable, is usually not as bad as arriving late by that same amount of time.

Another possible function is nonlinear, taking a form such as

$$f(t) = \alpha([t^* - t]^+)^2 + \beta([t - t^*]^+)^2. \quad (10.5)$$

In this function, the penalty associated with early or late arrival grows faster and faster the farther the arrival time from the target. This may occur if, for instance, being ten minutes late is more than ten times as bad as being one minute late. Special cases of these functions arise when $\alpha = \beta$ (the function becomes symmetric), or when $\alpha = 0$ (there is no penalty for early arrival, but only for late arrival). This function is also differentiable everywhere, in contrast to equation (10.4) which is not differentiable at t^* . For certain algorithms this may be advantageous.

To each of these schedule delay functions $f(t)$, one can add the cost of the path arriving at time t (the sum of the link costs c_{ij} along the way) to yield the total cost of travel. We assume that travelers will choose both the departure time and the path to minimize this sum. The algorithm from Section 10.3.1 can be used to find both this ideal departure time and the path. The only change is that artificial links connecting time-expanded destination nodes ($s : t$) to super-destinations s now have a cost of $f(t)$, rather than zero. At termination, the departure time t_0^* minimizing $L_r^{t_0^*}$ corresponds to the least total cost, and the forwardnode labels trace out the path.

To demonstrate this algorithm, again consider the network in Figure 10.6, but with the arrival time penalty function

$$f(t) = [10 - t]^+ + 2[t - 10]^+, \quad (10.6)$$

which suggests that the traveler wishes to arrive at time 10, and that late arrival is twice as costly as early arrival.

As before, the time horizon is $T = 20$ and the destination is node 4, and the cost of each link is equal to its travel time. Table 10.4 shows the cost and backnode labels at the conclusion of the algorithm, which runs in exactly the same way as before except that the artificial destination links ($4 : t, 4$) have a cost equal to $f(t)$. At the conclusion of the algorithm, we can identify the total cost on the shortest path from node 1 to node 4, now including the penalty for arriving early or late at the destination.

Table 10.4 also shows the labels for the super-origin and super-destination, below the labels for the time-expanded nodes. The backnode label for the super-destination tells us that the least-cost path arrives at node 4 at time 9; the q_4^9 label tells us the least-cost path there comes through node 3 at time 4; q_3^4 tells us the least-cost path there comes through node 1 at time 1, and q_1^1 brings us to the super-origin. Therefore, we should depart the origin at time 1, and follow the path $[1, 3, 4]$ to arrive at the destination at time 9, with a total

Table 10.4: Labels at conclusion of arrival-penalty time-dependent shortest path, to node 4 in Figure 10.6.

t	L_1^t	L_2^t	L_3^t	L_4^t	q_1^t	q_2^t	q_3^t	q_4^t
20	0	5	5	10	1	1 : 15	2 : 20	3 : 15
19	0	5	5	10	1	1 : 14	2 : 19	3 : 14
18	0	5	5	10	1	1 : 13	2 : 18	3 : 13
17	0	5	5	10	1	1 : 12	2 : 17	3 : 12
16	0	5	5	10	1	1 : 11	2 : 16	3 : 11
15	0	5	5	10	1	1 : 10	2 : 15	3 : 10
14	0	5	5	∞	1	1 : 9	2 : 14	-1
13	0	5	5	12	1	1 : 8	2 : 13	2 : 6
12	0	5	5	10	1	1 : 7	2 : 12	3 : 7
11	0	5	5	11	1	1 : 6	2 : 11	2 : 5
10	0	5	5	∞	1	1 : 5	2 : 10	-1
9	0	5	∞	8	1	1 : 4	-1	3 : 4
8	0	5	∞	∞	1	1 : 3	-1	-1
7	0	5	5	∞	1	1 : 2	1 : 2	-1
6	0	5	∞	6	1	1 : 1	-1	3 : 1
5	0	5	∞	∞	1	1 : 0	-1	-1
4	0	∞	3	∞	1	-1	1 : 1	-1
3	0	∞	∞	∞	1	-1	-1	-1
2	0	∞	∞	∞	1	-1	-1	-1
1	0	∞	1	∞	1	-1	1 : 0	-1
0	0	∞	∞	∞	1	-1	-1	-1

(L, q) labels at super-origin 1: $(0, -1)$

(L, q) labels at super-destination 4: $(9, 4 : 9)$

cost of $c_4 = 9$. Of this cost, 8 units are due to travel time (difference between arrival and departure times), and 1 unit is due to the arrival time penalty from equation (10.6) with $t = 9$.

Departing earlier, at $t = 0$, on the same path would reduce the travel time to 6, but increase the early arrival penalty to 4. The total cost of leaving at $t = 0$ is thus higher than departing one time step later. Leaving at $t = 2$ and following the same path increases the travel time cost to 10. Since this means arriving at time 12, there is a late penalty cost of 4 added, resulting in a total travel cost of 14. By comparing all possible paths and departure times, you can verify that it is impossible to have a total cost less than 9.

Considering costs associated with departure time, rather than arrival time, is done in essentially the same way, by assigning a nonzero cost to the artificial links connecting the super-origin r to the time-expanded nodes $r(t)$. It is thus possible to have only departure time penalties, only arrival time penalties, both, or neither, depending on whether the artificial origin links and artificial destination links have nonzero costs.

10.4 Dynamic A^*

Because the time-expanded network transforms the time-dependent shortest problem into the classical, static shortest path problem from Section 2.4, any of the algorithms described there can be applied. The time-expanded network is acyclic, so it is fastest to use the algorithm from Section 2.4.1 — and indeed this is all that the algorithm in Section 10.2.2 is, using the time labels as a topological order.

Section 2.4.4 also presented the A^* algorithm, which provides a single path from one origin to one destination, rather than all shortest paths from one origin to all destinations, or all origins to one destination. By focusing on a single origin and destination, A^* can often find a shortest path much faster than a one origin-to-all destinations algorithm. The tradeoff is that the algorithm has to be repeated many times, once for every OD pair, rather than once for every origin or destination. In static assignment, one-to-all or all-to-one algorithms are preferred because, because in many cases, running A^* for each OD pair takes more time than running a one-to-all algorithm for each origin.

In dynamic traffic assignment with fixed departure times, however, the number of “origins” in the time-expanded network is multiplied by the number of departure times. If one were to write a full time-dependent OD matrix, the number of entries in this matrix is very large: in a network with 1000 centroids and 1000 time steps, there are 1 *billion* entries, one for every origin, every destination, and departure time. This is much larger than the number of vehicles that will be assigned, so almost every entry in this matrix will be zero. In such cases, A^* can work much better, only being applied to origins, destinations, and departure times with a positive entry in the matrix.

As discussed in Section 2.4.4, an effective estimate for A^* in traffic assignment problems is to use the free-flow travel costs. As a preprocessing step at the start of traffic assignment, you can use an all-to-one static shortest path algorithm to find the least-cost travel cost g_j^s from every node j to every destination s at free flow. For the remainder of the traffic assignment algorithm, you can then use g_j^s as the estimates for A^* . This is quite effective in practical networks.

10.5 Historical Notes and Further Reading

(These sections are incomplete in this beta version of the text, and will be substantially expanded in the complete first edition.)

Several authors have discussed how the time-dependent shortest path problem differs from the static case; both label-setting (Dreyfus, 1969) and label-correcting (Cooke and Halsey, 1969; Ziliaskopoulos and Mahmassani, 1994) methods exist. Chabini (1999) showed how the time index as a topological order in the time-expanded network. The use of free-flow times as estimates in time-dependent A^* is from Boyles (2013).

Section 2.5 described how the shortest path problem can be generalized to

Table 10.5: Link costs for Exercises 1 and 10.

Entry time	Link 1	Link 2	Link 3	Link 4	Link 5
0	5	1	3	1	1
1	6	1	3	6	1
2	4	2	3	5	2
3	5	3	3	4	4
4	7	5	3	3	2
5	8	8	3	2	1

include stochastic link costs, where the link costs are drawn from some probability distribution and the path with minimum expected cost is sought. You may recall that this problem was not too difficult to address in the static case. However, if travel times are *both* time-dependent and stochastic, more care is needed (Hall, 1986; Fu and Rilett, 1998), because Bellman's principle need not hold. Examples of algorithms to handle this issue are given in Hall (1986) and Miller-Hooks and Mahmassani (2000).

10.6 Exercises

1. [13] Table 10.5 shows time-dependent costs on five links, for different entry times. Which links have costs satisfying the FIFO principle?
2. [23] Prove that waiting is never beneficial in a FIFO network where link costs are equal to travel time.
3. [34] Prove that there is an acyclic time-dependent shortest path in a FIFO network where link costs are equal to travel time.
4. [53] In this chapter, we assumed it is impossible to travel beyond the time horizon, by not creating time-expanded links if they arrive at a downstream node after \bar{T} . Another alternative is to assume that travel beyond \bar{T} is permitted, but that travel times and costs stop changing after that point and take constant values. (Perhaps free-flow times after the peak period is over.)
 - (a) Modify the time-expanded network concept to handle this assumption. (The network should remain finite.)
 - (b) Modify the algorithm in Section 10.2.2 to work in this setting.
5. [63] Another way to handle the time horizon is to assume that the link travel times and costs are *periodic* with length \bar{T} . (For instance, \bar{T} may be 24 hours, and so entering a link at $t = 25$ hours would be the same as at $t = 1$ hour.) First repeat Exercise 4 with this assumption. Then prove that the modified algorithm you create will converge to the correct time-dependent shortest paths for any possible departure time.

Table 10.6: Backnode labels for Exercise 9.

Entry time	Node 1	Node 2	Node 3	Node 4
6	-1	4	3	2
5	-1	4	3	-1
4	-1	-1	-1	-1
3	-1	4	-1	-1
2	-1	4	-1	-1
1	-1	-1	-1	-1

Table 10.7: Cost labels for Exercise 9.

Entry time	Node 1	Node 2	Node 3	Node 4
6	0	1	5	5
5	0	2	4	∞
4	0	∞	∞	∞
3	0	2	∞	∞
2	0	1	∞	∞
1	0	∞	∞	∞

6. [31] Modify the algorithm in Section 10.3.1 to become an all-to-one algorithm, that finds least-cost paths from all origins and all departure times to one destination (arrival at any time is permitted).
7. [34] Show that the classical form of Bellman's principle (Section 2.4 holds in a FIFO, time-dependent network where the link costs are equal to the link travel time.)
8. [32] Find the time-dependent shortest path when departing node 1 in the network shown in the *left* panel of Figure 10.3, departing at time 0.
9. [23] Tables 10.6 and 10.7 show the backnode and cost labels for a time-dependent shortest path problem, where the destination is node 4, and the time horizon is 6. (Figure 10.7 shows the network topology.) What is the shortest path from node 1 to node 4, when departing at time 1?
10. [22] In the same network as Exercise 9, fill in the forwardnode and cost labels for time 0. The link costs are as in Table 10.5, and waiting is not allowed at nodes.
11. [45] Consider the network in Figure 10.8.
 - (a) Verify that the travel times satisfy the FIFO principle.
 - (b) Find the shortest paths between nodes 1 and 4 when departing at $t = 0$, $t = 2$, and $t = 10$.

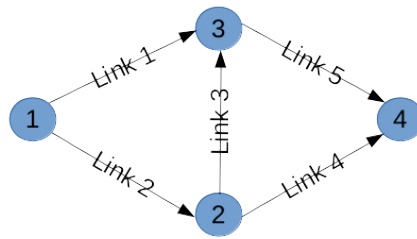


Figure 10.7: Network for Exercises 9 and 10.

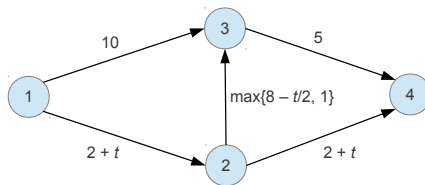


Figure 10.8: Network for Exercise 11.

- (c) For what departure times would the travel times on paths $[1, 2, 4]$ and $[1, 3, 4]$ be equal?
12. [23] Solve the time-dependent shortest path problem on the network in Figure 10.6 using the algorithm in Section 10.2.1. Comment on the amount of work needed to solve the problem using this algorithm, compared to the way it was solved in the text.
 13. [51] In the schedule delay equation (10.4), we typically assume $0 \leq \alpha \leq 1 \leq \beta$ for peak-hour commute trips. What counterintuitive behavior would occur if any of these three inequalities were violated?
 14. [12] Find the optimal departure times and paths from nodes 2 and 3 in the network in Figure 10.6, with equation (10.6) as the arrival time penalty. (You can answer this question directly from Table 10.3.)
 15. [36] Find the optimal departure times and paths from nodes 1, 2, and 3 in the network in Figure 10.6, if the arrival time penalty function is changed to $f(t) = ([12 - t]^+)^2 + 2([t - 12]^+)^2$.
 16. [0] According to the penalty function in Exercise 15, what is the desired arrival time?
 17. [88] Implement all of the algorithms described in this chapter, and test them on transportation networks with different characteristics (number of

origins, destinations, nodes, links, time intervals, ratio of links to nodes, with or without waiting, FIFO or non-FIFO, etc.). What algorithms perform best under what circumstances? For circumstances most resembling real-world transportation networks, what performs best?

Chapter 11

Dynamic User Equilibrium

The previous two chapters presented separate perspectives on dynamic traffic modeling. Chapter 9 described the network loading problem, in which driver behavior (as represented by path choices and possibly departure times) was known, and where we sought to represent the resulting traffic and congestion patterns on the network. Chapter 10 described the time-dependent shortest path and departure time choice problems, in which the network state (as represented by travel times and costs) was known, and where we sought to identify how drivers would behave. This chapter synthesizes these two perspectives through the concept of dynamic user equilibrium, defined as driver choices and network traffic which are mutually consistent, given the network loading and driver behavior assumptions. The dynamic user equilibrium principle, and how the network loading and driver behavior models can be connected, are the subjects of Section 11.1. Section 11.2 then presents several algorithms that can be used to solve for dynamic equilibria. Such algorithms are almost always heuristics, since realistic network loading models are not amenable to exact analysis, and indeed Section 11.3 shows that dynamic user equilibrium need not exist; and if it exists, it need not be unique. This section also provides examples to show that the dynamic user equilibrium solution need not minimize total travel time in a network, and that providing additional capacity to a network can increase total travel time, a dynamic analogue of the Braess paradox.

11.1 Towards Dynamic User Equilibrium

This section has two major goals, aimed at reconciling the network loading problem of Chapter 9 and the time-dependent shortest path algorithms of Chapter 10. In particular, we will need to solve these problems sequentially, and repeatedly, as shown in Figure 11.1. The first goal of this section is to provide the mechanics to link the output of each problem to the input needed by the other. Following these, we will be equipped to define dynamic user equilibrium formally. Fixed point and variational inequality formulations are given as well.

11.1.1 Flow Representation

The first question to address is how to represent the choices of all the travelers on the network. In describing network loading, Chapter 9 took an aggregate approach. All of the link models in that chapter can be seen as fluid models, where vehicles are infinitely divisible rather than discrete entities. However, the algorithms in Chapter 10 took a disaggregate approach, and found paths and departure times for a single vehicle. There are several ways to reconcile these perspectives.

The method adopted in this chapter is to store the number of vehicles departing on each path during each time interval. Denote by h_t^π the number of vehicles which start traveling on path π during the t -th interval. These values do not need to be integers; all that matters is that they are nonnegative, and that all paths connecting the same origin and destination sum to the total demand between these zones for each time interval. The h_t^π values can be collected into a single matrix H , whose dimensions are equal to the number of paths and time intervals. With this notation, the feasible set can be written as

$$\bar{H} = \left\{ H \in \mathbb{R}_+^{\bar{T} \times |\Pi|} : \sum_{\pi \in \Pi_{r,s}} h_t^\pi = d_t^{rs} \quad \forall (r, s) \in Z^2, t \right\}. \quad (11.1)$$

The main advantage of this notation is that the behavior is clear: by tracking some auxiliary variables in the network loading (as described in Section 11.1.3, at any point in time we can see exactly which vehicles are on which links, and can trace these to the paths the vehicles must follow). These paths can be connected directly with the paths found in a time-dependent shortest path algorithm. A disadvantage of this approach is that the number of paths grows exponentially with the network size. In practice, “column generation” schemes are popular, in which paths π are only identified when found by a shortest path algorithm, and h_t^π values only need be calculated and stored for paths to which travelers have been assigned.

This is not the only possible way to represent travel choices. A link-based representation requires fewer variables. For each turning movement $[h, i, j]$, each destination s , and each time interval t , let $\alpha_{hij,s}^t$ denote the proportion of travelers arriving at node i , via approach (h, i) , during the t -th time interval, who will exit onto link (i, j) *en route* to destination s . This approach mimics the flow splitting rules often found in traffic microsimulation software. The number of variables required by this representation grows with the network size, but not at the exponential rate required by a path-based approach. One can show that the path-based and link-based representation of route choice are equivalent in the sense that there exist α values which represent the same network state as any feasible set of H values, regardless of the network loading model, and that one can identify the H values corresponding to a given set of α values (see Exercise 1). The primary disadvantage of this representation is that the behavior is less clear: one cannot trace the path of any vehicle throughout the network deterministically (although one can do so stochastically, making a turn

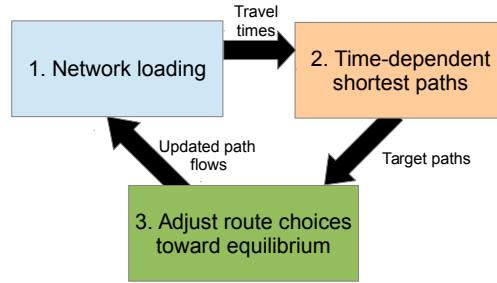


Figure 11.1: Iterative dynamic traffic assignment framework, with details from previous chapters.

at each node by treating the α values as probabilities, identifying the arrival times at successive nodes using the procedure in Section 11.1.2).

Both of these methods adopt the aggregate, continuous-flow perspective of Chapter 9. Yet another way to represent travel choices is to adopt the individual perspective of the algorithms in Chapter 10, and explicitly model each vehicle as a discrete agent with one specific path. This is the most behavioral way to represent choices: each individual vehicle is assigned one path, with no divisibility or fractional flows. Downsides of this approach are scalability — if the demand doubles but the network topology is unchanged, the individual vehicle approach would require twice as many variables, whereas the continuous methods would not require any more variables — and increased difficulty in implementing the network loading algorithms. It is certainly possible to implement discrete versions of merges, diverges, the cell transmission model, and so forth, but one must be careful about rounding. For instance, if the time step is chosen so that the capacity of a link is less than half a vehicle per timestep, consistently rounding numbers to the nearest integer would mean no vehicles can ever exit. Stochastic rounding, or accumulating a continuous flow value which can then be rounded, can address these difficulties.

11.1.2 Travel time calculation

Once network loading is complete, a time-dependent shortest path algorithm can be applied to determine the cost-minimizing routes for travelers. To do so, we need the travel times $\tau_{ij}(t)$ on each link (i, j) for travelers entering at each time t . The network loading models do not provide this directly, but rather give the cumulative counts N at the upstream and downstream ends of each link, at each time interval. However, this information will suffice for calculating the travel times. For link (i, j) at time t , the upstream count $N_{ij}^{\uparrow}(t)$ gives the total number of vehicles which have entered the link by time t , while the downstream count $N_{ij}^{\downarrow}(t)$ gives the total number of vehicles which left the link by time t .

Assume for a moment that $N_{ij}^{\uparrow}(t)$ and $N_{ij}^{\downarrow}(t)$ are strictly increasing, continu-

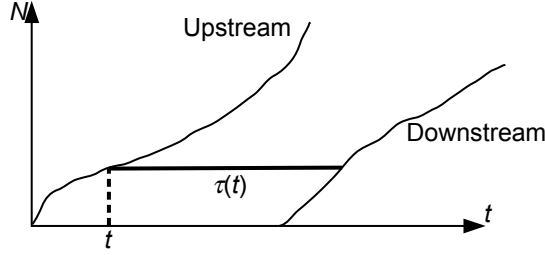


Figure 11.2: Obtaining travel times from plots of the cumulative counts N^\uparrow and N^\downarrow .

ous functions of t . If this is the case, we can define inverse functions $T_{ij}^\uparrow(n)$ and $T_{ij}^\downarrow(n)$, respectively giving the times when the n -th vehicle entered the link and left the link. The travel time for the n -th vehicle is then the difference between these: $T_{ij}^\downarrow(n) - T_{ij}^\uparrow(n)$. Graphically, this can be seen as the horizontal difference between the upstream and downstream N curves. (Figure 11.2). Then, to find the travel time for a vehicle entering the link at time t , we simply evaluate this difference for $n = N_{ij}^\uparrow(t)$:

$$\tau_{ij}(t) = T_{ij}^\downarrow(N_{ij}^\uparrow(t)) - T_{ij}^\uparrow(N_{ij}^\uparrow(t)) = T_{ij}^\downarrow(N_{ij}^\uparrow(t)) - t \quad (11.2)$$

since T_{ij}^\uparrow and N_{ij}^\uparrow are inverse functions.

A little bit of care must be taken because $N_{ij}^\uparrow(t)$ and $N_{ij}^\downarrow(t)$ are not strictly increasing functions of time unless there is always a positive inflow and outflow rate for link (i, j) . Furthermore, we often introduce a time discretization. For both of these reasons, the inverse functions $T_{ij}^\uparrow(n)$ and $T_{ij}^\downarrow(n)$ may not be well-defined. We thus need to modify equation (11.2) in a few ways:

- We must ensure that $\tau_{ij}(t)$ is always at least equal to the free-flow travel time on the link. The danger is illustrated in Figure 11.3, where no vehicles enter or leave the link for an extended period of time. The horizontal distance between the N^\uparrow and N^\downarrow curves at their closest point is small (the dashed line in the figure), but this does not reflect the actual travel time of any vehicle. In reality, a vehicle entering the link when it is completely empty, and when there is no downstream bottleneck, would experience free-flow conditions on the link.
- If the link has no outflow for an interval of time, then $N_{ij}^\downarrow(t)$ will be constant over that interval. This frequently happens with traffic signals. In this case, there are multiple values of time where $N_{ij}^\downarrow(t) = n$. The correct way to resolve this is to define $T_{ij}^\downarrow(n)$ to be the *earliest* time for

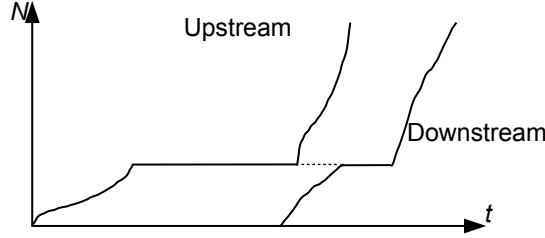


Figure 11.3: Cumulative counts are flat when there is no inflow or outflow.

which $N_{ij}^\downarrow(t) = n$:

$$T_{ij}^\downarrow(n) = \min_t \left\{ t : N_{ij}^\downarrow(t) = n \right\}. \quad (11.3)$$

- In discrete time, the time at which the n -th vehicle departs may not line up with a multiple of Δt , so there may be no known point where $N_{ij}^\downarrow(t)$ is exactly equal to n . In this case, it is appropriate to interpolate between the last time point where $N_{ij}^\downarrow(t) < n$, and the first time point where $N_{ij}^\downarrow(t) \geq n$.

With these modifications to how T_{ij}^\downarrow is calculated, the formula (11.2) can be used to calculate the travel times on each link and arrival time.

The travel time on a path π for a traveler departing at time t , denoted $C^\pi(t)$, can then be calculated sequentially. If the path is $\pi = [r, i_1, i_2, \dots, s]$, then the traveler departs origin r at time t , and arrives at node i_1 at time $t + \tau_{ri_1}(t)$. The travel time on link (i_1, i_2) is then $\tau_{i_1 i_2}(t + \tau_{ri_1}(t))$, so the traveler arrives at i_2 at time $t + \tau_{ri_1}(t) + \tau_{i_1 i_2}(t + \tau_{ri_1}(t))$, and so forth. Writing out this formula can be a bit cumbersome, but calculating it in practice is quite simple: it is nothing more than accumulating the travel times of the links in the path, keeping track of the time at which each link is entered.

11.1.3 Determining splitting proportions

The network loading requires knowledge of when and where vehicles enter the network, and the splitting proportions p at diverges and general intersections. In dynamic traffic assignment, this is reflected indirectly, through the variables $h^\pi(t)$, denoting the number of travelers departing path π at time t . We do not specifically give the proportions p as a function of time, because we do not know when any traveler will reach any node before actually performing the network loading. Path choice is a behavioral parameter associated with travelers, so it is easier to reconcile path choices with behavior if they are expressed this way, rather than by simply specifying the turning fractions at each node.

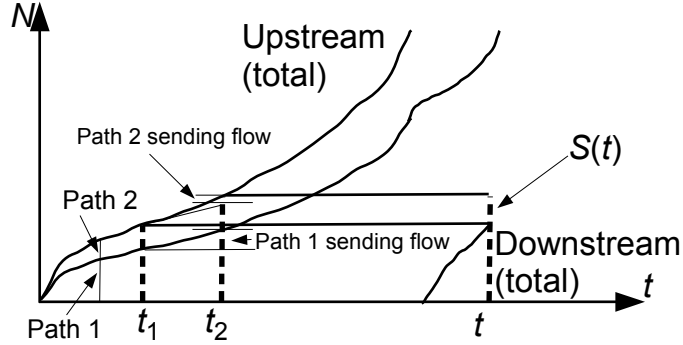


Figure 11.4: Disaggregating the sending flow by path.

That said, the network loading does in fact need $p_{hij}(t)$ values for each turning movement $[h, i, j]$ at a diverge or general intersection, for each time period t . These are obtained by examining the vehicles comprising the sending flow $S_{hi}(t)$, and calculating the fraction of these vehicles whose path includes link (i, j) as the next link. One way to do this is to disaggregate the cumulative count values $N^\uparrow(t)$ and $N^\downarrow(t)$ calculated at the upstream and downstream ends of each link. For each path π in the network, and for every link (h, i) and time t , define $N_{hi,\pi}^\uparrow(t)$ and $N_{hi,\pi}^\downarrow(t)$ to be the total number of vehicles using path π which have respectively entered and left link (h, i) by time t . Clearly we have

$$N_{hi}^\uparrow(t) = \sum_{\pi \in \Pi} N_{hi,\pi}^\uparrow(t) \quad N_{hi}^\downarrow(t) = \sum_{\pi \in \Pi} N_{hi,\pi}^\downarrow(t) \quad \forall (h, i) \in A, t. \quad (11.4)$$

Then, the sending flow for each link and time interval can be disaggregated in the same way, with $S_{hi,\pi}(t)$ defined as the number of vehicles in the sending flow which are using path π . Assume that we are calculating sending flow for link (h, i) at time t , and have determined $S_{hi}(t)$. At this point in time, the total number of vehicles which has left this link is $N_{hi}^\downarrow(t)$. Therefore, the vehicles in the sending flow are numbered in the range $N_{hi}^\downarrow(t)$ to $N_{hi}^\downarrow(t) + S_{hi}(t)$. Using the inverse functions (11.2), the times at which these vehicles entered link (h, i) are in the range $T^\uparrow(N_{hi}^\downarrow(t))$ to $T^\uparrow(N_{hi}^\downarrow(t) + S_{hi}(t))$. Denote these two times by t_1 and t_2 , respectively. Then the disaggregate sending flow $S_{hi,\pi}(t)$ is the number of vehicles on path π which entered link (h, i) between t_1 and t_2 , that is,

$$S_{hi,\pi}(t) = N_{hi,\pi}^\uparrow(t_2) - N_{hi,\pi}^\uparrow(t_1). \quad (11.5)$$

This procedure is illustrated in Figure 11.4.

The proportion of travelers in the sending flow S_{hi} wishing to turn onto link (i, j) can now be calculated. Let $\Pi_{[h,i,j]}(t)$ be the set of paths which arrive at

node (h, i) at time t and immediately continue onto link (i, j) . Then

$$p_{hij}(t) = \frac{\sum_{\pi \in \Pi_{[h,i,j]}(t)} S_{hi,\pi}(t)}{S_{hi}(t)} \quad (11.6)$$

for use in diverge or general intersection node models. Then, after a node model produces the actual flows $y_{hij}(t)$ between links, the disaggregated cumulative counts must be updated as well.

Let $y_{hi}(t) = \sum_{(i,j) \in \Gamma(i)} y_{hij}(t)$ be the total flow which leaves link (h, i) during time t . Then all vehicles which entered the link between t_1 and $T_{hi}^\uparrow(N^\downarrow(t)_{hi} + y_{hi})$ can leave the link (use t_3 to denote this latter time), so we update the downstream aggregate counts to

$$N_{hi,\pi}^\downarrow(t + \Delta t) = N_{hi,\pi}^\uparrow(t_3) \quad (11.7)$$

and the upstream counts to

$$N_{ij,\pi}^\uparrow(t + \Delta t) = N_{hi,\pi}^\downarrow(t + \Delta t). \quad (11.8)$$

where (h, i) is the link immediately upstream of (i, j) on path π .

This formula is an approximation, since it assumes that the vehicles on different paths in the sending flow are uniformly distributed. In reality, vehicles towards the start of the sending flow may have a different mix of paths than vehicles towards the end of the sending flow. This means that the p value (calculated based on the entire sending flow) and the actual vehicles which are moved (which are from the start of the sending flow) may not be entirely consistent. In practice, this violation is typically small. If one wishes to obtain an exact formula, an iterative process can be used to ensure that the proportions p are consistent with the proportion of vehicles between t_1 and t_3 .

11.1.4 Principle of dynamic user equilibrium

With the procedures in the previous two subsections, we can now iteratively perform network loading and find time-dependent shortest paths. The dynamic traffic assignment problem is to find a mutually consistent solution between these two models. Let H be a matrix of path flows indicating the number of vehicles departing on each path at each time, and let T be a matrix of path travel times, indicating the travel time for each path and departure time. Then the network loading can be concisely expressed by

$$T = \mathcal{N}(H) \quad (11.9)$$

where \mathcal{N} is a function encompassing whatever network loading is performed, mapping the path flows to path travel times. Likewise, let the assumed behavioral rule be denoted by \mathcal{B} , which indicates the allowable path flow matrices H when the travel times are T . For instance, one may assume that departure times are fixed, but only minimum-travel time paths must be used; that minimum-travel time paths must be used but the departure times must minimize total

cost (travel time plus schedule delay); or some other principle. In general, there may be multiple H matrices which satisfy this principle, as when multiple paths have the same, minimal travel time, in which case \mathcal{B} actually denotes a *set* of H matrices. So, we can express the behavioral consistency rule by

$$H \in \mathcal{B}(T). \quad (11.10)$$

We are now in a position to define the solution to the dynamic traffic assignment problem as a fixed point problem: *the path flow matrix H is a dynamic user equilibrium if*

$$H \in \mathcal{B}(\mathcal{N}(H)). \quad (11.11)$$

That is, the path flow matrix must be consistent with the driver behavior assumptions, when the travel times are obtained from that same path flow matrix. The most common behavioral rule is that departure times are fixed, but only minimum-travel time paths will be used. In this case the principle can be stated more intuitively as *all used paths connecting the same origin to the same destination at the same departure time must have equal and minimal travel time*. If departure times can be varied to minimize total cost, then the principle can be stated as *all used paths connecting the same origin to the same destination have the same total cost, regardless of departure time*.

The dynamic user equilibrium solution can also be stated as a solution to a variational inequality. For the case of fixed departure times, the set of feasible path flow matrices is given by

$$\bar{H} = \left\{ H \in \mathbb{R}_+^{\bar{T} \times |\Pi|} : \sum_{\pi \in \Pi_{r,s}} h_t^\pi = d_t^{r,s} \quad \forall (r,s) \in Z^2, t \right\}, \quad (11.12)$$

that is, matrices with nonnegative entries where the sum of all path flows connecting the origin r to destination s at departure time t is the corresponding value in the time-dependent OD matrix $d_t^{r,s}$. Then, the dynamic user equilibrium solution \hat{H} satisfies

$$\mathcal{N}(\hat{H}) \cdot (\hat{H} - H) \leq 0 \quad \forall H \in \bar{H}, \quad (11.13)$$

where the product \cdot is the Frobenius product, obtained by treating the matrices as vectors and calculating their dot product.

In the case of departure time choice, we can define $S = \mathcal{S}(H)$ to be the matrix of total costs for each path and departure time, where \mathcal{S} is obtained by composing the schedule delay function (10.4) with the network loading mapping \mathcal{N} . In this case, the set of feasible path flow matrices is given by

$$\hat{H} = \left\{ H \in \mathbb{R}_+^{\bar{T} \times |\Pi|} : \sum_{\pi \in \Pi_{r,s}} \sum_t h_t^\pi = d^{r,s} \quad \forall (r,s) \in Z^2 \right\}, \quad (11.14)$$

where $d^{r,s}$ is the OD matrix giving total flows between each origin and destination throughout the analysis period. The variational inequality in this case is

$$\mathcal{S}(\hat{H}) \cdot (\hat{H} - H) \leq 0 \quad \forall H \in \hat{H}. \quad (11.15)$$

Although these fixed point and variational inequality formulations encode the equilibrium principle, and concretely specify the problem at hand, little else can be proven concretely except in the case of simple link and node models (such as point queues). The vast majority of theoretical results on variational inequalities and fixed point problems relies on regularity conditions, such as continuity or monotonicity of the mappings \mathcal{N} or \mathcal{H} . For realistic link models and node models, these mappings are not continuous. For example, queue spillback imposes discontinuities on the network loading mapping. This means that we usually cannot prove that dynamic user equilibrium always exists or is unique, and indeed later in this chapter we present counterexamples where this need not be the case. However, the variational inequality can be used to define gap measures that can measure the degree of compliance with the equilibrium principle. In practice, whether the increased realism of a dynamic traffic model outweighs these disadvantages is case-dependent, and these considerations enter into how the appropriate modeling tools are chosen.

11.2 Solving for Dynamic Equilibrium

This section describes how dynamic user equilibrium solutions can be found. All of the algorithms in this section follow the same general framework, based on iterating between network loading, finding time-dependent shortest paths, and a path updating procedure, and they share common termination criteria. Three specific path updating procedures are discussed here: the convex combinations method, simplicial decomposition, and gradient projection. These are all heuristics, without making stronger assumptions on the properties of the network loading. This is demonstrated more fully in the next section, where we show that dynamic traffic assignment does not always share the neat solution properties of static assignment. These methods are described only for the case of fixed departure times, but it is not difficult to extend any of them to the case of departure time choice with schedule delay.

11.2.1 General framework

The general framework for almost all dynamic traffic assignment algorithms involves iterating three steps: network loading, as discussed in Chapter 9, time-dependent shortest paths, as discussed in Chapter 10, and updating the path flow matrix H , as discussed here. The algorithm takes the following form:

1. Initialize the path flow matrix H to a feasible value in \bar{H} .
2. Perform the network loading using path flows H , obtaining path travel times $T = \mathcal{N}(H)$.
3. Identify the time-dependent shortest paths.
4. Check termination criteria; if sufficiently close to dynamic user equilibrium, stop.

5. Update the path flow matrix H and return to step 2.

A few steps in this algorithm warrant further explanation. The choice of the initial solution is arbitrary. One reasonable choice is to identify one or more low travel-time paths between each OD pair based on free-flow travel times, and to divide the demand in the OD matrix between them. The effort and time involved in identifying the initial solution should be balanced against the benefit obtained by a more careful choice, as opposed to starting with a simpler choice and just running additional iterations.

The termination criteria in step 4 can take several forms. The most theoretically sound termination criterion is a gap measure, comparing the path flows H and travel times T with the assumed behavioral rule for travelers. Such measures are preferred to simpler criteria, such as stopping when the path flows do not change much from iteration to iteration — if the latter occurs, there is no way to tell whether this is because we are near an equilibrium solution, or because the algorithm is unable to make any more progress and has gotten stuck near a non-equilibrium solution. For the case of fixed demand, the behavioral rule is that travelers must take a least travel-time path between their origin and destination, for their departure time. After performing the time-dependent shortest path step, we can calculate the travel time on such a path for each OD pair (r, s) and departure time t ; call this value $\tau_{r,s,t}^*$. If all travelers in the network were on such paths, then the total travel time experienced by all travelers in the network would be given by

$$SPTT = \sum_{(r,s) \in Z^2} \sum_t d_t^{rs} \tau_{r,s,t}^*. \quad (11.16)$$

This is often known as the *shortest path travel time*. This is contrasted with the *total system travel time*, which is the actual total travel time spent by all travelers on the network:

$$TSTT = \sum_{\pi \in \Pi} \sum_t h_{\pi}^t \tau_{\pi}^t = H \cdot T. \quad (11.17)$$

Clearly $TSTT \geq SPTT$, and $TSTT = SPTT$ only if all travelers are on least-time paths, which corresponds to dynamic user equilibria. Therefore, the gap between these values is a quantitative measure of how close the path flows H are to satisfying the equilibrium principle. Two common gap values involve normalizing the difference $TSTT - SPTT$ in two ways. The *relative gap* γ normalizes this difference by SPTT,

$$\gamma = \frac{TSTT - SPTT}{SPTT}, \quad (11.18)$$

while the *average excess cost* normalizes this difference by the total number of travelers on the network,

$$AEC = \frac{TSTT - SPTT}{\sum_{(r,s) \in Z^2} \sum_t d_t^{rs}}. \quad (11.19)$$

The algorithm can be terminated whenever either of these measures is sufficiently small. An advantage of the average excess cost is that it is measured in time units, and has the intuitive interpretation as being the average difference between a traveler's actual travel time, and the shortest path travel time available to them. Similar expressions can be derived for departure time choice, with schedule delay.

Finally, the last step (updating the path flow matrix) requires the most care, and this section presents three alternatives. The convex combinations method is the simplest and most economical in terms of computer memory. The simplicial decomposition method converges faster, by storing previous solutions and using them to find a better improvement direction for H . The third method is gradient projection, which involves computation of an approximate derivative of path travel times with respect to path flow. This derivative enables the use of Newton's method to identify the amount of flow to shift between paths.

11.2.2 Convex combinations method

The convex combinations method is the simplest way to update the path choice matrix. In this method, after computing time-dependent shortest paths, one identifies a "target matrix" H^* which would give the path flows travelers would choose if the path travel times were held constant at their current values T . For the case of fixed departure times, for each OD pair (r, s) and departure time t , all of the demand d_t^{rs} is loaded onto its time-dependent shortest path in H^* , and all other path flows are set to zero. For the case of departure time choice, for each OD pair (r, s) the departure time t^* and corresponding path with minimal schedule delay is identified, and all d^{rs} travelers associated with this OD pair are assigned to that path and departure time, with all other path flows set to zero. This is often called an *all-or-nothing assignment*, because it involves choosing a single alternative to assign an entire class of travelers.

The basic form of the convex combinations method updates H using the formula

$$H \leftarrow \lambda H^* + (1 - \lambda)H, \quad (11.20)$$

where $\lambda \in [0, 1]$ is a step size showing how far to move in the direction of the target. If $\lambda = 0$, then the path flow matrix is unchanged, whereas if $\lambda = 1$, the path flow matrix is set equal to the target. Intermediate values produce a solution which is a weighted average of the current solution H and the target H^* . There are a number of ways to choose the step sizes λ . The simplest alternative is to use a decreasing sequence of values, choosing for the i -th iteration the step size $\lambda_i = 1/(i + 1)$, so the step sizes form the sequence $1/2, 1/3, 1/4$, and so on. It is common to use sequences such that $\sum \lambda_i = +\infty$ and $\sum \lambda_i^2 < \infty$, because we do not want the sequence of H values to converge too quickly (or else it may stop short of the equilibrium), but if the step size drops too slowly, the algorithm is likely to "overshoot" the equilibrium and oscillate.

More sophisticated variations are possible as well. Rather than using a fixed step size, λ can be chosen to minimize, say, the relative gap or average excess

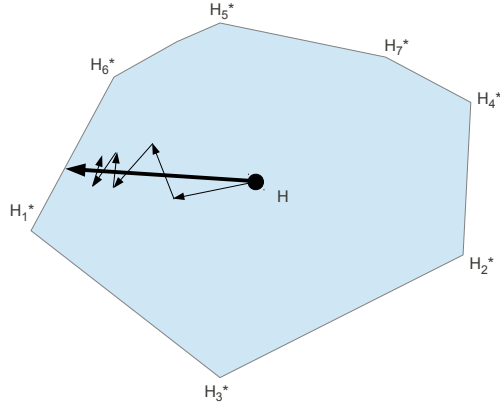


Figure 11.5: Simplicial decomposition can move towards non-corner points.

cost. While this likely decreases the number of iterations required to reach a small gap, the computation at each iteration is increased. In particular, each λ value tested requires an entire network loading step to determine the corresponding gap, which can be a significant time investment. It is also possible to vary the λ value for different path flows and departure times. Some popular heuristics are to use larger λ values for OD pairs which are further from equilibrium, or to vary λ for different departure times — since travel times for later departure times depend on the path choices for earlier departure times, it may not make sense to invest much effort in equilibrating later departure times until earlier ones have stabilized.

11.2.3 Simplicial decomposition

The simplicial decomposition algorithm extends the convex combinations method by remembering the target matrices H^* from earlier iterations. This requires more computer memory, but storing these previous target matrices leads to a more efficient update of the path flow vector. In particular, choosing an all-or-nothing assignment as a target matrix essentially restricts the choice of search direction to the corner points of the feasible region. As a result, when the equilibrium solution is reached, the convex combinations method will “zig-zag,” taking a number of short, oblique steps rather than a more direct step towards the equilibrium problem. (Figure 11.5). Simplicial decomposition is able to do so by combining multiple target points into the search direction. In this algorithm, the set \mathcal{H} is used to store all of the target path flow matrices found thus far.

A second notion in simplicial decomposition is that of a “restricted equilibrium.” Given a set $\mathcal{H} = \{H_1^*, H_2^*, \dots, H_k^*\}$, we say that a time-dependent path flow matrix \hat{H} is a *restricted equilibrium relative to \mathcal{H}* if it solves the variational

inequality

$$\mathcal{N}(\hat{H}) \cdot (\hat{H} - H_i^*) \leq 0 \quad \forall H_i^* \in \mathcal{H}. \quad (11.21)$$

This is different from the variational inequality (11.13) because the only possible choices for H are the matrices in the set \mathcal{H} , rather than *any* of the feasible H matrices satisfying (11.12).

Effectively, H is a restricted equilibrium if none of the targets in \mathcal{H} lead to improving directions in the sense that the total system travel time would be reduced by moving to some $H_i^* \in \mathcal{H}$ while fixing the travel times at their current values.

At a high level, simplicial decomposition works by iterating between adding a new target matrix to \mathcal{H} , and then finding a restricted equilibrium using the current matrices in \mathcal{H} . In practice, it is too expensive to exactly find a restricted equilibrium at each iteration. Instead, several “inner iteration” steps are taken to move towards a restricted equilibrium with the current set \mathcal{H} before looking to add another target. In each inner iteration, the current solution H is adjusted to $H + \mu\Delta H$, where μ is a step size and ΔH is a direction which moves toward restricted equilibrium. One good choice for this direction is

$$\Delta H = \frac{\sum_{H_i^* \in \mathcal{H}} [\mathcal{N}(H) \cdot (H - H_i^*)]^+ (H_i^* - H)}{\sum_{H_i^* \in \mathcal{H}} [\mathcal{N}(H) \cdot (H - H_i^*)]^+} \quad (11.22)$$

This rather intimidating-looking formula is actually quite simple. It is nothing more than a weighted average of the directions $H_i^* - H$ (potential moves toward each target in \mathcal{H}), where the weight for each potential direction is the extent to which it improves upon the current solution: $[\mathcal{N}(H) \cdot (H - H_i^*)]$ is the reduction in total system travel time obtained by moving from H to H_i^* while holding travel times constant. If this term is negative, there is no need to move in that direction, so the weight is simply set to zero. The denominator is simply the sum of the weights, which serves as a normalizing factor.

The step size μ is chosen through trial-and-error. One potential strategy is to iteratively test μ values in some sequence (say, 1, 1/2, 1/4, ...) until we have found a solution acceptably closer to restricted equilibrium than H .¹ “Acceptably closer” can be calculated using the *restricted average excess cost*

$$AEC' = \frac{\mathcal{N}(H) \cdot H - \min_{H_i^* \in \mathcal{H}} \{\mathcal{N}(H) \cdot H_i^*\}}{\sum_{(r,s) \in Z^2} \sum_t d_t^{rs}} \quad (11.23)$$

which is similar to the average excess cost, but instead of using the shortest path travel time uses the best available target vector in \mathcal{H} .

Unlike the version of simplicial decomposition used for the static traffic assignment problem (Chapter 6), there is no guarantee that a sufficiently small choice of μ will result in a reduction of restricted average excess cost. However, in practice, this rule seems to work acceptably.

¹Those familiar with nonlinear optimization may see parallels between this and the Armijo rule.

Putting all of this together, for simplicial decomposition, step 5 of the dynamic traffic assignment algorithm in Section 11.2.1 involves performing all of these steps as a “subproblem”:

5. **Subproblem:** Find an approximate restricted equilibrium H using only the vectors in \mathcal{H} .
 - (a) Find the improvement direction ΔH using equation (11.22).
 - (b) Update $H \leftarrow H + \mu\Delta H$, with μ sufficiently small (to reduce AEC').
 - (c) Perform network loading to update travel times.
 - (d) Return to step (a) of subproblem unless AEC' is small enough.

Furthermore, the set \mathcal{H} must be managed. It is initialized to be empty, and whenever time-dependent shortest paths are found, a new all-or-nothing assignment (the one which would have been chosen as the sole target in the convex combinations method) is added to \mathcal{H} .

11.2.4 Gradient projection

The gradient project method updates the path flow matrix H by using Newton’s method. Consider first the simpler problem of trying to shift flows between two paths π_1 and π_2 (for the same departure time) to equalize their travel times. Write $\tau_1(h_1, h_2)$ and $\tau_2(h_1, h_2)$ to denote the travel times on the two paths as a function of the flows h_1 and h_2 on the two paths. If we were to shift Δh vehicles from path 1 to path 2, the difference in travel times between the paths is given by

$$g(\Delta h) = \tau_1(h_1 - \Delta h, h_2 + \Delta h) - \tau_2(h_1 - \Delta h, h_2 + \Delta h). \quad (11.24)$$

We want to choose Δh so that the difference $g(\Delta h)$ between the path travel times is equal to zero.

Using one step of Newton’s method, an approximate value of Δh is given by

$$\Delta h = -\frac{g(0)}{g'(0)}. \quad (11.25)$$

The numerator $g(0)$ is simply the difference in travel times between the two paths before any flow is shifted. To calculate the denominator, we need to know how the difference in travel times will change as flow is shifted from π_1 to π_2 . Computing the derivative of (11.24) with the help of the chain rule, we have

$$g'(0) = \left(\frac{\partial t_1}{\partial h_2} + \frac{\partial t_2}{\partial h_1} \right) - \left(\frac{\partial t_1}{\partial h_1} + \frac{\partial t_2}{\partial h_2} \right). \quad (11.26)$$

This formula is not easy to evaluate exactly, but we can make a few approximations. The first term in (11.26) reflects the impact the flow on one path has on the *other* path’s travel time, while the second term reflects the impact the flow on one path has on *its own* travel time. Typically, we would expect the second

effect will be larger in magnitude than the first effect (although exceptions do exist). So, as a first step we can approximate the derivative as

$$g'(0) \approx - \left(\frac{\partial t_1}{\partial h_1} + \frac{\partial t_2}{\partial h_2} \right). \quad (11.27)$$

The next task is to calculate the derivative of a path travel time with respect to flow along this path. Unlike in static traffic assignment, there is no closed-form expression mapping flows to travel times, but rather a network loading procedure must be used. For networks involving triangular fundamental diagrams (which can include the point queue model, cf. Section 9.5.3), the derivative of the travel time on a single link (i, j) with respect to flow entering at time t can be divided into two cases. In the first case, suppose that the vehicle entering link (i, j) at time t exits at $t' = t + \tau_{ij}(t)$, and that the link is demand-constrained at that time (that is, all of the sending flow $S_{ij}(t')$ is able to move). In this case, even if a marginal unit of flow is added at this time, the link will remain demand-constrained, and all of the flow will still be able to move. No additional delay would accrue, so the derivative of the link travel time is

$$\frac{d\tau_{ij}(t)}{dh} = 0. \quad (11.28)$$

In the second case, suppose that at time t' flows leaving link (i, j) are constrained either by the capacity of the link or by the receiving flow of a downstream link. In this case, not all of the flow is able to depart the link, and a queue has formed at the downstream end of (i, j) . The clearance rate for this queue is given by $y_{ij}(t')$, so the incremental delay added by one more vehicle joining the queue is $1/y_{ij}(t')$, and

$$\frac{\tau_{ij}(t)}{dh} = \frac{1}{y_{ij}(t')}. \quad (11.29)$$

(The denominator of this expression cannot be zero, since t' is the time at which a vehicle is leaving the link.)

Therefore, we can calculate the derivative of an entire path's travel time inductively. Assume that $\pi = [r, i_1, i_2, \dots, s]$, and that t_i gives the travel time for arriving at each node i in the path. Then dt_π/dh_π is obtained by summing expressions (11.28) and (11.29) for the uncongested and congested links in this path, taking care to use the correct time indices:

$$\frac{dt_\pi}{dh_\pi} \approx \sum_{(i,j) \in \pi} \frac{1}{y_{ij}(t_j)} [y_{ij}(t_j) < S_{ij}(t_j)], \quad (11.30)$$

where the brackets are an indicator function equal to one if the statement in brackets is true, and zero if false.

We are almost ready to give the formula for (11.27), but we can make one more improvement to the approximation. Two paths may share a certain number of links in common. Define the *divergence node* of two paths to be node

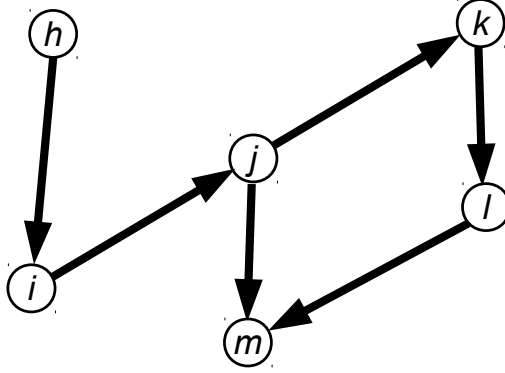


Figure 11.6: Node j is the divergence node of these two paths.

where the paths diverge for the first time. (Figure 11.6). Prior to the divergence node, the two paths include the same links, so there will be no effect of shifting flow from one path to the other. Therefore, the sum in (11.30) need only be taken for links beyond the divergence node. Even if the paths rejoin at a later point, they may do so at different times, so we cannot say that there is no effect of shifting flow between common links downstream of a divergence node. So, if $d(\pi_1, \pi_2)$ is the divergence node, then we have

$$\frac{dt_\pi}{dh_\pi} \approx \sum_{(i,j) \in \pi: (i,j) > d(\pi_1, \pi_2)} \frac{1}{y_{ij}(t_j)} [y_{ij}(t_j) < S_{ij}(t_j)], \quad (11.31)$$

where the “ $>$ ” notation for links indicates links downstream of a node in a path. This expression can then be used in (11.27) and (11.25) to find the approximate amount of flow which needs to be shifted from π_1 to π_2 to equalize their costs.

The procedure for updating H can now be described as follows:

5. **Subproblem:** Perform Newton updates between all paths and the target path found.
 - (a) For each path π with positive flow, let ρ be the least-travel time path connecting the same OD pair and departure time.
 - (b) Calculate Δh using equations (11.31), (11.27), and (11.25), using π as the first path and ρ as the second.
 - (c) If $\Delta h < h_\pi$, then update $h_\rho \leftarrow h_\rho + \Delta h$ and $h_\pi \leftarrow h_\pi - \Delta h$. Otherwise, set $h_\rho \leftarrow h_\rho + h_\pi$ and $h_\pi \leftarrow 0$.

As stated, the path travel times and derivatives are not updated after each shift is performed. Doing so would increase accuracy, but greatly increase computation time since a complete network loading would have to be performed.

11.3 Properties of Dynamic Equilibria

The dynamic traffic assignment problem is much less well-behaved than the static traffic assignment problem, where a unique equilibrium provably exists under natural conditions. This is because the more realistic network loading procedures used in dynamic traffic assignment lack the regularity properties which make static assignment more convenient mathematically. Recall that the dynamic user equilibrium path flows solve the variational inequality

$$\mathcal{N}(\hat{H})(\hat{H} - H) \leq 0 \quad \forall H \in \bar{H}. \quad (11.32)$$

Showing existence of a solution to this variational inequality would typically rely on showing that \mathcal{N} is continuous. However, when queue spillback is modeled, the travel times are discontinuous in the path flow variables. Showing uniqueness of a solution to the variational inequality typically involves showing that \mathcal{N} has some flavor of monotonicity, but common node models can be shown to violate monotonicity even when we restrict attention to networks involving only a single diverge and merge. Thus, the failure of existence or uniqueness results is directly traced to the features that make dynamic traffic assignment a more realistic model. As an analyst, you should be aware of this tradeoff.

This section catalogs a few examples of dynamic traffic assignment problems highlighting some unusual or counterintuitive results. We begin with an example where the choice of node model results in no dynamic user equilibrium solution existing. Two examples of equilibrium nonuniqueness are given, first where multiple user equilibrium solutions exist, and second where literally *all* feasible path flow matrices are user equilibrium solutions, even though they vary widely in total system travel time. We conclude with a dynamic equivalent to the Braess paradox which was developed by Daganzo, in which increasing the capacity on a link can make network-wide conditions arbitrarily worse, because of queue spillback.

11.3.1 Existence: competition at merges

The network in Figure 11.7 has two origin-destination pairs (A to B, and C to D). Aside from these four centroids, there are two nodes representing intersections. Turns are not allowed at these intersections, so $\Xi(1) = \{[A, 1, 2], [C, 1, D]\}$ and $\Xi(2) = \{[1, 2, B], [C, 2, D]\}$. As a result, each OD pair has only two routes available to it: $[A, B]$ and $[A, 1, 2, B]$ for A to B, and $[C, 1, D]$ and $[C, 2, D]$ for C to D. For the sake of convenience, call these four routes “top,” “bottom,” “left,” and “right,” respectively. The figure shows the *travel times* with each link. Two links in the network also carry a toll, expressed in the same units as travel time, and drivers choose routes to minimize the sum of travel time and toll. Capacities and jam densities are large enough that congestion will never arise, and all links will remain at free flow.

Nodes 1 and 2 have a distinctive node model, representing absolute priority of one approach over the other. For Node 1, all flow on the movement $[A, 1, 2]$

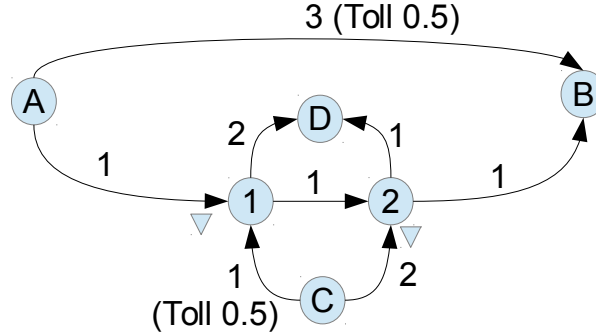


Figure 11.7: Network for example with no equilibrium solution. Link travel times shown.

must yield to flow on $[C, 1, D]$. This can be expressed with the following node model:

$$y_{C1D} = \min \{S_{C1}, R_{1D}\} \quad (11.33)$$

$$y_{A12} = \begin{cases} 0 & \text{if } y_{C1D} > 0 \\ \min \{S_{A1}, R_{12}\} & \text{otherwise} \end{cases} \quad (11.34)$$

Likewise, for Node 2, all flow on the movement $[C, 2, D]$ must yield to flow on $[1, 2, B]$. That is, traffic from C to D has priority at node 1; and traffic from A to B has priority at node 2. These relationships are indicated on the figure with triangles next to the approach that must yield.

Each OD pair has one unit of demand, that departs during the same time interval. We now show that there is no assignment of demand to paths that satisfies the equilibrium principle, requiring any used path to have minimal travel time. We begin first by showing that any equilibrium solution cannot split demand among multiple paths; each OD pair must place all of its flow on just one path or the other. With the given demand, the cost on the bottom path is either 3 (if there is no flow on the left path from C to D) or 4 (if there is), in either case, the cost is different from that on the top path (3.5). All demand will thus use either the top path or the bottom path, whichever is lower. Similarly, for the OD pair from C to D, the cost on the right path is either 3 or 4, whereas that on the left is either 3.5 or 4.5. Both paths cannot have equal travel time, so at most one path can be used.

Thus, there are only four solutions that can possibly be equilibria: all flow from A to B must either be on the top path or the bottom path; and all flow from C to D must either be on the left path or the right path. First consider the case when all flow from A to B is on the top path. The left path would then have a cost of 3.5, and the right path a cost of 3. So all flow from A to B must be on the right path. But then the travel times on the top and bottom paths are

3.5 and 3, respectively, so this solution cannot be an equilibrium (the travelers from A to B would prefer the bottom path, contradicting our assumption).

So now consider the other possibility, when all flow from A to B is on the bottom path. The travel times on the left and right paths are now 3.5 and 4, so all travelers from C to D would pick the left path. But then the travel times on the top and bottom paths are 3.5 and 4, respectively, so this solution is not an equilibrium either.

Therefore, there is *no* assignment of vehicles to paths that satisfies the principle of user equilibrium. This network is essentially the Ginger/Harold “matching pennies” game from Section 1.3 encoded into dynamic traffic assignment. Game theorists often resolve this type of situation by proposing a *mixed-strategy equilibrium*, in which the players randomize their actions. One can show that a mixed-strategy equilibrium exists in this network (Exercise 8), but this equilibrium concept is usually not applied to transportation planning. This is for both computational reasons (when there are many players, as in practical planning networks, computing mixed-strategy equilibria is hard) and for modeling reasons (typically travelers do not randomize their paths with the intent of “outsmarting” other travelers).

Mathematically, the reason no equilibrium exists is because the node model specified by (11.33) and (11.34) is not continuous in the sending and receiving flows.

This is similar to static traffic assignment, where equilibrium existence cannot be guaranteed if the link performance functions are not continuous. In dynamic traffic assignment, when building node models that capture absolute priority (as at yields, two-way stops, or signals with permitted turns), ensuring continuity is difficult. In part, this is the motivation for using ratios of α values in Chapter 9 to reflect priorities at merges and general intersections.

But even if the link and node models are chosen to be continuous, dynamic equilibrium need not exist. Repeating the arguments made in Chapter 5.2 would also require the travel time calculations to be continuous in the link cumulative entrances and exits N^\uparrow and N^\downarrow . The use of a minimum to disambiguate multiple possible values of travel times in Equation (11.3) can mean that small changes in a link’s cumulative entries or exits could cause a large change in the travel time (imagine how the length of the dashed line in Figure 11.3 would change if one of the corner points were shifted slightly). Continuity could be assured only if the cumulative counts were *strictly* increasing, that is, if vehicles were constantly flowing into and out of each link. Therefore, although one can construct dynamic traffic assignment models which guarantee existence of equilibria, these require quite strong and significant restrictions on the network loading and travel time calculation procedures.

11.3.2 Uniqueness: Nie’s merge

The network in Figure 11.8 consists of a single diverge and merge. One can visualize this network as a stylized version of a freeway lane drop (Figure 11.9), where the reduction from two lanes to one lane reduces the capacity from 80

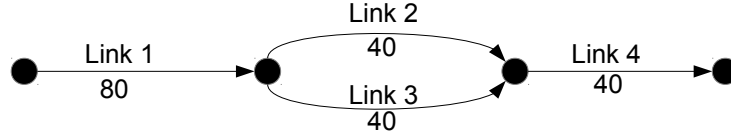


Figure 11.8: Nie's merge network. All capacity values in vehicles per minute.

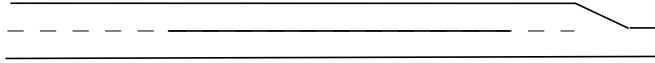


Figure 11.9: A physical interpretation of Nie's merge.

vehicles per minute to 40 vehicles per minute, and where drivers either merge early (choosing the top lane at the diverge point) or late (waiting until the lane drop itself). The inflow rate from upstream is 80 vehicles per minute. Since the capacity of the downstream exit is only 40 vehicles per minute, the excess vehicles will form one or more queues in this network.

To simplify the analysis, assume that the time horizon is short enough that none of these queues will grow to encompass the entire length of the link. With this assumption, queue spillback can be ignored, and we can focus on the issue of route choice. Furthermore, under these assumptions, S_1 is always 80 vehicles per minute, and R_2 , R_3 , and R_4 are always 40 vehicles per minute. We work with a timestep of one minute, and will express all flow quantities in vehicles per minute.²

In this network, there is only one choice of routes (the top or bottom link at the diverge). We will restrict our attention to path flow matrices H where the proportions of vehicles choosing the top and bottom routes are the same for all departure times, and show that there are multiple equilibrium solutions even when limiting our attention to such H matrices. There may be still more equilibria where the proportion of vehicles choosing each path varies over time.

Therefore, the ratio h_{124}/h_{134} is constant for all time intervals, which means that the splitting fractions p_{12} and p_{13} at the diverge point are also constant, and equal to $h_{124}/40$ and $h_{134}/40$, respectively.

Using the diverge and merge models from Section 9.2.3, we can analyze the queue lengths and travel times on each link as a function of these splitting

²This is larger than what is typically used in practice, but simplifies the calculations and does not affect the conclusions of this example.

fractions.

It turns out that three distinct dynamic user equilibria exist:

Equilibrium I: If $p_{12} = 1$ and $p_{13} = 0$, then the diverge formula (9.17) gives the proportion of flow which can move as

$$\phi = \min \left\{ 1, \frac{40}{80}, \frac{40}{0} \right\} = \frac{1}{2}. \quad (11.35)$$

Therefore, the transition flows at each time step are $y_{12} = 40$ and $y_{13} = 0$. Since $y_{12} < S_{12}$, a queue will form on the upstream link, and its sending flow will remain at $S_1 = 80$. Therefore, once the first vehicles reach the merge, we will have $S_2 = 40$ and $S_3 = 0$. Applying these proportions, together with $R_4 = 40$, the merge formula gives $y_{24} = 40$ and $y_{34} = 0$. There will be no queue on link (2,4), so both link (2,4) and link (3,4) are at free-flow. This solution is an equilibrium: the two paths through the network only differ by the choice of link 2 or link 3, and both of these have the same travel time since they are both at free-flow. In physical terms, this corresponds to all drivers choosing to merge early; the queue forms upstream of the point where everyone chooses to merge, and there is no congestion downstream.

Equilibrium II: If $p_{12} = 0$ and $p_{13} = 1$, the solution is exactly symmetric to that of Equilibrium I. A queue will again form on the upstream link, now because all drivers are waiting to take the bottom link. In physical terms, this corresponds to all drivers choosing to merge early, but to the lane which is about to drop. They then all merge back when the lane actually ends. This solution does not seem especially plausible, but it does satisfy the equilibrium condition.

Equilibrium III: If $p_{12} = p_{13} = 1/2$, drivers wish to split equally between the top and bottom links. The proportion of flow which can move at the diverge is

$$\phi = \min \left\{ 1, \frac{40}{40}, \frac{40}{40} \right\} = 1, \quad (11.36)$$

so all vehicles can move and there is no queue at the diverge: $y_{12} = y_{13} = 40$. Once these vehicles reach the merge, we will have $S_2 = S_3 = 40$ and $R_4 = 40$. The merge formula (9.12) then gives $y_{24} = y_{34} = 20$, so queues will form on both merge links. However, since the inflow and outflow rates of links 2 and 3 are identical, the queues will have identical lengths, and so the travel times on these links will again be identical. Therefore, this solution satisfies the principle of dynamic user equilibrium. In physical terms, this is the case when no drivers change lanes until the lane actually ends, and queueing occurs at the merge point.

By examining the intermediate solutions, we can calculate the travel times on both paths for different values of the fraction p_{12} . (We do not need to state the

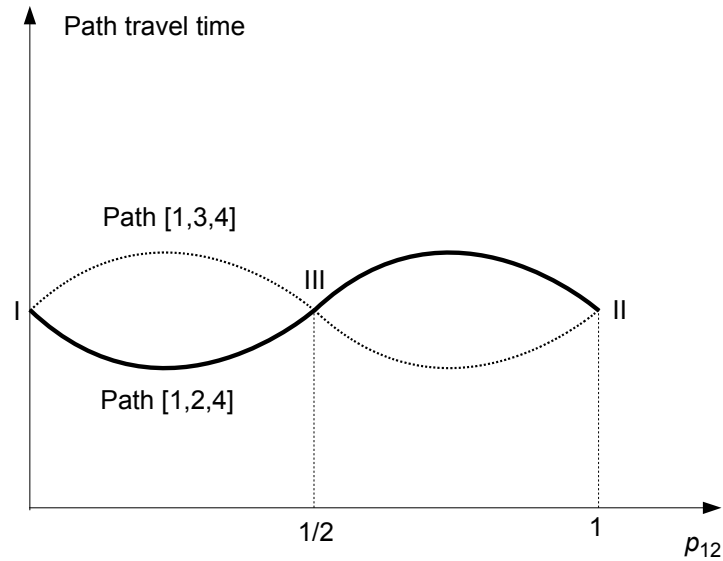


Figure 11.10: Travel time on top and bottom paths as a function of splitting fraction, with three equilibrium solutions marked.

values of the other fraction, since $p_{13} = 1 - p_{12}$.) This is shown in Figure 11.10, and the three equilibria correspond to the crossing points of the paths. Note that all three of the equilibria share the same equilibrium travel time. Although the queues in Equilibrium III are only half as long as those in Equilibria I and II, being split between two links, the outflow rates of these queues are also only half as great (20 veh/min instead of 40 veh/min).

This example shows that the dynamic user equilibrium solution is not unique, even in an extremely simple network. This nonuniqueness has practical consequences. The effect of a potential improvement to link 2 or 3 will depend crucially on the number of travelers on the link, which varies widely in all three equilibria. One criterion for distinguishing which of these equilibria is more likely is *stability*, which explores what would happen if the equilibria were slightly perturbed. As shown in Figure 11.10, if one begins at Equilibrium I and perturbs p_{13} to a small value (reducing p_{12} by the same amount), we see that the travel time on the top path increases, whereas the travel time on the bottom path decreases. It may seem odd that increasing flow on the bottom path decreases its travel time — what is happening is that congestion at the diverge decreases (lowering the travel time of both paths), but congestion forms on the top link at the merge (increasing the travel time just of the top path). Superimposing these effects produces the result in Figure 11.10. As a result, travelers will switch from the (slower) top path to the (faster) bottom one, moving us even further away from Equilibrium I. Therefore, Equilibrium I is not stable. The same analysis holds for Equilibrium II.

Equilibrium III, on the other hand, is stable. If a few travelers switch from the top to the bottom path, the travel time on the top path decreases and that on the bottom path increases. Therefore, travelers will tend to switch back to the top path, restoring the equilibrium. The same holds if travelers switch from the bottom path to the top path. This gives us reason to believe that Equilibrium III is more likely to occur in practice than Equilibrium I or II. This type of analysis is much more complicated in larger networks, and for the most part is completely unexplored. Coming to a better understanding of the implications of nonuniqueness in large networks, as well as techniques for addressing this, is an important research question.

11.3.3 Nie's merge redux

Modifying the merge network from the previous section, we can produce an even more pathological result. We now eliminate the lane drop entirely, but preserve the diverge/merge network, by setting the capacity of the downstream link to 80 vehicles per minute, the same as the upstream link. This might represent a roadway section where lane changing is prohibited, perhaps in a work zone or with a high-occupancy/toll lane. Now, at the merge point $S_2 + S_3 \leq R_4$ no matter what the flow pattern is on the network. This means that the merge will always be freely-flowing, and there will be no queues at the merge. Links 2, 3, and 4 will be at free-flow regardless of the path flows H .

However, queues can still occur at the *diverge*, where the analysis is the same as before. If $p_{12} = p_{13} = 1/2$, no queues will form at the diverge point, whereas for any other values of p_{12} and p_{13} formula (11.35) will predict queues on the upstream link, reaching their maximum length if $p_{12} = 1$ or $p_{13} = 1$. However, *all of these solutions satisfy the principle of equilibrium* because the only delay occurs on the upstream link, which is common to both paths. No matter what the values of p are, there is zero relative gap or average excess cost, since no traveler can reduce his or her travel time by choosing a different route. Choosing a different route could influence the travel time of those further upstream, but under the assumption that drivers act only to minimize their own travel time, this influence on other travelers is of no concern.

Figure 11.11 shows the queue lengths and total system travel time as p varies. The system-optimal solution is unique: if $p_{12} = p_{13} = 1/2$ there are no queues in the system and all vehicles experience free-flow travel times. For any other values of p_{12} and p_{13} , a queue will form at the diverge and some delay is experienced. Yet *all* possible p values satisfy the equilibrium principle, since the delay is upstream of the diverge and drivers cannot choose another route to minimize their own travel time. Furthermore, as the time horizon grows longer, the difference in total travel time between the worst of the equilibria (either $p_{12} = 0$ or $p_{12} = 1$) and the system optimum solution can grow arbitrarily large. This is in contrast with the bounded “price of anarchy” which can often be found for static traffic assignment (cf. Section 5.4.3).

This effect less of a practical concern than the original version of Nie's merge — there are good behavioral reasons to doubt that a significant imbalance of

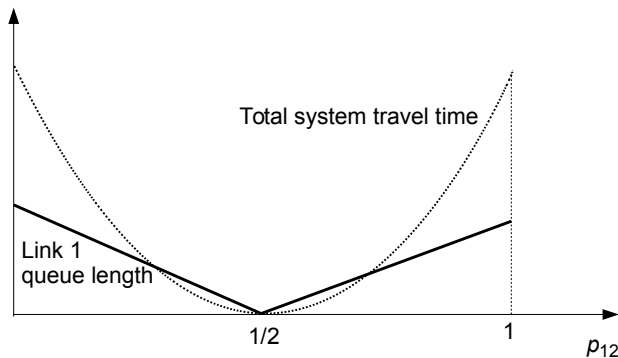


Figure 11.11: Queue length and total travel time for different splitting proportions p in the modified Nie’s merge network. Note that all p values represent equilibria.

travelers will choose one alternative over another identical one. One argument is from entropy principles (Section 5.2.2): if travelers have the same behavior assumption, it is unlikely they would all choose one route over another with equal travel time. Furthermore, the assumption of a triangular fundamental diagram implies that travel speeds remain at free-flow for all subcritical densities. In practice the speed will drop slightly due to variations in preferred speeds and difficulties in overtaking at higher density, so drivers would likely prefer the route chosen by fewer travelers.

However, from the standpoint of *modeling* the fact that all feasible solutions are equilibria poses significant challenges.

Literally any solution will have zero gap, and if an all-or-nothing assignment is chosen as the initial solution (as is sometimes done in implementations), dynamic traffic assignment software will report that a perfect equilibrium has been reached.

This example shows that initial solutions for dynamic traffic assignment should be carefully chosen, perhaps by spreading vehicles over multiple paths, or breaking ties stochastically in shortest path algorithms to avoid assigning all vehicles to the same path in the first all-or-nothing assignment.

11.3.4 Efficiency: Daganzo’s paradox

The previous section showed that the worst user equilibrium solution can be arbitrarily worse than the system optimal solution, in terms of total travel time. However, there was still one user equilibrium solution that was also system optimal (the case where $p_{12} = p_{13} = 1/2$). This need not be in the case. Here we present an example where the *only* user equilibrium solution can be arbitrarily worse than system optimum.

Furthermore, it is “paradoxical” in the sense that increasing the capacity of the only congested link on the network can make the problem worse, and that

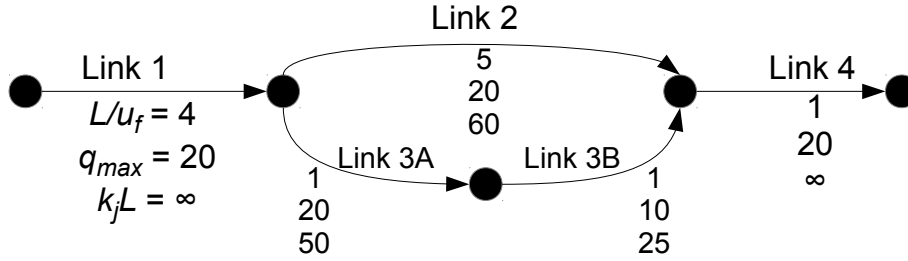


Figure 11.12: Network for Daganzo's paradox.

reducing the capacity on this link can improve system conditions! In this sense, it is a dynamic equivalent of the Braess paradox from Section 4.3.

Many have criticized the Braess paradox on the grounds that the link performance functions used in static assignment are unrealistic. In the example shown below, queue spillback (a feature unique to dynamic network loading) is actually the critical factor in the paradox.

See the network in Figure 11.12, where time is measured in units of time steps Δt . Like the networks in the two previous sections, it consists of a single merge and diverge. However, the free-flow travel times and capacities on the top and bottom links are now different: the top route is longer, but has a higher capacity, while the bottom route is shorter at free-flow, but has a bottleneck limiting the throughput on this route — link 3B has only half the capacity of 3A. We will use the spatial queue model of Section 9.1.3 to propagate traffic flow, although the same results would be obtained with an LWR-based model or anything else which captures queue spillback and delay. The input demand is constant, at 20 vehicles per time step.

The capacity on the top route is high enough to accommodate all of the demand; if all of this demand were to be assigned to this route, the travel time would be 10 minutes per vehicle. Assigning all vehicles to the top route is neither the user equilibrium nor the system optimum solution, but it does give an upper bound on the average vehicle travel time in the system optimal assignment — it is possible to do better than this if we assign some vehicles to the bottom, shorter route.

To derive the user equilibrium solution, notice that initially all vehicles will choose the bottom path, since it has the lower travel time at free flow. A queue will start forming on link 3A, since the output capacity is only 10 vehicles per time step (because of the series node model from Section 9.2.1, and the capacity of link 3B) and vehicles are entering at double this rate. As the queue grows, the travel time on link 3A will increase as well. With the spatial queue model and these inflow and outflow rates, you can show that the travel time for the n -th vehicle entering the link is

$$1 + \frac{n}{q_{in}} \left(\frac{q_{in}}{q_{out}} - 1 \right) = 1 + \frac{n}{20} \tag{11.37}$$

as long as the queue has not spilled back (see Exercise 9.3).

Based on equation (11.37), when the 60th vehicle enters the link, it will spend four time units on link 3A, and thus its travel time across the entire bottom path would be the same as if it had chosen the top path. At this point there are 40 vehicles on link 3A, as can be seen by drawing the N^\uparrow and N^\downarrow curves for this link. From this point on, vehicles will split between the top and bottom paths to maintain equal travel times.

So far, so good; the first 60 vehicles that enter the network have a travel time of less than 10 minutes, and all the rest have a travel time of exactly 10 minutes. Now see what happens if we increase the capacity on link 3B, with the intent of alleviating congestion by improving the bottleneck capacity. If the capacity on 3B increases from 10 to 12, the story stays the same, except that it is the first 90 vehicles that have a travel time of less than 10 minutes. We can see this by setting (11.37) equal to four (the time needed to equalize travel times on the top and bottom paths), but with $q_{in} = 12$ instead of 10, and solving for n . Network conditions indeed have improved. But if the capacity increases still further, to 15, then equation (11.37) tells us that it is only after 180 vehicles have entered the system that travelers would start splitting between the top and bottom links. By tracing the N^\uparrow and N^\downarrow curves, at this point in time 120 vehicles will have exited link 3A, meaning the queue length would be 60 vehicles. **But the jam density of the bottom link only allows it to hold 50 vehicles.** The queue will thus spill upstream of the diverge node, and in this scenario, *no* vehicles will opt to take the top path. By the time a driver reaches the diverge point, the number of vehicles on the bottom link is 50, giving a travel time on 3A of $3\frac{1}{3}$ minutes, and a total travel time of $9\frac{1}{3}$ minutes from origin to destination. This is less than the travel time on the top path, so drivers prefer waiting in the queue to taking the bypass route.

As a result, *all* drivers will choose the bottom path, and the queue on the origin centroid connector will grow without bound, as will the travel times experienced by vehicles entering the network later and later. By increasing the length of time vehicles enter the network at this rate, we can make the delays as large as we like.

We thus see that even with dynamic network loading, increasing the capacity on the only bottleneck link can make average travel times worse — and in fact arbitrarily worse than the system optimum solution, where the average travel time is less than 10 minutes. The reason for this phenomenon is the interaction between queue spillback and selfish route choice: in the latter scenario it would be better for the system for some drivers to choose the top route, even though it would worsen their individual travel time.

Of course, if the capacity on 3B were increased even further, all the way to 20 vehicles per time step, delays would drop again since there would be no bottleneck at all. Exercise 11 asks you to plot the average vehicle delay in this network as the capacity on link 3B varies from 0 to 25 vehicles per time step.

11.3.5 Conclusion

The purpose of these examples is to show that dynamic user equilibrium is complex. Guaranteeing existence or uniqueness of dynamic equilibrium requires making strong assumptions on traffic flow propagation. However, for some practical applications, using a more realistic traffic flow model is more important than mathematical properties of the resulting model. Many people find comfort in the fact that we can at best solve for an equilibrium approximately, and thus dismiss the question of whether an equilibrium “truly” exists as akin to asking how many angels can dance on the head of a pin.

We emphasize that existence and uniqueness are not simply mathematical abstractions, and that they have significant implications for practice: if an equilibrium does not exist, should we really be ranking projects based on equilibrium analysis? If multiple equilibria exist, what should we plan for? Can we even find them all? At the same time, we acknowledge that using static equilibrium to sidestep these difficulties is often unacceptable. For many applications, the assumptions in link performance functions are simply too unrealistic. Such is the nature of mathematical modeling in engineering practice. No tool is right for every task, but rather experienced practitioners know how to match the available tools to the job at hand. By reading this book, we hope that you have gained the insight to understand the advantages and disadvantages of both static and dynamic traffic assignment models, and to make educated decisions about the right tool for a particular project.

Finally, more research is needed on these topics, understanding the significance of equilibrium nonexistence and nonuniqueness, particularly in practical networks and not just in “toy” networks such as the ones in this section. As researchers, we would be delighted for you to contribute to work in this field.

11.4 Historical Notes and Further Reading

(These sections are incomplete in this beta version of the text, and will be substantially expanded in the complete first edition.)

Formulating dynamic traffic assignment models is more difficult than doing so for static traffic assignment. For this reason, this chapter has eschewed highly detailed mathematical formulations of the equilibrium principle in favor of simply expressing the dynamic user equilibrium principle and proposing heuristics.

It is particularly difficult to formulate this problem if we seek a user equilibrium, rather than a system optimum, and if there are multiple destinations. For this reason, the first DTA models (Merchant and Nemhauser, 1978a,b) were restricted to a single destination and a system optimum was sought. These authors used an “exit function” link model which is no longer common; using more recent link models, it is possible to find the system-optimum solution with a single destination by solving a linear program (Ziliaskopoulos, 2000). When there are multiple destinations, ensuring that flow to one destination does not over-

take that headed to another imposes nonconvexity on the feasible flows (Carey, 1992).

For this reason, variational inequality formulations are more common than mathematical optimization. Examples of these include Friesz et al. (1993), Wie et al. (1995), and Chen and Hsueh (1998). Optimal control approaches have also been proposed (Friesz et al., 1989; Ran et al., 1993), as have formulations as a nonlinear complementarity problem (Ban et al., 2012). Fixed point approaches are also common — Bar-Gera (2005) and Bellei et al. (2005) are just two examples — but as with the static assignment problem, are more useful for specifying the problem than for solving it.

Using link or path flows as the main decision variable is the most intuitive choice, and therefore the most common in the literature. However, the use of splitting proportions is becoming more common (Nezamuddin and Boyles, 2015; Long et al., 2012; Gentile, 2016).

For the convex combinations and simplicial decomposition algorithms, refer to the references in Section 6.6. For gradient projection as specialized to dynamic network loading, see Nezamuddin and Boyles (2015) and Gentile (2016).

The equilibrium existence counterexample in Section 11.3.1 is from Waller (2006). The uniqueness counterexample in Section 11.3.2 is from Nie (2010b), and its special case in Section 11.3.3 is from Boyles et al. (2013). The efficiency counterexample in Section 11.3.4 is from Daganzo (1998). One consequence of these counterexamples is that queue spillback significantly complicates the finding and interpretation of dynamic user equilibria; see also the discussion in Boyles and Ruiz Juri (2019).

11.5 Exercises

- [37] (*Equivalence of link-based and path-based flow representations*). Given splitting proportions $\alpha_{hij,s}^t$ for each destination s , time interval t , and turning movement $[h, i, j]$, show how “equivalent” path flow values h_t^π can be found. Then, if given path flows h_t^π , show how “equivalent” $\alpha_{hij,s}^t$ values can be found. (“Equivalent” means that the link cumulative counts N^\uparrow and N^\downarrow would be the same for all time steps after performing network loading, possibly with a small error due to time discretization that would shrink to zero as $\Delta t \rightarrow 0$.)
- [37] Consider the four-link network shown in Figure 11.13, and perform network loading using the cell transmission model. (Each link is one cell long.) During the first time interval, 10 vehicles enter Link 1 on the top path; during the second time interval, 5 vehicles enter Link 1 on the top path and 5 on the bottom path; and during the third time interval, 10 vehicles enter Link 1 on the bottom path. No other vehicles enter the network.

Interpolating as necessary, what time do the first and last vehicles on the top path exit cell 4? the first and last vehicles on the bottom path? What

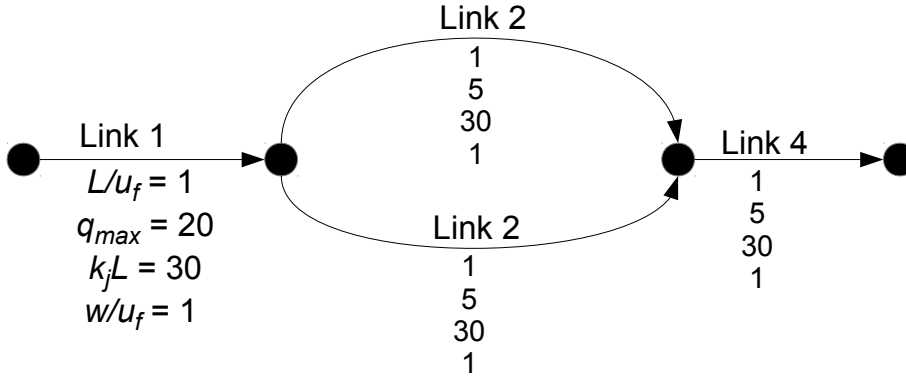


Figure 11.13: Network for Exercise 2.

is the derivative of the travel time on the top path for a vehicle leaving at the start of interval 2?

- [12] In the course of the convex combinations algorithm, assume that the H and H^* matrices are as below, and $\lambda = 1/3$. What is the new H matrix?

$$H = \begin{bmatrix} 6 & 12 \\ 30 & 24 \end{bmatrix} \quad H^* = \begin{bmatrix} 18 & 0 \\ 0 & 54 \end{bmatrix} \quad (11.38)$$

- [23] Consider a network with only one origin-destination pair connected by four paths, with three departure time intervals. Departure times are fixed. At some point in the simplicial decomposition algorithm, \mathcal{H} contains the following three matrices:

$$\begin{bmatrix} 20 & 0 & 0 & 0 \\ 0 & 10 & 0 & 0 \\ 0 & 0 & 30 & 0 \end{bmatrix} \begin{bmatrix} 0 & 20 & 0 & 0 \\ 0 & 10 & 0 & 0 \\ 0 & 30 & 0 & 0 \end{bmatrix} \begin{bmatrix} 20 & 0 & 0 & 0 \\ 0 & 0 & 0 & 10 \\ 0 & 0 & 30 & 0 \end{bmatrix} \quad (11.39)$$

and the current path flow and travel time matrices are:

$$H = \begin{bmatrix} 14 & 6 & 0 & 0 \\ 0 & 8 & 0 & 2 \\ 0 & 9 & 21 & 0 \end{bmatrix} \quad T(H) = \begin{bmatrix} 20 & 20 & 24 & 27 \\ 30 & 34 & 37 & 40 \\ 44 & 35 & 36 & 40 \end{bmatrix} \quad (11.40)$$

What are the unrestricted and restricted average excess costs of the current solution? What is the matrix ΔH ?

- [35] Assume that there is a single OD pair, two paths, and three departure times; 15 vehicles depart during the first, 10 during the second, and 5 during the third. Let $H_{\pi t}$ denote the number of vehicles departing on path π at time t , and that the path travel times are related to the path

flows by the following equations:³

$$\begin{aligned} T_{11} &= 15 \\ T_{21} &= 13 + 3H_{11} + H_{21} \\ T_{12} &= 15 + H_{11}^2 \\ T_{22} &= 13 + H_{11} + 3H_{12} + H_{21}^2 + H_{22} \\ T_{13} &= 15 + H_{12}^2 + \frac{1}{2}H_{11}^2 \\ T_{23} &= 13 + H_{12} + 3H_{13} + H_{22}^2 + H_{23} \end{aligned}$$

Find the path flows obtained from three iterations of the convex combinations method with step sizes $\lambda_1 = 1/2$, $\lambda_2 = 1/4$, and $\lambda_3 = 1/6$ (so you should find a total of four H^* matrices, counting the initial matrix, and take three weighted averages). Your initial matrix should be the shortest paths with zero flow. What is the resulting AEC? *Break any ties in favor of path 1.*

6. [35] Using the same network and demand as in Exercise 5, now assume that you are solving the same problem with simplicial decomposition, and at some stage \mathcal{H} contains the following two matrices:

$$\begin{bmatrix} 15 & 0 \\ 0 & 10 \\ 5 & 0 \end{bmatrix} \quad \begin{bmatrix} 15 & 0 \\ 10 & 0 \\ 0 & 5 \end{bmatrix}$$

and that the current solution is

$$H = \begin{bmatrix} 15 & 0 \\ 4 & 6 \\ 3 & 2 \end{bmatrix}$$

- What is the average excess cost of the current solution?
 - What is the restricted average excess cost?
 - What is the search direction from H ?
 - Give the updated H matrix and new restricted average excess cost after taking a step with $\mu = 0.1$.
 - If we terminate the subproblem and return to the master algorithm, what matrix (if any) do we add to \mathcal{H} ?
7. [35] Again using the network and demand from Exercise 5, now start with the initial solution

$$H = \begin{bmatrix} 10 & 5 \\ 5 & 5 \\ 0 & 5 \end{bmatrix}$$

³In practice these would come from performing network loading and calculating path travel times as described in this chapter; these functions are provided here for your convenience.

Using the gradient projection method, identify a new H and T matrix. Try both exact and quasi-Newton steps, and see which gives the greater reduction in AEC .

8. [46] Find the mixed-strategy equilibrium in the network in Figure 11.7. (For each of the two vehicles, indicate the probability it will choose each of the two paths available to it; these probabilities should be such that the *expected* travel times are the same for both options.)
9. [35] Consider a network with one origin, one destination, two possible departure time intervals (1 and 2), and two paths (A and B). A total of 16 vehicles depart during time interval 1, and 4 depart during time interval 2. Let t_τ^π refer to the travel time on the π -th path when departing at the τ -th time interval, with h_τ^π defined similarly for path flows. Suppose the travel times are related to the path flows as follows:

$$\begin{aligned}t_1^A &= 0.5h_1^A + 5h_2^A + 6 \\t_1^B &= 0.5h_1^B + 3h_2^B + 10 \\t_2^A &= 0.3h_1^A + 0.6h_2^A + 0.8 \\t_2^B &= 0.2h_1^B + 0.4h_2^B + 2\end{aligned}$$

and consider the following four path flow matrices:

$$H_1 = \begin{bmatrix} 0 & 16 \\ 4 & 0 \end{bmatrix} \quad H_2 = \begin{bmatrix} 12 & 4 \\ 2 & 2 \end{bmatrix} \quad H_3 = \begin{bmatrix} 8 & 8 \\ 2 & 2 \end{bmatrix} \quad H_4 = \begin{bmatrix} 16 & 0 \\ 0 & 4 \end{bmatrix}$$

(M. Netter, 1972)⁴

- (a) Which of the above path flow matrices satisfy the principle of user equilibrium?
- (b) Which of the above path flow matrices represent *efficient* equilibria?
- (c) Perturb H_3 in the following way: adjust h_2^A from 2 to 2.1 (so h_2^B becomes 1.9), changing all of the travel times. Then, adjust h_1^A and h_1^B until $t_1^A = t_1^B$ (restoring equilibrium for the first departure time). Then, adjust h_2^A and h_2^B until $t_2^A = t_2^B$ (restoring equilibrium for the second departure time), and so on, alternating between the two departure times, until you reach a new equilibrium solution. Which equilibrium solution do you end up at? Is H_3 stable?
10. [33] Consider an instance of the Daganzo paradox network (Figure 11.12), but with travel times $\tau_2 = 10$, $\tau_{3A} = 4$ and $\tau_{3B} = 1$, capacities of $c_{3B} = 10$ and $c_1 = c_2 = c_{3A} = c_4 = 50$ vehicles per time step, and an inflow rate of $Q = 50$ vehicles per time step. Assume that the maximum number of vehicles which can fit on link 3A is $(k_j L)_{3A} = 300$ vehicles.

⁴If you are wondering how the travel time on a path can be influenced by drivers leaving at a later time, recall that FIFO violations can occur due to phenomena such as express lanes opening, allowing “later” vehicles to overtake “earlier” ones and delay them in the network.

- (a) How many vehicles should be on link 3A at any time to maintain equilibrium?
 - (b) Does this queue fit on the link?
 - (c) The capacity on 3A is now increased to $c_{3A} = 25$. At the new equilibrium, have drivers' travel times increased, decreased, or stayed the same?
 - (d) The capacity is now increased to $c_{3A} = 50$. At the new equilibrium, have drivers' travel times increased, decreased, or stayed the same (relative to $c_{3A} = 20$)?
11. [55] In the network of Figure 11.12, find the equilibrium solution in terms of the capacity on link 3B. Plot the average travel time as this capacity ranges between 0 and 25 vehicles per time step. Assume a time horizon of 20 time steps. (You may do this either by deriving a closed-form expression for the average travel time, or by using one of the algorithms in this chapter to solve for the approximate equilibrium solution.)

Appendix A

Mathematical Concepts

This appendix reviews the mathematical background needed to understand the mathematical formulations in this book. This involves understanding certain properties of matrices, sets, and functions. The descriptions are relatively terse, because there are countless textbooks and online resources with additional explanations and examples. It is also a very incomplete overview, only covering concepts which will be used later in the book and largely assuming that the reader has seen this material at some point in the past. The section also collects a number of useful results related to matrices, sets, and functions. The proofs of most of these are left as exercises.

A.1 Indices and Summation

We commonly work with attributes of links or nodes, such as the total number of trips starting or ending at a node, or the travel time on a link. In such cases, it makes sense to use the same variable letter for the number of trips produced at a node (say P), and to indicate the productions at a specific node with a subscript or superscript (P_1 or P^3 .) Unfortunately, the superscript notation can be ambiguous, since the superscript in P^3 can be interpreted both as an exponent, and as the index referring to a particular node. In most cases it will be clear which meaning is intended, and parentheses can be used in cases which are unclear: $(P^3)^2$ is the square of the productions at node 3. To avoid confusion, subscripts are generally preferred to superscripts, but superscripts can make the notation more compact when there are multiple indices. An example is δ_{ij}^π from Section 1.5, where the variable δ is indexed both by a link (i, j) and a path π . We have striven to be consistent with which indices appear as subscripts and which appear as superscripts. However, in a few occasions, being absolutely rigid in this regard would create cluttered formulas. In such cases, we have sacrificed rigor for readability, and hope that you are not thrown off by an index typically appearing as a subscript in the superscript position, and vice versa.

Subscript or superscript indices also allow formulas to be written more com-

pactly. A common example is the summation notation, such as

$$\sum_{i=1}^5 P_i = P_1 + P_2 + P_3 + P_4 + P_5. \quad (\text{A.1})$$

The left-hand side of Equation (A.1) is used as shorthand for the right-hand side. More formally, the left-hand side instructs us to choose all the values of i between 1 and 5 (inclusive); collect the terms P_i for all of these values of i ; and to add them together. A variant of this notation is

$$\sum_{i \in N} P_i, \quad (\text{A.2})$$

which expresses the sum of the productions from all nodes in the set N . Here i ranges over all elements in the set N , rather than between two numbers as in (A.1). We can also add conditions to this range by using a colon. If we only wanted to sum over nodes whose products are less than, say, 500, we can write

$$\sum_{i \in N: P_i < 500} P_i. \quad (\text{A.3})$$

When it is exceptionally clear what values or set i is ranging over, we can simply write

$$\sum_i P_i, \quad (\text{A.4})$$

but this “abbreviated” form should be used sparingly, and avoided if there is any ambiguity at all as to what values i should take in the sum.

When there are multiple indices, a double summation can be used:

$$\sum_{i=1}^3 \sum_{j=2}^3 x_{ij} = x_{12} + x_{22} + x_{32} + x_{13} + x_{23} + x_{33}. \quad (\text{A.5})$$

You can think of a double sum either as a “nested” sum:

$$\sum_{i=1}^3 \sum_{j=2}^3 x_{ij} = \sum_{i=1}^3 \left(\sum_{j=2}^3 x_{ij} \right) = \sum_{i=1}^3 (x_{i2} + x_{i3}), \quad (\text{A.6})$$

which expands to the same thing as the right-hand side of (A.5), or as summing over all combinations of i and j such that i is between 1 and 3, and j is between 2 and 3. Triple sums, quadruple sums, and so forth behave in exactly the same way, and are common when there are many indices.

The summation index is often called a *dummy variable*, because the indices do not have an absolute meaning. Rather, they are only important insofar as they point to the correct numbers to add up. For instance, $\sum_{i=1}^5 P_i$ and $\sum_{i=0}^4 P_{i+1}$ are exactly the same, because both expressions have you add up P_1

through P_5 . Likewise, $\sum_{i=1}^5 P_i$ and $\sum_{j=1}^5 P_j$ are exactly the same, the fact that we are counting from 1 to 5 using the variable j instead of i is of no consequence.

Related to this, it is *wrong* to refer to a summation index outside of the sum itself. A formula such as $x_i + \sum_{i=1}^5 y_i$ is incorrect. For the formula to make sense, x_i needs to refer to one specific value of i . But using i as the index in the sum $\sum_{i=1}^5$ means that i must range over all the values between 1 and 5. Does y_i refer to the index of summation (which ranges from 1 to 5), or to the one specific value of i used outside the sum? If you want to refer to a specific node, as well as to index over all nodes, you can add a prime to one of them, as in

$$x_i + \sum_{i'=1}^5 y_{i'}, \quad (\text{A.7})$$

or you can either use a different letter altogether, as in

$$x_i + \sum_{j=1}^5 y_j. \quad (\text{A.8})$$

Both conventions are common.

In transportation network analysis, we frequently have to sum over all of the links which are in the forward or reverse star of a node (Section 2.3). If the link flows are denoted by a variable x , the total flow entering node i is the sum of the link flows in its reverse star. This can be written in several ways:

$$\sum_{(h,i) \in \Gamma^{-1}(i)} x_{hi} \qquad \sum_{(h,i) \in A} x_{hi} \quad (\text{A.9})$$

The notation on the right can be a bit confusing at first glance, since it looks like we are summing the flow of *every* link (the whole set A), rather than just the links entering node i . The critical point is that in this formula, i refers to one specific node which was previously chosen and defined outside of the summation. *The only variable we are summing over in Equation (A.9) is h .* In this light, i is fixed, and the right-most sum is over all the values of h such that (h, i) is a valid link (that is, $(h, i) \in A$). These are exactly the links which form the reverse star of i .

Almost all of the sums we will see in this book involve only a finite number of terms. These sums are much easier to work with than infinite sums, and have the useful properties listed below. (Some of these properties do not always apply to sums involving infinitely many terms.)

1. You can factor constants out of a sum:

$$\sum_i c x_i = c \sum_i x_i, \quad (\text{A.10})$$

no matter what values i ranges over. This follows from the distributive property for sums: $a(b + c) = ab + ac$. (A “constant” here is any term which does not depend on i .)

2. If what you are summing is itself a sum, you can split them up:

$$\sum_i (x_i + y_i) = \sum_i x_i + \sum_i y_i. \quad (\text{A.11})$$

This follows from the commutative property for sums: $a + b = b + a$, so you can rearrange the order in which you add up the terms.

3. You can exchange the order of a double summation:

$$\sum_i \sum_j x_{ij} = \sum_j \sum_i x_{ij} \quad (\text{A.12})$$

no matter what values i and j range over. This also follows from the commutative property.

None of these properties are anything new; they simply formalize how we can operate with the \sum notation using the basic properties of addition. These properties can also be combined. It is common to exchange the order of a double sum, and then factor out a term, to perform a simplification:

$$\sum_i \sum_j c_j x_{ij} = \sum_j \sum_i c_j x_{ij} = \sum_j \left(c_j \sum_i x_{ij} \right). \quad (\text{A.13})$$

The last step is permissible because c_j is a “constant” relative to a sum over i . We could not do this step at the beginning, because c_j is not a constant relative to a sum over j . This kind of manipulation is helpful if, say, $\sum_i x_{ij}$ has a convenient form or a previously-calculated value.

Occasionally it is useful to refer to products over an index. In analogy with the summation notation \sum used for sums over an index, products over an index are written with the notation \prod . As an example

$$\prod_{i=1}^5 P_i = P_1 P_2 P_3 P_4 P_5 \quad (\text{A.14})$$

Exercise 3 asks you to investigate which of the properties of summation notation carry over to the product notation.

A.2 Vectors and Matrices

A *scalar* is a single real number, such as 2, -3 , $\sqrt{2}$, or π . The notation $x \in \mathbb{R}$ means that x is a real number (a scalar). A *vector* is a collection of scalars; the *dimension* of a vector is the number of scalars it contains. For instance, $[3 \ -5]$ is a two-dimensional vector, $[6 \ 0 \ \pi]$ is a three-dimensional vector, and so forth. It is possible to write vectors either horizontally, with its component scalars in a row, or vertically, with its component scalars in a column. For the most part it doesn't matter whether vectors are written in a row or

in a column; the exception is in formulas involving multiplication between vectors and matrices, as described below, where row and column vectors must be distinguished. Vectors are usually denoted with boldfaced lower-case letters, like \mathbf{x} ; the notation $\mathbf{x} \in \mathbb{R}^n$ indicates that \mathbf{x} is an n -dimensional vector.¹ (So, $[3 \ -5] \in \mathbb{R}^2$ and $[6 \ 0 \ \pi] \in \mathbb{R}^3$.) Individual components of vectors are often denoted with subscripts or superscripts, as in x_1 or x^1 . The *zero vector* is a vector with zeros for all of its components, and is denoted $\mathbf{0}$.

Two vectors of the same dimension can be added together by adding the corresponding components of each vector. If $\mathbf{x} = [1 \ 2]$ and $\mathbf{y} = [3 \ 4]$, then $\mathbf{x} + \mathbf{y} = [4 \ 6]$. Multiplying a vector by a scalar means multiplying each component of the vector by the scalar, so $3\mathbf{x} = [3 \ 6]$ and $-\mathbf{y} = [-3 \ -4]$. The *dot product* of two vectors of the same dimension is defined as

$$\mathbf{x} \cdot \mathbf{y} = \sum_i x_i y_i, \quad (\text{A.15})$$

where the sum is taken over all vector components; with the example above, $\mathbf{x} \cdot \mathbf{y} = 1 \times 3 + 2 \times 4 = 11$.

The dot product of two vectors is a scalar. The magnitude of a vector \mathbf{x} is given by its *norm* $|\mathbf{x}| = \sqrt{\mathbf{x} \cdot \mathbf{x}}$. This norm provides a measure of distance between two vectors; the distance between \mathbf{x} and \mathbf{y} is given by $|\mathbf{x} - \mathbf{y}|$.

The dot product can also be written

$$\mathbf{x} \cdot \mathbf{y} = |\mathbf{x}||\mathbf{y}| \cos \theta \quad (\text{A.16})$$

where θ is the angle between the vectors \mathbf{x} and \mathbf{y} if they are both drawn from a common point. In particular, \mathbf{x} and \mathbf{y} are perpendicular if $\mathbf{x} \cdot \mathbf{y} = 0$.

A collection of vectors $\mathbf{x}_1, \dots, \mathbf{x}_n$ is *linearly independent* if the only solution to the equation $a_1\mathbf{x}_1 + \dots + a_n\mathbf{x}_n = \mathbf{0}$ is $a_1 = \dots = a_n = 0$. Otherwise, these vectors are *linearly dependent*.

A matrix is a rectangular array of scalars. If a matrix has m rows and n columns, it is called an $m \times n$ matrix, and is an element of $\mathbb{R}^{m \times n}$. A matrix is *square* if it has the same number of rows and columns. In this book, matrices are denoted by boldface capital letters, such as \mathbf{X} or \mathbf{Y} . In the examples that follow, let \mathbf{X} , \mathbf{Y} , and \mathbf{Z} be defined as follows:

$$\mathbf{X} = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \quad \mathbf{Y} = \begin{bmatrix} 0 & -1 \\ -3 & 5 \end{bmatrix} \quad \mathbf{Z} = \begin{bmatrix} -1 & 1 & -2 \\ 3 & -5 & 8 \end{bmatrix}.$$

Elements of matrices are indexed by their row first and column second, so $x_{11} = 1$ and $x_{12} = 2$.

Addition and scalar multiplication of matrices works in the same way as with vectors, so

$$\mathbf{X} + \mathbf{Y} = \begin{bmatrix} 1+0 & 2-1 \\ 3-3 & 4+5 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 0 & 9 \end{bmatrix}$$

¹There are some exceptions; for instance, shortest-path labels are traditionally denoted with an upper-case L .

and

$$5\mathbf{X} = \begin{bmatrix} 5 & 10 \\ 15 & 20 \end{bmatrix}.$$

The transpose of a matrix \mathbf{A} , written \mathbf{A}^T , is obtained by interchanging the rows and columns, so that if \mathbf{A} is an $m \times n$ matrix, \mathbf{A}^T is an $n \times m$ matrix. As examples we have

$$\mathbf{X}^T = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix} \quad \mathbf{Z}^T = \begin{bmatrix} -1 & 3 \\ 1 & -5 \\ -2 & 8 \end{bmatrix}.$$

Matrix multiplication is somewhat less intuitive. Two matrices can only be multiplied together if the number of columns in the first matrix is the same as the number of rows in the second. (The reason for this will become clear when the operation is defined.) For instance, you can multiply a 2×3 matrix by a 3×3 matrix, but you can't multiply a 2×3 matrix by another 2×3 matrix. This immediately suggests that the order of matrix multiplication is important, since two matrices may have compatible dimensions in one order, but not in the other. Multiplying an $m \times n$ matrix by a $n \times p$ matrix creates an $m \times p$ matrix, defined as follows. Let $\mathbf{A} \in \mathbb{R}^{m \times n}$ and $\mathbf{B} \in \mathbb{R}^{n \times p}$. Then the product $\mathbf{C} = \mathbf{AB} \in \mathbb{R}^{m \times p}$ has elements defined as

$$c_{ij} = \sum_{k=1}^n a_{ik}b_{kj}. \quad (\text{A.17})$$

If you imagine that each row of the first matrix is treated as a vector, and that each column of the second matrix is treated as a vector, then each component of the product matrix is the dot product of a row from the first matrix and a column from the second. For this dot product to make sense, these row and column vectors have to have the same dimension, that is, the number of columns in the first matrix must equal the number of rows in the second. Using the matrices defined above, we have

$$\mathbf{XY} = \begin{bmatrix} 1 \times 0 + 2 \times -3 & 1 \times -1 + 2 \times 5 \\ 3 \times 0 + 4 \times -3 & 3 \times -1 + 4 \times 5 \end{bmatrix} = \begin{bmatrix} -6 & 9 \\ -12 & 17 \end{bmatrix}.$$

Observe that the element in the first row and first column of the product matrix is the dot product of the first row from \mathbf{X} and the first column from \mathbf{Y} ; the element in the first row and second column of the product is the dot product of the first row from \mathbf{X} and the second column of \mathbf{Y} ; and so forth. You should be able to verify that

$$\mathbf{XZ} = \begin{bmatrix} 5 & -9 & 14 \\ 9 & -17 & 26 \end{bmatrix}.$$

A matrix and a vector can be multiplied together, if you interpret a row vector as an $1 \times n$ matrix or a column vector as a $m \times 1$ matrix. A dot product of two vectors \mathbf{x} and \mathbf{y} of equal dimension can be written as a matrix multiplication:

$\mathbf{x} \cdot \mathbf{y} = \mathbf{x}^T \mathbf{y}$ if they are both column vectors, or as \mathbf{xy}^T if they are both row vectors. Here, the distinction between row and column vectors is important, because matrices can only be multiplied if their dimensions are compatible. In matrix multiplications, the convention used in this text is that vectors such as \mathbf{x} are column vectors, and a row vector is denoted by \mathbf{x}^T . Again, this distinction is only relevant in matrix multiplication, and for other purposes row and column vectors can be treated interchangeably.

It is worth repeating that matrix multiplication is not commutative, so that usually $\mathbf{XY} \neq \mathbf{YX}$ (and in fact both products may not even exist, depending on the dimensions of \mathbf{X} and \mathbf{Y}), although there are some exceptions. You may wonder why this seemingly-complicated definition of matrix multiplication is used instead of other, seemingly simpler, approaches. One reason is that this definition actually ends up representing real-world calculations more frequently than other definitions. For instance, it can be used to compactly write a set of equations, as is common in optimization problems (see Section B.3).

A square matrix \mathbf{A} is *symmetric* if $a_{ij} = a_{ji}$ (in other words, for a symmetric matrix $\mathbf{A} = \mathbf{A}^T$), and it is *diagonal* if $a_{ij} = 0$ unless $i = j$ (that is, all its elements are zero except on the diagonal from upper-left to lower-right). A very special diagonal matrix is the *identity matrix*, which has 1's along the diagonal and 0's everywhere else. The notation \mathbf{I} denotes an identity matrix of any size, so we can write

$$\mathbf{I} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad \text{or} \quad \mathbf{I} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

In practice, the dimension is usually obvious because of the requirements of matrix multiplication. The identity matrix has the unique property that $\mathbf{AI} = \mathbf{IA} = \mathbf{A}$ for any matrix \mathbf{A} .

A square matrix \mathbf{A} is *invertible* if there is another square matrix (call it \mathbf{A}^{-1}) such that $\mathbf{AA}^{-1} = \mathbf{A}^{-1}\mathbf{A} = \mathbf{I}$. Multiplying by an inverse matrix should be thought of as the equivalent of “matrix division.” Just as it is not possible to divide by all numbers (division by zero is undefined), not all matrices have inverses. One can show that a square matrix is invertible if and only if the vectors forming its rows are linearly independent. The matrices seen in this book will all be invertible.

Computing the matrix inverse is a bit tedious, and is rarely needed in transportation network analysis. However one special case is worth mentioning:

Proposition A.1. *A diagonal matrix \mathbf{A} is invertible if and only if all its diagonal entries are nonzero; in this case, its inverse \mathbf{A}^{-1} is also a diagonal matrix, whose diagonal entries are the reciprocal of the diagonal entries in \mathbf{A} .*

So, for instance,

$$\left(\begin{bmatrix} 5 & 0 \\ 0 & -3 \end{bmatrix} \right)^{-1} = \begin{bmatrix} 1/5 & 0 \\ 0 & -1/3 \end{bmatrix}.$$

Finally, a symmetric matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ is *positive definite* if the matrix product $\mathbf{x}^T \mathbf{A} \mathbf{x}$ is strictly positive for any nonzero vector $\mathbf{x} \in \mathbb{R}^n$, and *positive semidefinite* if $\mathbf{x}^T \mathbf{A} \mathbf{x} \geq 0$ for any $\mathbf{x} \in \mathbb{R}^n$ whatsoever. As examples, consider the matrices

$$\mathbf{A} = \begin{bmatrix} 10 & 1 \\ 1 & 5 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} \quad \mathbf{C} = \begin{bmatrix} 1 & 10 \\ 10 & 5 \end{bmatrix}.$$

The matrix \mathbf{A} is positive definite, because for any vector $\mathbf{x} = [x_1 \ x_2]$, the matrix product is

$$\mathbf{x}^T \mathbf{A} \mathbf{x} = [x_1 \ x_2] \begin{bmatrix} 10 & 1 \\ 1 & 5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = 10x_1^2 + 2x_1x_2 + 5x_2^2.$$

The expression on the right is always positive, because it can be rewritten as

$$9x_1^2 + 4x_2^2 + (x_1 + x_2)^2.$$

None of those three terms can be negative, and since $\mathbf{x} \neq \mathbf{0}$, at least one of these terms is strictly positive.

The matrix \mathbf{B} is not positive definite, since if $\mathbf{x} = [1 \ 0]$ then $\mathbf{x}^T \mathbf{B} \mathbf{x} = 0$, which is not strictly positive. However, it is positive semidefinite, since the matrix product is

$$\mathbf{x}^T \mathbf{B} \mathbf{x} = [x_1 \ x_2] \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = x_2^2$$

which is surely nonnegative.

The matrix \mathbf{C} is neither positive definite nor positive semidefinite, since if $\mathbf{x} = [1 \ -1]$ then

$$\mathbf{x}^T \mathbf{C} \mathbf{x} = [1 \ -1] \begin{bmatrix} 1 & 10 \\ 10 & 5 \end{bmatrix} \begin{bmatrix} 1 \\ -1 \end{bmatrix} = -14 < 0.$$

Notice that for a matrix to be positive definite or semidefinite, we have to check whether a condition holds for all possible nonzero vectors. If the condition fails for even one vector, the matrix is not positive definite or semidefinite.

Checking positive definiteness or semidefiniteness can be tedious. One test is to find all the eigenvalues of the matrix; if they are all positive, the matrix is positive definite, and if all are non-negative it is positive semidefinite. However, diagonal matrices arise fairly often in transportation network analysis, and this case is easy.

Proposition A.2. *A diagonal matrix is positive definite if and only if all its diagonal entries are strictly positive. A diagonal matrix is positive semidefinite if and only if all its diagonal entries are nonnegative.*

There are also times where we will apply the concept of positive definiteness to non-symmetric matrices. The idea is the same – we want $\mathbf{x}^T \mathbf{A} \mathbf{x}$ to be strictly

positive for all nonzero \mathbf{x} – but the eigenvalue test does not apply, and there are non-symmetric matrices which have strictly positive eigenvalues but do not satisfy $\mathbf{x}^T \mathbf{A} \mathbf{x} > 0$ for all nonzero \mathbf{x} . However, we can form the *symmetric part* of the matrix \mathbf{A} by calculating $\frac{1}{2}(\mathbf{A} + \mathbf{A}^T)$. It is easy to show that this is always a symmetric matrix, and that $\mathbf{x}^T \mathbf{A} \mathbf{x} > 0$ if and only if $\mathbf{x}^T (\frac{1}{2}(\mathbf{A} + \mathbf{A}^T)) \mathbf{x} > 0$. So, when we refer to a non-symmetric matrix being positive definite, what we will mean is that its symmetric part $\frac{1}{2}(\mathbf{A} + \mathbf{A}^T)$ is positive definite.

The *determinant* of a square matrix is occasionally useful in transportation network analysis (but less so than in other applied mathematics fields). For a 1×1 matrix, its determinant is simply the value of the single entry in the matrix. For an $n \times n$ matrix, the determinant can be computed by the following procedure: select any row or column of the matrix; for each entry in this row or column, compute the determinant of the $(n - 1) \times (n - 1)$ matrix resulting from deleting both the row and column of this entry; and alternately add and subtract the resulting determinants. For our purposes, the determinant can be used to concisely express other matrix properties. For instance, one can show that a square matrix is invertible if and only if its determinant is not zero, and that it is positive definite if and only if the determinants of all the square submatrices including the element in the first row and column are positive. It also appears in the characterization of totally unimodular matrices, which are discussed in Section C.5.2 as a property of certain integer optimization problems which makes them easier to solve.

A.3 Sets

A *set* is a collection of any type of objects, denoted by a plain capital letter, such as X or Y . In transportation network analysis, we work with sets of numbers, sets of nodes, sets of links, sets of paths, sets of origin-destination pairs, and so forth. Sets can contain either a finite or infinite number of elements. As examples, let's work with the sets

$$X = \{1, 2, 4\} \quad Y = \{1, 3, 5, 7\},$$

using curly braces to denote a set, and commas to list the elements. Set membership is indicated with the notation \in , so $1 \in X$, $1 \in Y$, $2 \in X$, but $2 \notin Y$. The *union* of two sets $X \cup Y$ is the set consisting of elements either in X or in Y (or both), so

$$X \cup Y = \{1, 2, 3, 4, 5, 7\}.$$

The *intersection* of two sets $X \cap Y$ is the set consisting only of elements in both X and Y , so

$$X \cap Y = \{1\}.$$

A set is a *subset* of another set, denoted by \subseteq , if *all* of its elements also belong to the second set. With these sets $X \not\subseteq Y$. Even though the element 1 is in both X and Y , the elements 2 and 4 are in X but not Y . We do have $X \cap Y \subset X$ and $X \cap Y \subset Y$, however.

In this book, sets will take one of three forms:

1. Sets that consist of a finite number of elements, which can be listed as with X and Y above. This includes the set of nodes in a network, the set of acyclic paths in a network, and so forth. For a finite set, the notation $|X|$ indicates the number of elements in X .
2. Sets which are intervals on the real line, such as $[-1, 1]$ or $(0, 2]$. These intervals are sets containing all real numbers between their endpoints; a square bracket next to an endpoint means that the endpoint is included in the set, while parentheses mean that the endpoint is not included. Intervals usually contain infinitely many elements.
3. Sets which contain all objects satisfying one or more conditions. For instance, the set $\{x \in \mathbb{R} : x^2 < 4\}$ contains all real numbers whose square is less than four; in this case it can be written simply as the interval $(-2, 2)$. A more complicated set is $\{(x, y) \in \mathbb{R}^2 : x + y \leq 3, |x| \leq |y|\}$. This set contains all two-dimensional vectors which (if x and y are the two components of the vector) satisfy both the conditions $x + y \leq 3$ and $|x| \leq |y|$. It can also be thought of as the intersection of the sets $\{(x, y) \in \mathbb{R}^2 : x + y \leq 3\}$ and $\{(x, y) \in \mathbb{R}^2 : |x| \leq |y|\}$. If there are *no* vectors which satisfy all of the conditions, the set is *empty*, denoted \emptyset .

We use the common mathematical conventions that \mathbb{R} is the set of all real numbers, and \mathbb{Z} is the set of all integers. If we want to restrict attention to non-negative real numbers or integers (i.e., positive or zero), the notations \mathbb{R}_+ and \mathbb{Z}_+ are used. Superscripts (e.g., \mathbb{R}^3 or \mathbb{Z}^5) indicate vectors of a particular dimension whose elements belong to a particular set: \mathbb{R}^3 is the set of 3-dimensional vectors of real numbers, and \mathbb{Z}^5 the set of 5-dimensional integer vectors. To refer to a matrix of a particular size, we indicate both dimensions in the superscript; for instance, $\mathbb{Z}_+^{3 \times 5}$ is the set of matrices with 3 rows and 5 columns, each of whose elements is a non-negative integer.

Sets of scalars or vectors can be described in other ways. Given any vector $\mathbf{x} \in \mathbb{R}^n$, the *ball of radius r* is the set

$$B_r(\mathbf{x}) = \{\mathbf{y} \in \mathbb{R}^n : |\mathbf{x} - \mathbf{y}| < r\}, \quad (\text{A.18})$$

that is, the set of all vectors whose distance to \mathbf{x} is less than r , where r is some positive number. A ball in one dimension is an interval; a two-dimensional ball is a circle; a three-dimensional ball is a sphere; a four-dimensional ball a hypersphere, and so on.

Given some set of real numbers X , the vector \mathbf{x} is a *boundary point* of X if *every* ball $B_r(\mathbf{x})$ contains both elements in X and elements not in X , no matter how small the radius r . Notice that the boundary points of a set need not belong to the set: 2 is a boundary point of the interval $(-2, 2)$. A set is *closed* if it contains all of its boundary points. A set X is *bounded* if every element of X is contained in a sufficiently large ball centered at the origin, that is, if there is

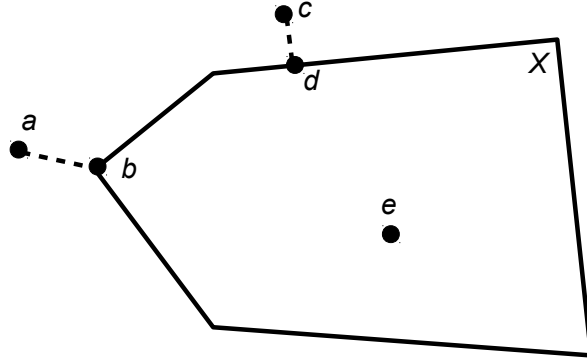


Figure A.1: Some examples of the projection operation.

some r such that $\mathbf{x} \in B_r(\mathbf{0})$ for all $\mathbf{x} \in X$. A set is *compact* if it is both closed and bounded.

These facts will prove useful:

Proposition A.3. Let $f(x_1, x_2, \dots, x_n)$ be any linear function, that is, $f(\mathbf{x}) = a_1x_1 + a_2x_2 + \dots + a_nx_n$ for some constants a_i , and let b be any scalar.

- (a) The set $\{\mathbf{x} \in \mathbb{R}^n : f(\mathbf{x}) = b\}$ is closed.
- (b) The set $\{\mathbf{x} \in \mathbb{R}^n : f(\mathbf{x}) \leq b\}$ is closed.
- (c) The set $\{\mathbf{x} \in \mathbb{R}^n : f(\mathbf{x}) \geq b\}$ is closed.

Proposition A.4. Let X and Y be any sets of scalars or vectors.

- (a) If X and Y are closed, so are $X \cap Y$ and $X \cup Y$.
- (b) If X and Y are bounded, so are $X \cap Y$ and $X \cup Y$.
- (c) If X and Y are compact, so are $X \cap Y$ and $X \cup Y$.

Combining Propositions A.3 and A.4, we can see that any set defined solely by linear equality or weak inequality constraints (any number of them) is closed.

If X is a closed set of n -dimensional vectors, and $\mathbf{y} \in \mathbb{R}^n$ is any n -dimensional vector, the *projection of \mathbf{y} onto X* , written $\text{proj}_X(\mathbf{y})$, is the vector \mathbf{x} in X which is “closest” to \mathbf{y} in the sense that $|\mathbf{x} - \mathbf{y}|$ is minimal. In Figure A.1, we have $\text{proj}_X(a) = b$, $\text{proj}_X(c) = d$, and $\text{proj}_X(e) = e$.

This subsection concludes with a discussion of what it means for a set to be *convex*. This notion is very important, and is described in more detail than the other concepts, starting with an intuitive definition.

If X is convex, geometrically this means that line segments connecting points of X lie entirely within X . For example, the set in Figure A.2 is convex, while

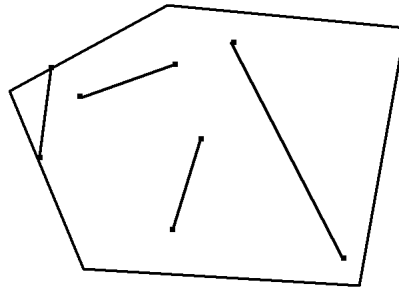


Figure A.2: A convex set.

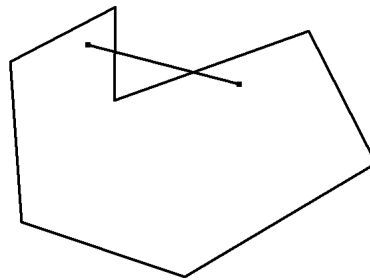


Figure A.3: A nonconvex set with a “bite” taken out of it.

those in Figure A.3 and Figure A.4 are not. Intuitively, a convex set cannot have any “holes” punched into it, or “bites” taken out of it.

Mathematically, we write this as follows:

Definition A.1. A set $X \subseteq \mathbb{R}^n$ is convex if, for all $x_1, x_2 \in X$ and all $\lambda \in [0, 1]$, the point $\lambda x_2 + (1 - \lambda)x_1 \in X$.

If this definition is not clear, notice that one way to express the line segment between any two points is $\lambda x_2 + (1 - \lambda)x_1$, and that you cover the entire line between x_1 and x_2 as λ varies between 0 and 1, regardless of how close or far apart these two points are located.

Example A.1. Show that the one-dimensional set $X = \{x : x \geq 0\}$ is convex.

Solution. Pick any $x_1, x_2 \geq 0$ and any $\lambda \in [0, 1]$. Because x_1, x_2 , and λ are all nonnegative, so are λx_2 and $(1 - \lambda)x_1$, and therefore so is $\lambda x_2 + (1 - \lambda)x_1$. Therefore $\lambda x_2 + (1 - \lambda)x_1$ belongs to X as well. ■

Example A.2. Show that the hyperplane $X = \{\mathbf{x} \in \mathbb{R}^n : \sum_{i=1}^n a_i x_i - b = 0\}$ is convex.

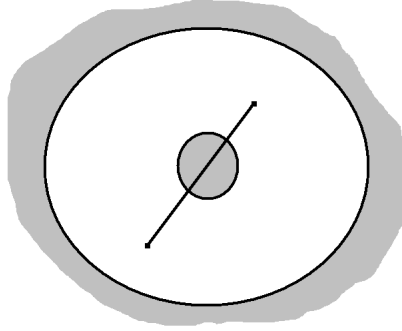


Figure A.4: A nonconvex set with a “hole” in it.

Solution. This set is the same as $\{\mathbf{x} \in \mathbb{R}^n : \sum_{i=1}^n a_i x_i = b\}$. Pick any $\mathbf{x}, \mathbf{y} \in X$ and any $\lambda \in [0, 1]$. Then

$$\begin{aligned} \sum_{i=1}^n a_i(\lambda y_i + (1-\lambda)x_i) &= \lambda \sum_{i=1}^n a_i y_i + (1-\lambda) \sum_{i=1}^n a_i x_i \\ &= \lambda b + (1-\lambda)b \\ &= b \end{aligned}$$

so $\lambda \mathbf{y} + (1-\lambda)\mathbf{x} \in X$ as well. ■

Sometimes, more complicated arguments are needed.

Example A.3. Show that the two-dimensional ball $B = \{[x \ y] \in \mathbb{R}^2 : x^2 + y^2 \leq 1\}$ is convex.

Solution. Pick any vectors $\mathbf{a}, \mathbf{b} \in B$ and any $\lambda \in [0, 1]$. We will write the components of these as $\mathbf{a} = [a_x \ a_y]$ and $\mathbf{b} = [b_x \ b_y]$. The point $\lambda \mathbf{b} + (1-\lambda)\mathbf{a}$ is the vector $[\lambda b_x + (1-\lambda)a_x \ \lambda b_y + (1-\lambda)a_y]$. To show that it is in B , we must show that the sum of the squares of these components is no greater than 1.

$$\begin{aligned} &(\lambda b_x + (1-\lambda)a_x)^2 + (\lambda b_y + (1-\lambda)a_y)^2 \\ &= \lambda^2(b_x^2 + b_y^2) + (1-\lambda)^2(a_x^2 + a_y^2) + 2\lambda(1-\lambda)(a_x b_x + a_y b_y) \\ &\leq \lambda^2 + (1-\lambda)^2 + 2\lambda(1-\lambda)(a_x b_x + a_y b_y) \end{aligned}$$

because $\mathbf{a}, \mathbf{b} \in B$ (and therefore $a_x^2 + a_y^2 \leq 1$ and $b_x^2 + b_y^2 \leq 1$). Notice that $a_x b_x + a_y b_y$ is simply the dot product of \mathbf{a} and \mathbf{b} , which is equal to $|\mathbf{a}||\mathbf{b}| \cos \theta$, where θ is the angle between the vectors \mathbf{a} and \mathbf{b} . Since $|\mathbf{a}| \leq 1$, $|\mathbf{b}| \leq 1$ (by definition of B), and since $\cos \theta \leq 1$ regardless of θ , $a_x b_x + a_y b_y \leq 1$. Therefore

$$\begin{aligned} &\lambda^2 + (1-\lambda)^2 + 2\lambda(1-\lambda)(a_x b_x + a_y b_y) \\ &\leq \lambda^2 + (1-\lambda)^2 + 2\lambda(1-\lambda) = (\lambda + (1-\lambda))^2 = 1 \end{aligned}$$

so the point $\lambda \mathbf{b} + (1 - \lambda)\mathbf{a}$ is in B regardless of the values of \mathbf{a} , \mathbf{b} , or λ . Thus B is convex. ■

Example A.4. Show that the set $C = \{[x \ y] \in \mathbb{R}^2 : x^2 + y^2 > 1\}$ is not convex.

Solution. Let $\mathbf{b}^1 = [2 \ 0]$, $\mathbf{b}^2 = [-2 \ 0]$, $\lambda = 1/2$. Then $\lambda \mathbf{b}^2 + (1 - \lambda)\mathbf{b}^1 = [0 \ 0] \notin C$ even though $\mathbf{b}^1, \mathbf{b}^2 \in C$ and $\lambda \in [0, 1]$. Therefore C is not convex. ■

Proving that a set is convex requires showing that something is true for *all* possible values of $x_1, x_2 \in X$, and $\lambda \in [0, 1]$, whereas disproving convexity only requires you to pick one combination of these values where the definition fails. Lastly, this result will prove useful in showing that a set is convex:

Proposition A.5. If X and Y are convex sets, so is $X \cap Y$.

A.4 Functions

A function is a mapping between sets. If the function f maps set X to set Y , then f associates *every* element of X with *some* single element of Y . (Note that not every element of Y needs to be associated with an element of X .) The set X is known as the *domain* of f . Examples include the familiar functions $f(x) = x^2$ and $g(x, y) = x^2 + y^2$. The function f maps \mathbb{R} to \mathbb{R} , while g maps \mathbb{R}^2 to \mathbb{R} . An example of a function which maps \mathbb{R}^2 to \mathbb{R}^2 is the vector-valued function

$$\mathbf{h}(x_1, x_2) = \begin{bmatrix} 3x_1 + 2x_2 \\ -x_1x_2 \end{bmatrix}$$

The inverse of a function f , denoted f^{-1} , “undoes” the mapping f in the sense that if $f(x) = y$, then $f^{-1}(y) = x$. As an example, if $f(x) = x^3$, then $f^{-1}(x) = \sqrt[3]{x}$. Not every function has an inverse, and inverse functions may need to be restricted to subsets of X and Y .

The composition of two functions f and g , denoted $f \circ g$ involves substituting function g into function f . If $f(x) = x^3$ and $g(x) = 2x + 4$, then $f \circ g(x) = (2x + 4)^3$. This can also be written as the function $f(g(x))$.

A function is *continuous* if, for any point $\hat{x} \in X$, the limit $\lim_{x \rightarrow \hat{x}} f(x)$ exists and is equal to $f(\hat{x})$. Intuitively, continuous functions can be drawn without lifting your pen from the paper. A function is *differentiable* if, for any point $x \in X$, the limit

$$\lim_{\Delta x \rightarrow 0} \frac{f(x + \Delta x) - f(x)}{|\Delta x|} \quad (\text{A.19})$$

exists; that limit is then called the *derivative* of f , and gives the slope of the tangent line at any point. It can be shown that any differentiable function must be continuous.

Proposition A.6. Let f and g be continuous functions. Then we have the following:

- The multiple αf is a continuous function for any scalar α .
- The sum $f + g$ is a continuous function.
- The product fg is a continuous function.
- The composition $f \circ g$ is a continuous function.

Furthermore, all of these results hold if “continuous” is replaced by “differentiable.”

Proposition A.7. Let X be a nonempty, convex set of n -dimensional vectors. Then the projection function $\text{proj}_X(\mathbf{x})$ is defined and continuous for all $\mathbf{x} \in \mathbb{R}^n$.

Differentiability is more complicated when dealing with functions of multiple variables (that is, functions whose domain is \mathbb{R}^2 , \mathbb{R}^3 , and so forth). The basic notion is the *partial derivative*, in which all variables except one are assumed constant, and an ordinary derivative is taken with respect to the remaining variable. For instance, if $f(x, y) = x^2 + 3xy + y^3$, the partial derivatives with respect to x and y are

$$\frac{\partial f}{\partial x} = 2x + 3y \quad \frac{\partial f}{\partial y} = 3x + 3y^2.$$

Second partial derivatives are found in the same way, taking partial derivatives of $\frac{\partial f}{\partial x}$ and $\frac{\partial f}{\partial y}$. If all of the second partial derivatives of a function are continuous at a point, the order of differentiation does not matter and

$$\frac{\partial^2 f}{\partial x \partial y} = \frac{\partial^2 f}{\partial y \partial x} \tag{A.20}$$

at that point.

All of these partial derivatives can be organized into different kinds of vectors and matrices. The *gradient* of a function is the vector of all its partial derivatives. For $f(x, y) = x^2 + 3xy + y^3$, this is

$$\nabla f(x, y) = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix} = \begin{bmatrix} 2x + 3y \\ 3x + 3y^2 \end{bmatrix}.$$

A point \mathbf{x} where $\nabla f(\mathbf{x}) = \mathbf{0}$ is called a *stationary point*. (If f is a function of a single variable, this is just a point where the derivative vanishes.) The *Hessian* of a function is the matrix of all of its second partial derivatives. For this same function, we have

$$Hf(x, y) = \begin{bmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x \partial y} \\ \frac{\partial^2 f}{\partial y \partial x} & \frac{\partial^2 f}{\partial y^2} \end{bmatrix} = \begin{bmatrix} 2 & 3 \\ 3 & 6y \end{bmatrix}$$

By equation (A.20), if the Hessian matrix contains continuous functions, it is symmetric.

Now consider a vector-valued function of multiple variables $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$. (That is, f has n input variables and produces an m -dimensional vector as output.) The *Jacobian* of f is a matrix of all of its *first* partial derivatives. For a concrete example, let

$$\mathbf{g}(x, y, z) = \begin{bmatrix} g_1(x, y, z) \\ g_2(x, y, z) \end{bmatrix} = \begin{bmatrix} x^2 + y^2 + z^2 \\ xyz \end{bmatrix}.$$

The Jacobian is the matrix

$$J\mathbf{g}(x, y, z) = \begin{bmatrix} \frac{\partial g_1}{\partial x} & \frac{\partial g_1}{\partial y} & \frac{\partial g_1}{\partial z} \\ \frac{\partial g_2}{\partial x} & \frac{\partial g_2}{\partial y} & \frac{\partial g_2}{\partial z} \end{bmatrix} = \begin{bmatrix} 2x & 2y & 2z \\ yz & xz & xy \end{bmatrix}$$

Be sure to note the difference between Hessians and Jacobians. The Hessian is defined for scalar-valued functions, and contains all the second partial derivatives. The Jacobian is defined for vector-valued functions, and contains all the first partial derivatives.

Finally, there is an important notion of function *convexity*. Confusingly, this is a different idea than set convexity discussed in Section A.3, although there are some relationships and similar ideas between them. Set and function convexity together play a pivotal role in optimization and network equilibrium problems, so function convexity is discussed at length here. Geometrically, a convex function lies below its secant lines. Remember that a secant line is the line segment joining two points on the function. As we see in Figure A.5, no matter what two points we pick, the function always lies below its secant line. On the other hand, in Figure A.6, not every secant line lies above the function: some lie below it, and some lie both above and below it. Even though we can draw some secant lines which are above the function, this isn't enough: *every* possible secant must lie above the function. For this concept to make sense, the domain X of the function must be a convex set, an assumption which applies for the remainder of this section

The following definition makes this intuitive notion formal:

Definition A.2. A function $f : X \rightarrow \mathbb{R}$ is convex if, for every $x_1, x_2 \in X$ and every $\lambda \in [0, 1]$,

$$f((1 - \lambda)x_1 + \lambda x_2) \leq (1 - \lambda)f(x_1) + \lambda f(x_2) \quad (\text{A.21})$$

and strictly convex if

$$f((1 - \lambda)x_1 + \lambda x_2) < (1 - \lambda)f(x_1) + \lambda f(x_2) \quad (\text{A.22})$$

for all distinct $x_1, x_2 \in X, \lambda \in (0, 1)$

Essentially, x_1 and x_2 are the two endpoints for the secant line. Since this entire line segment must be above the function, we need to consider every point between x_1 and x_2 . This is what λ does: as λ varies between 0 and 1, the points $\lambda x_2 + (1 - \lambda)x_1$ cover every point between x_1 and x_2 . You can think of

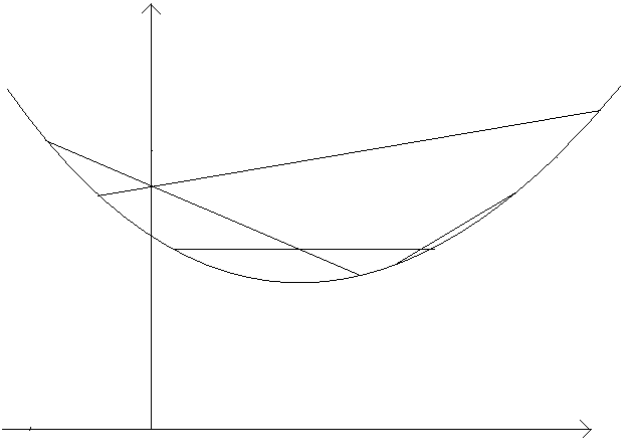


Figure A.5: A convex function lies below all of its secants.

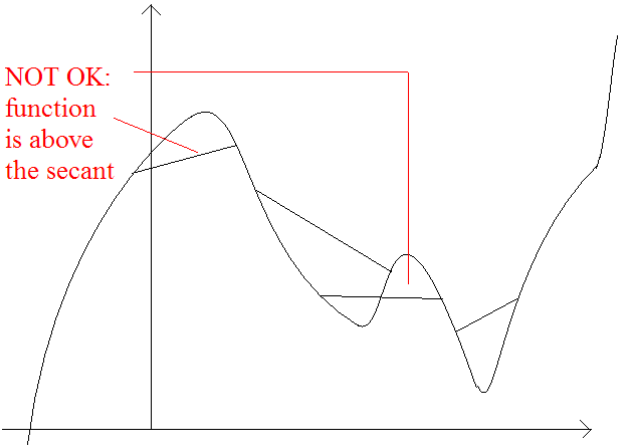


Figure A.6: A nonconvex function does not lie below all of its secants.

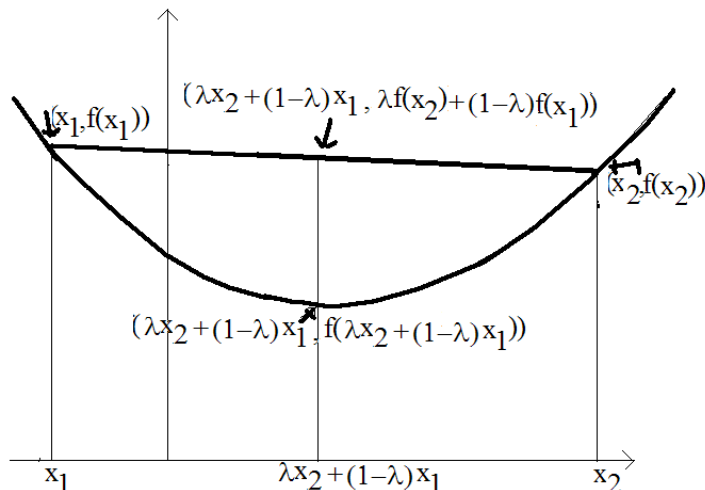


Figure A.7: All of the relevant points for the definition of convexity.

$\lambda x_2 + (1 - \lambda)x_1$ as a “weighted average,” where λ is the weight put on x_2 . For $\lambda = 0$, all the weight is on x_1 . For $\lambda = 1$, all the weight is on x_2 . For $\lambda = 1/2$, equal weight is put on the two points, so the weighted average is the midpoint. $\lambda = 1/3$ corresponds to the point a third of the way between x_1 and x_2 .

So we need to say that, at all such intermediate x values, the value of f is lower than the y -coordinate of the secant. The value of the function at this point is simply $f((1 - \lambda)x_1 + \lambda x_2)$. Because the secant is a straight line, its y -coordinate can be seen as a weighted average of the y -coordinates of its endpoints, that is, $f(x_1)$ and $f(x_2)$. This weighted average can be written as $(1 - \lambda)f(x_1) + \lambda f(x_2)$, so requiring the function to lie below the secant line is exactly the same as enforcing condition (A.21) for all possible secant lines: that is, for all $x_1, x_2 \in X$ and all $\lambda \in [0, 1]$.

Figure A.7 explains this in more detail. Along the horizontal axis, the secant endpoints x_1 and x_2 are shown, along with an intermediate point $\lambda x_2 + (1 - \lambda)x_1$. The y -coordinates are also shown: at the endpoints, these are $f(x_1)$ and $f(x_2)$. At the intermediate point, the y -coordinate of the function is $f(\lambda x_2 + (1 - \lambda)x_1)$, while the y -coordinate of the secant is $\lambda f(x_2) + (1 - \lambda)f(x_1)$. Because the function is convex, the former can be no bigger than the latter. Time spent studying this diagram is very well spent. Make sure you understand what each of the four points marked on the diagram represents, and why the given mathematical expressions correctly describe these points. Make sure you see what role λ plays: as λ increases from 0 to 1, the central vertical line moves from x_1 to x_2 . (What would happen if we picked x_1 and x_2 such that $x_1 > x_2$? What if $x_1 = x_2$?)

Example A.5. *Is the function $f(x) = |x|$, $x \in \mathbb{R}$ convex? Is it strictly convex?*

Solution. To see if f is convex, we need to see if (A.21) is true; to see if it is strictly convex, we need to check (A.22). Furthermore, these inequalities have to be true for *every* $x_1, x_2 \in \mathbb{R}$, and *every* $\lambda \in [0, 1]$. It is not enough to simply pick a few values randomly and check the equations. So, we have to work symbolically. In this case,

$$\begin{aligned} f((1-\lambda)x_1 + \lambda x_2) &= |(1-\lambda)x_1 + \lambda x_2| \\ &\leq |(1-\lambda)x_1| + |\lambda x_2| \quad \text{by the triangle inequality} \\ &= (1-\lambda)|x_1| + \lambda|x_2| \quad \text{because } \lambda, 1-\lambda \geq 0 \\ &= (1-\lambda)f(x_1) + \lambda f(x_2) \end{aligned}$$

Therefore (A.21) is satisfied, so f is convex. To show that it is strictly convex, we would have to show that the inequality

$$|(1-\lambda)x_1 + \lambda x_2| < |(1-\lambda)x_1| + |\lambda x_2|$$

can be replaced by a *strict* inequality $<$. However, we can't do this: for example, if $x_1 = 1$, $x_2 = 2$, $\lambda = 0.5$, the left side of the inequality ($|1/2 + 2/2| = 3/2$) is exactly equal to the right side ($|1/2| + |2/2| = 3/2$). So f is not strictly convex. ■

Note that proving that $f(x)$ is convex requires a *general* argument, where proving that $f(x)$ was not strictly convex only required a single counterexample. This is because the definition of convexity is a “for all” or “for every” type of argument. To prove convexity, you need an argument that allows for all possible values of x_1 , x_2 , and λ , whereas to disprove it you only need to give one set of values where the necessary condition doesn't hold.

Example A.6. *Show that every linear function $f(x) = ax + b$, $x \in \mathbb{R}$ is convex, but not strictly convex.*

Solution.

$$\begin{aligned} f((1-\lambda)x_1 + \lambda x_2) &= a((1-\lambda)x_1 + \lambda x_2) + b \\ &= a((1-\lambda)x_1 + \lambda x_2) + ((1-\lambda) + \lambda)b \\ &= (1-\lambda)(ax_1 + b) + \lambda(ax_2 + b) \\ &= (1-\lambda)f(x_1) + \lambda f(x_2) \end{aligned}$$

So we see that inequality (A.21) is in fact satisfied as an *equality*. That's fine, so every linear function is convex. However, this means we can't replace the inequality \leq with the strict inequality $<$, so linear functions are not strictly convex. ■

Sometimes it takes a little bit more work, as in the following example:

Example A.7. *Show that $f(x) = x^2$, $x \in \mathbb{R}$ is strictly convex.*

Solution. Pick x_1, x_2 so that $x_1 \neq x_2$, and pick $\lambda \in (0, 1)$.

$$\begin{aligned} f((1-\lambda)x_1 + \lambda x_2) &= ((1-\lambda)x_1 + \lambda x_2)^2 \\ &= (1-\lambda)^2 x_1^2 + \lambda^2 x_2^2 + 2(1-\lambda)\lambda x_1 x_2 \end{aligned}$$

What to do from here? Comparing term by term, because $\lambda \in (0, 1)$, we know that $\lambda^2 < \lambda$ and $(1-\lambda)^2 < 1-\lambda$. Therefore $(1-\lambda^2)x_1^2 < (1-\lambda)x_1^2$ and $\lambda^2 x_2^2 < \lambda x_2^2$. This takes care of two of the terms, all we have left is Since $x_1 \neq x_2$, $(x_1 - x_2)^2 > 0$. Expanding, this means that $x_1^2 + x_2^2 > 2x_1 x_2$. This means that

$$\begin{aligned} (1-\lambda)^2 x_1^2 + \lambda^2 x_2^2 + 2(1-\lambda)\lambda x_1 x_2 &< (1-\lambda)^2 x_1^2 + \lambda^2 x_2^2 + (1-\lambda)(\lambda)(x_1^2 + x_2^2) \\ &= (1-\lambda)x_1^2 + \lambda x_2^2 \text{ (after some algebra)} \\ &= (1-\lambda)f(x_1) + \lambda f(x_2) \end{aligned}$$

which proves strict convexity. ■

This last example shows that proving convexity can be difficult and nonintuitive, even for simple functions like x^2 . The good news is that there are often simpler conditions that we can check. These conditions involve the first and second derivatives of a function.

Proposition A.8. *Let $f : X \rightarrow \mathbb{R}$ be a differentiable function, where X is a subset of \mathbb{R} . Then f is convex if and only if*

$$f(x_2) \geq f(x_1) + f'(x_1)(x_2 - x_1)$$

for all $x_1, x_2 \in X$.

Proposition A.9. *Let $f : X \rightarrow \mathbb{R}$ be twice differentiable, where X is a subset of \mathbb{R} , and let f be twice differentiable on X . Then f is convex if and only if $f''(x) \geq 0$ for all $x \in X$*

Equivalent conditions for strict convexity can be obtained in a natural way, changing \geq to $>$ and requiring that x_1 and x_2 be distinct in Proposition A.8. Proposition A.9 also changes slightly in this case; $f''(x) > 0$ is sufficient for strict convexity but is not necessary. You are asked to prove these statements in the exercises. Essentially, Proposition A.8 says that f lies *above* its tangent lines (Figure A.8), while Proposition A.9 says that f is always “curving upward.” (A convex function lies *above* its tangents, but *below* its secants.)

These conditions are usually easier to verify than that of Definition A.2.

Example A.8. *Show that $f(x) = x^2$ is strictly convex using Proposition A.8*

Solution. Pick any $x_1, x_2 \in \mathbb{R}$ with $x_1 \neq x_2$. We have $f'(x_1) = 2x_1$, so we need to show that

$$x_2^2 > x_1^2 + 2x_1(x_2 - x_1)$$

Expanding the right-hand side and rearranging terms, we see this is equivalent to

$$x_1^2 - 2x_1 x_2 + x_2^2 > 0$$

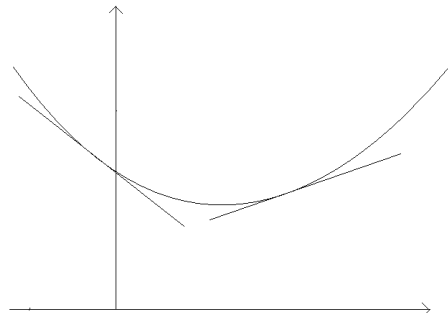


Figure A.8: A convex function lies above its tangents.

or

$$(x_1 - x_2)^2 > 0$$

which is clearly true since $x_1 \neq x_2$. Thus f is strictly convex. ■

Example A.9. Show that $f(x) = x^2$ is strictly convex using Proposition A.9

Solution. $f''(x) = 2 > 0$ for all $x \in \mathbb{R}$, so f is strictly convex. ■ Much simpler! If f is differentiable (or, better yet, twice differentiable) checking these conditions is almost always easier.

Furthermore, once we know that some functions are convex, we can use this to show that many other combinations of these functions must be convex as well.

Proposition A.10. If f and g are convex functions, and α and β are positive real numbers, then $\alpha f + \beta g$ is convex as well.

Proposition A.11. If f and g are convex functions, then $f \circ g$ is convex as well.

Some common convex functions are $|x|$, x^2 , e^x , and $ax + b$. So, Proposition A.10 tells us that $3x^2 + 4|x|$ is convex. It also tells us that any quadratic function $ax^2 + bx + c$ is convex as long as $a > 0$. Proposition A.11 says that the composition of two convex functions is convex as well. For instance, e^{x^2} is convex, and $x^4 = (x^2)^2$ is convex as well.

What about functions of more than one variable? The “shortcut” conditions in Propositions A.8 and A.9 only apply if the domain of f is one-dimensional. It turns out that very similar conditions can be given for the multivariable case. The multi-dimensional equivalent of the first derivative is the gradient, and the equivalent of the second derivative is the Hessian.

The equivalent conditions on convexity are

Proposition A.12. *Let $f : X \rightarrow \mathbb{R}$ be a function whose gradient exists everywhere on X . Then f is convex if and only if*

$$f(\mathbf{x}_2) \geq f(\mathbf{x}_1) + \nabla f(x_1) \cdot (\mathbf{x}_2 - \mathbf{x}_1)$$

for all $\mathbf{x}_1, \mathbf{x}_2 \in X$.

Proposition A.13. *Let $f : X \rightarrow \mathbb{R}$ be a function whose Hessian exists everywhere on X . Then f is convex if and only if $H(f)$ is positive semidefinite for all $\mathbf{x} \in X$; and f is strictly convex if $H(f)$ is positive definite for all $\mathbf{x} \in X$.*

Unfortunately, neither of these is as easy to check as the single-dimension equivalents. In particular, it is rather tedious to check whether or not a matrix is positive semidefinite or not. Fortunately, in many important cases in transportation network analysis, the Hessian is diagonal and Proposition A.2 applies.

As a final note, one useful link between convex sets and convex functions is the following result:

Proposition A.14. *If $g : \mathbb{R}^n \rightarrow \mathbb{R}$ is a convex function, then the set $X = \{x : g(x) \leq 0\}$ is a convex set.*

A.5 Exercises

- [22] For this exercise, let $x_{11} = 5$, $x_{12} = 6$, $x_{13} = 7$, $x_{21} = 4$, $x_{22} = 3$, and $x_{23} = 9$. Evaluate each of these sums.
 - $\sum_{i=1}^2 \sum_{j=1}^3 x_{ij}$
 - $\sum_{i=1}^2 \sum_{j=2}^3 x_{ij}$
 - $\sum_{j=1}^3 x_{1j}$
 - $\sum_{j=1}^2 \sum_{i=1}^3 x_{ji}$
 - $\sum_{j=1}^3 \sum_{k=0}^1 x_{k+1,j}$
 - $\sum_{i=1}^2 \sum_{j=i}^3 x_{ij}$
- [22] Repeat Exercise 1, but with products \prod instead of sums \sum .
- [33] Section A.1 lists three properties of the summation notation \sum . Formulate and prove analogous properties for the product notation \prod . You can assume that the product involves only a finite number of factors.
- [22] Prove Proposition A.1.
- [24] Prove Proposition A.2.
- [53] Prove that if a matrix is invertible, its inverse matrix is unique.

7. [26] For each of the sets below, identify its boundary points and indicate whether or not it is closed, whether or not it is bounded, and whether or not it is convex.
- (a) $(5, 10]$
 - (b) $[4, 6]$
 - (c) $(0, \infty)$
 - (d) $\{x : |x| > 5\}$
 - (e) $\{x : |x| \leq 5\}$
 - (f) $\{(x, y) : 0 \leq x \leq 4, -3 \leq y \leq 3\}$
 - (g) $\{(x, y) : 4x - y = 1\}$
8. [34] Prove Proposition A.3.
9. [34] Prove Proposition A.4.
10. [53] Identify the projections of the following points on the corresponding sets. You may find it helpful to draw sketches.
- (a) The point $x = 3$ on the set $[0, 1]$.
 - (b) The point $x = \frac{1}{2}$ on the set $[0, 1]$.
 - (c) The point $(2, 5)$ on the unit circle $x^2 + y^2 = 1$.
 - (d) The point $(2, 5)$ on the line $x + y = 1$.
 - (e) The point $(2, 5)$ on the line segment between $(0, 1)$ and $(1, 0)$.
 - (f) The point $(2, 3)$ on the line segment between $(0, 1)$ and $(1, 0)$.
 - (g) The point $(1, 2, 3)$ on the sphere $x^2 + y^2 + z^2 = 2$.
11. [20] Prove Proposition A.5.
12. [23] Find the inverses of the following functions.
- (a) $f(x) = 3x + 4$
 - (b) $f(x) = 5e^x - 1$
 - (c) $f(x) = 1/x$
13. [57] Prove Proposition A.6 from first principles, using the definitions of continuity and differentiability in the text.
14. [65] Prove Proposition A.7, using the formal definition of continuity.
15. [22] Calculate the gradients of the following functions:
- (a) $f(x_1, x_2, x_3) = x_1^2 + 2x_2x_3 + x_3^2$
 - (b) $f(x, y) = \frac{xy}{x^2 + y^2 + 1}$
 - (c) $f(x_1, x_2, d_1, d_2) = 3x_1 + 5x_2^2 + 3(d_1 + 4)^2 + 5(d_2 - 1)^3$

16. [25] Calculate the Jacobian matrices of the following functions:

$$(a) f(x, y) = \begin{bmatrix} y \\ -x \end{bmatrix}$$

$$(b) f(x_1, x_2, x_3) = \begin{bmatrix} x_1^2 + x_2 + x_3 \\ 3x_2 - 5x_3 \end{bmatrix}$$

$$(c) f(x, y) = \begin{bmatrix} 3x + y \\ 3y + x \\ xy \end{bmatrix}$$

17. [26] Calculate the Hessian matrices of functions from Exercise 15.

18. [35] Determine which of the following functions are convex, strictly convex, or neither. Justify your answer rigorously.

$$(a) f(x) = \log x \text{ where } x \geq 1.$$

$$(b) f(x) = -x^3 \text{ where } x \leq 0.$$

$$(c) f(x) = \sin x \text{ where } x \in \mathbb{R}.$$

$$(d) f(x) = e^{x^2 - 2x - 3} \text{ where } x \in \mathbb{R}.$$

$$(e) f(x) = \begin{cases} 0 & \text{if } x \leq 0 \\ x & \text{if } x \geq 0 \end{cases}$$

19. [35] Determine which of the following sets is convex. Justify your answer rigorously.

$$(a) X = \{(x_1, x_2) \in \mathbb{R}^2 : 4x_1 - 3x_2 \leq 0\}$$

$$(b) X = \{x \in \mathbb{R} : e^x \leq 4\}$$

$$(c) X = \{(x_1, x_2, x_3) \in \mathbb{R}^3 : x_1 = 0\}$$

$$(d) X = \{(x_1, x_2, x_3) \in \mathbb{R}^3 : x_1 \neq 0\}$$

$$(e) X = \{1\}$$

20. [31] Show that the integral of an increasing function is a convex function.

21. [54] Show that a differentiable function f of a single variable is convex if and only if $f(x) + f'(x)(y - x) \leq f(y)$ for all x and y .

22. [54] Show that a twice-differentiable function f of a single variable is convex if and only if f'' is everywhere nonnegative.

23. [68] Prove the claims of Exercises 21 and 22 for functions of multiple variables.

24. [32] Page 466 states that " $f''(x) > 0$ is sufficient for strict convexity but not necessary." Give an example of a strictly convex function where $f''(x) = 0$ at some point.

Appendix B

Optimization Concepts

This appendix provides a brief introduction on formulating optimization problems. We provide examples of linear, nonlinear, and integer formulations. The focus in this appendix is on the basic terminology and formulation of these problems. Specifically, we provide a detailed explanation of representing optimization formulations using index and matrix notations. The following appendix goes into more detail on solution methods.

B.1 Components of an Optimization Problem

An optimization problem is a mathematical model that can help decision makers arrive at decisions which optimize a specific goal, given the constraints they face. Every optimization problem has three components: an objective function, decision variables, and constraints. When one talks about *formulating* an optimization problem, it means translating a “real-world” problem into the mathematical equations and variables which comprise these three components:

Objective function: A mathematical function which represent the goal of the decision maker, to be either maximized or minimized.

Decision variables: Variables representing the factors or choices that can be controlled by the decision maker, either directly or indirectly.

Constraints: Equations or inequalities representing restrictions on the values the decision variables can take.

The objective function, often¹ denoted f or z , reflects a single quantity to be either maximized or minimized. Examples in the transportation world include “minimize congestion”, “maximize safety”, “maximize accessibility”,

¹But not always! Often problem-specific notation will be used, such as c for cost, and so forth. The notation given in this section is what is traditionally used if we are referring to a generic optimization problem outside of a specific context.

“minimize cost”, “maximize pavement quality”, “minimize emissions,” “maximize revenue,” and so forth. You may object to the use of a *single* objective function, since real-world problems typically involve many different and conflicting objectives from different stakeholders, and this objection is certainly valid. There are several reasons why the scope of this book is restricted to single objectives. From a historical perspective, if we don’t know how to optimize a single objective function, then we have no hope of being able to optimize multiple objectives simultaneously, since the latter build on the former. From a pedagogical perspective, the methods of multi-objective optimization are much more complex, and the basic concepts of optimization are best learned in a simpler context first. From a mathematical perspective, the definition of “simultaneously optimize” is very tricky, since there is probably no plan which will, say, simultaneously minimize congestion and agency cost — therefore multiobjective optimization is “fuzzier,” and this fuzziness can be confusing when first taught. So, don’t be alarmed by this restriction to a single objective, but do keep it in the back of your head if you use optimization beyond this book.

The decision variables (often, but not always, denoted as the vector \mathbf{x}) reflect aspects of the problem that you (or the decision maker) can control. This can include both variables you can directly choose, as well as variables which you indirectly influence by the choice of other decision variables. For example, if you are a private toll road operator trying to maximize your profit, you can directly choose the toll, so that is a decision variable. You can’t directly choose how many people drive on the road — but because that’s influenced by the toll you chose, the toll road volume should be a decision variable as well. However, you should avoid including extraneous decision variables. Every decision variable in your formulation should either directly influence the objective function, or influence another decision variable that affects the objective function.

Constraints represent any kind of limitation on the values that the decision variables can take. The most intuitive types of constraints are those which directly and obviously limit the choices you can make: you can’t exceed a budget, you are required by law to provide a certain standard of maintenance, you are not allowed to change the toll by more than \$1 from its current value, and so forth. One very frequent mistake is to omit “obvious” constraints (e.g., the toll can’t be negative). An optimization formulation must be complete and not leave out any constraint, no matter how obvious it may seem to you. Real-world optimization problems are solved by computers, for which nothing is “obvious.” The second type of constraint is required to ensure consistency among the decision variables. Following the toll road example from the previous paragraph, while you can influence both the toll (directly) and the roadway volume (indirectly), the roadway volume must be consistent with the toll you chose. This relationship must be reflected by an appropriate constraint; say, a demand function $d(\tau)$ giving the demand d for the tollway in terms of the toll τ , which can be estimated in a variety of econometric ways. Constraints are the primary way these linkages between decision variables can be captured.

A particular choice of decision variables \mathbf{x} is *feasible* if it satisfies all of the constraints, and it is *optimal* if it maximizes or minimizes the objective function

(whichever is appropriate for the given problem). For some problems, we may need to distinguish local and global optima. Section B.6.1 discusses this in greater detail. In short, we usually want to find a global optimum, but for some complicated optimization problems a local optimum may be the best we can do.

The following example explains these definitions:

Example B.1. *A plant produces two types of concrete (Type 1 and Type 2). The owner of the plant makes a profit of \$90 for each truckload of Type 1 concrete produced and a profit of \$120 for each truckload of Type 2 concrete produced. Three materials are required to produce concrete: cement, aggregate, and water. The plant requires 30 units of cement, 50 units of aggregate, and 60 units of water to produce one truckload of Type 1 concrete. The plant requires 40 units of cement, 20 units of aggregate, and 90 units of water to produce one truckload of Type 2 concrete. The plant is supplied 2000 units of cement, 2500 units of aggregate, and 4000 units of water daily. How many truckloads of Type 1 and Type 2 cement should the plant produce to maximize daily profit?*

In any optimization problem, the first step is to identify the decision variables or the variables which can be controlled by the decision maker. In this problem, the decision variables are the daily truckloads of Type 1 and Type 2 concrete produced by the plant.

Let x represent the daily truckloads of Type 1 concrete produced by the plant and y represent the daily truckloads of Type 2 concrete produced by the plant.

The second step is to identify the constraints or restrictions on the decision variables. In this problem there are restrictions on the total volume of cement, aggregate, and water supplied to the plant daily which limits the total amount of Type 1 and Type 2 concrete which can be produced.

Each truckload of Type 1 concrete requires 30 units of cement. Therefore, to produce x truckloads of Type 1 concrete requires $30x$ units of cement. Each truckload of Type 2 concrete requires 40 units of cement. Therefore, to produce y truckloads of Type 2 concrete requires $40y$ units of cement. The plant is supplied 2000 units of cement daily. Therefore, the constraint on the total cement consumed by the plant can be written as below:

$$30x + 40y \leq 2000 \quad (\text{B.1})$$

The plant is supplied 2500 units of aggregate and 4000 units of water daily. Along similar lines, the constraint on the total volume of aggregate and water consumed by the plant can be written as below:

$$50x + 20y \leq 2500 \quad (\text{B.2})$$

$$60x + 90y \leq 4000 \quad (\text{B.3})$$

In addition, the daily truckloads of Type 1 and Type 2 concrete produced has to be greater than or equal to zero. The non-negativity constraint can be represented as:

$$x \geq 0, y \geq 0 \quad (\text{B.4})$$

Each Type 1 concrete truckload generates a profit of \$90. Therefore, x truckloads of Type 1 concrete generates \$90 x profit. Each Type 2 concrete truckload generates a profit of \$120. Therefore, y truckloads of Type 2 concrete generates a profit of \$120 y . The goal of the decision maker is to maximize this daily profit which can be written as:

$$\max 90x + 120y \tag{B.5}$$

The entire problem can be summarized as below. The following set of equations represents an optimization formulation or a mathematical programming formulation for the concrete plant profit maximizing problem.

$$\begin{array}{ll} \min_{x,y} & 90x + 120y \\ \text{s.t.} & 50x + 20y \leq 2500 \\ & 60x + 90y \leq 4000 \\ & x, y \geq 0 \end{array}$$

Here x and y are written underneath max to indicate that these are the decision variables. In cases where it is obvious what the decision variables are, we sometimes omit writing them below the max or min in the objective. More complicated optimization problems might use letters to name things other than decision variables, and in those cases it is helpful to explicitly write down which variables the decision maker can affect. The abbreviation “s.t.” stands for “subject to” or “such that” and indicates the constraints.

The above example has linear objective functions and constraints. Optimization formulations can also have nonlinear functions as shown in the example below. The objective in the above example is to maximize the profit. When the decision maker is concerned with controlling costs rather than profit, the objective function often involves minimization.

Example B.2. *A business has the option of setting up concrete plants at two locations. The daily cost operating a plant at Location 1 per unit of production has two components: a fixed cost of 20 and an additional cost which increases by 0.5 for every unit of production. Similarly, the daily cost of operating a plant at Location 2 per unit production has a fixed cost of 12 and an additional cost which increases by 1.2 for every unit of production. The business has committed to supplying at least 12 units of concrete daily. What is the total amount to be produced at each location to minimize cost and satisfy demand?*

There are two decision variables in this problem: the production in plant at Location 1 and production in plant at Location 2. Let x denote the units of production in plant at Location 1 and y represent the units of production in plant at Location 2.

The business has agreed to supply at least 12 units of concrete daily. Therefore, the sum of production in plants at Location 1 and Location 2 must be greater than equal to 12.

$$x + y \geq 12 \quad (\text{B.6})$$

Also, the production at both plants cannot be negative which is represented as:

$$x \geq 0, y \geq 0 \quad (\text{B.7})$$

The daily cost of operating a plant at Location 1 per unit of production is $20 + 0.5x$. Since the plant produces x units of concrete, the total cost of operating a plant at Location 1 is $(20 + 0.5x)x$. Similarly, the total cost of operating a plant at Location 2 is $(12 + 1.2y)y$. The goal of the business is to minimize the total cost of operation which is given as:

$$\min (20 + 0.5x)x + (12 + 1.2y)y \quad (\text{B.8})$$

The optimization formulation can thus be summarized as below:

$$\begin{array}{ll} \min_{x,y} & 20x + 0.5x^2 + 12y + 1.2y^2 \\ \text{s.t.} & x + y \geq 12 \\ & x, y \geq 0 \end{array}$$

Optimization models are widely used in resource allocation where a fixed resource has to be allocated among various tasks to maximize or minimize objectives. A common variant of the resource allocation for civil engineers is the budget allocation problem. In the budget allocation problem, a fixed budget has to be distributed among various projects. The objective can vary depending on the nature of the decision maker. For example, private entities might be interested in maximizing returns or profit whereas public entities might be interested in maximizing social welfare or equity.

Example B.3. *A business is seeking to invest in two public-private partnership projects. Project 1 yields an expected return of \$10 and a standard deviation of \$5 for each dollar invested. Project 2 has an expected return of \$12 and a standard deviation of \$7 for each dollar invested. For any investment portfolio, the business's utility is described as $U = \mu - \frac{1}{2}\sigma$, where μ represents the average return on investment and σ the standard deviation of return. The business has a total of \$1000 to invest. Assume that the returns on investments on both projects are independent. Determine the amount of money to be invested in both projects so that the business can maximize its utility?*

There are two decision variables in this problem: the amount of money invested in Project 1 (x), and the amount of money invested in Project 2 (y).

We know the total amount of money available is \$1000. Therefore:

$$x + y \leq 1000 \quad (\text{B.9})$$

Also, the business cannot invest negative money in either project:

$$x \geq 0, y \geq 0 \quad (\text{B.10})$$

Project 1 yields an average return of \$10 for each dollar invested. Therefore, with an investment of \$ x , the average return is \$ $10x$. Project 2 yields an average return of \$12 for each dollar invested. Therefore, with an investment of \$ y , the average return is \$ $12y$. The total average return on investment is given as $10x + 12y$.

The standard deviation of return on project 1 is \$5 for each dollar invested and for project 2 is \$7 for each dollar invested. Therefore, the standard deviation of return on investments is given as $\sqrt{25x^2 + 49y^2}$.

The business's utility is given as: $U = \mu - \frac{1}{2}\sigma = (10x + 12y) - \frac{1}{2}\sqrt{25x^2 + 49y^2}$. The optimization formulation can be summarized as follows:

$$\begin{array}{ll} \min_{x,y} & 10x + 12y - 0.5\sqrt{25x^2 + 49y^2} \\ \text{s.t.} & x + y \leq 1000 \\ & x, y \geq 0 \end{array}$$

B.2 Index Notation

In the previous section, we used the symbols x and y to represent the decision variables. Each symbol corresponds to a scalar. In this section, we will develop an optimization formulation for a problem with more decision variables and constraints. You will notice that using scalar based symbols will become cumbersome as the size of the formulation increases. We will then introduce the set and index notations which will help represent the optimization formulation in a compact manner using more convenient notation. First, let us consider the following problem.

Example B.4. *A business has five factories located at Pittsburgh, Boston, Austin, Los Angeles, and Miami producing two types of window frames. It costs \$10 and \$26 to produce one unit of window frame of Type 1 and Type 2 respectively. The business has committed to supplying at least 300 units of window frame of Type 1 and at least 200 units of window frame of Type 2 each week. Due to the number of workers employed at each location, the maximum number of window frames which can be produced at Pittsburgh, Boston, Austin, Los Angeles, and Miami are 100, 125, 100, 125, and 50 respectively. The cost of producing one unit of window frame of Type 1 at Pittsburgh, Boston, Austin, Los Angeles, and Miami is \$10, \$10, \$25, \$30, and \$30 respectively. The cost of producing one unit of window frame of Type 2 at Pittsburgh, Boston, Austin, Los Angeles, and Miami is \$40, \$40, \$15, \$20, and \$20 respectively. Determine the number of window frames of Type 1 and Type 2 which the business has to produce at each location with the objective of minimizing cost?*

Let us first identify the decision variables. In this problem, the business has to decide how many units of window frames of Type 1 and Type 2 to produce at the five locations: Pittsburgh, Boston, Austin, Los Angeles, and Miami. Therefore, there are 10 decision variables.

Let a , b , c , d , and e represent the number of units of window frames of Type 1 to be produced per week at Pittsburgh, Boston, Austin, Los Angeles, and Miami respectively. Let p , q , r , s , and t represent the number of units of window frames of Type 2 to be produced per week at Pittsburgh, Boston, Austin, Los Angeles, and Miami respectively.

The business has committed to supplying at least 300 units of window frames of Type 1 per week. Therefore, the total amount of window frames of Type 1 produced per week must be greater than or equal to 300.

$$a + b + c + d + e \geq 300 \quad (\text{B.11})$$

The business has committed to supplying at least 200 units of window frames of Type 2 per week. Therefore, the total amount of window frames of Type 2 produced per week must be greater than or equal to 200.

$$p + q + r + s + t \geq 200 \quad (\text{B.12})$$

The Pittsburgh factory can produce a maximum of 100 units of window frames per week. Therefore, the sum of the total number of window frames of Type 1 produced per week at Pittsburgh and the total number of window frames of Type 2 produced per week at Pittsburgh must be less than or equal to 100.

$$a + p \leq 100 \quad (\text{B.13})$$

The factory at Boston can produce a maximum of 125 units of window frames per week. Therefore,

$$b + q \leq 125 \quad (\text{B.14})$$

The factory at Austin can produce a maximum of 100 units of window frames per week. Therefore,

$$c + r \leq 100 \quad (\text{B.15})$$

Along similar lines, the factories at Los Angeles and Miami are constrained to produce a maximum of 125 and 50 window frames respectively.

$$d + s \leq 125 \quad (\text{B.16})$$

$$e + t \leq 50 \quad (\text{B.17})$$

The number of window frames of both types produced per week at all locations has to be greater than or equal to zero. The non-negativity constraints are represented as:

$$a \geq 0, b \geq 0, c \geq 0, d \geq 0, e \geq 0 \quad (\text{B.18})$$

$$p \geq 0, q \geq 0, r \geq 0, s \geq 0, t \geq 0 \quad (\text{B.19})$$

The cost of producing a units of window frames of Type 1 and p units of window frames of Type 2 at Pittsburgh is $10a + 40p$. Along similar lines, the cost of producing specific number of window frames of both types at the other four locations can be determined. The total production cost is the sum of production costs at each location which is given as: $10a + 10b + 25c + 30d + 30e + 40p + 40q + 15r + 20s + 20t$. The objective is to minimize the total production costs.

The optimization formulation can thus be summarized as shown below

$$\begin{array}{ll} \min_{a, \dots, e, p, \dots, t} & 10a + 10b + 25c + 30d + 30e + 40p + 40q + 15r + 20s + 20t \\ \text{s.t.} & a + b + c + d + e \geq 300 \\ & p + q + r + s + t \geq 200 \\ & a + p \leq 100 \\ & b + q \leq 125 \\ & c + r \leq 100 \\ & d + s \leq 125 \\ & e + t \leq 50 \\ & a, b, c, d, e \geq 0 \\ & p, q, r, s, t \geq 0 \end{array}$$

This formulation is correct, but a bit unwieldy. To someone looking just at the formulation, it is hard to tell which variables refer to what. Index notation, presented in the following subsections, can improve the presentation of formulations like this one.

B.2.1 Single index notation

This section introduces sets and indices, and explains how to use notation based on them to present formulations concisely. First imagine that the business had factories at ten locations and was producing five types of window frames. We would need 50 different symbols to denote the window frame types produced at all locations! The formulation can be represented in a simpler way by using set and index notation. This section first presents the single index notation. A more compact multiple index notation is provided in the following section.

A set is a collection of similar objects. Let \mathcal{I} denote the set of all locations. For the above example, let

$$\mathcal{I} = \{\text{Pittsburgh, Boston, Austin, Los Angeles, Miami}\}. \quad (\text{B.20})$$

In the above example, we are given the names of locations of five factories. In some cases, we may not know or we may not be interested in the names of the locations. All we may know or care about is that there are five factories located in five different places. In such case, the set \mathcal{I} can be represented as

$$\mathcal{I} = \{1, 2, 3, 4, 5.\} \quad (\text{B.21})$$

The above notation can become cumbersome if the number of locations is high. So a shorter form representation is

$$\mathcal{I} = \{1, \dots, 5.\} \quad (\text{B.22})$$

The above notation can be further generalized for any value n denoting number of locations as

$$\mathcal{I} = \{1, \dots, n.\} \quad (\text{B.23})$$

An index is used to refer to any element of the set. The symbol \in represents “an element of”. Therefore, $i \in \mathcal{I}$ denotes an index i which is an element of the set \mathcal{I} . Thus when $\mathcal{I} = \{\text{Pittsburgh, Boston, Austin, Los Angeles, Miami}\}$, i can refer to any of Pittsburgh, Boston, Austin, Los Angeles, or Miami. Indices can be combined with symbols to represent decision variables and input parameters in a concise manner.

In the previous example, we used the symbols a, b, c, d , and e to represent the number of units of window frames of Type 1 to be produced per week at Pittsburgh, Boston, Austin, Los Angeles, and Miami respectively. Using the set-index notation, let x_i denote the number of units of window frames per week of Type 1 to be produced at location $i \in \mathcal{I}$. Thus a corresponds to $x_{\text{Pittsburgh}}$, b corresponds to x_{Boston} , and so on.

The business has to produce at least 300 units of window frames of Type 1 per week. This constraint can be represented as:

$$x_{\text{Pittsburgh}} + x_{\text{Boston}} + x_{\text{Austin}} + x_{\text{Los Angeles}} + x_{\text{Miami}} \geq 300 \quad (\text{B.24})$$

The above constraint can be succinctly represented using the summation operator Σ as

$$\sum_{i \in \mathcal{I}} x_i \geq 300. \quad (\text{B.25})$$

If the set $\mathcal{I} = \{1, \dots, 5\}$, then the above constraint can also be represented using either of the following equations:

$$\sum_{i=1}^5 x_i \geq 300 \quad (\text{B.26})$$

$$\sum_{1 \leq i \leq 5} x_i \geq 300. \quad (\text{B.27})$$

Similarly, let y_i denote the number of units of window frames per week of Type 2 to be produced at location $i \in \mathcal{I}$. As before, the constraint on minimum

number of units of Type 2 window frames produced can be written as any of the following::

$$\sum_{i \in \mathcal{I}} y_i \geq 200 \quad (\text{B.28})$$

$$\sum_{i=1}^5 y_i \geq 200 \quad (\text{B.29})$$

$$\sum_{1 \leq i \leq 5} y_i \geq 200 \quad (\text{B.30})$$

Now, look at the constraints which limits the number of window frames produced at each location. Consider the case where the city names are written explicitly, so $\mathcal{I} = \{\text{Pittsburgh, Boston, Austin, Los Angeles, Miami}\}$. The constraints are:

$$x_{\text{Pittsburgh}} + y_{\text{Pittsburgh}} \leq 100 \quad (\text{B.31})$$

$$x_{\text{Boston}} + y_{\text{Boston}} \leq 125 \quad (\text{B.32})$$

$$x_{\text{Austin}} + y_{\text{Austin}} \leq 100 \quad (\text{B.33})$$

$$x_{\text{Los Angeles}} + y_{\text{Los Angeles}} \leq 125 \quad (\text{B.34})$$

$$x_{\text{Miami}} + y_{\text{Miami}} \leq 50 \quad (\text{B.35})$$

This will become very cumbersome to write as the number of locations and window frame types increase. Let u_i represent the maximum amount of window frame which can be produced at location $i \in \mathcal{I}$. For example, $u_{\text{Pittsburgh}} = 100$, $u_{\text{Los Angeles}} = 125$. The symbol \forall represents “for all”. Therefore, $\forall i \in \mathcal{I}$ implies for all the elements in the set \mathcal{I} . All of these constraints can then be written as a single equation:

$$x_i + y_i \leq u_i \quad \forall i \in \mathcal{I} \quad (\text{B.36})$$

The above equation denotes for each element $i \in \mathcal{I}$, the constraints $x_i + y_i \leq u_i$ holds. Similarly the non-negativity constraints can be written as:

$$x_i \geq 0 \quad \forall i \in \mathcal{I} \quad (\text{B.37})$$

$$y_i \geq 0 \quad \forall i \in \mathcal{I} \quad (\text{B.38})$$

When $\mathcal{I} = \{1, \dots, 5\}$, the production limit at each location and non-negativity constraints can also be represented using either of the two following set of equations:

$$x_i + y_i \leq u_i \quad \forall i = 1, \dots, 5 \quad (\text{B.39})$$

$$x_i \geq 0 \quad \forall i = 1, \dots, 5 \quad (\text{B.40})$$

$$y_i \geq 0 \quad \forall i = 1, \dots, 5 \quad (\text{B.41})$$

or:

$$x_i + y_i \leq u_i \quad \forall 1 \leq i \leq 5 \quad (\text{B.42})$$

$$x_i \geq 0 \quad \forall 1 \leq i \leq 5 \quad (\text{B.43})$$

$$y_i \geq 0 \quad \forall 1 \leq i \leq 5 \quad (\text{B.44})$$

The objective function for the formulation was:

$$\min 10a + 10b + 25c + 30d + 30e + 40p + 40q + 15r + 20s + 20t \quad (\text{B.45})$$

Using the index-set notation, the objective function can be written

$$\begin{aligned} \min & 10x_{\text{Pittsburgh}} + 10x_{\text{Boston}} + 25x_{\text{Austin}} + 30x_{\text{Los Angeles}} + 30x_{\text{Miami}} \\ & + 40y_{\text{Pittsburgh}} + 40y_{\text{Boston}} + 15y_{\text{Austin}} + 20y_{\text{Los Angeles}} + 20y_{\text{Miami}}. \end{aligned} \quad (\text{B.46})$$

Let v_i represent the cost of producing one window frame of Type 1 at location $i \in \mathcal{I}$ and w_i represent the cost of producing one window frame of Type 2 at location $i \in \mathcal{I}$. For example $v_{\text{Miami}} = 30$, $w_{\text{Miami}} = 20$. The objective function can now be succinctly represented as:

$$\min \sum_{i \in \mathcal{I}} (v_i x_i + w_i y_i) \quad (\text{B.47})$$

When $\mathcal{I} = \{1, \dots, 5\}$ the objective function can also be represented as:

$$\min \sum_{i=1}^5 (v_i x_i + w_i y_i) \quad (\text{B.48})$$

or:

$$\min \sum_{1 \leq i \leq 5} (v_i x_i + w_i y_i) \quad (\text{B.49})$$

Therefore, the formulation can be rewritten as

$$\begin{aligned} \min_{x_i, y_i} & \sum_{i \in \mathcal{I}} (v_i x_i + w_i y_i) \\ \text{s.t.} & \sum_{i \in \mathcal{I}} x_i \geq 300 \\ & \sum_{i \in \mathcal{I}} y_i \geq 200 \\ & x_i + y_i \leq u_i \quad \forall i \in \mathcal{I} \\ & x_i \geq 0 \quad \forall i \in \mathcal{I} \\ & y_i \geq 0 \quad \forall i \in \mathcal{I} \end{aligned}$$

or equivalently, with $\sum_{i=1}^5$ or $\sum_{1 \leq i \leq 5}$ on the summations.

In addition to making the formulation more compact, the set index notation also makes the formulation easier to understand, and easier to change (if there were more cities, all we would have to change is the definition of the set \mathcal{I} or the number 5 to whatever the new number of cities is). Notice also that now it is important to specify that x_i and y_i are the decision variables: u_i , v_i , and w_i now represent *given* problem data which we cannot change.

B.2.2 Multiple index notation

This section introduces a notation that further simplifies the example presented in the previous subsection. Previously we had set

$$\mathcal{I} = \{\text{Pittsburgh, Boston, Austin, Los Angeles, Miami}\}$$

or $\mathcal{I} = \{1, \dots, 5\}$. Two sets of decision variables x_i , y_i were used to represent window frames of Type 1 and 2 produced at all locations $i \in \mathcal{I}$. If the business was producing ten types of window frames, then even the single index notation becomes cumbersome, as we would need ten different subscripted symbols, x_i, y_i, z_i, \dots and so on.

This issue can be addressed using multiple indices. Let us introduce the set \mathcal{J} to model the two types of window frames. Similar to the set of locations, \mathcal{J} can be defined in two ways: $\mathcal{J} = \{\text{Type 1, Type 2}\}$ or $\mathcal{J} = \{1, 2\}$. Let j be an index referring to any element in \mathcal{J} , i.e., $j \in \mathcal{J}$.

Instead of using two symbols with subscripts for location, we will define the decision variable using one symbol with two subscripts. Let x_{ij} denote the number of type $j \in \mathcal{J}$ window frames produced per week at location $i \in \mathcal{I}$.

When the sets were $\mathcal{I} = \{\text{Pittsburgh, Boston, Austin, Los Angeles, Miami}\}$ and $\mathcal{J} = \{\text{Type 1, Type 2}\}$, in the single index notation two equations (one for Type 1, other for Type 2) were used to represent the total production must be greater than equal to demand.

$$\sum_{i \in \mathcal{I}} x_i \geq 300 \tag{B.50}$$

$$\sum_{i \in \mathcal{I}} y_i \geq 200 \tag{B.51}$$

Using the double index notation, the two equations can be summarized into one equation as shown below, by introducing b_j to be the demand for window frames of type $j \in \mathcal{J}$. For this example, $b_{\text{Type 1}} = 300$ and $b_{\text{Type 2}} = 200$.

$$\sum_{i \in \mathcal{I}} x_{ij} \geq b_j \quad \forall j \in \mathcal{J} \tag{B.52}$$

Pay close attention to the two set element references, $\forall j \in \mathcal{J}$ on the right hand side and $i \in \mathcal{I}$ underneath the summation operator. The $\forall j \in \mathcal{J}$ on the right hand side ensures that the equation $\sum_{i \in \mathcal{I}}()$ is repeated for each element $j \in \mathcal{J}$ as shown below.

$$x_{\text{Pittsburgh},j} + x_{\text{Boston},j} + x_{\text{Austin},j} + x_{\text{Los Angeles},j} + x_{\text{Miami},j} \geq b_j \quad \forall j \in \mathcal{J} \quad (\text{B.53})$$

Since the set \mathcal{J} has two elements, the equation is repeated twice representing the demand needing to be met for Type 1 and Type 2 window frames. In the single index notation $x_i + y_i \leq u_i \quad \forall i \in \mathcal{I}$ is used to represent the constraint on maximum window frames which can be produced at each location. In the double index notation, the left hand side can be made more compact using a summation operator as shown below.

$$\sum_{j \in \mathcal{J}} x_{ij} \leq u_i \quad \forall i \in \mathcal{I} \quad (\text{B.54})$$

Pay close attention to the indices over which the summation ($j \in \mathcal{J}$) is happening and the elements on the right hand side ($i \in \mathcal{I}$). The above expression repeats the following equation for each element or each location $i \in \mathcal{I}$.

$$x_{i,\text{Type 1}} + x_{i,\text{Type 2}} \leq u_i \quad \forall i \in \mathcal{I} \quad (\text{B.55})$$

In the single index notation, two sets of equations were used to represent the non-negativity conditions - $x_i \geq 0 \quad \forall i \in \mathcal{I}$ and $y_i \geq 0 \quad \forall i \in \mathcal{I}$. This can be concisely represented using a single equation as follows:

$$x_{ij} \geq 0 \quad \forall i \in \mathcal{I}, j \in \mathcal{J} \quad (\text{B.56})$$

You are enforcing x_{ij} to be ≥ 0 for each element or location $i \in \mathcal{I}$ as well as window frame type $j \in \mathcal{J}$. The order in which you reference the elements and sets on the right hand side after \forall does not matter, i.e., the following two equations represent the same non-negativity constraints.

$$x_{ij} \geq 0 \quad \forall i \in \mathcal{I}, j \in \mathcal{J} \quad (\text{B.57})$$

$$x_{ij} \geq 0 \quad \forall j \in \mathcal{J}, i \in \mathcal{I} \quad (\text{B.58})$$

Using the single index notation, the objective function was represented as $\min \sum_{i \in \mathcal{I}} (v_i x_i + w_i y_i)$. In the double index notation the objective function can be represented as $\min \sum_{i \in \mathcal{I}} (v_i x_{i,\text{Type 1}} + w_i x_{i,\text{Type 2}})$. We can make the representation even more compact and more intuitive by defining v_{ij} as the cost of producing one window frame of type $j \in \mathcal{J}$ at location $i \in \mathcal{I}$. The objective function can then be defined as:

$$\min \sum_{i \in \mathcal{I}} \sum_{j \in \mathcal{J}} v_{ij} x_{ij} \quad (\text{B.59})$$

Note that the order in which we sum the objective function does not matter, i.e., $\sum_{i \in \mathcal{I}} \sum_{j \in \mathcal{J}} v_{ij} x_{ij} = \sum_{j \in \mathcal{J}} \sum_{i \in \mathcal{I}} v_{ij} x_{ij}$. (See Section A.1).

The formulation in double index notation is finally:

$$\begin{aligned}
\min_{x_{ij}} \quad & \sum_{i=1}^5 \sum_{j=1}^2 v_{ij} x_{ij} \\
\text{s.t.} \quad & \sum_{i \in \mathcal{I}} x_{ij} \geq b_j \quad \forall j \in \mathcal{J} \\
& \sum_{j \in \mathcal{J}} x_{ij} \leq u_i \quad \forall i \in \mathcal{I} \\
& x_{ij} \geq 0 \quad \forall i \in \mathcal{I}, j \in \mathcal{J}
\end{aligned}$$

B.2.3 Additional information on index notation

This section provides a brief overview of some mathematical notation which is commonly seen in the optimization literature.

The set \mathbb{R} contains all real numbers. The set of all non-negative real numbers is commonly represented as \mathbb{R}_+ . Another way to represent the non-negativity constraint $x_{ij} \geq 0 \quad \forall i \in \mathcal{I}, j \in \mathcal{J}$ is shown below:

$$x_{ij} \in \mathbb{R}_+ \quad \forall i \in \mathcal{I}, j \in \mathcal{J} \quad (\text{B.60})$$

In the above equation each decision variable x_{ij} is restricted to lie in the set of non-negative real numbers. If the decision variables can be positive or negative real numbers or zero, then the above equation can be modified as:

$$x_{ij} \in \mathbb{R} \quad \forall i \in \mathcal{I}, j \in \mathcal{J} \quad (\text{B.61})$$

The set \mathbb{Z} contains the set of all integers.² The set of all non-negative integers is commonly represented as \mathbb{Z}_+ . In certain type of optimization problems, called integer programs, the decision variables are restricted to be integers or non-negative integers which can be represented as follows:

$$x_{ij} \in \mathbb{Z} \quad \forall i \in \mathcal{I}, j \in \mathcal{J} \quad (\text{B.62})$$

$$x_{ij} \in \mathbb{Z}_+ \quad \forall i \in \mathcal{I}, j \in \mathcal{J} \quad (\text{B.63})$$

In some cases, there will be certain constraints defined on a subset of indices. Let us assume in the above example, all the window frames produced on the east coast of the United States are first transported to a warehouse before delivery to retailers. The warehouse can store only 150 window frames each week. This restriction can be represented by the following constraint equation:

$$x_{\text{Boston, Type 1}} + x_{\text{Boston, Type 2}} + x_{\text{Miami, Type 1}} + x_{\text{Miami, Type 2}} \leq 150 \quad (\text{B.64})$$

The above constraint can be more concisely represented by defining a subset $\mathcal{K}(\mathcal{I}) = \{i \in \mathcal{I} : i \text{ lies on East Coast}\}$. In this expression $:$ corresponds to

²The use of the letter ‘Z’ is from the German word “Zahlen,” meaning “numbers.”

“such that”. Another way to present this information is using the $|$ symbol. $\mathcal{K}(\mathcal{I}) = \{i \in \mathcal{I} | i \text{ lies on East Coast}\}$. Given the definition of the subset $\mathcal{K}(\mathcal{I})$ (which can also be represented as $\mathcal{K}_{\mathcal{I}}$), the constraint can be more succinctly presented as:

$$\sum_{k \in \mathcal{K}(\mathcal{I})} \sum_{j \in \mathcal{J}} x_{kj} \leq 150 \quad (\text{B.65})$$

The set notation can also be used to represent various mathematical expressions of decision variables in a clean manner. Let $\mathcal{I} = \{1, \dots, 10\}$. We want to represent the expression $x_2 + x_4 + x_6 + x_8 + x_{10}$ in a concise manner. One way to do this would be to define $\mathcal{K}(\mathcal{I}) = \{i \in \mathcal{I} : i \text{ is even}\}$ and then write:

$$x_2 + x_4 + x_6 + x_8 + x_{10} = \sum_{k \in \mathcal{K}(\mathcal{I})} x_k \quad (\text{B.66})$$

Another way to represent this would be to indicate the condition expression in the summation operator itself

$$x_2 + x_4 + x_6 + x_8 + x_{10} = \sum_{\{i \in \mathcal{I} : i \text{ is even}\}} x_i = \sum_{\{i \in \mathcal{I} | i \text{ is even}\}} x_i \quad (\text{B.67})$$

Along similar lines:

$$x_3 + x_4 + x_5 = \sum_{\{i \in \mathcal{I} : 3 \leq i \leq 5\}} x_i = \sum_{\{i \in \mathcal{I} | 3 \leq i \leq 5\}} x_i \quad (\text{B.68})$$

$$x_7 + x_8 + x_9 + x_{10} = \sum_{\{i \in \mathcal{I} : i \geq 7\}} x_i = \sum_{\{i \in \mathcal{I} | i \geq 7\}} x_i \quad (\text{B.69})$$

$$(\text{B.70})$$

B.3 Vector and Matrix Notations

Vector and matrix notation, introduced in Section A.2, are also very widely used to represent optimization formulations concisely. This section explains these conventions. To start, when we write equations or inequalities involving vectors, we mean that they apply to every component of those vectors. For example, $\mathbf{x} \geq \mathbf{b}$, means that *every* element in \mathbf{x} is greater than or equal to the corresponding element in \mathbf{b} .

$$\mathbf{x} \geq \mathbf{b} \implies \begin{cases} x_1 \geq b_1 \\ x_2 \geq b_2 \\ \vdots \\ x_n \geq b_n \end{cases} \quad (\text{B.71})$$

Thus $\mathbf{x} \geq \mathbf{0}$ implies each element of \mathbf{x} is greater than or equal to zero. Similarly,

$$\mathbf{Ax} = \mathbf{b} \implies \begin{cases} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = b_2 \\ \vdots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n = b_n \end{cases} \quad (\text{B.72})$$

Also,

$$\mathbf{b}^T \mathbf{x} = b_1x_1 + b_2x_2 + \cdots + b_nx_n = \sum_{i=1}^n b_i x_i \quad (\text{B.73})$$

is another way to express the dot product of two vectors \mathbf{x} and \mathbf{b} .

Given the above information, we now reformulate the window frame optimization formulation using vectors and matrices. Let $\mathbf{x} = (x_1, \dots, x_{10})$ represent a column vector of decision variables, i.e.,

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \dots \\ x_{10} \end{bmatrix} \quad (\text{B.74})$$

In the vector \mathbf{x} , let x_1, x_2, x_3, x_4 , and x_5 represent the five decision variables corresponding to the amount of window frame of Type 1 produced at the five locations and x_6, x_7, x_8, x_9 , and x_{10} represent the five decision variables corresponding to the amount of window frame of Type 2 produced at the five locations. Let $\mathbf{c} = (c_1, \dots, c_{10})$ represent a column vector of costs where c_1, c_2, c_3, c_4 , and c_5 represent the cost of producing one unit of Type 1 window frame at the five different locations and c_6, c_7, c_8, c_9 , and c_{10} represent the cost of producing one unit of Type 2 window frame at the five different locations.

$$\mathbf{c} = \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \\ c_6 \\ c_7 \\ c_8 \\ c_9 \\ c_{10} \end{bmatrix} = \begin{bmatrix} 10 \\ 10 \\ 25 \\ 30 \\ 30 \\ 40 \\ 40 \\ 15 \\ 20 \\ 20 \end{bmatrix} \quad (\text{B.75})$$

The objective function in this case is:

$$\min 10x_1 + 10x_2 + 25x_3 + 30x_4 + 30x_5 + 40x_6 + 40x_7 + 15x_8 + 20x_9 + 20x_{10} \quad (\text{B.76})$$

Let \mathbf{c}^T represent the transpose of the vector \mathbf{c} , so \mathbf{c}^T is a row vector. The objective function can then be compactly represented as:

$$\min \mathbf{c}^T \mathbf{x} = \sum_{i=1}^{10} c_i x_i \quad (\text{B.77})$$

The production of window frame of Type 1 must be greater than 300 and window frame of Type 2 must be higher than 200.

$$x_1 + x_2 + x_3 + x_4 + x_5 \geq 300 \quad (\text{B.78})$$

$$x_6 + x_7 + x_8 + x_9 + x_{10} \geq 200 \quad (\text{B.79})$$

Let us represent the above set of constraints in a compact manner. Define a matrix \mathbf{A} as:

$$\mathbf{A} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \quad (\text{B.80})$$

Let \mathbf{b} be a column vector of demands.

$$\mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} = \begin{bmatrix} 300 \\ 200 \end{bmatrix} \quad (\text{B.81})$$

The constraints can now be represented as

$$\mathbf{Ax} = \begin{bmatrix} x_1 + x_2 + x_3 + x_4 + x_5 \\ x_6 + x_7 + x_8 + x_9 + x_{10} \end{bmatrix} \geq \begin{bmatrix} 300 \\ 200 \end{bmatrix} = \mathbf{b} \quad (\text{B.82})$$

or in a compact form, simply as:

$$\mathbf{Ax} \geq \mathbf{b} \quad (\text{B.83})$$

The constraints on the number of window frames produced at each location is given as:

$$x_1 + x_6 \leq 100 \quad (\text{B.84})$$

$$x_2 + x_7 \leq 125 \quad (\text{B.85})$$

$$x_3 + x_8 \leq 100 \quad (\text{B.86})$$

$$x_4 + x_9 \leq 125 \quad (\text{B.87})$$

$$x_5 + x_{10} \leq 50 \quad (\text{B.88})$$

Define a column vector \mathbf{u} and matrix \mathbf{U} as:

$$\mathbf{u} = \begin{bmatrix} 100 \\ 125 \\ 100 \\ 125 \\ 50 \end{bmatrix} \quad (\text{B.89})$$

$$\mathbf{U} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \quad (\text{B.90})$$

Thus the production at each location being lesser than the capacity can be written in a compact manner as:

$$\mathbf{U}\mathbf{x} \leq \mathbf{u} \quad (\text{B.91})$$

The nonnegativity constraints can be represented as $\mathbf{x} \geq \mathbf{0}$. Thus the final optimization formulation can be given as:

$$\begin{array}{ll} \min_{\mathbf{x}} & \mathbf{c}^T \mathbf{x} \\ \text{s.t.} & \mathbf{A}\mathbf{x} \geq \mathbf{b} \\ & \mathbf{U}\mathbf{x} \leq \mathbf{u} \\ & \mathbf{x} \geq \mathbf{0} \end{array}$$

Another way to represent the non-negativity constraints is $\mathbf{x} \in \mathbb{R}_+^{10}$. In the general case, when the decision variable vector \mathbf{x} has n elements, the nonnegativity constraints can be represented as $\mathbf{x} \in \mathbb{R}_+^n$. If the decision variables are restricted to the set of positive integers, then $\mathbf{x} \in \mathbb{Z}_+^n$.

B.4 Examples of Basic Optimization Problems

This section provides several examples of optimization problems. The first is a classic optimization problem known as the *transportation problem*.

Example B.5. (Transportation problem.) *A timber company has three mills which produces wooden frames and five markets. The three mills can produce 20, 40, and 30 units of wooden frames respectively on a daily basis. The daily demands for the wooden frames at the five markets are 15, 30, 20, 15, and 10 respectively. The cost to transport the wooden frames from the three mills to the five markets are shown below:*

Mill Location/Market	1	2	3	4	5
1	22	13	24	31	47
2	35	24	11	27	38
3	44	33	25	11	26

Formulate an optimization problem for satisfying demands at the markets while minimizing the transportation costs from the mills.

Let \mathcal{I} and \mathcal{J} denote the set of mill and market locations, respectively. The decision variables in this problem are x_{ij} , each of which represents the volume of wooden frames to be sent from location $i \in \mathcal{I}$ to location $j \in \mathcal{J}$. For example,

x_{23} corresponds to the volume of wooden frames to be sent from Mill 2 to Market 3.

Let u_i represent the amount of wooden frames which can be produced at mill $i \in \mathcal{I}$ and d_j represent the amount of wooden frames needed at market $j \in \mathcal{J}$. For example, $u_3 = 30$ and $d_4 = 15$.

At each mill, the total volume of wooden frames transported must be less than or equal to the capacity of the mill. For example, at Mill 2:

$$x_{21} + x_{22} + x_{23} + x_{24} + x_{25} \leq 40 \quad (\text{B.92})$$

This can be represented generally as:

$$\sum_{j \in \mathcal{J}} x_{ij} \leq u_i \quad \forall i \in \mathcal{I} \quad (\text{B.93})$$

Similarly, demand at each market must be met. The demand constraint can be represented as:

$$\sum_{i \in \mathcal{I}} x_{ij} \geq d_j \quad \forall j \in \mathcal{J} \quad (\text{B.94})$$

In addition, the volume of wooden frames transported between mill and market locations cannot be negative.

$$x_{ij} \geq 0 \quad \forall i \in \mathcal{I}, \forall j \in \mathcal{J} \quad (\text{B.95})$$

Let c_{ij} represent the cost to transport a wooden frame from mill $i \in \mathcal{I}$ to location $j \in \mathcal{J}$. For example $c_{24} = 27$. The objective is to minimize the total transportation costs which is:

$$\min \sum_{i \in \mathcal{I}} \sum_{j \in \mathcal{J}} c_{ij} x_{ij} \quad (\text{B.96})$$

The final formulation for the transportation problem can be summarized as:

$$\begin{aligned} \min_{x_{ij}} \quad & \sum_{i \in \mathcal{I}} \sum_{j \in \mathcal{J}} c_{ij} x_{ij} \\ \text{s.t.} \quad & \sum_{j \in \mathcal{J}} x_{ij} \leq u_i \quad \forall i \in \mathcal{I} \\ & \sum_{i \in \mathcal{I}} x_{ij} \geq d_j \quad \forall j \in \mathcal{J} \\ & x_{ij} \geq 0 \quad \forall i \in \mathcal{I}, \forall j \in \mathcal{J} \end{aligned}$$

This formulation is an example of a *linear program*. In a linear program, the objective functions and constraints are linear functions of the decision variables, and the solution can be any real number satisfying the constraints (it does not have to be an integer). Notice that in the above example, we are not restricting the number of wooden frames to be transported between each pair of mill and market to be integer. However, the solution to this specific linear program will always yield integer solutions as long as the input data is integer, although this is not true in general (see Section C.5.2).

We now modify this problem by considering an additional factor.

Example B.6. *The trucks used in transporting wooden frames from Mill 1 are very old, the frames may be damaged because of their poor suspension systems. To prevent this, additional packing material is needed, and the amount depends on the destination market location. The table below shows the amount of packing material per frame for each mill and market combination (Mills 2 and 3 have newer trucks that do not require special packaging.)*

Mill Location/Market	1	2	3	4	5
1	3	7	3	1	0
2	0	0	0	0	0
3	0	0	0	0	0

If Mill 1 has 21 units of packing material available each day, formulate the problem of meeting demands while minimizing transportation costs.

Let r_{ij} represent the amount of packaging needed to transport a wooden frame from mill $i \in \mathcal{I}$ to market $j \in \mathcal{J}$. For example, $r_{12} = 7$ and $r_{34} = 0$. Let R_i represent the total packing material available at mill i each day, so $R_1 = 21$. R_2 and R_3 can be assigned the value zero, because they do not need to have any packing material available. The linear transportation problem formulation can be modified by adding an additional resource constraint as shown below:

$$\begin{aligned}
 \min_{x_{ij}} \quad & \sum_{i \in \mathcal{I}} \sum_{j \in \mathcal{J}} c_{ij} x_{ij} \\
 \text{s.t.} \quad & \sum_{j \in \mathcal{J}} x_{ij} \leq u_i \quad \forall i \in \mathcal{I} \\
 & \sum_{i \in \mathcal{I}} x_{ij} \geq d_j \quad \forall j \in \mathcal{J} \\
 & \sum_{j \in \mathcal{J}} r_{ij} x_{ij} \leq R_i \quad \forall i \in \mathcal{I} \\
 & x_{ij} \geq 0 \quad \forall i \in \mathcal{I}, \forall j \in \mathcal{J}
 \end{aligned}$$

It turns out that adding the packing material constraint changes the optimization problem in such a way that the optimal solutions may not be integers. If this condition is important, we must enforce it with an additional constraint, by replacing

$$x_{ij} \geq 0 \quad \forall i \in \mathcal{I}, \forall j \in \mathcal{J} \tag{B.97}$$

with

$$x_{ij} \in \mathbb{Z}_+ \quad \forall i \in \mathcal{I}, \forall j \in \mathcal{J} \tag{B.98}$$

The formulation now becomes an *integer program*. The methods needed to solve integer programs are different than the methods used to solve linear program, and integer programs are much harder to solve. Sometimes, it may be adequate to solve the problem as a linear program and then convert its optimal solution to an integral one, say, by rounding — if the values of the decision

variables are in the hundreds or thousands, the effect of rounding is likely small. However, for some integer programs this can lead to very poor solutions.

We now return to the original transportation problem formulation of Example B.5 without the packing material constraint. In that example, the objective functions and constraints are all linear functions of the decision variables. In many real world applications, it might not be possible to use linear functions to represent the objective function or constraints. The next example modifies the transportation costs to reflect “diseconomies of scale,” where the unit cost of shipping increases with the quantity (perhaps the most efficient trucks are used first, but as more and more frames are shipped, you have to start using older and less fuel-efficient trucks).

Example B.7. Assume now that the unit cost of transporting wooden frames between mill $i \in \mathcal{I}$ and market $j \in \mathcal{J}$ is $c_{ij} + x_{ij}$. (For example, if 10 wooden frames are being transported between mill $i \in \mathcal{I}$ and location $j \in \mathcal{J}$, then the cost of transporting each frame between the two locations is $c_{ij} + 10$, and the total transportation cost would be $(c_{ij} + 10) \times 10$). Formulate an optimization problem to meet the demands at the markets while minimizing transportation costs.

In this formulation, if x_{ij} wooden frames are being transported then the total transportation cost between mill $i \in \mathcal{I}$ and market $j \in \mathcal{J}$ is $(c_{ij} + x_{ij}) \times x_{ij}$. The constraints are the same, but the objective function now changes:

$$\begin{aligned} \min_{x_{ij}} \quad & \sum_{i \in \mathcal{I}} \sum_{j \in \mathcal{J}} (c_{ij} + x_{ij}) x_{ij} \\ \text{s.t.} \quad & \sum_{j \in \mathcal{J}} x_{ij} \leq u_i \quad \forall i \in \mathcal{I} \\ & \sum_{i \in \mathcal{I}} x_{ij} \geq d_j \quad \forall j \in \mathcal{J} \\ & x_{ij} \geq 0 \quad \forall i \in \mathcal{I}, \forall j \in \mathcal{J} \end{aligned}$$

In this formulation, the objective function is no longer linear in the decision variables. The above formulation is an example of a *nonlinear program*. In a nonlinear program, either the objective function or any of the constraints is nonlinear. If we wanted to enforce the decision variables to be integers, we would need to replace

$$x_{ij} \geq 0 \quad \forall i \in \mathcal{I}, \forall j \in \mathcal{J} \tag{B.99}$$

with

$$x_{ij} \in \mathbb{Z}_+ \quad \forall i \in \mathcal{I}, \forall j \in \mathcal{J} \tag{B.100}$$

in the final formulation. This leads to a *nonlinear integer program*, which again requires different solution methods.

In the linear variant of the transportation problem, we assume a unit cost of transportation between each mill and market location. For example, the unit cost of transportation between mill 2 and market 3 is 11. Therefore, the total

cost of transporting 10 wooden frames between mill 2 and market 3 is 110. The total cost of transporting 50 wooden frames between mill 2 and market 3 is 550. However in the real world, there is often discounts available in bulk transport. For example, the unit cost of transportation might be 11 up to 30 wooden frames and 5 for the wooden frames above 30. This is called economy of scale. Therefore, the total cost of transporting 50 wooden frames would be $30 \times 11 + 20 \times 5 = 430$. The presence of economy of scale in transportation makes the linear variant of the transportation problem an

Integer programs are also used in modeling selections or incorporating “yes or no” decisions. In such problems often a decision variable which can only take the values 0 (for “no”) or 1 (for “yes”) is used to model the selection decisions. This is called a *binary variable*.

Example B.8. *The timber company is now interested in establishing factories to produce wooden doors. The potential sites for establishing the factories are the same as the current locations of the timber mills. There is a fixed cost (which has been amortized to a daily cost) of 300, 350, and 450 associated with establishing the factories at the three locations. Factories built at each site would respectively have daily production capacities of 40, 80, and 60 doors. The daily demands for the wooden doors at the five markets are 35, 15, 50, 20, and 40 respectively. The unit cost of transporting wooden frames is assumed to be the same as the unit cost of transporting wooden frames which are given in the linear transportation example. Formulate an optimization problem to meet the demands at the markets while minimizing the facility location and transportation costs.*

As in Example B.5, let \mathcal{I} and \mathcal{J} represent the set of potential factory locations and markets respectively. There are two sets of decision variables. The first set of decision variables y_i takes the value 1 if a facility is located at $i \in \mathcal{I}$ and 0 otherwise. For example the values $y_1 = 1$, $y_2 = 0$, and $y_3 = 1$ would mean that factories are opened at sites 1 and 3. The second set of decision variables x_{ij} represents the volume of demand at market $j \in \mathcal{J}$ served by a facility at location $i \in \mathcal{I}$.

Let d_j represent the demand for wooden doors at market $j \in \mathcal{J}$. Therefore $d_1 = 35, d_2 = 15$ and so on. Let u_i represent the capacity of the factory at location $i \in \mathcal{I}$. Note that $u_1 = 40, u_2 = 80, u_3 = 60$.

If a factory is located at $i \in \mathcal{I}$, then the total volume of wooden doors supplied to all markets cannot exceed the production capacity of the factory. If a factory is not located at $i \in \mathcal{I}$, then the total volume of total volume of wooden doors supplied to all markets must be zero. This constraint can be represented as

$$\sum_{j \in \mathcal{J}} x_{ij} \leq u_i y_i \quad \forall i \in \mathcal{I}. \quad (\text{B.101})$$

Note that when $y_i = 1$, the right hand side becomes u_i (total production at an open factory cannot exceed its capacity). When $y_i = 0$ the right hand side

becomes zero (nothing can be produced at a factory which was never opened). Similar to the transportation problem, the total volume supplied to each market location must meet the demand:

$$\sum_{i \in \mathcal{I}} x_{ij} \geq d_i \quad \forall j \in \mathcal{J}. \quad (\text{B.102})$$

In addition to the non-negativity constraints on the variable x_{ij} , there is also an additional binary restriction on the facility location decision variable y_i .

$$x_{ij} \geq 0 \quad \forall i \in \mathcal{I}, \forall j \in \mathcal{J} \quad (\text{B.103})$$

$$y_i \in \{0, 1\} \quad \forall i \in \mathcal{I} \quad (\text{B.104})$$

The objective function has two cost components: the cost of establishing the facility and the transportation costs. The expression for the transportation costs is the same as in the linear transportation problem example. Let f_i represent the cost of establishing a facility at location $i \in \mathcal{I}$. The total facility location cost can be given as $\sum_{i \in \mathcal{I}} f_i y_i$. Therefore the objective function is:

$$\min \sum_{i \in \mathcal{I}} f_i y_i + \sum_{i \in \mathcal{I}} \sum_{j \in \mathcal{J}} (c_{ij} + x_{ij}) x_{ij} \quad (\text{B.105})$$

The final formulation is

$$\begin{aligned} \min_{x_{ij}, y_i} \quad & \sum_{i \in \mathcal{I}} f_i y_i + \sum_{i \in \mathcal{I}} \sum_{j \in \mathcal{J}} (c_{ij} + x_{ij}) x_{ij} \\ \text{s.t.} \quad & \sum_{j \in \mathcal{J}} x_{ij} \leq u_i y_i \quad \forall i \in \mathcal{I} \\ & \sum_{i \in \mathcal{I}} x_{ij} \geq d_i \quad \forall j \in \mathcal{J} \\ & x_{ij} \geq 0 \quad \forall i \in \mathcal{I}, \forall j \in \mathcal{J} \\ & y_i \in \{0, 1\} \quad \forall i \in \mathcal{I} \end{aligned}$$

In this formulation, the facility location decision variable is restricted to be either 0 or 1 and cannot be any real number. Therefore, the above formulation is an example of an integer program. Binary variables are also common in project or portfolio selection problems, which appear often in civil engineering. The next example shows such a problem.

Example B.9. *A business is seeking to invest in two private-public partnership projects. There is an option investing \$400, \$200, \$300, \$150, and \$250 in projects 1 through 5 respectively. The expected and standard deviation of return on the five projects is shown in the table below.*

Project	Expected Value	Stdev
1	5000	300
2	2000	150
3	3500	125
4	2000	225
5	3750	75

The returns on the five projects have a correlation coefficient of +0.1. For any investment portfolio, the business balances risk and expected return with the utility function $U = \mu - \frac{1}{2}\sigma$ where μ is the average return on investment and σ the standard deviation of return. The business has a total of \$1000 to invest. Formulate an optimization problem for determining the investment projects to maximize its utility.

Let \mathcal{I} represent the set of projects. In this case, let $\mathcal{I} = \{1, 2, 3, 4, 5\}$.

The decision variables in this problems should reflect which projects are selected for investment. This can be done using binary decision variables x_i which take the value 1 if project $i \in \mathcal{I}$ is selected, and 0 if not.

The business has a total budget of \$1000. Let B represent the total budget and b_i represent the amount which needs to be invested in each project $i \in \mathcal{I}$. For example $b_4 = 150$ and $b_5 = 250$. The budget constraint can be enforced as:

$$\sum_{i \in \mathcal{I}} b_i x_i \leq B \quad (\text{B.106})$$

Now let us tackle the objective function. Let μ_i represent the average return of investment in project $i \in \mathcal{I}$. The average return on investment μ is given as:

$$\mu = 5000x_1 + 2000x_2 + 3500x_3 + 2000x_4 + 3750x_5 = \sum_{i \in \mathcal{I}} \mu_i x_i \quad (\text{B.107})$$

Given the standard deviation of returns σ_i for each $i \in \mathcal{I}$ and the correlation coefficient $\rho_{ij} = 0.1$ for all $i \in \mathcal{I}$ and $j \in \mathcal{I}$. The standard deviation of return is given as:

$$\begin{aligned} \sigma &= \sqrt{300^2 x_1 + \dots + 225^2 x_4 + 75^2 x_5 + 0.1 \times 300 \times 150 + \dots} \\ &= \sqrt{\sum_{i \in \mathcal{I}} \sigma_i^2 x_i + \sum_{i \in \mathcal{I}} \sum_{j \in \mathcal{I}; j \neq i} \rho_{ij} \sigma_i \sigma_j x_i x_j} \quad (\text{B.108}) \end{aligned}$$

The optimization formulation is

$$\begin{aligned} \min_{x_i} \quad & \sum_{i \in \mathcal{I}} \mu_i x_i - \frac{1}{2} \sqrt{\sum_{i \in \mathcal{I}} \sigma_i^2 x_i + \sum_{i \in \mathcal{I}} \sum_{j \in \mathcal{I}; j \neq i} \rho_{ij} \sigma_i \sigma_j x_i x_j} \\ \text{s.t.} \quad & \sum_{i \in \mathcal{I}} b_i x_i \leq B \\ & x_i = \{0, 1\} \quad \forall i \in \mathcal{I} \end{aligned}$$

The above formulation is a nonlinear integer program. The nonlinearity arises from two sources: both the square root and the $x_i x_j$ interaction terms in the objective function. The integrality arises from the binary decision variables.

B.5 More Examples of Optimization Formulations

The previous section was a tour of common types of optimization problems: linear programs, integer programs, and nonlinear programs. This section provides additional examples of how optimization problems might be formulated in a transportation engineering setting. As you read through these, think about how they relate to the concepts defined in the previous section (which ones are linear, integer, etc.).

Example B.10. (Transit frequency setting.) *You are working for a public transit agency in a city, and must decide the frequency of service on each of the bus routes. The bus routes are known and cannot change, but you can change how the city's bus fleet is allocated to each of these routes. (The more buses assigned to a route, the higher the frequency of service.) Knowing the ridership on each route, how should buses be allocated to routes to minimize the total waiting time?*

Solution. To formulate this as an optimization problem, we need to identify an objective, decision variables, and constraints. For this problem, as we do this we will be faced with other assumptions which must be made. With this and other such problems, there may be more than one way to write down an optimization problem, what matters is that we *clearly state all of the assumptions made*. Another good guiding principle is to *start with the simplest model which captures the important behavior*, which can then be refined by relaxing assumptions or replacing simple assumptions with more realistic ones.

We need some notation, so let R be the set of bus routes. We know the current ridership on each route d_r . Here we will make our first assumption: that the demand on each route is inelastic and will not change based on the service frequency. Like all assumptions, it is not entirely true but in some cases may be close enough to the truth to get useful results; if not, you should think about what you would need to replace this assumption, which is always a good exercise. We are also given the current fleet size (which we will denote N), and must choose the number of buses associated with each route (call this n_r) — thus the decision variables in this problem are the n_r values.

The objective is to minimize the total waiting time, which is the sum of the waiting time for the passengers on each route. How long must passengers wait for a bus? If we assume that travelers arrive at a uniform rate, then the average waiting time will be half of the service headway. How is the headway related to the number of buses on the route n_r ? Assuming that the buses are evenly

dispersed throughout the time period we are modeling, and assuming that each bus is always in use, then the headway on route r will be the time required to traverse this route (T_r) divided by the number of buses n_r assigned to this route. So, the average delay per passenger is half of the headway, or $T_r/(2n_r)$, and the total passenger delay on this route is $(d_r T_r)/(2n_r)$. This leads us to the objective function

$$D(\mathbf{n}) = \sum_{r \in R} \frac{d_r T_r}{2n_r} \quad (\text{B.109})$$

in which the total delay is calculated by summing the delay associated with each route.

What constraints do we have? Surely we must run at least one bus on each route (or else we would essentially be canceling a route), and in reality as a matter of policy there may be some lower limit on the number of buses assigned to each route; for route r , call this lower bound L_r . Likewise, there is some upper bound U_r on the number of buses assigned to each route as well. So, we can introduce the constraint $L_r \leq n_r \leq U_r$ for each route r .

Putting all of these together, we have the optimization problem

$$\begin{aligned} \min_{\mathbf{n}} \quad & D(\mathbf{n}) = \sum_{r \in R} \frac{d_r T_r}{2n_r} \\ \text{s.t.} \quad & n_r \geq L_r \quad \forall r \in R \\ & n_r \leq U_r \quad \forall r \in R \\ & \sum_{r \in R} n_r \leq N \end{aligned}$$

■

Example B.11. (Scheduling maintenance.) *You are responsible for scheduling routine maintenance on a set of transportation facilities (such as pavement sections or bridges.) The state of these facilities can be described by a condition index which ranges from 0 to 100. Each facility deteriorates at a known, constant rate (which may differ between facilities). If you perform maintenance during a given year, its condition will improve by a known amount. Given an annual budget for your agency, when and where you should perform maintenance to maximize the average condition of these facilities? You have a 10 year planning horizon.*

Solution. In contrast to the previous example, where the three components of the optimization problem were described independently, from here on problems will be formulated in a more organic way, describing a model built from the ground up. (This is how optimization models are usually described in practice.) After describing the model in this way, we will identify the objective function, decision variables, and constraints to write the optimization problem in the usual form. We start by introducing notation based on the problem statement.

Let F be the set of facilities, and let c_f^t be the condition of facility f at the **end**³ of year t , where t ranges from 1 to 10. Let d_f be the annual deterioration on

³This word is intentionally emphasized. In this kind of problem it is very easy to get

facility f , and i_f the amount by which the condition will improve if maintenance is performed. So, if no maintenance is performed during year t , then

$$c_f^t = c_f^{t-1} - d_f \quad \forall f \in F, t \in \{1, 2, \dots, 10\} \quad (\text{B.110})$$

and if maintenance is performed we have

$$c_f^t = c_f^{t-1} - d_f + i_f \quad \forall f \in F, t \in \{1, 2, \dots, 10\} \quad (\text{B.111})$$

Both of these cases can be captured in one equation with the following trick: let x_f^t equal one if maintenance is performed on facility f during year t , and 0 if not. Then

$$c_f^t = c_f^{t-1} - d_f + x_f^t i_f \quad \forall f \in F, t \in \{1, 2, \dots, 10\} \quad (\text{B.112})$$

Finally, the condition can never exceed 100 or fall below 0, so the full equation for the evolution of the state is

$$c_f^t = \begin{cases} 100 & \text{if } c_f^{t-1} - d_f + x_f^t i_f > 100 \\ 0 & \text{if } c_f^{t-1} - d_f + x_f^t i_f < 0 \\ c_f^{t-1} - d_f + x_f^t i_f & \text{otherwise} \end{cases} \quad (\text{B.113})$$

for all $f \in F$ and $t \in \{1, 2, \dots, 10\}$. Of course, for this to be usable we need to know the current conditions of the facilities, c_f^0 .

The annual budget can be represented this way: let k_f be the cost of performing maintenance on facility f , and B^t the budget available in year t . Then

$$\sum_{f \in F} k_f x_f^t \leq B^t \quad \forall t \in \{1, \dots, 10\}. \quad (\text{B.114})$$

For the objective, we need the average condition across all facilities and all years; this is simply $\frac{1}{10|F|} \sum_{f \in F} \sum_{t=1}^{10} c_f^t$. The obvious decision variables are the maintenance variables x_f^t , but we also have to include c_f^t because these are influenced by the maintenance variables. As constraints, we need to include the state evolution equations (B.113), the budget constraints (B.114), and, less obviously the requirement that x_f^t be either 0 or 1. Putting it all together, we have the optimization problem

$$\begin{aligned} \max_{\mathbf{x}, \mathbf{c}} \quad & \frac{1}{10|F|} \sum_{f \in F} \sum_{t=1}^{10} c_f^t \\ \text{s.t.} \quad & \sum_{f \in F} k_f x_f^t \leq B^t \quad \forall t \in \{1, \dots, 10\} \\ & c_f^t \text{ is given by (B.113)} \quad \forall f \in F, t \in \{1, 2, \dots, 10\} \\ & x_f^t \in \{0, 1\} \quad \forall f \in F, t \in \{1, 2, \dots, 10\} \end{aligned}$$

■

confused about what occurs at the start of period t , at the end of period t , during the middle of period t , etc.

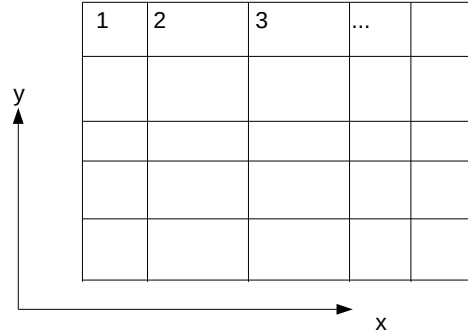


Figure B.1: Coordinates and labeling of intersections for Example B.12

In this example, pay close attention to the use of formulas like (B.114), which show up very frequently in optimization. It is important to make sure that every “index” variable in the formula is accounted for in some way. Equation (B.114) involves the variables x_f^t , but for which facilities f and time periods t ? The facility index f is summed over, while the time index t is shown at right as $\forall t \in \{1, \dots, 10\}$. This means that a copy of (B.114) exists for each time period, and in each of these copies, the left-hand side involves a sum over all facilities at that time. Therefore the one line (B.114) actually includes 10 constraints, one for each year. It is common to forget to include things like $\forall t \in \{1, \dots, 10\}$, or to try to use an index of summation outside of the sum as well (e.g., an expression like $B_f - \sum_{f \in F} k_f x_f^t$), which is meaningless. Make sure that all of your indices are properly accounted for!

In this next example, the objective function is less obvious.

Example B.12. (A facility location problem.) *In a city with a grid network, you need to decide where to locate three bus terminals. Building the terminal at different locations costs a different amount of money. Knowing the home locations of customers throughout the city who want to use the bus service, where should the terminals be located to minimize the construction cost and walking distance customers have to walk? Assume that each customer will walk from their home location to the nearest terminal.*

Solution. This problem could easily become very complicated if we take into account the impact of terminal locations on bus routes, so let’s focus on what the problem is asking for: simply locating terminals to minimize walking distance from customers’ home locations.

Number each of the intersections in the grid network (Figure B.1) from 1 to I , the total number of intersections. Assume that the terminals will be located at these intersections, and let the variables L_1 , L_2 , and L_3 denote the numbers of the intersections where terminals will be built. Let $C(i)$ be the cost of building a terminal at location i , so the total cost of construction is $C(L_1) + C(L_2) + C(L_3)$.

Let P be the set of customers, and let H_p denote the intersection that is the home location of customer p .

How can we calculate the walking distance between two intersections (say, i and j)? Figure B.1 shows a coordinate system superimposed on the grid. Let $x(i)$ and $y(i)$ be the coordinates of intersection i in this system. Then the walking distance between points i and j is

$$d(i, j) = |x(i) - x(j)| + |y(i) - y(j)|. \quad (\text{B.115})$$

This is often called the *Manhattan distance* between two points, after one of the densest grid networks in the world.

So what is the walking distance $D(p)$ for customer p ? The distance from p to the first terminal is $d(H_p, L_1)$, to the second terminal is $d(H_p, L_2)$, and to the third is $d(H_p, L_3)$. The passenger will walk to whichever is closest, so $D(p, L_1, L_2, L_3) = \min\{d(H_p, L_1), d(H_p, L_2), d(H_p, L_3)\}$ and the total walking distance is $\sum_{p \in P} D(p, L_1, L_2, L_3)$.

For this problem, the decision variables and constraints are straightforward: the only decision variables are L_1 , L_2 , and L_3 and the only constraint is that these need to be integers between 1 and I . The tricky part is the objective function: we are instructed both to minimize total cost as well as total walking distance. We have equations for each of these, but we can only have one objective function. In these cases, it is common to form a *convex combination* of the two objectives, introducing a weighting parameter $\lambda \in [0, 1]$. That is, let

$$f(L_1, L_2, L_3) = \lambda[C(L_1) + C(L_2) + C(L_3)] + (1 - \lambda) \left[\sum_{p \in P} D(p, L_1, L_2, L_3) \right] \quad (\text{B.116})$$

Look at what happens as λ varies. If $\lambda = 1$, then the objective function reduces to simply minimizing the cost of construction. If $\lambda = 0$, the objective function is simply minimizing the total walking distance. For a value in between 0 and 1, the objective function is a weighted combination of these two objectives, where λ indicates how important the cost of construction is relative to the walking distance.

For concreteness, the optimization problem is

$$\begin{array}{ll} \min_{L_1, L_2, L_3} & \lambda[C(L_1) + C(L_2) + C(L_3)] + (1 - \lambda) \left[\sum_{p \in P} D(p, L_1, L_2, L_3) \right] \\ \text{s.t.} & L_f \in \{1, 2, \dots, I\} \quad \forall f \in \{1, 2, 3\} \end{array}$$

■

In the final example in this section, finding a mathematical representation of a solution is more challenging.

Example B.13. (Shortest path problem.) *Figure B.2 shows a road network, with the origin and destination marked. Given the travel time on each roadway link, what is the fastest route connecting the origin to the destination?*

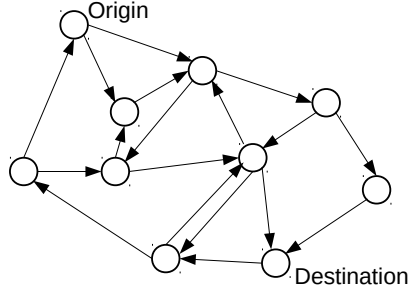


Figure B.2: Roadway network for Example B.13

Solution. We've already presented some algorithms for solving this problem in Section 2.4, but here we show how it can be placed into the general framework of optimization problems.

Notation first: let the set of nodes be N , and let r and s represent the origin and destination nodes. Let the set of links be A , and let t_{ij} be the travel time on link (i, j) . So far, so good, but how do we represent a route connecting two intersections?

Following Example B.11, introduce binary variables $x_{ij} \in \{0, 1\}$, where $x_{ij} = 1$ if link (i, j) is part of the route, and $x_{ij} = 0$ if link (i, j) is not part of the route. The travel time of a route is simply the sums of the travel times of the links in the route, which is $\sum_{(i,j) \in A} t_{ij} x_{ij}$.

We now have an objective function and decision variables, but what of the constraints? Besides the trivial constraint $x_{ij} \in \{0, 1\}$, we need constraints which require that the x_{ij} values actually form a contiguous path which starts at the origin r and ends at the destination s . We do this by introducing a *flow conservation constraint* at each intersection. For node i , recall that $\Gamma(i)$ denotes the set of links which leave node i , and $\Gamma^{-1}(i)$ denotes the set of links which enter node i .

Consider any contiguous path connecting intersection r and s , and examine any node i . One of four cases must hold:

Case I: Node i does not lie on the path at all. Then all of the x_{ij} values should be zero for $(i, j) \in \Gamma(i) \cup \Gamma^{-1}(i)$.

Case II: Node i lies on the path, but is not the origin or destination. Then $x_{ij} = 1$ for exactly one $(i, j) \in \Gamma(i)$, and for exactly one $(i, j) \in \Gamma^{-1}(i)$.

Case III: Node i is the origin r . Then all x_{ij} values should be zero for $(i, j) \in \Gamma^{-1}(i)$, and $x_{ij} = 1$ for exactly one $(i, j) \in \Gamma(i)$.

Case IV: Node i is the destination s . Then all x_{ij} values should be zero for $(i, j) \in \Gamma(i)$, and $x_{ij} = 1$ for exactly one $(i, j) \in \Gamma^{-1}(i)$.

An elegant way to combine these cases is to look at the differences $\sum_{(i,j) \in \Gamma(i)} x_{ij} - \sum_{(h,i) \in \Gamma^{-1}(i)} x_{hi}$. For cases I and II, this difference will be 0; for case III, the difference will be +1, and for case IV, the difference will be -1.

So, this leads us to the optimization formulation of the shortest path problem:

$$\begin{aligned}
 \min_{\mathbf{x}} \quad & \sum_{(i,j) \in A} t_{ij} x_{ij} \\
 \text{s.t.} \quad & \sum_{(i,j) \in \Gamma(i)} x_{ij} - \sum_{(h,i) \in \Gamma^{-1}(i)} x_{hi} = \begin{cases} 1 & \text{if } i = r \\ -1 & \text{if } i = s \\ 0 & \text{otherwise} \end{cases} \quad \forall i \in N \\
 & x_{ij} \in \{0, 1\} \quad \forall (i, j) \in A
 \end{aligned} \tag{B.117}$$

■

A careful reader will notice that if the four cases are satisfied for a solution, then the equations (B.117) are satisfied, but the reverse may not be true. Can you see why, and is that a problem?

There is often more than one way to formulate a problem: for instance, we might choose to minimize congestion by spending money on capacity improvements, subject to a budget constraint. Or, we might try to minimize the amount of money spent, subject to a maximum acceptable limit on congestion. Choosing the “correct” formulation in this case may be based on which of the two constraints is harder to adjust (is the budget constraint more fixed, or the upper limit on congestion?). Still, you may be troubled by this seeming imprecision. This is one way in which modeling is “as much art as science,” which is not surprising — since the human and political decision-making processes optimization tries to formalize, along with the inherent value judgments, (what truly is the objective?) are not as precise as they seem on the surface. One hallmark of a mature practitioner of optimization is a flexibility with different formulations of the same underlying problem, and a willingness to engage in “back-and-forth” with the decision maker as together you identify the best formulation for a particular scenario. In fact, in some cases it may not matter. The theory of duality (which is beyond the scope of this book) shows that these alternate formulations sometimes lead to the same ultimate decision, which is comforting.

B.6 General Properties of Optimization Problems

After seeing some concrete examples of optimization formulations, we now define a general form for optimization problems. This form is useful because it allows us to talk about properties of optimization problems, or methods for solving them, in a way that is independent of any specific context (shipping, investment, routing, etc.). Any of the optimization problems discussed in this book can be represented in this form, perhaps after some conversions or transformation.

The general form is

$$\begin{aligned} \min_{\mathbf{x}} \quad & f(\mathbf{x}) \\ \text{s.t.} \quad & g_i(\mathbf{x}) \geq b_i \quad \forall i = 1, 2, \dots, m \\ & h_i(\mathbf{x}) = 0 \quad \forall i = 1, 2, \dots, l \end{aligned}$$

If any variables are restricted to be integers, those constraints are listed in addition to the ones given above.

In the above problem $\mathbf{x} = (x_1, x_2, \dots, x_n)$ is a vector of decision variables. *Decision variables* are factors which are under the control of the decision maker. The function $f(\mathbf{x})$ is the *objective function* which represents the cost or utility associated with a decision \mathbf{x} . The equations $g_i(\mathbf{x}) \geq b_i$ and $h_i(\mathbf{x}) = 0$ are *constraint functions* which impose restrictions on the range of values the decision variables can take. Let $X = \{\mathbf{x} \in \mathbb{R}^n : g_i(\mathbf{x}) \geq b_i \forall i = 1, 2, \dots, m; h_i(\mathbf{x}) = 0 \forall i = 1, 2, \dots, l\}$. The set X is called the *feasible region* which represents the set of values which the decision variables can take or the set of values which satisfies all constraint functions. Any value $\mathbf{x} \in X$ is a *feasible solution* to the optimization problem. The goal of the mathematical problem is to determine the *optimal solution* which is defined as the feasible solution with the smallest objective function value, i.e., to find $\mathbf{x}^* \in X$ such that $f(\mathbf{x}^*) \leq f(\mathbf{x}) \forall \mathbf{x} \in X$.

Depending on the mathematical properties of the objective and constraint functions there are different categories of optimization problems. Linear optimization problems are mathematical programs where the functions $f(\mathbf{x}), g_i(\mathbf{x}) \forall i = 1, 2, \dots, m, h_i(\mathbf{x}) \forall i = 1, 2, \dots, m$ are linear. If any of the objective or constraint functions are nonlinear, then we have nonlinear optimization problems. Integer programs are mathematical programs where the decision variables have restricted to a set of integers. Depending on the nature of the mathematical program, there are different types of solution algorithms with varying degrees of efficiency.

B.6.1 Local and global solutions

Objective functions can have both *local* and *global* maximum or minimum solutions (Figure B.3). Let f be a function defined on a region $X \subseteq \mathbb{R}^n$. Recall from Section A.3 that a ball centered on \mathbf{x} of radius ϵ is the set containing all points of \mathbb{R}^n whose distance to \mathbf{x} is less than ϵ .

Definition B.1. A point $\mathbf{x}^* \in X$ is a *local minimum* of a function $f(\mathbf{x})$ if there is a ball B centered on \mathbf{x} with positive radius such that

$$f(\mathbf{x}) \geq f(\mathbf{x}^*) \quad \forall \mathbf{x} \in B \cap X$$

Definition B.2. A point $\mathbf{x}^* \in X$ is a *global minimum* of a function $f(\mathbf{x})$ if

$$f(\mathbf{x}) \geq f(\mathbf{x}^*) \quad \forall \mathbf{x} \in X$$

Strict local and global minima are obtained by replacing the inequalities in the above definitions with strict inequalities:

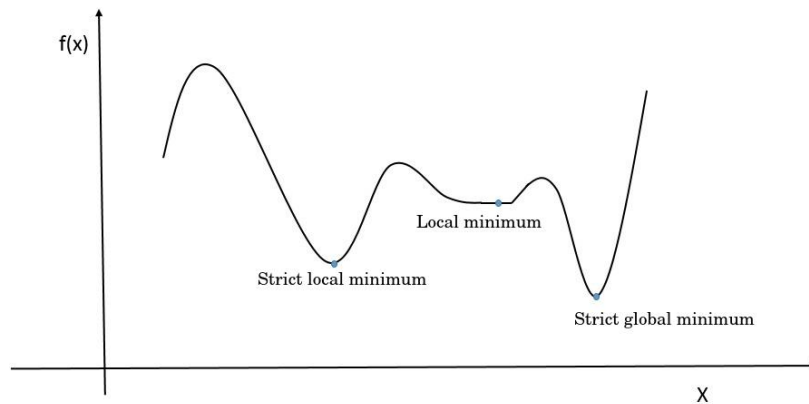


Figure B.3: Local, strict local, and global minima.

Definition B.3. A point $\mathbf{x}^* \in X$ is a strict local minimum of a function $f(\mathbf{x})$ if there is a ball B centered on \mathbf{x} with positive radius such that

$$f(\mathbf{x}) > f(\mathbf{x}^*) \quad \forall \mathbf{x} \in B \cap X, \mathbf{x} \neq \mathbf{x}^*$$

Definition B.4. A point $\mathbf{x}^* \in X$ is a global minimum of a function $f(\mathbf{x})$ if

$$f(\mathbf{x}) > f(\mathbf{x}^*) \quad \forall \mathbf{x} \in X, \mathbf{x} \neq \mathbf{x}^*$$

The corresponding terms for maximization problems are defined in analogous ways, by reversing the signs of the inequalities.

B.6.2 Objective function transformations

The following results are very useful in showing cases when two superficially different optimization problems may in fact be the same. The first result shows that it is easy to convert any maximization problem to a minimization problem (or vice versa) by negating the objective function. The second shows that constants may be freely added or subtracted to objective functions without changing the optimal solutions. The third shows that multiplication or division by a positive constant does not change the optimal solutions.

Proposition B.1. If the feasible set is X , the solution $\hat{\mathbf{x}}$ is a global maximum of f if and only if $\hat{\mathbf{x}}$ is a global minimum of $-f$ for the same feasible set X .

Proof. For the first part, assume that $\hat{\mathbf{x}}$ is the global maximum of f on the set X . Then $f(\hat{\mathbf{x}}) \geq f(\mathbf{x})$ for all $x \in X$. Multiplying both sides by -1 reverses the sign of the inequality, giving $-f(\hat{\mathbf{x}}) \leq -f(\mathbf{x})$ for all X , so $\hat{\mathbf{x}}$ is a global minimum for $-f$. For the second part, assume that $\hat{\mathbf{x}}$ is the global minimum of $-f$, so $-f(\hat{\mathbf{x}}) \leq -f(\mathbf{x})$ for all $x \in X$. Again multiplying both sides by -1 gives $f(\hat{\mathbf{x}}) \geq f(\mathbf{x})$ for all $x \in X$, so $\hat{\mathbf{x}}$ is a global maximum of f . \square

This result is convenient, because we do not need to develop two different theories for maximization and minimization problems. Instead, we can focus just on one. In this book, we develop results for minimization problems, a fairly common convention in engineering.⁴ Whenever you encounter a maximization problem, you can convert it to a minimization problem by negating the objective function, then using the minimization results and procedures. (Of course, this choice is completely arbitrary; we could just as well have chosen to develop results only for maximization problems and in fact some other fields do just this.)

Proposition B.2. *Let $\hat{\mathbf{x}}$ be an optimal solution when the objective function is $f(\mathbf{x})$ and the feasible set is X . Then for any constant b not depending on \mathbf{x} , $\hat{\mathbf{x}}$ is also an optimal solution when the objective function is $f(\mathbf{x}) + b$ and the feasible set is X .*

Proof. By Proposition B.1, with no loss of generality we can assume that the optimization problem is a minimization problem and that $\hat{\mathbf{x}}$ is a global minimum. Therefore $f(\hat{\mathbf{x}}) \leq f(\mathbf{x})$ for all $x \in X$. We can add the constant b to both sides of the equation, so $f(\hat{\mathbf{x}}) + b \leq f(\mathbf{x}) + b$ for all $x \in X$. Therefore $\hat{\mathbf{x}}$ is also a global minimum (and thus optimal) when the objective function is $f(\mathbf{x}) + b$. □

Proposition B.3. *Let $\hat{\mathbf{x}}$ be an optimal solution when the objective function is $f(\mathbf{x})$ and the feasible set is X . Then for any constant $c > 0$ not depending on \mathbf{x} , $\hat{\mathbf{x}}$ is also an optimal solution when the objective function is $cf(\mathbf{x})$ and the feasible set is X .*

Proof. See Exercise 6. □

B.6.3 Existence of solutions

It is possible to formulate optimization problems that do not have solutions — either because the constraints are contradictory and there is no feasible solution satisfying all of them, or because of the way the objective function and constraints are defined. Optimization problems of the first type are called *infeasible*. For example, the problem “minimize x subject to $x \leq 0$ ” has no optimal solution — whatever x you choose, you can find an x which is even more negative. Optimization problems of this type, where the objective can be made as negative as you like, are called *unbounded*. Some problems lack solutions even without falling into either of these categories; for example, “minimize x subject to $x > 0$ ” has no optimal solution, for whatever x you choose, $x/2$ is feasible but smaller. This is why we tend to avoid strict inequalities in optimization problems. But there are still problematic optimization formulations even without strict inequalities, such as “minimize $1/x$ subject to $x \geq 1$.”

⁴Economists often use maximization problems as their standard convention.

Practically speaking, if you are in a situation like the ones described above, you should take another look at your optimization problem: perhaps one constraint can be relaxed (maybe penalized in the objective, rather than strictly excluding solutions), or perhaps you missed a constraint that should have been there. Systems with unbounded or truly infeasible solutions are rare. This subsection provides a mathematical perspective on the topic, giving conditions on the constraints and objective function which can guarantee existence of a solution.

Keeping with our standard convention, we consider a problem of the form “minimize $f(\mathbf{x})$ subject to $\mathbf{x} \in X$,” where the set X contains all solutions satisfying all constraints. Weierstrass’ Theorem identifies gives sufficient conditions, under which an optimal solution exists.

Theorem B.1. (Weierstrass’ Theorem.) *Let f be a continuous, real-valued function defined on X , and let X be a non-empty, closed, and bounded set. The optimization problem $\{\min f(x) : x \in X\}$ has a minimum solution.*

(For definitions of the terms in this theorem, see Appendix A.)

Example B.14. *Consider the optimization formulation $\min(x - 5)^2, x \in X = \{x : x \in \mathbb{R}, 3 \leq x \leq 7\}$. Does a minimum exist?*

Solution. The function $(x - 5)^2$ is continuous and real valued. The feasible region is non-empty, closed, and bounded. Therefore, the above optimization problem has a minimum. By plotting the function, we can see that the minimum occurs at $x = 5$. ■

Example B.15. *Consider the optimization formulation $\min(x - 5)^2, x \in X = \{x : x \in \mathbb{R}, 5 < x \leq 7\}$. Does a minimum exist?*

Solution. In the above variant, the feasible region is not closed. Given any $y \in X$, we will be able to find another $z \in X$, such that $f(z) < f(y)$. For example, if we pick $y = 5.001$, we can always find another $z = 5.0001$ whose objective function value is smaller. Therefore, no minimum exists in this case. ■

Note that Weierstrass’ Theorem provides sufficient and not necessary conditions (more on this in next subsection). The continuity, closed, bounded assumption are not required for minima to exist. We can have situations where minima exist without the above conditions being satisfied.

Example B.16. *Consider the optimization formulation $\min(x - 5)^2, x \in X = \{x : x \in \mathbb{R}, x > 3\}$. Does a minimum exist?*

Solution. In the above optimization problem, the feasible region is not closed or bounded. However, we do know that the minimum exists at $x = 5$. ■

Example B.17. *Consider the objective function*

$$f(x) = \begin{cases} 1 & x = 0 \\ 5 & \text{otherwise} \end{cases} .$$

The feasible region is $X = \{x : x \in \mathbb{R}\}$. Does a minimum exist?

Solution. In this example, the function is discontinuous, the feasible region is not closed or bounded. However, we still have a minimum at $x = 0$. ■

B.6.4 Necessary and sufficient conditions

The previous subsection presented Weierstrass' Theorem, which gave sufficient conditions for the existence of a minimum solution. Some students are unfamiliar with the distinction between “necessary” and “sufficient” conditions, so we explain them here in the context of this theorem. Let us consider two statements A and B . Saying that A is a *necessary* condition for B means that B is true only if A is true, written $B \implies A$ (read “ B implies A ”). For example, every square is also a rectangle. This means that a shape can be a square only if it is a rectangle; being a rectangle is necessary for being a square, so we can identify A with “the shape is a rectangle,” and B with “the shape is a square.” Notice that A being true does not imply that B is true; there are rectangles which are not squares.

In the above situation, we can also say that B is a *sufficient* condition for A , meaning that if A is true, B must also be true. For example, “being a square” is sufficient for “being a rectangle.” In a statement like $B \implies A$, the arrow points from the sufficient condition to the necessary one. When B is a sufficient condition for A , A can be true without B being true.

Used alone, the terms “necessary” or “sufficient” mean that A and B are not exactly the same thing, but that one includes cases that the other does not have. For example, there are some rectangles which are not squares. If A and B *do* mean exactly the same thing, we say that A is necessary *and* sufficient for B ; A is true if and only if (abbreviated “iff”) B is true. This is written $A \iff B$, and means that both $A \implies B$ and $B \implies A$ are true. An example are the statements “ n is an even integer” and “ $n = 2k$ for some integer k .” Each statement is true exactly when the other is true; in other words, they are *equivalent* and we can replace one statement by the other whenever we like.

In Weierstrass' Theorem, statement A is “the function f is continuous and real-valued, and X is non-empty, closed, and bounded,” and statement B is “the function has a minimum value.” Condition A is sufficient, because any time A is true, B is true as well. However, condition A is not necessary, because there are examples where A is false but B is still true (Examples B.14 and B.15 above).

B.7 Exercises

- [11] Why is it generally a bad idea to use strict inequalities ($>$, $<$) in mathematical programs?
- [12] Mathematically specify an optimization problem which has no optimal solution, and an optimization problem which has multiple optimal solutions.

Table B.1: Data for Exercise 7

Bridge f	Initial condition c_f^0	Deterioration rate d_f	Repairability i_f	Maintenance cost k_f
1	85	5	3	2
2	95	10	1	4
3	75	1	1	3
4	80	6	6	1
5	100	3	3	1

3. [24] For the following functions, identify all stationary points and global optima.
- $f(x) = x^4 - 2x^3$
 - $f(x_1, x_2) = 2x_1^2 + x_2^2 - x_1x_2 - 7x_2$
 - $f(x_1, x_2) = x_1^6 + x_2^6 - 3x_1^2x_2^2$
 - $f(x_1, x_2, x_3) = (x_1 - 4)^2 + (x_2 - 2)^2 + x_3^2 + x_1x_2 + x_1x_3 + x_2x_3$
4. [42] Express the constraint “ $x \in \mathbb{Z}$ ” (that is, x must be an integer) in the form $g(x) = 0$ for some continuous function g . (This shows that nonlinear programming can’t be any easier than integer programming!)
5. [30] Can you ever improve the value of the objective function at the optimal solution by adding constraints to a problem? If yes, give an example. If no, explain why not.
6. [21] Prove Proposition B.3.
7. [27] Write out the objective and all of the constraints for the maintenance scheduling problem of Example B.11, using the bridge data shown in Table B.1, where the maintenance cost is expressed in millions of dollars. Assume a two-year time horizon and an annual budget of \$5 million.
8. [44] Reformulate the transit frequency-setting problem of Example B.10 so that the objective is to minimize the total amount of money spent, while achieving a pre-specified level of service (maximum delays).
9. [53] You are responsible for allocating bridge maintenance funding for a state, and must develop optimization models to assist with this for the current year. Political realities and a strict budget will require you to develop multiple formulations for the purposes of comparison.

Let \mathcal{B} denote the set of bridges in the state. There is an “economic value” associated with the condition of each bridge: the higher the condition, the higher the value to the state, because bridges in worse condition require more routine maintenance, impose higher costs on drivers who must drive slower or put up with potholes, and carry a higher risk of unforeseen failures requiring emergency maintenance. To represent this, there is a value

function $V_b(x_b)$ associated with each bridge $b \in \mathcal{B}$, giving the economic value of this bridge if x_b is spent on maintenance this year. You have a total budget of X to spend statewide. The state is also divided into n districts, and each bridge belongs to one district. Let \mathcal{D}_i denote the set of bridges which belong to district i ; different districts may contain different numbers of bridges.

In the following, be sure to define any additional notation you introduce. There is more than one possible formulation that achieves the stated goals, so feel free to explain any parts of your formulations which may not be self-evident.

- (a) **Maximize average benefits:** Formulate an optimization model (objective function, decision variables, and constraints) to maximize the average economic value of bridges across the state.
- (b) **Equitable allocation:** The previous model may recommend spending much more money in some districts than others, which is politically infeasible. Formulate an optimization model based on maximizing the total economic value to the state, but ensuring that each district receives a relatively fair share of the total budget X .
- (c) **Equitable benefits:** There is a difference between a fair allocation of money, and a fair allocation of *benefits* (say, if the functions V_b differ greatly between districts). Defining *benefit* as the difference between the economic value after investing in maintenance, and the “do-nothing” economic value, formulate an optimization model which aims to achieve a relatively fair distribution of benefits among districts.

10. [34] Consider the mathematical program

$$\begin{aligned}
 \max \quad & 3x_1 + 5x_2 + 6x_3 \\
 \text{s.t.} \quad & x_1 + x_2 + x_3 = 1 \\
 & 2x_1 + 3x_2 + 2x_3 \leq 5 \\
 & x_2 + x_3 \leq 1/2 \\
 & x_1, x_2, x_3 \geq 0
 \end{aligned}$$

- (a) Are all of the constraints needed?
 - (b) Solve this problem graphically.
11. [57] In an effort to fight rising maintenance costs and congestion, a state transportation agency is considering a toll on a certain freeway segment during the peak period. Imposing a toll accomplishes two objectives at once: it raises money for the state, and also reduces congestion by making some people switch to less congested routes, travel earlier or later, carpool, take the bus, and so forth. Suppose that there are 10000 people who would

want to drive on the freeway if there was no toll and no congestion, but the number of people who actually do is given by

$$x = 10000e^{(15-6\tau-t)/500}$$

where τ is the roadway toll (in dollars) and t is the travel time (in minutes). (That is, the higher the toll, or the higher the travel time, the fewer people drive.) The travel time, in turn, is given by

$$t = 15 \left[1 + 0.15 \left(\frac{x}{c} \right)^4 \right]$$

minutes, where $c = 8000$ veh/hr is the roadway capacity during rush hour. Regulations prohibit tolls exceeding \$10 at any time. Citizens are unhappy with both congestion and having to pay tolls. After conducting multiple surveys, the agency has determined that citizen satisfaction can be quantified as

$$s = 100 - t - \tau/5$$

- (a) Formulate two nonlinear programs based on this information, where the objectives are to either (a) maximize revenue or (b) maximize citizen satisfaction.
 - (b) Solve both of these problems, reporting the optimal values for the decision variables and the objective function.
 - (c) Comment on the two solutions. (e.g., do they give a similar toll value, or very different ones? How much revenue does the state give up in order to maximize satisfaction?)
 - (d) Name at least two assumptions that went into formulating this problem. Do you think they are realistic? Pick one of them, and explain how the problem might be changed to eliminate that assumption and make it more realistic.
12. [59] You are asked to design a traffic signal timing at the intersection of 8th & Grand. Assuming a simple two-phase cycle (where Grand Avenue moves in phase 1, and 8th Street in phase 2), no lost time when the signal changes, and ignoring turning movements, the total delay at the intersection can be written as

$$\frac{\lambda_1(c - g_1)^2}{2c \left(1 - \frac{\lambda_1}{\mu_1}\right)} + \frac{\lambda_2(c - g_2)^2}{2c \left(1 - \frac{\lambda_2}{\mu_2}\right)}$$

where g_1 and g_2 are the effective green time allotted to Grand Avenue and 8th Street, $c = g_1 + g_2$ is the cycle length, λ_1 and μ_1 are the arrival rate and saturation flow for Grand Avenue, and λ_2 and μ_2 are the arrival rate and saturation flow for 8th Street.

All signals downtown are given a sixty-second cycle length to foster good progression, and the arrival rate and saturation flow are 2200 veh/hr and

3600 veh/hr for Grand Avenue, respectively, and 300 veh/hr and 1900 veh/hr for 8th Street. Furthermore, no queues can remain at the end of the green interval; this means that $\mu_i g_i$ must be at least as large as $\lambda_i c$ for each approach i .

- (a) Why does the constraint $\mu_i g_i \geq \lambda_i c$ imply that no queues will remain after a green interval?
 - (b) Formulate a nonlinear program to minimize total delay.
 - (c) Simplify the nonlinear program so there is only one decision variable, and solve this nonlinear program using the bisection method of Section 3.3.2 (terminate when $b - a \leq 1$).
 - (d) Write code to automate this process, and perform a sensitivity analysis by plotting the effective green time on Grand Ave as λ_1 varies from 500 veh/hr to 3000 veh/hr, in increments of 500 veh/hr. Interpret your plot.
 - (e) Identify two assumptions in the above model. Pick one of them, and describe how you would change your model to relax that assumption.
13. [26] Find the global minima of the following functions in two ways: the bisection method, and using Newton's method to directly find a stationary point (if the Newton step leaves the feasible region, move to the boundary point closest to where Newton's method would go). Run each method for five iterations, and see which is closer to the optimum, making the comparison based on the value of the objective function at the final points.
- (a) $f(x) = -\arctan x$, $x \in [0, 10]$
 - (b) $f(x) = x \sin(1/(100x))$, $x \in [0.015, 0.04]$
 - (c) $f(x) = x^3$, $x \in [5, 15]$
14. [51] Give a network and a feasible solution \mathbf{x} to the mathematical formulation of the shortest path problem in Example B.13 where the links with $x_{ij} = 1$ do not form a contiguous path between r and s , as alluded to at the end of the example.

Appendix C

Optimization Techniques

This appendix contains additional information on optimization problems, and specific techniques for solving them. In turn, we present solution methods for one-dimensional optimization, linear optimization problems, unconstrained nonlinear optimization, constrained nonlinear optimization, and optimization problems with integrality constraints. We close with an introduction to metaheuristics, general-purpose techniques which can be applied to any optimization problem, but without guarantees of finding an optimal solution.

C.1 More Line Search Algorithms for One-Dimensional Optimization

Section 3.3.2 presented the bisection method as a way of solving optimization problems with a single decision variable x ; a convex, differentiable objective function f ; and a single, “interval” constraint of the form $a \leq x \leq b$. (Actually, we can relax the assumption that f is convex to f being *unimodal*, that is, if every local minimum is also a global minimum. Convexity is sufficient, but not necessary, for the function to be unimodal.) Most optimization problems have more than one variable, but solution algorithms often rely on simpler “subproblems” which are often one-dimensional. The Frank-Wolfe method from Section 6.2.2 is an example of this.

This section presents two other line search methods that can be applied to similar problems. The *golden section* method requires more iterations than bisection, but does not require taking the derivative of f . It is useful if f is not differentiable, or if the derivative is cumbersome to evaluate. *Newton’s method*, on the other hand, requires the stronger assumption that f is twice differentiable (and requires calculating both first and second derivatives), but usually requires fewer iterations than bisection.

C.1.1 Golden section method

The golden section method does not use derivatives to narrow down where the minimum lies. Instead, the logic is as follows: assume we know the value of the function at *two* intermediate points (call them c and d), as well as at the endpoints a and b . These points are in the order $a < c < d < b$. If f is unimodal, then there are only two possibilities: either $f(a) \geq f(c) \geq f(d)$, or $f(c) \leq f(d) \leq f(b)$. The reason is that a unimodal function must decrease before its minimum is reached, and increase afterwards. So if $f(c) \geq f(d)$, then the minimum must be to the right of c , and the function must be decreasing everywhere to the left of c , so $f(a) \geq f(c) \geq f(d)$. If $f(c) \leq f(d)$, then using a similar argument we know that $f(c) \leq f(d) \leq f(b)$. In the first case, we know that the minimum is somewhere in the interval $[c, b]$, and in the second case we know it is somewhere in the interval $[a, d]$. In either case, we have reduced the width of the interval where the minimum can lie, and we iterate by choosing two more intermediate points in the reduced interval, on till convergence.

This rule works no matter how the two intermediate points are chosen. To make the method as efficient as possible, we want to re-use points already found if possible. For instance, in the first case above, the interval is reduced to $[c, b]$. The point d is within this interval, and can serve as one of the two points we use at the next iteration. We also want to reduce the interval as much as possible from one iteration to the next. Since we don't know in advance which end we will trim from, c should be as far from the left endpoint as d is from the right, that is, $c - a = b - d$. To be able to re-use the point d in the next iteration, we want d to be the same fraction of the distance from c to b as c is from a to b . (Figure C.1). The same logic applies if we are in the second case and must eliminate the upper end.

Using these principles to solve for c and d , we see that c should be to the right of a by $(3 - \sqrt{5})/2 \approx 38.2\%$ of the total length of the interval $[a, b]$, and that d should be to the right of a by $(\sqrt{5} - 1)/2 \approx 61.8\%$ of the length of $[a, b]$. Both of our principles are satisfied: c is 38.2% to the right of a , and d is 38.2% to the left of b ; and point d lies 38.2% of the way from c to b , just as c is 38.2% of the way from a to b . This means that the lengths of the intervals $[c, b]$ and $[a, b]$ have the *golden ratio* of 1 to 1.618. (This ratio appears often in nature, art, design, and mathematics, being very closely related to the Fibonacci sequence.)

With this in mind, here are the steps of the algorithm:

- Step 0: Initialize.** Let $\theta = (3 - \sqrt{5})/2$, set the iteration counter $k = 0$, and initialize $a_0 = a$, $b_0 = b$, $c_0 = a + \theta(b - a)$, $d_0 = b - \theta(b - a)$.
- Step 1: Determine which end to eliminate.** If $f(c_k) \geq f(d_k)$, go to step 2; otherwise, go to step 3.
- Step 2: Eliminate lower end.** Update $a_{k+1} = c_k$, $b_{k+1} = b_k$, $c_{k+1} = d_k$, and $d_{k+1} = c_k + \theta(b_k - c_k)$. Skip to step 4.
- Step 3: Eliminate upper end.** Update $a_{k+1} = a_k$, $b_{k+1} = d_k$, $c_{k+1} = a_k + \theta(c_k - a_k)$, and $d_{k+1} = c_k$. Proceed to step 4.

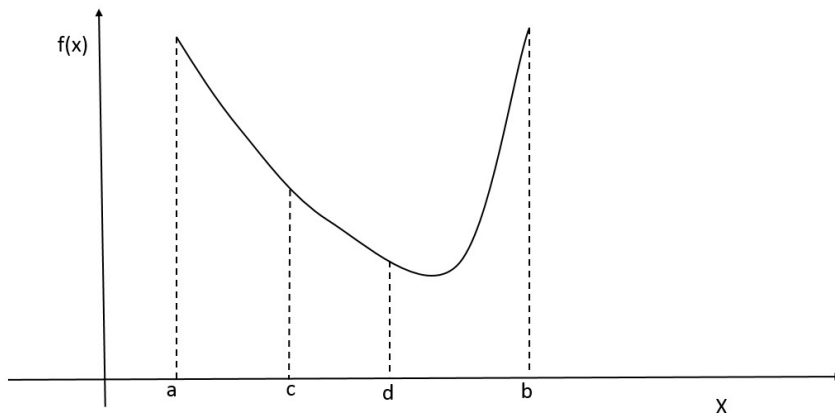


Figure C.1: Golden section method.

Step 4: Iterate. Increase the counter k by 1 and check the termination criterion. If $b_k - a_k < \epsilon$, then terminate; otherwise, return to step 1.

Example C.1. Find the minimum of the function $f(x) = (x - 1)^2 + e^x$ in the interval $[0, 2]$, within a tolerance of $\epsilon = 0.01$.

Solution. In the initialization phase, we set $k = 0$, $a_0 = 0$, $b_0 = 2$, $\epsilon = 0.01$, and

$$\begin{aligned} c_0 &= a_0 + \theta(b_0 - a_0) &&= 0.764 \\ d_0 &= b_0 - \theta(b_0 - a_0) &&= 1.236 \end{aligned}$$

We now proceed to step 1. Since $f(c_1) = 2.2025 < f(d_1) = 3.4975$, we decide to eliminate the upper end and perform step 3.

$$\begin{aligned} a_1 &= a_0 = 0 && b_1 &= d_0 = 1.236 \\ c_1 &= a_0 + \theta(c_0 - a_0) = 0.4722d_1 && &= c_0 = 0.764 \end{aligned}$$

The interval is still wider than the tolerance ϵ , so we return to the first step. Now, $f(c_2) = 1.882 < f(d_2) = 2.2025$, so we again eliminate the upper end by performing step 3.

$$\begin{aligned} a_2 &= a_1 = 0 \\ b_2 &= d_1 = 0.764 \\ c_2 &= a_1 + \theta(c_1 - a_1) = 0.2918 \\ d_2 &= c_1 = 0.4721 \end{aligned}$$

Table C.1: Demonstration of the golden section algorithm with $f(x) = (x - 1)^2 + e^x$, $x \in [0, 2]$.

k	a_k	b_k	c_k	d_k	$f(c_k)$	$f(d_k)$
0	0	2	0.764	1.236	2.20254	3.49751
1	0	1.236	0.472152	0.764	1.88206	2.20254
2	0	0.764	0.291848	0.472152	1.84038	1.88206
3	0	0.472152	0.180362	0.291848	1.86946	1.84038
4	0.180362	0.472152	0.291848	0.360688	1.84038	1.84304
5	0.180362	0.360688	0.249247	0.291848	1.84669	1.84038
6	0.249247	0.360688	0.291848	0.318118	1.84038	1.8395
7	0.291848	0.360688	0.318118	0.334391	1.8395	1.84012
8	0.291848	0.334391	0.308100	0.318118	1.83956	1.8395
9	0.308100	0.334391	0.318118	0.324348	1.8395	1.83963
10	0.308100	0.324348	0.314306	0.318118	1.83948	1.8395
11	0.308100	0.318118	0.311926	0.314306	1.8395	1.83948
12	0.311926	0.318118	0.314306	0.315753	1.83948	1.83949

We proceed as follows until $b_k - a_k < \epsilon$, with the remaining steps shown in Table C.1. The optimal value is $\frac{a_{13} + b_{13}}{2} = 0.3150$. ■

We used this same example for bisection in Section 3.3.2. Golden section required more iterations than bisection, to find a solution with the same tolerance. This is because bisection reduces the interval width by half at each iteration, whereas golden section reduces it by $\theta \approx 38\%$. On the other hand, we did not have to calculate any derivatives. The value of the objective function f was enough.

Example C.2. Find the minimum of the function $f(x) = 1 + e^{-x} \sin(-x)$ in the interval $[0, 3]$ with a tolerance of 0.01.

Solution. See Table C.2 below. The optimal value is $\frac{a_{13} + b_{13}}{2} = 0.7860$. (Using calculus, the optimal solution is actually $\pi/4 \approx 0.7854$.) ■

C.1.2 Newton's method

You may recall Newton's method from calculus: to find a zero of some function g , the method starts with an initial guess, which is iteratively updated by finding where the linear approximation of g at that point has a zero. Under the right conditions, Newton's method is almost miraculously fast, exhibiting quadratic convergence (roughly speaking, the number of correct digits doubles at each iteration). There are a few downsides: it can only be applied if g is differentiable, and Newton's method may not converge at all if you start too far away from the root, or if the shape of the function makes Newton's method "overshoot" and end up further away than it started. Nevertheless, Newton's method and its variants are very commonly used in optimization. In this appendix, we will

Table C.2: Demonstration of the golden section algorithm with $f(x) = 1 + e^{-x} \sin(-x)$, $x \in [0, 3]$.

k	a_k	b_k	c_k	d_k	$f(c_k)$	$f(d_k)$
0	0	3	1.146	1.854	0.710349	0.849629
1	0	1.854	0.708228	1.146	0.679624	0.710349
2	0	1.146	0.437772	0.708228	0.726369	0.679624
3	0.437772	1.146	0.708228	0.875457	0.679624	0.680064
4	0.437772	0.875457	0.604968	0.708228	0.689418	0.679624
5	0.604968	0.875457	0.708228	0.77213	0.679624	0.67766
6	0.708228	0.875457	0.77213	0.811575	0.67766	0.67782
7	0.708228	0.811575	0.747707	0.77213	0.678073	0.67766
8	0.747707	0.811575	0.77213	0.787178	0.67766	0.677604
9	0.77213	0.811575	0.787178	0.796507	0.677604	0.677643
10	0.77213	0.796507	0.781442	0.787178	0.677608	0.677604
11	0.781442	0.796507	0.787178	0.790752	0.677604	0.677612
12	0.781442	0.790752	0.784999	0.787178	0.677603	0.677604

use it in several ways; and several of the algorithms mentioned in the main text make use of it as well.

Here we are specifically concerned with solving a one-dimensional optimization problem with an interval constraint. If the objective function f is convex, then it is enough to find a point where the derivative f' vanishes. So, we simply apply Newton's method to the *derivative*, with $g \equiv f'$, to try to find \hat{x} such that $f'(\hat{x}) = 0$. Newton's method uses the derivative of g , which ends up being the *second* derivative f'' . (If f is not twice-differentiable, Newton's method cannot be applied.) We have to make one minor modification to Newton's method: the line search cannot leave the feasible region $[a, b]$, so we truncate the search at these boundary points. This can actually be helpful, since it prevents Newton's method from diverging. There are still cases where Newton's method can fail; an example is given in Example C.5 below.

Unlike bisection or golden section, Newton's method is not an "interval reduction" method, where we gradually shrink the range of possible values where the optimum can lie. So we need a different way to measure convergence. It is common to stop when f' is "close enough" to zero; we will let ϵ' denote this value.

The steps of Newton's method for line search are:

Step 0: Initialize. Set the iteration counter $k = 0$, and initialize x_0 to any point in $[a, b]$. (If you have a good guess as to the minimum point, it can greatly speed things up.)

Step 1: Check convergence. If $|f'(x_k)| < \epsilon'$, then terminate.

Step 2: Calculate recommended shift. Create a candidate point $\tilde{x} = x_k - f'(x_k)/f''(x_k)$.

Step 3: Ensure feasibility. Project candidate onto the feasible region $x_{k+1} = \text{proj}_{[a,b]}(\tilde{x})$ by setting $x_{k+1} = a$ if $\tilde{x} < a$, $x_{k+1} = b$ if $\tilde{x} > b$, and $x_{k+1} = \tilde{x}$ otherwise.

Step 4: Iterate. Increase k by 1, and return to step 1.

Example C.3. Find the minimum of the function $f(x) = (x - 1)^2 + e^x$ in the interval $[0, 2]$ using Newton's method, with $\epsilon' = 0.0337$. (In this and the next example, ϵ' is chosen to make the tolerance comparable to the ϵ value used for bisection and golden section; for this function, when $|f'(x)| < 0.0337$, x is within 0.01 of its optimal value.)

Solution. Start by computing the formulas for the first and second derivative of f , since we will be using these often: $f'(x) = 2(x - 1) + e^x$, and $f''(x) = 2 + e^x$.

In the initialization phase, we set $k = 0$. For an initial guess, choose $x_0 = 1$. (This makes for a fair comparison, since this is the starting point for bisection.) The first and second derivatives are equal to e and $e + 2$, respectively, so the new candidate point is $\tilde{x} = 1 - e/(e + 2) = 0.4239$. This lies within the boundary $[0, 2]$, so we accept the candidate point as the next solution: $x_1 = 0.4239$.

At this new point, the first and second derivatives equal 0.3756 and 3.528, so the next candidate is $0.4239 - 0.3756/3.528 = 0.3174$. We accept the candidate as the new point, so $x_2 = 0.3174$. At x_2 , the derivative is $f'(x_2) = 0.0084 < \epsilon'$, so we terminate and report 0.3174 as the optimal solution. ■

Notice that Newton's method achieved in only two iterations the level of precision bisection reached in eight, and golden section reached in thirteen! At this point, the solution given by Newton's method differs from the true optimum x by roughly 2×10^{-3} . One more iteration of Newton's method would reduce this error to 1×10^{-6} , and yet another would reduce it to 3×10^{-13} . This is what we mean when we say its convergence rate is miraculous!

Example C.4. Find the minimum of the function $f(x) = 1 + e^{-x} \sin(-x)$ in the interval $[0, 3]$ using Newton's method, with $\epsilon' = 0.00645$. (Again, this choice of ϵ' ensures that when Newton's method terminates, x is within 0.01 of its optimal value.)

Solution. Start by computing the formulas for the first and second derivative of f , since we will be using these often: $f'(x) = e^{-x}(\sin x - \cos x)$, and $f''(x) = 2e^{-x} \cos x$.

In the initialization phase, we set $k = 0$. For an initial guess, choose $x_0 = 1.5$. (This makes for a fair comparison, since this is the starting point for bisection.) The first and second derivatives are equal to 0.2068 and 0.03157, respectively, so the new candidate point is $\tilde{x} = 1.5 - 0.2068/0.03157 = -5.051$. This is outside of the feasible interval $[0, 3]$, so we project the candidate back onto the feasible region, choosing $x_1 = 0$.

At this new point, the first and second derivatives equal -1 and 2 , so the next candidate is $0 - (-1)/2 = 0.5$. We accept the candidate as the new point, so $x_2 = 0.5$. Another two iterations are needed: $x_3 = 0.5 - (-0.2415)/(1.065) =$

0.7268, and $x_4 = 0.7822$. At x_4 , the derivative is $|f'(x_4)| = 0.0021 < \epsilon'$, so we terminate and report 0.7822 as the optimal solution. ■

Newton's method is not infallible, however, as shown by the next example.

Example C.5. Apply Newton's method to the function $f(x) = x \tan^{-1} x - \frac{1}{2} \log(1 + x^2)$ over $x \in [-2, 2]$, starting with the initial guess $x_0 = 1.3917452$. (This formula looks unwieldy, but plot the function! It actually looks quite nice, and you can spot the minimum immediately by inspection.)

Solution. The first and second derivatives are $f'(x) = \tan^{-1} x$ and $f''(x) = 1/(x^2 + 1)$. Calculating f' and f'' at the initial guess x_0 , we find the next point to be $x_1 = -1.3914752$ — exactly the opposite of where we started! You can guess what happens next. Calculating from x_1 , we find the next point to be $x_2 = +1.3914752$, and so on *ad infinitum*. ■

Again, this is a “nice” function — it is even strictly convex! Yet Newton's method fails to find the optimal solution. You can verify that if we start with an initial guess closer to zero, Newton's method will converge, and if the initial guess is further away, Newton's method will diverge to whatever the endpoints are of the feasible interval.

C.2 Linear Programming

This section briefly outlines the properties of a linear optimization problem (commonly called a *linear program*), terminology associated with solutions of linear programs, and two solution methods: a graphical method suitable for small problems, and the simplex method which is widely used for practical problems. Recall that a linear program is an optimization where the objective function and constraints are all linear in the decision variables; the constraints may either be equalities ($=$) or weak inequalities (\geq, \leq).

C.2.1 Basic properties and graphical solution method

This section will predominantly use the following example:

$$\begin{array}{ll} \min_{x,y} & 10x + 26y \\ \text{s.t.} & 11x + 3y \geq 21 \\ & 6x + 20y \geq 39 \\ & x + y \leq 9 \\ & x, y \geq 0 \end{array}$$

The feasible region for this problem is shown in Figure C.2 as the area shaded in yellow.

The optimal solution to this linear program is the point (x, y) in the feasible region which minimizes $10x + 26y$. In this case, the optimal solution is $(1.5, 1.5)$, with a corresponding objective function value of 54. It is not immediately obvious how we found this point, since there is an infinite number of feasible solutions.

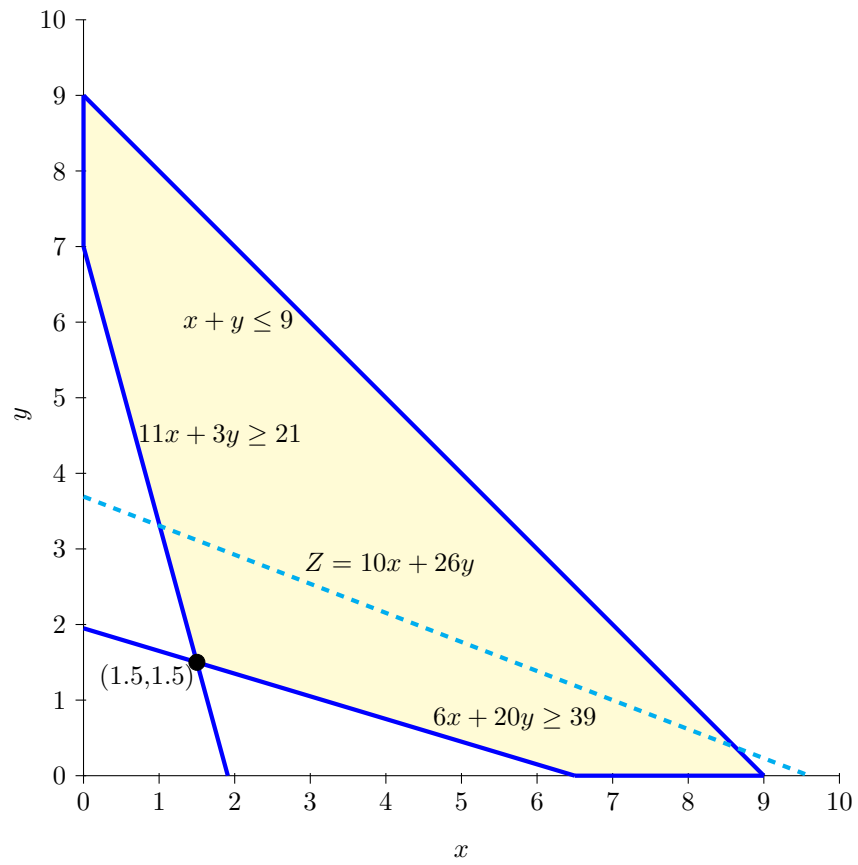


Figure C.2: An illustrative linear program.

This point happens to lie at a “corner” of the feasible region. This is not a coincidence. If a linear program has an optimal solution, and if its feasible region has a corner, then there is an optimal solution that lies at such a corner point. As a result, we can usually confine our attention to the corner points, of which there are only a finite number. In this particular case, the corner point feasible solutions are $(0, 9)$, $(0, 7)$, $(1.5, 1.5)$, $(6.5, 0)$ and $(9, 0)$. Checking in each in turn, we see that the objective function is minimized at $(1.5, 1.5)$, so this is the optimal solution.

This provides us with a method for solving a linear program, if the linear program has an optimal solution and its feasible region has at least one corner.

- Plot all the constraints and identify the feasible region.
- Determine the feasible corner points.
- Evaluate the objective function at the feasible corner points.
- The “best” feasible corner point is the optimal solution

For small problems this can be done graphically, as in Figure C.2. For larger problems, it is difficult to identify all the corner points by inspection, and a different method is needed.

This intuitive description is enough for our purposes; for readers wanting a more technical definition of a “corner,” we can define it as a point that is not the midpoint of any line segment contained in the feasible region. These points are also known as *vertices*, or *extreme points*. A point along an edge of the feasible region can be drawn as the midpoint of a line segment drawn along this edge; a point in the interior of the feasible region is the midpoint of many different line segments. But a line segment drawn in the feasible region can only have a corner as one of its endpoints, never the midpoint.

Why do we need the caveats “if a linear program has an optimal solution” and “if its feasible region has a corner”? Consider the examples below. Modify the constraints so the optimization problem has the following form:

$$\begin{array}{ll} \min_{x,y} & 10x + 26y \\ \text{s.t.} & x + y \leq 9 \\ & y \leq 5 \\ & y \geq 0. \end{array}$$

This produces the feasible region shown in Figure C.3. There is no restriction on how small x can be, and as x tends to $-\infty$, so does the objective function. Therefore there is no optimal solution; the problem is said to be *unbounded* since there is no limit to how much the objective function can be minimized. (The same would hold true for a maximization problem if it is possible for the objective function to be made arbitrarily large.)

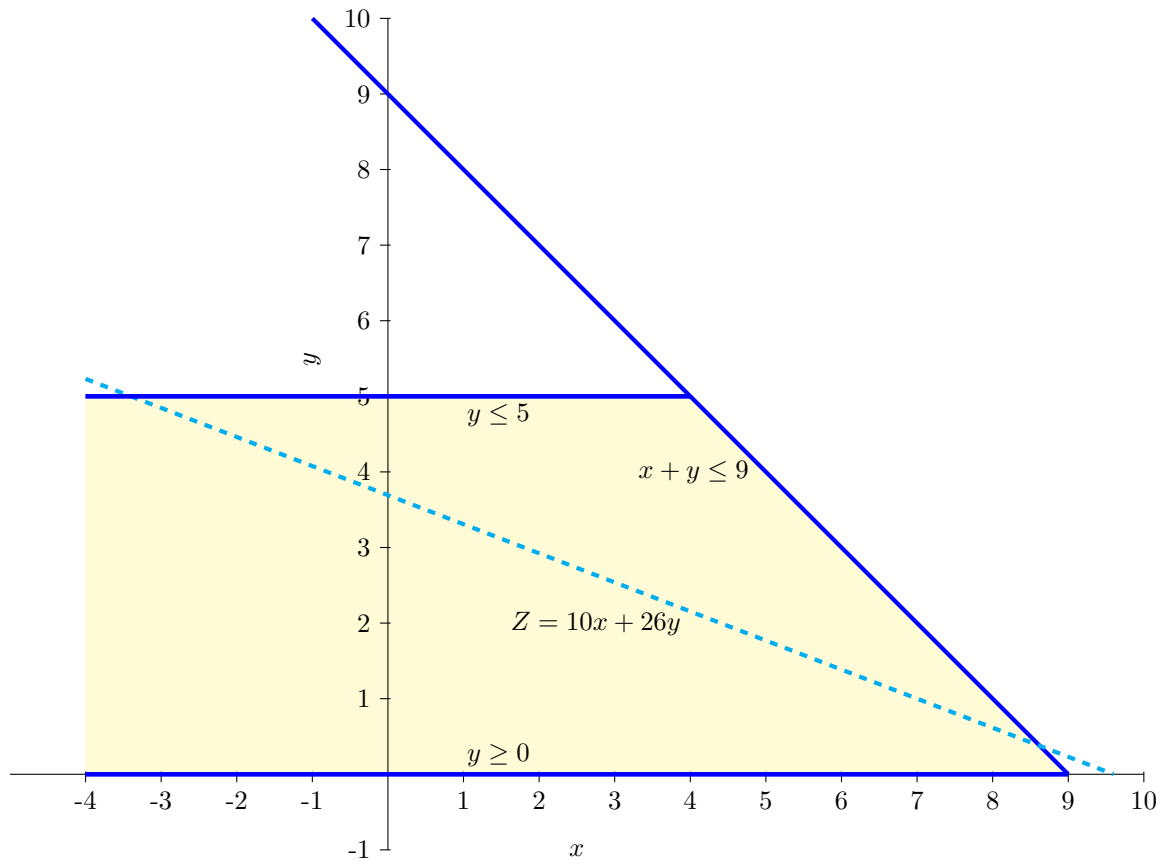


Figure C.3: An unbounded linear program.

Or, consider this optimization problem:

$$\begin{array}{ll} \min_{x,y} & 10x + 26y \\ \text{s.t.} & x + y \leq 9 \\ & y \leq 5 \\ & x \geq 9.5 \\ & y \geq 0. \end{array}$$

There is no solution which obeys all the constraints (see Figure C.4). The constraints $x + y \leq 9$ and $y \geq 0$ mean that x cannot be greater than 9; however there is also the constraint $x \geq 9.5$. Such a problem is called *infeasible*, because there is not even a feasible solution (let alone an optimal one). As a rule of thumb, unbounded or infeasible problems often mean that you have missed something in your formulation. In the real world, it is not possible to produce “infinitely good” solutions (surely there is some limitation; this would be a constraint you have missed), and there is usually some possible course of action, even if it is very unpleasant (your constraints are too restrictive).

Here is an optimization problem which has an optimal solution, but its feasible region has no corners (Figure C.5):

$$\begin{array}{ll} \min_{x,y} & x + y \\ \text{s.t.} & x + y \geq 0. \end{array}$$

Any solution on the line $x + y = 0$ is feasible and optimal. So this problem has infinitely many optimal solutions, but no corner points. These examples show that the statement “at least one corner point is optimal, so we only have to look at corner points” needs a few technical qualifications (that there is in fact an optimal solution, and a corner point). In practice these kinds of counterexamples are rare.

Also note that this statement does not say that *only* corner points may be optimal, even with the qualifications that there are corner points and optimal solutions. There very well may be a non-corner point which is optimal. But in such cases there is *also* an optimal corner point, so it is still acceptable to just check the corner points. We will still find an optimal solution that way. An example of such a problem is

$$\begin{array}{ll} \min_{x,y} & 12x + 40y \\ \text{s.t.} & 11x + 3y \geq 21 \\ & 6x + 20y \geq 39 \\ & x + y \leq 9 \\ & x, y \geq 0. \end{array}$$

(See Figure C.6) Any feasible point on the line segment between (1.5, 1.5) and (6.5, 0) has the same, optimal objective function value of 78. So a solution like (3.5, 0.75) is optimal even though it is not a corner point. However, the corner points (1.5, 1.5) and (6.5, 0) are both optimal as well, so we can still find

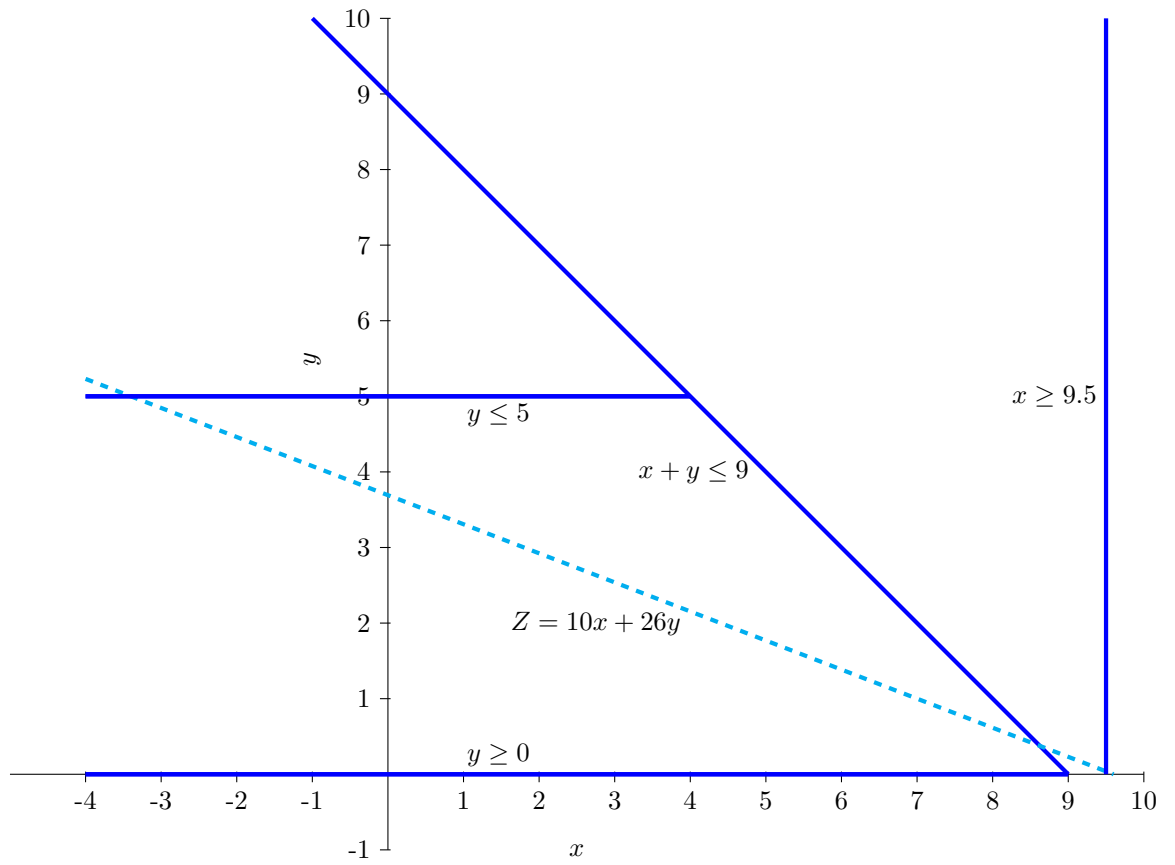


Figure C.4: An infeasible linear program.

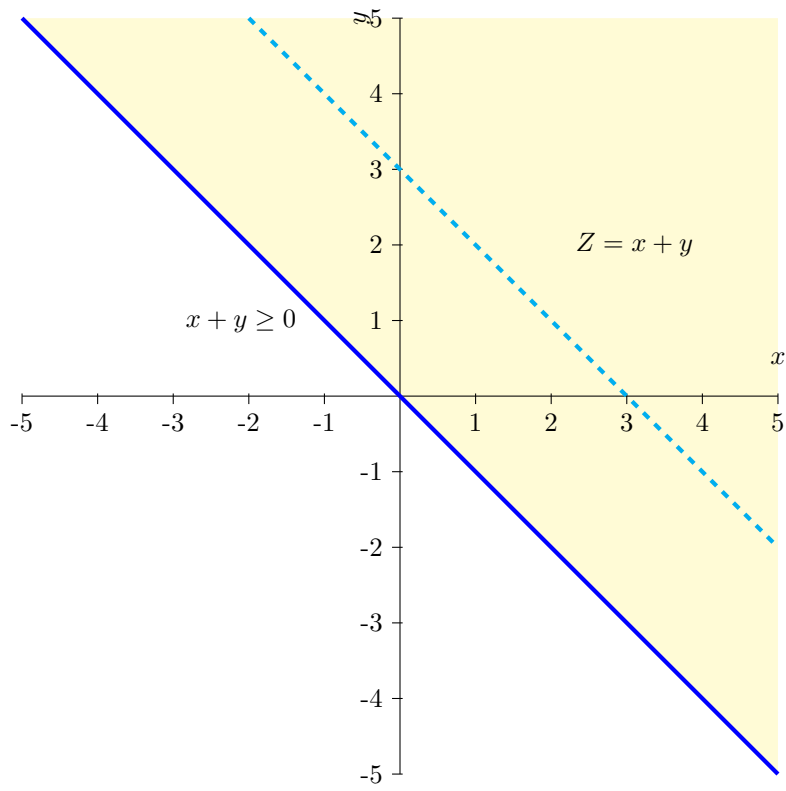


Figure C.5: An linear program with no corner points.

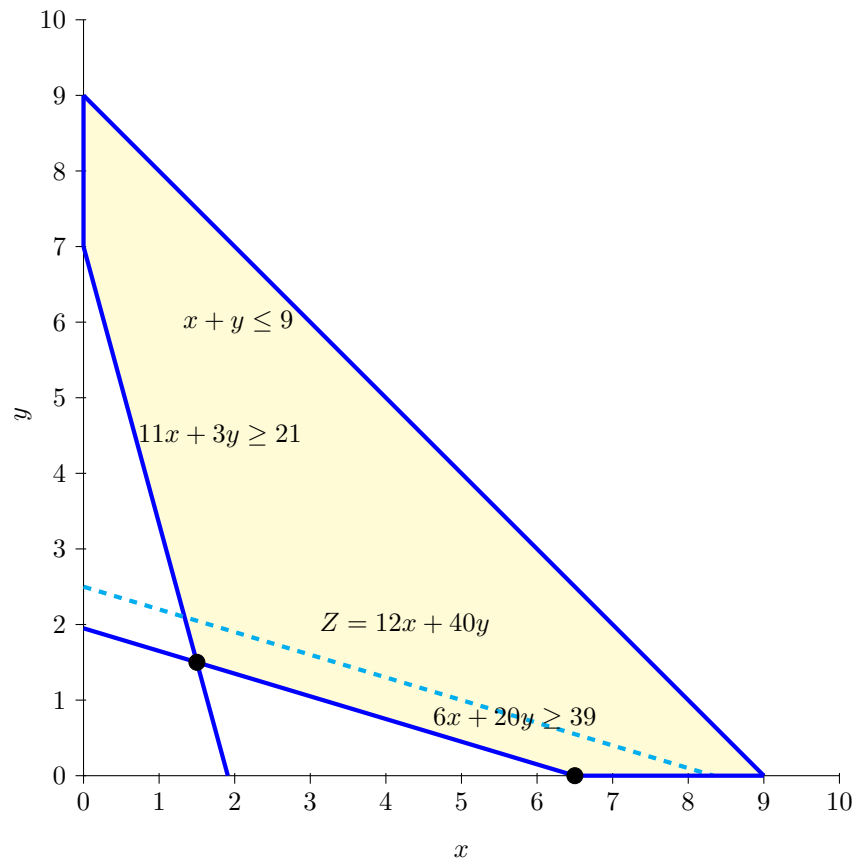


Figure C.6: Multiple optimal solution

an optimal solution if we just check corners. (For most purposes it is fine to identify a single optimal solution, Identifying all optima is usually unnecessary.)

This graphical method is good for building intuition and illustrating the main ideas behind solving linear programs. However, it becomes difficult as problems become larger. A problem with three decision variables would require a 3D drawing; a problem with four decision variables would require drawing in 4 dimensions, which is beyond the scope of most of us mere mortals. Practical-sized problems commonly involve thousands or even millions of decision variables, so a different technique is needed. However, the idea of focusing on corner points can be translated to problems of this scale. To do so requires a systematic procedure for generating and checking corner points. This procedure is called the *simplex algorithm*. The mathematical tools for doing this rely on linear algebra, since corner points can be identified by solving systems of linear equations (in the examples in this section, each corner point was the intersection of two lines). The remainder of this section is focused on presenting this method.

C.2.2 Standard form of a linear program

The simplex algorithm is easier to explain if the linear program is converted to a standard form. This standard form is not restrictive, and you can transform any linear program into an equivalent one which is in standard form. A linear program is in standard form if:

- The objective function is to be minimized.
- All decision variables have a non-negativity constraint (that is, $x_i \geq 0$ is a constraint for every decision variable x_i).
- All other constraints are equality (=) constraints.

In the previous section, we considered this linear program:

$$\begin{array}{ll} \min_{x,y} & 10x + 26y \\ \text{s.t.} & 11x + 3y \geq 21 \\ & 6x + 20y \geq 39 \\ & x + y \leq 9 \\ & x, y \geq 0 \end{array}$$

This is a minimization problem where all variables must be non-negative; but the remaining constraints are inequalities, not equalities. We can fix this by introducing three new variables s_1 , s_2 , and s_3 , and rewriting the optimization problem:

$$\begin{array}{ll} \min_{x,y,s_1,s_2,s_3} & 10x + 26y \\ \text{s.t.} & 11x + 3y - s_1 = 21 \\ & 6x + 20y - s_2 = 39 \\ & x + y + s_3 = 9 \\ & x, y, s_1, s_2, s_3 \geq 0. \end{array}$$

This problem is equivalent to the original one in the sense that any solution to the standard-form problem can easily be translated to a solution in the original problem (just ignore the new variables); and a solution is feasible to one problem if and only if it is feasible to the other problem. Given a solution (x, y) to the original problem, let $s_1 = 11x + 3y - 21$, $s_2 = 6x + 20y - 30$, and $s_3 = 9 - (x + y)$. If that solution is feasible, these new variables must all be non-negative, and the three equality constraints in the standard-form problem will be satisfied. The reverse is true as well; for instance, since $11x + 3y = 21 + s_1$ in a feasible solution to the standard-form problem, and since $s_1 \geq 0$, we must have $11x + 3y \geq 21$ and the first constraint in the original problem is also satisfied. The new variables introduced (s_1 , s_2 , and s_3) are often called *slack* variables, because they show how much the left-hand side of the constraint can be changed before it is violated.

This technique can be used to ensure that all constraints are equalities. Similarly, non-standard problems violating the other requirements can also be converted into equivalent, standard-form linear programs. If the objective function is maximization, rather than minimization, we can multiply the objective function by -1 . (Minimizing $-f$ is the same as maximizing f ; see Proposition B.1 in Section B.6.2.) If there is a decision variable which is “free” (it does not have to be non-negative), it can be replaced with two new decision variables. For instance, if the variable y can be either negative or positive, everywhere y appears in the optimization problem we can replace it with $y^+ - y^-$, where y^+ and y^- are two new decision variables with non-negativity constraints ($y^+ \geq 0$, $y^- \geq 0$). This works because we represent any value (positive or negative) as a subtraction of two positive numbers.

It is common to write linear programs in matrix-vector form, since this is more convenient when there are a large number of decision variables and constraints. In matrix notation, the linear program in standard form can be written as:

$$\begin{array}{ll} \min_{\mathbf{x}} & \mathbf{c} \cdot \mathbf{x} \\ \text{s.t.} & \mathbf{A}\mathbf{x} = \mathbf{b} \\ & \mathbf{x} \geq \mathbf{0} \end{array}$$

where \mathbf{x} is the vector of decision variables, \mathbf{c} is a vector representing the coefficients in the objective function, \mathbf{b} is a vector representing the right-hand side of each equality constraint, and \mathbf{A} is the matrix of coefficients in all of the equality constraints. (Section A.2 provides a review of vector and matrix operations if you are unfamiliar with them.) For the above example in standard form, these vectors and matrices are as follows:

$$\mathbf{x} = \begin{bmatrix} x \\ y \\ s_1 \\ s_2 \\ s_3 \end{bmatrix}$$

$$\mathbf{c} = \begin{bmatrix} 10 \\ 26 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$\mathbf{A} = \begin{bmatrix} 11 & 3 & -1 & 0 & 0 \\ 6 & 20 & 0 & -1 & 0 \\ 1 & 1 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{b} = \begin{bmatrix} 21 \\ 39 \\ 9 \end{bmatrix}$$

C.2.3 Simplex algorithm

Usually, a linear program has an infinite number of feasible solutions. However, as discussed above, it is enough to look at corner points of the feasible region, since at least one of them will be optimal (assuming that the problem has an optimal solution, and that there is a corner point, conditions we will take as given in this subsection). The simplex algorithm is designed to systematically search these corner points. Starting from one corner point, it moves to an adjacent corner point with an equal or smaller objective function value. Since there are a finite number of corner points, and since linear programs have the property that you can reach an optimal corner from any other by a sequence of adjacent corners with nonincreasing objective function values, this procedure is guaranteed to find an optimal solution. A corner point can be identified by solving systems of equations representing a subset of the constraints, so the simplex method is presented in terms of linear algebra. This subsection provides the steps of the simplex method, along with some intuitive justification for them, but we do not provide rigorous proofs of correctness, nor do we handle all of the “corner cases” that can arise when applying the simplex method. A reader interested in these details is referred to a standard text in linear programming, such as Bertsimas and Tsitsiklis (1997).

Assume we are given a standard-form linear program, where the decision variable \mathbf{x} is a vector of dimension n , the objective function coefficient \mathbf{c} is a vector of dimension n , \mathbf{b} is a vector of dimension m , and the coefficient matrix \mathbf{A} is a matrix of dimension $m \times n$. We can assume there are at least as many variables as equality constraints ($n \geq m$); otherwise some of the constraints are redundant and can be removed. (Indeed, if the number of independent constraints is equal to the number of variables, there is only one feasible solution!) Let \mathbf{A}_i be the vector corresponding to the i -th column of the matrix \mathbf{A} .

$$A_i = \begin{bmatrix} a_{i1} \\ a_{i2} \\ \vdots \\ a_{im} \end{bmatrix}$$

$$\begin{array}{ll} \min_{\mathbf{x}} & \mathbf{c} \cdot \mathbf{x} \\ \text{s.t.} & \mathbf{A}\mathbf{x} = \mathbf{b} \\ & \mathbf{x} \geq 0 \end{array}$$

A *basis* B refers to a set of m linearly independent columns of \mathbf{A} . These columns are represented by the square matrix \mathbf{A}_B , which is of size $m \times m$. We will use NB (for “nonbasic”) to represent all of the other columns of \mathbf{A} . By limiting ourselves to the decision variables x_i corresponding to the columns in the basis, we now have exactly the same number of decision variables and constraints. Fixing all of the other decision variables to zero, we can now solve the m equality constraints in terms of these m basic variables. The resulting solution \mathbf{x}_B corresponds to an intersection point of the constraints. If $\mathbf{x}_B \geq \mathbf{0}$, then this solution satisfies all of the constraints, and is therefore feasible. The resulting value of \mathbf{x} , whose basic components solve the constraints ($\mathbf{x}_B = \mathbf{A}_B^{-1}\mathbf{b}$), and whose nonbasic components are zero ($\mathbf{x}_N = \mathbf{0}$) form a corner point solution.

Every corner point can be identified in this way. So the question is to find some basis B corresponding to an optimal solution. Starting from some initial basis, the simplex method looks for an adjacent basis (one which is the same except for one column) with a lower value of the objective function. If one exists, we update the basis and repeat the process to try to find a better one. If no adjacent basis is better, we terminate with the optimal solution. The simplex method can determine whether an adjacent basis is better by computing the *reduced cost*, the rate of change in the objective function if a nonbasic column were to enter the basis in place of one of the existing columns. Since the objective function is minimization, a negative reduced cost will represent a better solution. Using matrix algebra, we can easily compute reduced costs, and new bases, using formulas shown below.

The steps of the simplex method are as follows:

1. Initialization:

- Determine a basis \mathbf{A}_B comprising of m linearly independent columns of \mathbf{A} .
- Calculate $\mathbf{x}_B = \mathbf{A}_B^{-1}\mathbf{b}$.
- If $\mathbf{x}_B \geq \mathbf{0}$, continue; otherwise, this basis is infeasible. Choose a different basis and repeat this step. If you have already tried every basis, the linear program is infeasible. Terminate the algorithm and report this.

- Set $\mathbf{x}_N = 0$.

2. Compute reduced costs:

- Compute the reduced cost of each nonbasic index $j \in N$ as $\bar{c}_j = c_j - \mathbf{c}_B^T \mathbf{A}_B^{-1} \mathbf{A}_j$, where \mathbf{c}_B is the column vector of objective function coefficients for the basic variables.
- If $\bar{c}_j \geq 0$ for all $j \in N$, then the current solution is optimal. Terminate the algorithm and return the solution.
- Otherwise, choose some nonbasic variable $j \in N$ with $\bar{c}_j < 0$. The column representing this variable will enter the basis.
- Set $d_j = 1$, and $d_i = 0$ for all $i \in N - \{j\}$.

3. Identify descent direction:

- Compute the basic direction $\mathbf{d}_B = -\mathbf{A}_B^{-1} A_j$.
- If $d_i \geq 0$ for all $i \in B$, the linear program is unbounded. Terminate the algorithm and report this.
- Otherwise, $d_i < 0$ for at least one $i \in B$. Compute the maximum step size

$$\theta = \min_{i \in B: d_i < 0} \left(\frac{-x_i}{d_i} \right). \quad (\text{C.1})$$

- Let k be the index of a variable achieving this minimum, with $\theta = -x_i/d_i$. The column representing this variable will leave the basis (to be replaced with column j).

4. Compute new solution:

- Set $x_i^{\text{new}} = x_i + \theta d_i$ for all variables i (both basic and nonbasic).
- Update the basis \mathbf{A}_B by replacing A_k with A_j .
- Update the index sets by removing k from B , and adding j . Likewise, remove j from N and add k .
- Update the vectors \mathbf{c}_B , \mathbf{x}_B , \mathbf{x}_N appropriately based on $\mathbf{x}_i^{\text{new}}$.
- Return to step 2.

As an example, we show how the simplex method will operate on the following standard-form linear program (Figure C.7 plots the feasible region).

$$\begin{array}{ll} \min_{x, y, s_1, \dots, s_4} & 10x + 26y \\ \text{s.t.} & 11x + 3y - s_1 = 21 \\ & 6x + 20y - s_2 = 39 \\ & x + y + s_3 = 9 \\ & -\frac{1}{3}x + y + s_4 = 6 \\ & 4x - 5y + s_5 = 20 \\ & x, y, s_1, s_2, s_3, s_4 \geq 0 \end{array}$$

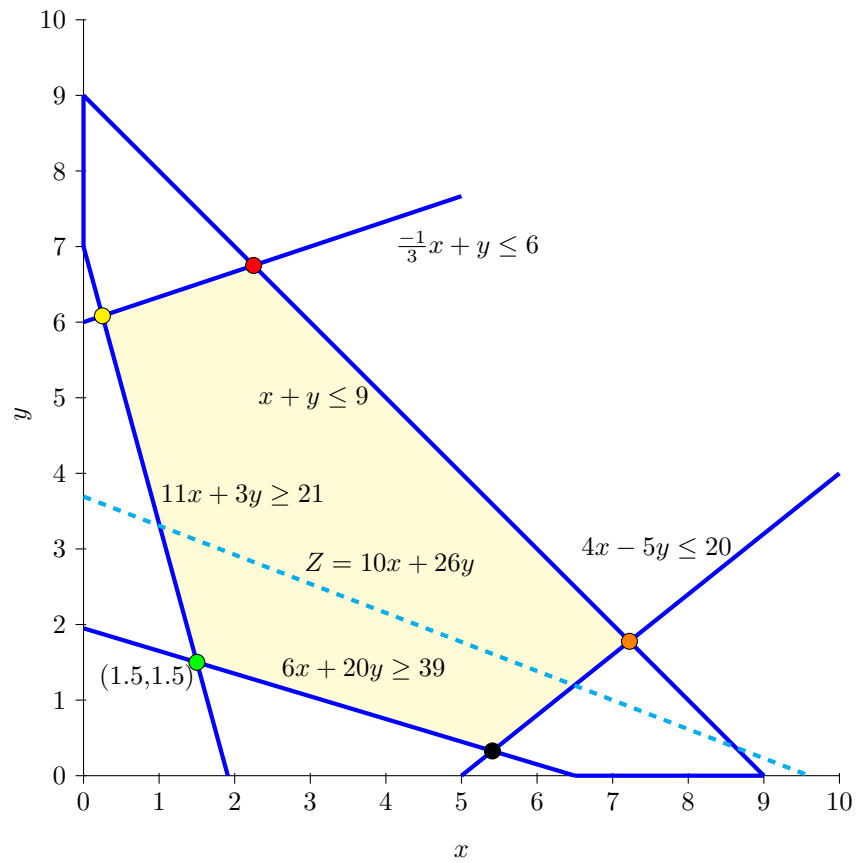


Figure C.7: Feasible Region of the LP

Initialization: Set $B = \{1, 2, 3, 4, 7\}$ and $N = \{5, 6\}$. Then

$$\mathbf{b} = \begin{bmatrix} 21 \\ 39 \\ 9 \\ 6 \\ 20 \end{bmatrix} \quad \mathbf{A}_B = \begin{bmatrix} 11 & 3 & -1 & 0 & 0 \\ 6 & 20 & 0 & -1 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ \frac{-1}{3} & 1 & 0 & 0 & 0 \\ 4 & -5 & 0 & 0 & 1 \end{bmatrix} \quad \mathbf{c}_B = \begin{bmatrix} 10 \\ 26 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$\mathbf{x}_B = \mathbf{A}_B^{-1} \mathbf{b} = \begin{bmatrix} x \\ y \\ s_1 \\ s_2 \\ s_5 \end{bmatrix} = \begin{bmatrix} 2.25 \\ 6.75 \\ 24.00 \\ 109.5 \\ 44.75 \end{bmatrix} \quad \mathbf{x}_N = \begin{bmatrix} s_3 \\ s_4 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

The values of x_B are non-negative, so this basis is a feasible corner point. The objective function $Z = \mathbf{c}_B^T \mathbf{x}_B = 198$. The solution corresponds to the red dot in Figure C.7.

Compute reduced costs: $N = \{5, 6\}$.

$$\bar{c}_5 = c_5 - \mathbf{c}_B^T \mathbf{A}_B^{-1} \mathbf{A}_5 = -14$$

$$\bar{c}_6 = c_6 - \mathbf{c}_B^T \mathbf{A}_B^{-1} \mathbf{A}_6 = -12$$

As both reduced costs are negative, we can pick either of them. Let us pick $j = 5$. Set $d_5 = 1, d_6 = 0$.

Identify descent direction:

Now $B = \{1, 2, 3, 4, 7\}$. Compute:

$$\mathbf{d}_B = -\mathbf{A}_B^{-1} \mathbf{A}_5 = \begin{bmatrix} -0.75 \\ -0.25 \\ -9.00 \\ -9.50 \\ 1.75 \end{bmatrix}$$

$$\theta = \min \left\{ \frac{2.25}{0.75}, \frac{6.75}{0.25}, \frac{24.0}{9.0}, \frac{109.5}{9.5} \right\} = \frac{24}{9}$$

The index $k = 3$.

Compute new solution:

$$\mathbf{x}^{\text{new}} = \mathbf{x} + \theta \mathbf{d} = \begin{bmatrix} 2.25 \\ 6.75 \\ 24 \\ 109.5 \\ 0 \\ 0 \\ 44.75 \end{bmatrix} + \theta \begin{bmatrix} -0.75 \\ -0.25 \\ -9.00 \\ -9.50 \\ 1.00 \\ 0 \\ 1.75 \end{bmatrix} = \begin{bmatrix} 0.25 \\ 6.0833 \\ 0 \\ 84.1667 \\ 2.6667 \\ 0 \\ 49.4167 \end{bmatrix}$$

The new $B = \{1, 2, 5, 4, 7\}$ and $N = \{3, 6\}$.

$$\mathbf{x}_B = \begin{bmatrix} x \\ y \\ s_3 \\ s_2 \\ s_5 \end{bmatrix} = \begin{bmatrix} 0.25 \\ 6.083 \\ 2.6667 \\ 84.1667 \\ 49.4167 \end{bmatrix} \quad \mathbf{x}_N = \begin{bmatrix} s_1 \\ s_4 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$\mathbf{A}_B = \begin{bmatrix} 11 & 3 & 0 & 0 & 0 \\ 6 & 20 & 0 & -1 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ -\frac{1}{3} & 1 & 0 & 0 & 0 \\ 4 & -5 & 0 & 0 & 1 \end{bmatrix} \quad \mathbf{c}_B = \begin{bmatrix} 10 \\ 26 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

The objective function $Z = \mathbf{c}_B^T \mathbf{x}_B = 160.66588$. The basic feasible solution corresponds to the yellow dot in Figure C.7.

Compute reduced costs: $N = \{3, 6\}$.

$$\bar{c}_3 = c_3 - \mathbf{c}_B^T \mathbf{A}_B^{-1} \mathbf{A}_3 = 1.5556$$

$$\bar{c}_6 = c_6 - \mathbf{c}_B^T \mathbf{A}_B^{-1} \mathbf{A}_6 = -21.333$$

As the reduced cost $c_6 < 0$, pick $j = 6$. Set $d_6 = 1, d_3 = 0$.

Identify descent direction:

Now $B = \{1, 2, 5, 4, 7\}$. Compute:

$$\mathbf{d}_B = -\mathbf{A}_B^{-1} \mathbf{A}_6 = \begin{bmatrix} 0.25 \\ -0.9167 \\ 0.6667 \\ -16.8333 \\ -5.5833 \end{bmatrix}$$

$$\theta = \min \left\{ \frac{6.083}{0.967}, \frac{84.1667}{16.8333}, \frac{49.4167}{5.5833} \right\} = \frac{84}{16.8333}$$

The index $k = 4$.

Compute new solution:

$$\mathbf{x}^{\text{new}} = \mathbf{x} + \theta \mathbf{d} = \begin{bmatrix} 0.25 \\ 6.0833 \\ 0 \\ 84.1667 \\ 2.6667 \\ 0 \\ 49.4167 \end{bmatrix} + \theta \begin{bmatrix} 0.25 \\ -0.9167 \\ 0.6667 \\ -16.8333 \\ 0.6667 \\ 1.0 \\ -5.5833 \end{bmatrix} = \begin{bmatrix} 1.5 \\ 1.5 \\ 0 \\ 0 \\ 6.0 \\ 5.0 \\ 21.5 \end{bmatrix}$$

The new $B = \{1, 2, 5, 6, 7\}$ and $N = \{3, 4\}$.

$$\mathbf{x}_B = \begin{bmatrix} x \\ y \\ s_3 \\ s_4 \\ s_5 \end{bmatrix} = \begin{bmatrix} 1.5 \\ 1.5 \\ 6.0 \\ 5.0 \\ 21.5 \end{bmatrix} \quad \mathbf{x}_N = \begin{bmatrix} s_1 \\ s_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$\mathbf{A}_B = \begin{bmatrix} 11 & 3 & 0 & 0 & 0 \\ 6 & 20 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ \frac{-1}{3} & 1 & 0 & 1 & 0 \\ 4 & -5 & 0 & 0 & 1 \end{bmatrix} \quad \mathbf{c}_B = \begin{bmatrix} 10 \\ 26 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

The objective function $Z = \mathbf{c}_B^T \mathbf{x}_B = 54.0$. The basic feasible solution corresponds to the green dot in Figure C.7.

Calculate the reduced cost: $N = \{3, 4\}$.

$$\bar{c}_3 = c_3 - \mathbf{c}_B^T \mathbf{A}_B^{-1} \mathbf{A}_3 = 0.2178$$

$$\bar{c}_4 = c_4 - \mathbf{c}_B^T \mathbf{A}_B^{-1} \mathbf{A}_4 = 1.2673$$

As both the reduced costs are positive, we have reached the optimal solution which is $x = 1.5, y = 1.5$ with an objective function value of 54.

The above procedure provides the basic framework for the simplex algorithm. There are a few issues that can arise when applying this algorithm. First, the issue of *degeneracy*, which occurs when a basic variable has the value of zero. Usually it is the nonbasic variables that are equal to zero; but sometimes we need to set a basic variable to zero to solve the system of equations. This can cause a few problems, and a poorly-designed implementation of the simplex algorithm may get stuck in an infinite loop in the presence of degeneracy, rotating among the same set of columns over and over again with no change in the objective function or the solution, just changing which zero variables count as basic and nonbasic. Degeneracy can be addressed by a good tiebreaking rule that can prevent these infinite loops. One such rule is *Bland's rule*: among all the indices j for which \bar{c}_j is a negative reduced cost, choose the first of them (smallest j) to enter the basis; and among all the indices k which achieve the minimum in (C.1), choose the first of them (smallest k) to leave the basis.

There are also a few steps which were not unambiguously specified in the algorithm above.

Identifying an initial basis: Choosing an initial basis which corresponds to a feasible solution can be difficult, especially if there are many columns. One solution to this problem is the “big- M method,” which generates an auxiliary variable y_i for each of the m equality constraints. To each of the constraints, we either add y_i (if the right-hand side b_i is positive) or subtract y_i (if b_i is negative). In the objective function, we add $M y_i$ for each of these auxiliary variables, and we add a non-negativity constraint

for each y_i . By adding these variables, it is easy to get an initial solution by setting $y_i = |b_i|$ for each constraint. If M is a large enough constant, this solution will have a very poor objective function value, and *any* feasible corner point solution to the original problem will have a lower objective. So, we can start the simplex method at this solution, and as it moves to better and better solutions it will eventually remove all of the artificial variables from the basis. The resulting optimal solution will be feasible to the original problem as well.

Selecting a nonbasic variable to enter the basis: In the description of the simplex method, we left the choice j of an entering variable arbitrary; we could choose any nonbasic variable with a negative reduced cost. If there is more than one choice, we have to decide how to proceed, and this choice can affect the number of iterations required. In the above example, in the first iteration, we had $\bar{c}_5 = -14$ and $\bar{c}_6 = -12$, either of them could have been chosen. We chose the nonbasic variable corresponding to index 5 (s_3) to enter the basis. The simplex algorithm then traveled to the basic feasible solution corresponding to the yellow dot (Figure C.7), and then reached the optimal solution in the next round. Therefore, when we picked s_3 , the simplex needed two more iterations to reach the optimal solution. If we had instead chosen the nonbasic variable corresponding to index 6 (s_4) to enter the basis, the algorithm would have traveled to the orange dot, then the black dot and finally reached the optimal solution. Therefore, the choice of index 6 would have resulted in one additional iteration. Unfortunately, there is no known way to predict in advance how many iterations will be required. There are two common methods which perform acceptably in practice: either pick the nonbasic variable with the most negative reduced cost; or among the nonbasic variables with negative reduced costs, pick the one with the smallest index.

C.3 Unconstrained Nonlinear Optimization

This section discusses optimization problems of the form $\min f(\mathbf{x})$, where \mathbf{x} is an n -dimensional real vector, with *no* constraints whatsoever. We begin by deriving optimality conditions, and then present methods for solving such problems.

C.3.1 Optimality conditions

In the main text, Section 3.3 gave optimality conditions for optimization problems with a convex objective function, linear equality constraints, and non-negativity constraints on the variables. Not all optimization problems fall into these categories, and this section provides some information on these cases. Recall that the purpose of optimality conditions is to provide mathematical equations or inequalities that can be used to check whether an optimal solution has been found or not. In theory, it may be possible to obtain an optimal solution by solving such a system of equations and inequalities directly. However,

this is often very difficult. Instead, the optimality conditions are mostly used in solution algorithms to know when an optimal solution has been found (or if we are close to optimality), and to provide guidance on how to improve a solution if it is not optimal. Throughout this discussion, we make heavy reference to “necessary” and “sufficient” conditions, explained in Section B.6.4.

This subsection deals with unconstrained nonlinear minimization problems of the form:

$$\min \{f(\mathbf{x}), \mathbf{x} \in \mathbb{R}^n\}$$

The objective function may or may not be convex. Some of the results require assumptions on differentiability, which we will state as needed. These results will be stated without proof; readers wanting more explanation are referred to the books by Bertsekas (2016) and Bazaraa et al. (2006).

This section presents several necessary and sufficient optimality conditions for unconstrained nonlinear minimization problems. Optimality conditions are important because they help identify if a given solution is optimal or not. This can help in algorithm development to check if we can stop the algorithm or proceed further. In certain specific cases, the optimality conditions can also help solve for the optimal solution or arrive at a set containing optimal solutions.

Definition C.1. *If the function f is differentiable, a stationary point is a value of \mathbf{x}^* such that $\nabla f(\mathbf{x}^*) = 0$.*

In many cases, stationary point is either a local minimum or local maximum. However, this is not always the case; for instance if $f(x) = x^3$, then $x^* = 0$ is a stationary point, but the function has neither a minimum or maximum there. A stationary point which is neither a local minimum or a local maximum is called a *saddle point*.

Theorem C.1. (First-order necessary conditions for local minima.) *If f is differentiable at a point \mathbf{x}^* which is a local minimum, then \mathbf{x}^* is also a stationary point.*

This result is a necessary condition; it is “first-order” because it refers to the first derivative. Therefore, if \mathbf{x}^* is a local minimum, then it is a stationary point. But stationary points need not be local minima; they could also be local maxima or saddle points, for instance.

Example C.6. *Determine the stationary points of the function $f(x) = (x-5)^2$. Does a minimum exist?*

Solution.

$$\nabla f(x^*) = 2(x^* - 5) = 0 \implies x^* = 5.$$

Thus $x^* = 5$ is the stationary point. By plotting the graph, one can easily determine that the stationary point corresponds to a local as well as global minimum. ■

Example C.7. *Determine the stationary points of the function $f(x) = x^3 + x^2 - x + 1$. Does a minimum exist?*

Solution.

$$\nabla f(x^*) = 3(x^*)^2 + 2x^* - 1 = 0 \implies x^* = 1/3, -1.$$

If you plot the graph, you will notice that $x^* = 1/3$ corresponds to a local minimum and $x^* = -1$ corresponds to a local maximum. ■

Example C.8. Determine the stationary points of the function $f(x_1, x_2) = 4(x_1 - 7)^2 + 5x_2^2(x_2 - 10)$.

Solution.

$$\frac{\partial f}{\partial x_1^*} = 8(x_1^* - 7) = 0 \implies x_1^* = 7$$

$$\frac{\partial f}{\partial x_2^*} = 3(x_2^*)^2 - 20x_2^* = 0 \implies x_2^* = 0, 20/3.$$

The two stationary points are $(7, 0)$ and $(7, 20/3)$. However, we do not have enough information to determine if the stationary points are minima, maxima, or saddle points. ■

If the first order necessary condition does not provide clarity on whether a stationary point is a minimum or not, a second-order necessary condition can be used if the function is twice differentiable:

Theorem C.2. (Second-order necessary conditions for local minima.) *If f is twice differentiable at a point \mathbf{x}^* which is a local minimum, then (i) \mathbf{x}^* is a stationary point, and (ii) the Hessian matrix $Hf(\mathbf{x}^*)$ is positive semidefinite.*

The theorem can be extended as follows: if \mathbf{x}^* is a local minimum, then \mathbf{x}^* is stationary and the Hessian is negative semidefinite there; if \mathbf{x}^* is a saddle point, then \mathbf{x}^* is stationary and the Hessian is neither positive semidefinite nor negative semidefinite. Again, these are *necessary* conditions. They must be true of every optimal solution, but not every point satisfying these conditions is optimal.

Example C.9. Consider the function $f(x) = (x - 5)^5$. Identify the stationary point and check if the Hessian is positive semidefinite.

Solution.

$$\nabla f(x^*) = 5(x^* - 5)^4 = 0 \implies x^* = 5.$$

The stationary point is $x^* = 5$. The Hessian at the stationary point is $f''(x^*) = 20(x^* - 5)^3$, which is nonnegative (actually zero) and thus positive semidefinite. However if you plot the function, you will notice that $x^* = 5$ does not correspond to a local minima, but instead a saddle point. ■

Example C.10. Consider the stationary point $(7, 0)$ of the function $f(x_1, x_2) = 4(x_1 - 7)^2 + 5x_2^2(x_2 - 10)$. Is $(7, 0)$ is a saddle point?

Solution. At $(7, 0)$ we have

$$\begin{aligned}\frac{\partial^2 f}{\partial x_1^2} &= 8 \\ \frac{\partial^2 f}{\partial x_1 \partial x_2} &= 0 \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} &= 0 \\ \frac{\partial^2 f}{\partial x_2^2} &= 6x_2 - 20,\end{aligned}$$

so the Hessian is given by

$$Hf = \begin{bmatrix} 8 & 0 \\ 0 & -20 \end{bmatrix}.$$

The Hessian is neither positive nor negative semidefinite. Therefore, the stationary point $(7, 0)$ is a saddle point. ■

A minor modification of the second-order necessary conditions gives a *sufficient* condition on optimality.

Theorem C.3. (Second-order sufficiency conditions for a strict local minimum.) *For a function f twice-differentiable at \mathbf{x}^* , \mathbf{x}^* is a strict local minimum if (i) \mathbf{x}^* is a stationary point and (ii) the Hessian matrix $H(\mathbf{x}^*)$ is positive definite.*

Likewise, if H is negative definite, then we know \mathbf{x}^* is a strict local maximum. As a sufficient condition, any point satisfying these conditions must be a local minimum. It is not necessary; for instance if $f(x) = x^2$, then $x^* = 0$ is a strict local minimum, but the Hessian is not positive definite there.

Example C.11. *Verify if the stationary points of the function $f(x) = x^3 + x^2 - x + 1$ are local minima. Does a minimum exist?*

Solution. The stationary points of the function are $x^* = 1/3$ and $x^* = -1$.

$$Hf(x^*) = 6(x^*) + 2$$

The Hessian is positive definite at $x^* = 1/3$ and negative definite at $x^* = -1$. Therefore, $x^* = 1/3$ is a strict local minimum and $x^* = -1$ is a strict local maximum. ■

Example C.12. *Consider the stationary point $(7, 20/3)$ of the function $f(x_1, x_2) = 4(x_1 - 7)^2 + 5x_2^2(x_2 - 10)$. Is $(7, 20/3)$ is a strict local minimum?*

Solution. At $(7, 20/3)$ we have

$$\begin{aligned}\frac{\partial^2 f}{\partial x_1^2} &= 8 \\ \frac{\partial^2 f}{\partial x_1 \partial x_2} &= 0 \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} &= 0 \\ \frac{\partial^2 f}{\partial x_2^2} &= 6x_2 - 20,\end{aligned}$$

so the Hessian is

$$Ff = \begin{bmatrix} 8 & 0 \\ 0 & 20 \end{bmatrix}.$$

This matrix is positive definite, so $(7, 20/3)$ is a strict local minimum. ■

Recall from Section 3.3 that if f is a convex function, then any local minimum must also be a global minimum, and that if f is strictly convex, any local minimum is the (unique) strict global minimum.

C.3.2 Solution framework

Unconstrained optimization problems are commonly solved using the following algorithmic framework; the details of steps 1–3 are filled in later in this subsection.

Step 0: Initialize. Set the iteration counter $k = 0$, and initialize \mathbf{x}_0 by choosing any vector in \mathbb{R}^n . (If you have a good guess as to the minimum point, it can greatly speed things up.)

Step 1: Check convergence. Evaluate one of the termination criteria (Section C.3.3.) If it is satisfied, return \mathbf{x}_k as an approximately-optimal solution, and stop.

Step 2: Determine a descent direction \mathbf{d}_k . (Section C.3.4; this step is where gradient descent and Newton's method differ.)

Step 3: Determine a step size α_k . (Section C.3.5.)

Step 4: Iterate. Update the solution, $\mathbf{x}_{k+1} \leftarrow \mathbf{x}_k + \alpha_k \mathbf{d}_k$, then increase k by 1, and return to step 1.

The following subsections spell out choices for Steps 1, 2, and 3. These choices are independent of each other, and you can combine any choice for one step with any choice for another.

C.3.3 Convergence criteria

There are several stopping criteria available to decide when to terminate. It is always best to select a stopping criterion to directly measure what we are trying to do. In this case, we know that the minimum of f must occur at a stationary point, which means $\nabla f = \mathbf{0}$. So, a natural stopping criteria is to terminate when ∇f is “close enough” to zero, that is, when $\|\nabla f\| < \epsilon$.

One possible difficulty with this choice is that you need a good intuition for what values of $\|\nabla f\|$ correspond to “good enough” solutions for your particular application, and this is not always easy. So, there are other convergence criteria that can also be used:

- You can stop when the solution stabilizes, $\|\mathbf{x}_k - \mathbf{x}_{k-1}\| < \epsilon$.
- You can stop when the objective function stabilizes, $|f(\mathbf{x}_k) - f(\mathbf{x}_{k-1})| < \epsilon$.
- You can normalize the previous two inequalities to reflect relative stability (e.g., stop when the objective function decreases by less than 1% between iterations).
- You can stop after a pre-specified number of iterations.
- You can stop after a certain amount of computation time has elapsed.

These methods are more intuitive to apply than $\|\nabla f\| < \epsilon$. The downside is that there is no guarantee that you are close to an optimal solution when they are satisfied. This is particularly obvious for the last criteria; but even for the earlier ones, there is no way to tell the difference between a solution “stabilizing” for a good reason (you are close to the minimum) or for a bad reason (the algorithm is stuck somewhere suboptimal but can’t make progress).

You can also use a combination of these methods; for instance, stopping when the objective changes by less than 1%, or after one hour of run time (whichever comes first).

C.3.4 Descent direction

There are two main ways to select the direction \mathbf{d}_k ; the idea in both cases is that the gradient points in the direction where the function *increases* as quickly as possible. Therefore, the negative of the gradient is the direction where the function decreases as quickly as possible. Since we are trying to minimize the objective, we will use the latter in some way.

In *gradient descent*, we simply use the gradient itself, choosing $\mathbf{d}_k = -\nabla f(\mathbf{x}_k)$. This method is simple and requires relatively little computation.

A more sophisticated choice is *Newton’s method*, which is based on the following logic: we can take a quadratic approximation to the objective function at the current point. We can identify the minimum point of that quadratic approximation more easily than the original function f (which may be much more complicated), and then choose the search direction \mathbf{d}_k to move towards

that minimum point. We will derive this direction, and then explain how it is related to Newton's method as you learned it in calculus, or as we used it in the previous section.

The quadratic approximation to f at \mathbf{x}_k is its second-order Taylor series, based on its gradient and Hessian:

$$f(\mathbf{x}) \approx f(\mathbf{x}_k) + (\mathbf{x} - \mathbf{x}_k)^T \nabla f(\mathbf{x}_k) + \frac{1}{2}(\mathbf{x} - \mathbf{x}_k)^T Hf(\mathbf{x}_k)(\mathbf{x} - \mathbf{x}_k)^T. \quad (\text{C.2})$$

Let \tilde{f} denote the right-hand side of equation (C.2). The minimum point of the quadratic approximation is the point where $\nabla \tilde{f}$ vanishes, that is, where

$$\nabla \tilde{f}(\mathbf{x}) = \nabla f(\mathbf{x}_k) + Hf(\mathbf{x}_k)(\mathbf{x} - \mathbf{x}_k) = 0. \quad (\text{C.3})$$

Solving for \mathbf{x} , we obtain

$$\mathbf{x} = \mathbf{x}_k - (Hf(\mathbf{x}_k))^{-1} \nabla f(\mathbf{x}_k). \quad (\text{C.4})$$

So, the search direction is

$$\mathbf{d}_k = \mathbf{x} - \mathbf{x}_k = -(Hf(\mathbf{x}_k))^{-1} \nabla f(\mathbf{x}_k). \quad (\text{C.5})$$

Why is this method related to Newton's method? Newton's method aims to find the zero of a function by finding the zero of its linear approximation. In this case, we are trying to find the minimum point of f , which occurs where $\nabla f = 0$. So, when we use Newton's method, we find where the linear approximation of ∇f vanishes. But if the gradient of a function is linear, then the original function is quadratic, and the gradient will be zero at the minimum point of this quadratic. Also recall the previous section, where in a one-dimensional setting we used Newton's method to try to find the point where $f'(x) = 0$, and ended up reducing x by $f'(x)/f''(x)$. The formula $\mathbf{d}_k = -(Hf(\mathbf{x}_k))^{-1} \nabla f(\mathbf{x}_k)$ is analogous: the gradient is the generalization of the first derivative; the Hessian is the generalization of the second derivative; and multiplying by the inverse of a matrix is the generalization of dividing by a scalar (just as multiplying by 2^{-1} is the same as dividing by 2). Indeed, in the single-dimensional case, equation (C.5) simplifies to

$$d_k = -f'(x_k)/f''(x_k), \quad (\text{C.6})$$

so the new point $x_{k+1} = x_k + d_k$ is given by exactly the same formula as in Step 2 of Newton's method for line search in Section C.1.2

Newton's method usually yields a "better" search direction, and fewer iterations are required to reach the minimum. However, it can be computationally expensive to calculate all the elements in the Hessian matrix, and to calculate its inverse. For this reason, there are a variety of "quasi-Newton" methods, which replace H^{-1} in equation (C.5) by another matrix which is easier to calculate, but is approximately the same.

C.3.5 Step size

There are several procedures available to determine the step size α_k . The simplest is to use a *constant step size*, where at each iteration $\alpha_k = \alpha$, for some pre-determined constant α . If the predetermined value is too big, the algorithm may overshoot the minimum, slowing convergence or even diverging. If the step size is too small, the algorithm may take a long time to converge. If you are determining the search direction using Newton's method, a choice of $\alpha = 1$ may actually perform quite well, since this will move you exactly to the minimum point of the quadratic approximation. This choice will probably not work well for gradient descent, since the "units" of the gradient are not the same as for the decision variables. So, if you use this approach, you should have a good intuition about the objective function.

A more sophisticated method is *exact line search*, where you choose the value of α_k so that $f(\mathbf{x}_k + \alpha_k \mathbf{d}_k)$ is minimized. This is a one-dimensional optimization problem, which can be solved using the methods of the previous section.¹ While this choice yields the greatest improvement in the objective function at each iteration, it can be computationally expensive to compute this minimum. (You would have to use the line search method at every iteration.)

A compromise between the two is a *backtracking line search*, which also tries to minimize $f(\mathbf{x}_k + \alpha_k \mathbf{d}_k)$, but in a loose way. Rather than hoping to find the exact minimum, we choose a value of α_k which is "good enough." Select a positive initial step size η , and two other constants β and γ , both strictly between 0 and 1. We then test the sequence of α_k values $\alpha_k = \eta, \beta\eta, \beta^2\eta, \dots$, stopping as soon as this inequality is satisfied:

$$f(\mathbf{x}_k + \alpha_k \mathbf{d}_k) - f(\mathbf{x}_k) < \gamma \alpha_k \nabla f(\mathbf{x}_k)^T \mathbf{d}_k \quad (\text{C.7})$$

The first value $\eta, \beta\eta, \beta^2\eta, \dots$ which satisfies this equation is chosen for α_k . The intuition in formula (C.7) is that the left-hand side shows how much f changes if we take a step of size α_k . We are trying to solve a minimization problem, so hopefully f decreases, and the left-hand side is negative. On the right-hand side, $\alpha_k \nabla f(\mathbf{x}_k)^T \mathbf{d}_k$ is how much we would expect f to decrease based on its linear approximation. Likewise, since \mathbf{d}_k is a direction in which f decreases, the right-hand side is also a negative number. We stop at the first choice of α_k for which the actual decrease in the objective function is at least a certain fraction γ of what we would expect from the linear approximation; this is exactly what the condition (C.7) checks. Typical values of the constants in this method are $\eta = 1$, $\beta = 1/2$, and $\gamma = 1/10$, but you should experiment with different values for your specific problem. This rule is often called the *Armijo rule*.

C.3.6 Examples

Example C.13. Apply the unconstrained optimization algorithm to the function $f(x) = x^2 - 10x + 20$. Terminate when $|x_k - x_{k-1}| < 0.01$, use gradient

¹You will need to impose an upper bound on α_k to do this, but in practice this is not usually very hard.

descent for the direction, and use a constant step size of $\alpha = 1$. Repeat with a constant step size of $\alpha = 0.1$.

Solution. From basic calculus we know that the minimum of the above function is at $x^* = 5$. For this example, we pick an initial value of $x_0 = 15$. At any point x_k , the descent direction is $d_k = -\nabla f(x_k) = 2x_k - 10$. Then, using a constant step size of $\alpha = 1$, we have the following:

At the initial point, the descent direction is $d_0 = -20$, so $x_1 = 15 - 1 \times 20 = -5$. Proceeding to the next iteration, we check the termination criterion. Since $|x_1 - x_0| > 0.01$, we continue. The new descent direction is $d_1 = -20$. Therefore, $x_2 = -5 + 1 \times 20 = 15$. Since $|x_2 - x_1| > 0.01$, we proceed to the next iteration. But we've returned to our initial point! Notice that the solutions will continue to oscillate between 15 and -5 , due to the large step size.

Repeating using a smaller constant step size of $\alpha = 0.1$ produces convergence: from the initial point, the descent direction is $d_1 = -20$ and $x_1 = 15 - 0.1 \times 20 = 13$. We have $|x_1 - x_0| > 0.01$, so we continue. At this new point, the descent direction is $d_1 = -16$. Therefore, $x_2 = 13 - 0.1 \times 16 = 11.4$. Subsequent iterations are shown in Table C.3. ■

In this example, the constant step size plays a major role in how quickly we converge (if at all). For reference, Table C.4 shows how many iterations are required for different choices. Similarly, the initial value chosen, and the parameters η, β, γ of an inexact line search play an important role in convergence.

Example C.14. Apply the unconstrained optimization algorithm to the function $f(x, y) = (x - 1)^4 + 5(y - 2)^4 + xy$. Terminate when $\|\mathbf{x}_k - \mathbf{x}_{k-1}\| < 0.01$, and use gradient descent. First use the algorithm with a constant step size of $\alpha = 0.0025$, then solve again using backtracking line search with $\eta = 1$, $\beta = 0.1$, and $\gamma = 0.1$.

Solution. Notice that there are two decision variables; we will use the vector $\mathbf{x} = (x, y)$ to describe both decision variables together. For a specific iteration, we will let their values be given by $\mathbf{x}_k = (x_k, y_k)$.

For this example, we pick an initial value of $(x_1, y_1) = (4, 4)$. At any point k , the descent direction is

$$\mathbf{d}_k = -\nabla f(x_k, y_k) = \begin{bmatrix} -4(x_k - 1)^3 - y_k \\ -20(y_k - 2)^3 - x_k \end{bmatrix}.$$

So from the initial point, the descent direction is

$$\mathbf{d}_0 = -\nabla f(x_0, y_0) = \begin{bmatrix} -4(x_0 - 1)^3 - y_0 \\ -20(y_0 - 2)^3 - x_0 \end{bmatrix} = \begin{bmatrix} -112 \\ -164 \end{bmatrix}$$

and the new point is

$$\mathbf{x}_1 = \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} = \begin{bmatrix} 4 \\ 4 \end{bmatrix} + 0.0025 \begin{bmatrix} -112 \\ -164 \end{bmatrix} = \begin{bmatrix} 3.72 \\ 3.59 \end{bmatrix}.$$

For convergence, we check if $\sqrt{(x_1 - x_0)^2 + (y_1 - y_0)^2} < 0.01$. This is not true, so we increase k to 1 and move to the next iteration.

Table C.3: Gradient descent applied to $f(x) = x^2 - 10x + 20$, with a constant step size $\alpha_k = 0.1$

k	x_k	$f(x_k)$
0	15	100
1	13.0	64.0
2	11.4	40.96
3	10.12	26.2144
4	9.096	16.7772
5	8.2768	10.7374
6	7.6214	6.8719
7	7.0972	4.3980
8	6.6777	2.8147
9	6.3422	1.8014
10	6.0737	1.1529
11	5.8590	0.7379
12	5.6872	0.4722
13	5.5498	0.3022
14	5.4398	0.1934
15	5.3518	0.1238
16	5.2815	0.0792
17	5.2252	0.0507
18	5.1801	0.0325
19	5.1441	0.0208
20	5.1153	0.0133
21	5.0922	0.0085
22	5.0738	0.0054
23	5.0590	0.0035
24	5.0472	0.0022
25	5.0378	0.0014

Table C.4: Number of iterations for gradient descent to minimize $f(x) = x^2 - 10x + 20$ with different step sizes

α_k	Iterations needed
0.1	27
0.2	14
0.3	9
0.4	7
0.5	3
0.6	7
0.7	10

The new descent direction is

$$\mathbf{d}_1 = -\nabla f(x_1, y_1) = \begin{bmatrix} -4(x_1 - 1)^3 - y_1 \\ -20(y_1 - 2)^3 - x_1 \end{bmatrix} = \begin{bmatrix} -84.04 \\ -84.11 \end{bmatrix}$$

and the new point is

$$\mathbf{x}_2 = \begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} 3.72 \\ 3.59 \end{bmatrix} + 0.0025 \begin{bmatrix} -84.08 \\ -84.11 \end{bmatrix} = \begin{bmatrix} 3.51 \\ 3.38 \end{bmatrix}.$$

Since $\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} > 0.01$, we increase k to 2 and continue. The algorithm converges after 64 iterations, at the point (1.66, 2.23). The objective function has a value of 3.91.

Solving again using backtracking line search, we have the same initial points and directions: $\mathbf{x}_0 = (4, 4)$ and $\mathbf{d}_0 = (-112, -164)$. However, we have to do a little more work to determine the step size α_0 . We start by testing $\alpha_0 = \eta = 1$. With this choice, the new solution would be

$$\begin{bmatrix} 4 \\ 4 \end{bmatrix} + 1 \begin{bmatrix} -112 \\ -164 \end{bmatrix} = \begin{bmatrix} -108 \\ -160 \end{bmatrix}.$$

The value of the objective function here is very large, approximately 3.58×10^9 , so the left-hand side of (C.7) is $3.58 \times 10^9 - 177 \approx 3.58 \times 10^9$. For the right hand side, we calculate

$$\gamma \alpha_0 \nabla f(\mathbf{x}_0)^t \mathbf{d}_0 = (0.1)(1) \begin{bmatrix} 112 & 164 \end{bmatrix} \begin{bmatrix} -112 \\ -164 \end{bmatrix} = -3944.$$

Inequality (C.7) is clearly false (the left-hand side is very positive, the right-hand side is negative), so we try again with $\alpha_0 = \beta\eta = 1/10$. This choice corresponds to the solution

$$\begin{bmatrix} 4 \\ 4 \end{bmatrix} + \frac{1}{10} \begin{bmatrix} -112 \\ -164 \end{bmatrix} = \begin{bmatrix} -7.2 \\ -12.4 \end{bmatrix}.$$

The objective function has a value of 2.19×10^5 , so the left-hand side of (C.7) is still approximately 2.19×10^5 , while the right-hand side is

$$\gamma \alpha_0 \nabla f(\mathbf{x}_0)^t \mathbf{d}_0 = \frac{1}{100} \begin{bmatrix} 112 & 164 \end{bmatrix} \begin{bmatrix} -112 \\ -164 \end{bmatrix} = -394.4.$$

which is again false. Trying again with $\alpha_0 = \beta^2\eta = 1/100$, the new solution is $\mathbf{x} = (2.88, 2.36)$, and the left and right-hand sides of (C.7) are now -157.6 and -39.4 , respectively. So we accept this choice: $\alpha_0 = 1/100$, and $\mathbf{x}_1 = (2.88, 2.36)$.

Proceeding similarly, you can verify that the next step sizes are $\alpha_1 = 1/10$, $\alpha_2 = 1/10$, $\alpha_3 = 1$, and $\alpha_4 = 0.1$, with the algorithm terminating after that step at the solution $\mathbf{x}_5 = (0.236, 1.774)$, with objective value 0.772. Notice that each iteration of backtracking line search required more work, but in the end we only had to perform five iterations, rather than sixty-four. ■

Example C.15. Apply the unconstrained optimization method to the function $f(x, y) = (x - 1)^4 + 5(y - 2)^4 + xy$. Terminate when $\|\mathbf{x}_k - \mathbf{x}_{k-1}\| < 0.01$, and use Newton's method with a constant step size of $\alpha = 1$.

Solution. We will use the same initial value $\mathbf{x}_0 = (4, 4)$. As before, the gradient at an arbitrary point is

$$\nabla f(x_k, y_k) = \begin{bmatrix} 4(x_k - 1)^3 + y_k \\ 20(y_k - 2)^3 + x_k \end{bmatrix},$$

and the Hessian is

$$Hf(x_k, y_k) = \begin{bmatrix} 12(x_k - 1)^2 & 1 \\ 1 & 60(y_k - 2)^2 \end{bmatrix}.$$

So, at the initial point we have

$$\nabla f(4.0, 4.0) = \begin{bmatrix} 4(4 - 1)^3 + 4 \\ 20(4 - 2)^3 + 4 \end{bmatrix} = \begin{bmatrix} 112 \\ 164 \end{bmatrix}$$

and

$$Hf(4.0, 4.0) = \begin{bmatrix} 108 & 1 \\ 1 & 240 \end{bmatrix}.$$

The search direction from Newton's method is

$$\mathbf{d}_0 = (Hf(\mathbf{x}_1))^{-1} \nabla f(\mathbf{x}_0) = \begin{bmatrix} -1.0307 \\ -0.6790 \end{bmatrix}.$$

Using the constant step size $\alpha = 1$, the new point is

$$\begin{bmatrix} x_1 \\ y_1 \end{bmatrix} = \begin{bmatrix} 4 \\ 4 \end{bmatrix} + 1 \begin{bmatrix} -1.0307 \\ -0.6790 \end{bmatrix} = \begin{bmatrix} 2.9692 \\ 3.3209 \end{bmatrix}$$

Since $\sqrt{(x_1 - x_0)^2 + (y_1 - y_0)^2} > 0.01$, we increment k to 1 and move to the next iteration.

At $(x_1, y_1) = (2.9692, 3.3209)$, the gradient and Hessian are

$$\nabla f(2.9692, 3.3209) = \begin{bmatrix} 4(2.9692 - 1)^3 + 3.3209 \\ 20(3.3209 - 2)^3 + 2.9692 \end{bmatrix} = \begin{bmatrix} 38.8672 \\ 49.069 \end{bmatrix}$$

and

$$Hf(2.9692, 3.3209) = \begin{bmatrix} 46.5353 & 1 \\ 1 & 104.696 \end{bmatrix},$$

respectively. So the search direction is

$$\mathbf{d}_1 = -(Hf(\mathbf{x}_1))^{-1} \nabla f(\mathbf{x}_1) = \begin{bmatrix} -0.7178 \\ -0.4618 \end{bmatrix}$$

and

$$\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} 2.9692 \\ 3.3209 \end{bmatrix} + 1 \begin{bmatrix} -0.7178 \\ -0.4618 \end{bmatrix} = \begin{bmatrix} 2.2513 \\ 2.8591 \end{bmatrix},$$

and so forth. The algorithm terminates in iteration 16, to the solution $\mathbf{x} = (0.2399, 1.756)$, with an objective function value of 0.7728. ■

C.4 Constrained Nonlinear Optimization

The main text derived optimality conditions for a specific class of nonlinear optimization problems including the static traffic assignment problem (Section 3.3). Likewise, Sections 6.2.1 and 6.2.2 introduced the method of successive averages and Frank-Wolfe algorithms for solving traffic assignment. This section discusses how these techniques can be applied to other kinds of nonlinear optimization problems involving linear constraints.

In this section, we will consider optimization problems of the form

$$\begin{aligned} \min_{\mathbf{x}=(x_1,\dots,x_n)} \quad & f(\mathbf{x}) \\ \text{s.t.} \quad & g_i(\mathbf{x}) \leq 0 \quad \forall i = 1, 2, \dots, m \\ & h_j(\mathbf{x}) = 0 \quad \forall j = 1, 2, \dots, \ell, \end{aligned}$$

where the g_i and h_j are functions representing the inequality and equality constraints, respectively. We will concisely denote the feasible region by X . That is, X is the set

$$X = \{\mathbf{x} \in \mathbb{R}^n : g_i(\mathbf{x}) \leq 0 \quad \forall i = 1, 2, \dots, m; h_j(\mathbf{x}) = 0 \quad \forall j = 1, 2, \dots, \ell\}.$$

We first derive optimality conditions that apply to a general constrained nonlinear optimization problem, and then specialize them to the case where all of the constraints are linear and the objective function is convex.

C.4.1 Optimality conditions

As with unconstrained optimization, there are first- and second-order necessary and sufficient conditions characterizing the local minima of constrained optimization problems. The main distinction is that optimal solutions can now lie along the boundary of the feasible region, as well as in its interior. It is possible to create examples where constraints behave irregularly. To address this, we will need to define *constraint qualifications*, which are essentially regularity conditions on the constraints. We begin with some definitions leading up to these.

Definition C.2. For any feasible solution $\mathbf{x} \in X$, an active constraint is an inequality constraint which is satisfied with equality. That is, the i -th inequality constraint is active if $g_i(\mathbf{x}) = 0$.

Definition C.3. For any feasible solution $\mathbf{x} \in X$, the active constraint set is the set of all active constraints at \mathbf{x} : $A(\mathbf{x}) = \{i : g_i(\hat{\mathbf{x}}) = 0\}$.

We now introduce several constraint qualifications, which we will discuss below.

Definition C.4. The linear independence constraint qualification holds at a feasible solution $\mathbf{x} \in X$ if the gradient vectors $\nabla g_i(\mathbf{x})$ for active constraints $i \in A(\mathbf{x})$, and for all equality constraints $\nabla h_j(\mathbf{x})$, are linearly independent.

Definition C.5. *The Mangasarian-Fromovitz constraint qualification holds at a feasible solution $\mathbf{x} \in \mathcal{A}$ if (i) the gradient vectors $\nabla h_j(\hat{\mathbf{x}})$ are linearly independent, and (ii) there exists a direction vector $\mathbf{d} \in \mathbb{R}^n$ such that $\nabla g_i(\hat{\mathbf{x}})^T \mathbf{d} < 0$ for every $i \in A(\mathbf{x})$ and $\nabla h_j(\hat{\mathbf{x}})^T \mathbf{d} = 0$ for all $j = 1, \dots, \ell$.*

The linear independence constraint qualification is stronger, and you can show that whenever it holds, the Mangasarian-Fromovitz constraint qualification also holds. There are other constraint qualifications as well. If the linear independence constraint qualification holds, then most other constraint qualification conditions hold as well. In most nonlinear optimization formulations representing traffic assignment, all of the constraints are linear, and a simpler constraint qualification holds:

Definition C.6. *The linearity constraint qualification holds if all of the functions g_i and h_j defining the constraints are affine functions.*

Perhaps the most famous necessary optimality conditions for constrained optimization are the Karush-Kuhn-Tucker (KKT) conditions. They make use of auxiliary variables $\boldsymbol{\mu}$ and $\boldsymbol{\lambda}$, often called *Lagrange multipliers*.

Theorem C.4. (First-order necessary conditions for a local minimum.) *Let f, g_i, h_j be continuously differentiable functions. If (i) \mathbf{x}^* is a local minimum of f , and (ii) any of the above constraint qualifications hold at \mathbf{x}^* , then there exist scalars $\mu_i, i = 1, 2, \dots, m$ and $\lambda_j, j = 1, \dots, \ell$, such that the following system holds:*

$$\nabla f(\mathbf{x}^*) + \sum_{i=1}^m \mu_i \nabla g_i(\mathbf{x}^*) + \sum_{j=1}^{\ell} \lambda_j \nabla h_j(\mathbf{x}^*) = \mathbf{0} \quad (\text{C.8})$$

$$h_j(\mathbf{x}^*) = 0 \quad \forall j = 1, 2, \dots, \ell \quad (\text{C.9})$$

$$g_i(\mathbf{x}^*) \leq 0 \quad \forall i = 1, 2, \dots, m \quad (\text{C.10})$$

$$\mu_i g_i(\mathbf{x}^*) = 0 \quad \forall i = 1, 2, \dots, m \quad (\text{C.11})$$

$$\mu_i \geq 0 \quad \forall i = 1, 2, \dots, m \quad (\text{C.12})$$

The above conditions can also be written in matrix form. Let $\boldsymbol{\mu}$ and $\boldsymbol{\lambda}$ be the vectors whose components are $\mu_1, \mu_2, \dots, \mu_m$ and $\lambda_1, \lambda_2, \dots, \lambda_\ell$, and likewise let $\mathbf{g}(\mathbf{x})$ and $\mathbf{h}(\mathbf{x})$ be vectors with components $g_i(\mathbf{x})$ and $h_j(\mathbf{x})$. Then we can rewrite the KKT conditions in terms of the Jacobians of \mathbf{g} and \mathbf{h} as

$$\begin{aligned} \nabla f(\mathbf{x}^*) + J\mathbf{g}(\mathbf{x}^*)\boldsymbol{\mu} + J\mathbf{h}(\mathbf{x}^*)\boldsymbol{\lambda} &= \mathbf{0} \\ \mathbf{h}(\mathbf{x}^*) &= \mathbf{0} \\ \mathbf{g}(\mathbf{x}^*) &\leq \mathbf{0} \\ \boldsymbol{\mu}^T \mathbf{g}(\mathbf{x}^*) &= 0 \\ \boldsymbol{\mu} &\geq \mathbf{0} \end{aligned}$$

Another way of representing the first order necessary condition is using the Lagrangian function

$$\mathcal{L}(\mathbf{x}, \boldsymbol{\mu}, \boldsymbol{\lambda}) = f(\mathbf{x}^*) + \sum_{i=1}^m \mu_i g_i(\mathbf{x}^*) + \sum_{j=1}^l \lambda_j h_j(\mathbf{x}^*)$$

We can rewrite the KKT functions in a simpler way using the Lagrangian. Taken as a vector, the partial derivatives of \mathcal{L} with respect to \mathbf{x} (written as $\nabla_{\mathbf{x}}\mathcal{L}$) form the left-hand side of the first KKT condition (C.8). Similarly, the partial derivatives of \mathcal{L} with respect to $\boldsymbol{\lambda}$ form the left-hand side of the second KKT condition (C.9). So we can write the KKT conditions as:

$$\begin{aligned}\nabla_{\mathbf{x}}\mathcal{L}(\mathbf{x}^*, \boldsymbol{\mu}, \boldsymbol{\lambda}) &= \mathbf{0} \\ \nabla_{\boldsymbol{\lambda}}\mathcal{L}(\mathbf{x}^*, \boldsymbol{\mu}, \boldsymbol{\lambda}) &= \mathbf{0}\end{aligned}$$

These are necessary conditions, meaning that they must be satisfied at any optimal solution. Without additional restrictions, they are not sufficient, meaning that there may be non-optimal points which also satisfy those conditions. Still, we can identify *all* the points which satisfy the KKT conditions to generate a set of “candidate solutions;” if an optimal solution exists it must be one of them. We give some examples of how to do this in the following subsection. For large-scale problems this is not a practical approach (the following subsection describes some methods that can be used), but it may be reasonable for problems with only a few decision variables and constraints, or where there is a special structure which further simplifies these conditions.

With additional conditions on the objective function and constraints, we can give stronger results based on the KKT conditions: For instance, under the linear independence constraint qualification, there are *unique* vectors of Lagrange multipliers $\boldsymbol{\mu}$ and $\boldsymbol{\lambda}$ satisfying the KKT conditions at a local minimum. The KKT conditions can also be *sufficient*, under additional restrictions. For example, by imposing convexity conditions on f and the g_i , and linearity on the h_j , we have this result:

Theorem C.5. (First-order sufficient conditions for global minima.) *Let f and all g_i be continuously differentiable convex functions, and let all h_j be linear functions. Then if there are $\mathbf{x}^* \in \mathbb{R}^n$, $\boldsymbol{\mu} \in \mathbb{R}^m$, and $\boldsymbol{\lambda} \in \mathbb{R}^\ell$ satisfying the following conditions, \mathbf{x}^* is a global minimum of f subject to $\mathbf{x} \in X$.*

$$\nabla f(\mathbf{x}^*) + \sum_{i=1}^m \mu_i \nabla g_i(\mathbf{x}^*) + \sum_{j=1}^{\ell} \lambda_j \nabla h_j(\mathbf{x}^*) = \mathbf{0} \quad (\text{C.13})$$

$$h_j(\mathbf{x}^*) = 0 \quad \forall j = 1, 2, \dots, \ell \quad (\text{C.14})$$

$$g_i(\mathbf{x}^*) \leq 0 \quad \forall i = 1, 2, \dots, m \quad (\text{C.15})$$

$$\mu_i g_i(\mathbf{x}^*) = 0 \quad \forall i = 1, 2, \dots, m \quad (\text{C.16})$$

$$\mu_i \geq 0 \quad \forall i = 1, 2, \dots, m \quad (\text{C.17})$$

In this result, the conditions on g_i and h_j imply that the feasible region X is a convex set; since f is also a convex function, we have a convex optimization problem, and therefore any local minimum is also global. This is what allowed us to convert the “local” necessary condition into a global sufficient condition.

As with unconstrained problems, we can also formulate a second-order sufficient condition, based on second partial derivatives.

Theorem C.6. (Second-order sufficient conditions for a local minimum.) *If the objective f and all g_i and h_j defining the constraints are twice continuously differentiable; the Hessian of \mathcal{L} with respect to \mathbf{x} is positive definite at \mathbf{x}^* , $\boldsymbol{\mu}$, and $\boldsymbol{\lambda}$; and $\mu_i > 0$ for any $i \notin A(\mathbf{x}^*)$; then any solution satisfying the KKT conditions is a strict local minimum of f .*

This condition can be relaxed slightly. Rather than requiring that the matrix $H_{\mathbf{xx}}$ consisting of all second partial derivatives $\partial^2 \mathcal{L} / \partial x_i \partial x_j$ be positive definite (requiring $\mathbf{d}^T H_{\mathbf{xx}}(\mathbf{x}^*) \mathbf{d} > 0$ for all nonzero $\mathbf{d} \in \mathbb{R}^n$), it is enough if $\mathbf{d}^T H_{\mathbf{xx}}(\mathbf{x}^*) \mathbf{d} > 0$ holds for any nonzero $\mathbf{d} \in \mathbb{R}^n$ such that $\nabla h_j(\mathbf{x}^*)^T \mathbf{d} = 0$ for all j , and $\nabla g_i(\mathbf{x}^*)^T \mathbf{d} = 0$ for all active $i \in A(\mathbf{x}^*)$.

C.4.2 Examples of applying KKT conditions

This section shows how the KKT conditions can be directly solved to identify candidate solutions, and how to then identify an optimal solution among these candidates. Again, this approach is not practical for large-scale problems, but is good for small problems, for building intuition about how the conditions work, and on occasion in problems with a special structure (for instance, if the conditions can be solved in closed form).

Example C.16. *Solve the following optimization problem:*

$$\begin{aligned} \min \quad & x_2 \\ \text{s.t.} \quad & (x_1 - 1)^2 + x_2^2 \leq 1 \\ & x_1 \geq 2 \end{aligned}$$

Are the KKT conditions satisfied here?

Solution. In this formulation, there is only a single feasible point $\mathbf{x} = (2, 0)$, so this solution must be optimal (both locally and globally). To examine the KKT conditions at this point, we start by writing the two constraints can be written as

$$g_1(x_1, x_2) = (x_1 - 1)^2 + x_2^2 - 1 \leq 0 \quad g_2(x_1, x_2) = -x_1 + 2 \leq 0,$$

and so their gradients are

$$\nabla g_1(x_1, x_2) = \begin{bmatrix} 2(x_1 - 1) \\ 2x_2 \end{bmatrix} \quad \nabla g_2(x_1, x_2) = \begin{bmatrix} -1 \\ 0 \end{bmatrix}.$$

The gradient of the objective function is

$$\nabla f(x_1, x_2) = \begin{bmatrix} 0 \\ 1 \end{bmatrix}.$$

From the first-order necessary conditions, the minimum should satisfy

$$\nabla f(\mathbf{x}^*) + \sum_{i=1}^m \mu_i \nabla g_i(\mathbf{x}^*) + \sum_{j=1}^{\ell} \lambda_j \nabla h_j(\mathbf{x}^*) = 0$$

For this particular example, this equation reduces to

$$\nabla f(\mathbf{x}^*) + \mu_1 \nabla g_1(\mathbf{x}^*) + \mu_2 \nabla g_2(\mathbf{x}^*) = 0$$

or

$$\begin{bmatrix} 0 \\ 1 \end{bmatrix} + \mu_1 \begin{bmatrix} 2(x_1^* - 1) \\ 2x_2^* \end{bmatrix} + \mu_2 \begin{bmatrix} -1 \\ 0 \end{bmatrix} = 0.$$

At $(x_1^*, x_2^*) = (2, 0)$, we have

$$\begin{bmatrix} 0 \\ 1 \end{bmatrix} + \mu_1 \begin{bmatrix} 2 \\ 0 \end{bmatrix} + \mu_2 \begin{bmatrix} -1 \\ 0 \end{bmatrix} = 0.$$

But there are no values of u_1 and u_2 for which the above equation is satisfied. So it is impossible to satisfy the KKT conditions at this point, even though it is optimal. How can this be? The answer is that the constraint qualifications are not satisfied at this point. The first constraint is not linear, so the linearity constraint qualification fails. The gradients of the two constraints are $(2, 0)$ and $(-1, 0)$, which are linearly dependent, so the linear independence constraint qualification also fails. Similarly, you can show that the Mangasarian-Fromovitz constraint qualification fails at this point. ■

This example highlights the importance of constraint qualifications. However, in traffic assignment, a majority of the formulations will have linear constraints, in which case the linear constraint qualification holds. In such cases, you do not have to worry further.

Note that convexity alone does not imply constraint qualification. In the above example both g_1 and g_2 are convex functions, so both the objective function and constraints are convex functions, so this is a convex optimization problem.

Example C.17. Solve the following optimization problem, where a , b , and c are positive real numbers.

$$\begin{aligned} \min \quad & x_1 + x_2 + x_3 \\ \text{s.t.} \quad & \frac{x_1^2}{a} + \frac{x_2^2}{b} + \frac{x_3^2}{c} = 1 \end{aligned}$$

Solution. We have $f(x_1, x_2, x_3) = x_1 + x_2 + x_3$ and a single equality constraint $h(x_1, x_2, x_3) = \frac{x_1^2}{a} + \frac{x_2^2}{b} + \frac{x_3^2}{c} - 1$. Calculating gradients, we have

$$\nabla f(x_1, x_2, x_3) = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \quad \nabla h(x_1, x_2, x_3) = \begin{bmatrix} \frac{2x_1}{a} \\ \frac{2x_2}{b} \\ \frac{2x_3}{c} \end{bmatrix},$$

and the condition

$$\nabla f(x_1^*, x_2^*, x_3^*) + \lambda \nabla h(x_1^*, x_2^*, x_3^*) = 0$$

becomes

$$\begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} + \lambda \begin{bmatrix} \frac{2x_1^*}{a} \\ \frac{2x_2^*}{b} \\ \frac{2x_3^*}{c} \end{bmatrix} = 0.$$

Solving, we get $x_1^* = -\frac{a}{2\lambda}$, $x_2^* = -\frac{b}{2\lambda}$, and $x_3^* = -\frac{c}{2\lambda}$.

We also have the condition $h(x_1^*, x_2^*, x_3^*) = 0$. For this problem, we compute as follows:

$$\begin{aligned} \nabla h(x_1^*, x_2^*, x_3^*) = 0 &\implies \frac{(x_1^*)^2}{a} + \frac{(x_2^*)^2}{b} + \frac{(x_3^*)^2}{c} = 1 \\ &\implies \frac{a}{4\lambda^2} + \frac{b}{4\lambda^2} + \frac{c}{4\lambda^2} = 1 \\ &\implies \lambda = \frac{\pm\sqrt{a+b+c}}{2} \end{aligned}$$

Therefore, we have two possible solutions for (x_1^*, x_2^*, x_3^*) :

$$\left(\frac{a}{\sqrt{a+b+c}}, \frac{b}{\sqrt{a+b+c}}, \frac{c}{\sqrt{a+b+c}} \right)$$

and

$$\left(-\frac{a}{\sqrt{a+b+c}}, -\frac{b}{\sqrt{a+b+c}}, -\frac{c}{\sqrt{a+b+c}} \right).$$

Checking both, the objective function is minimized at the second point, so this is optimal. ■

Example C.18. Write down the KKT conditions for the following optimization problem.

$$\begin{aligned} \min \quad & x_1^2 + x_2^2 \\ \text{s.t.} \quad & 11x_1 + 3x_2 \geq 21 \\ & 6x_1 + 20x_2 \geq 39 \\ & x_1 + x_2 \leq 9 \\ & x_1 \geq 0 \\ & x_2 \geq 0 \end{aligned}$$

Solution. Since all constraints are linear, the linear constraint qualification is satisfied and the KKT conditions are indeed necessary. We can rewrite the optimization problem in the form needed for the KKT conditions:

$$\begin{aligned} \min \quad & x_1^2 + x_2^2 \\ \text{s.t.} \quad & g_1(x_1, x_2) = -11x_1 - 3x_2 + 21 \leq 0 \\ & g_2(x_1, x_2) = -6x_1 - 20x_2 + 39 \leq 0 \\ & g_3(x_1, x_2) = x_1 + x_2 - 9 \leq 0 \\ & g_4(x_1, x_2) = -x_1 \leq 0 \\ & g_5(x_1, x_2) = -x_2 \leq 0. \end{aligned}$$

We calculate gradients as follows:

$$\begin{aligned} \nabla f(x_1, x_2) &= \begin{bmatrix} 2x_1 \\ 2x_2 \end{bmatrix} & \nabla g_1(x_1, x_2) &= \begin{bmatrix} -11 \\ -3 \end{bmatrix} & \nabla g_2(x_1, x_2) &= \begin{bmatrix} -6 \\ -20 \end{bmatrix} \\ \nabla g_3(x_1, x_2) &= \begin{bmatrix} 1 \\ 1 \end{bmatrix} & \nabla g_4(x_1, x_2) &= \begin{bmatrix} -1 \\ 0 \end{bmatrix} & \nabla g_5(x_1, x_2) &= \begin{bmatrix} 0 \\ -1 \end{bmatrix}. \end{aligned}$$

With these gradients, the first KKT condition (C.8) reduces to the two equations

$$\begin{aligned} 2x_1^* - 11\mu_1 - 6\mu_2 + \mu_3 - \mu_4 &= 0 \\ 2x_2^* - 3\mu_1 - 20\mu_2 + \mu_3 - \mu_5 &= 0. \end{aligned}$$

The second KKT condition (C.9) is ignored since there are no equality constraints. The third KKT condition (C.10) ensures feasibility:

$$\begin{aligned} g_1(x_1^*, x_2^*) \leq 0 &\implies -11x_1^* - 3x_2^* + 21 \leq 0 \\ g_2(x_1^*, x_2^*) \leq 0 &\implies -6x_1^* - 20x_2^* + 39 \leq 0 \\ g_3(x_1^*, x_2^*) \leq 0 &\implies x_1^* + x_2^* - 9 \leq 0 \\ g_4(x_1^*, x_2^*) \leq 0 &\implies -x_1^* \leq 0 \\ g_5(x_1^*, x_2^*) \leq 0 &\implies -x_2^* \leq 0. \end{aligned}$$

The fourth ensures complementarity (C.11), that the μ_i values must be zero unless the constraint is active:

$$\begin{aligned} \mu_1 g_1(x_1^*, x_2^*) = 0 &\implies \mu_1(-11x_1^* - 3x_2^* + 21) = 0 \\ \mu_2 g_2(x_1^*, x_2^*) = 0 &\implies \mu_2(-6x_1^* - 20x_2^* + 39) = 0 \\ \mu_3 g_3(x_1^*, x_2^*) = 0 &\implies \mu_3(x_1^* + x_2^* - 9) = 0 \\ \mu_4 g_4(x_1^*, x_2^*) = 0 &\implies \mu_4(-x_1^*) = 0 \\ \mu_5 g_5(x_1^*, x_2^*) = 0 &\implies \mu_5(-x_2^*) = 0 \end{aligned}$$

Finally, we require that the Lagrange multipliers all be non-negative:

$$\mu_1, \mu_2, \mu_3, \mu_4, \mu_5 \geq 0.$$

■ If desired, the optimum solution can be found by finding values of \mathbf{x} and $\boldsymbol{\mu}$ satisfying all these conditions. One way to do this is to start by identifying all 32 combinations of active/inactive constraints. For each, set $\mu_i = 0$ for the inactive constraints, and solve equations (C.8) and (C.11) for the remaining variables as a system of equations. If the solution satisfies the feasibility conditions (C.10) and has non-negative Lagrange multipliers, it satisfies all KKT conditions, and is a candidate solution. Among these, the solutions with the lowest objective function value are the optima.

We illustrate this procedure in the following example, which has fewer constraints:

Example C.19. *Solve the following optimization problem.*

$$\begin{aligned} \min \quad & x_1^2 + 2x_2^2 \\ \text{s.t.} \quad & x_1 + 2x_2 \leq 6 \\ & x_2 \geq x_1 + 2 \end{aligned}$$

Solution. Converting to the form required by the KKT conditions, we have $f(x_1, x_2) = x_1^2 + 2x_2^2$, and constraints $g_1(x_1, x_2) = x_1 + 2x_2 - 6$ and $g_2(x_1, x_2) = x_1 - x_2 + 2$, with gradients

$$\nabla f(x_1, x_2) = \begin{bmatrix} 2x_1 \\ 4x_2 \end{bmatrix} \quad \nabla g_1(x_1, x_2) = \begin{bmatrix} 1 \\ 2 \end{bmatrix} \quad \nabla g_2(x_1, x_2) = \begin{bmatrix} 1 \\ -1 \end{bmatrix}.$$

The KKT conditions for this problem are

$$\begin{aligned} 2x_1^* + u_1 + \mu_2 &= 0 \\ 4x_2^* + 2u_1 - \mu_2 &= 0 \end{aligned}$$

$$\begin{aligned} x_1^* + 2x_2^* - 6 &\leq 0 \\ x_1^* - x_2^* + 2 &\leq 0 \end{aligned}$$

$$\begin{aligned} \mu_1(x_1^* + 2x_2^* - 6) &= 0 \\ \mu_2(x_1^* - x_2^* + 2) &= 0 \end{aligned}$$

$$\mu_1, \mu_2 \geq 0.$$

The functions defining the constraints are linear, so the KKT conditions are indeed necessary. We now find candidate solutions by considering all combinations of active and inactive constraints.

Case I: Both constraints inactive:

In this case $\mu_1 = \mu_2 = 0$, so the equality conditions reduce to

$$\begin{aligned} 2x_1^* &= 0 \\ 4x_2^* &= 0 \end{aligned}$$

.

The solution is $(x_1^*, x_2^*) = (0, 0)$. However $(0, 0)$ violates the constraint $g_2(x_1, x_2) = x_1^* - x_2^* + 2 \leq 0$. So, there is no optimum solution where both constraints are inactive.

Case II: Only the second constraint is active:

In this case $\mu_1 = 0$. The equality conditions reduce to

$$\begin{aligned} 2x_1^* + \mu_2 &= 0 \\ 4x_2^* - \mu_2 &= 0 \\ \mu_2(x_1^* - x_2^* + 2) &= 0. \end{aligned}$$

The first two equations give $x_1^* = -\mu_2/2$ and $x_2^* = \mu_2/4$. Because the second constraint is active, we know $x_1^* - x_2^* + 2 = 0$. Therefore, the third equation will be satisfied automatically. Substituting $x_1^* = -\mu_2/2$ and $x_2^* = \mu_2/4$ into the active constraint, we have $\mu_2 = 8/3$, and therefore $x_1^* = 4/3$ and $x_2^* = 2/3$.

This solution satisfies both constraints, and the Lagrange multipliers are non-negative, so it is a candidate for optimality.

Case III: Only the first constraint is active:

In this case $\mu_2 = 0$, and the equality conditions are:

$$\begin{aligned} 2x_1^* + \mu_1 &= 0 \\ 4x_2^* + 2\mu_1 &= 0 \\ \mu_1(x_1^* + 2x_2^* - 6) &= 0 \end{aligned}$$

Proceeding in the same way, the first two equations require $x_1^* = -\mu_1/2$ and $x_2^* = -\mu_1/2$. Likewise, because we assume the first constraint is active, we can replace the third equation with $x_1^* + 2x_2^* = 6$. But with the values of x_1^* and x_2^* for this case, this simplifies to $-\mu_1 = 6$. This violates the requirement $\mu_1 \geq 0$, and therefore there cannot be an optimal solution corresponding to this case.

Case IV: Both constraints are active:

In this case, we know both constraints are satisfied with equality:

$$\begin{aligned} x_1^* + 2x_2^* &= 6 \\ x_1^* - x_2^* &= -2 \end{aligned}$$

The only solution to these equations is $x_1^* = 2/3$ and $x_2^* = 8/3$. It remains to see whether there are Lagrange multipliers satisfying the rest of the KKT conditions. Substituting into the first two conditions, we have

$$\begin{aligned} \mu_1 + \mu_2 &= -2x_1^* = -\frac{4}{3} \\ 2\mu_1 - \mu_2 &= -4x_2^* = -\frac{32}{3}. \end{aligned}$$

Solving this system, we find $\mu_1 < 0$, violating the non-negativity condition, and establishing that the solution where both constraints are active cannot be optimal.

We have exhausted all combinations of active and inactive constraints, and are left with a single candidate solution (from Case II). Therefore, this one is the optimum, and the solution to the problem is $x_1^* = -\frac{4}{3}$ and $x_2^* = \frac{2}{3}$. If there were multiple candidate solutions, we would evaluate the objective function at each to see which one(s) are minima. ■

C.4.3 Algorithms for linear constraints

The remainder of this section will describe four algorithms for solving nonlinear optimization problems with linear constraints. The first two algorithms are the method of successive averages (MSA) and Frank-Wolfe (FW) algorithm. Both of these are called convex combination methods. The third and fourth methods, gradient projection (GP) and manifold suboptimization (MS), modify the direction of steepest descent to ensure feasibility. The algorithms are presented in increasing order of complexity and efficiency. All of these algorithms will converge to a local optimum. If the objective is convex, this local optimum will also be globally optimal.

We focus here on nonlinear optimization problems with linear constraints, as the traffic assignment problem have this structure. These four algorithms have all been applied to traffic assignment as well. In Chapter 6, these algorithms are presented specifically in this context. Here we treat them more generally, as they can be used for other nonlinear optimization problems as well.

There are many ways to modify these algorithms, and the best implementation often depends on the specific problem structure and the computing environment being used. For example, all four methods have the common structure of “find an improving direction” and “choose a step size in that direction.” This section mainly emphasizes the former, since the algorithms differ more in this regard. However, there are alternative choices for the latter as well. Our presentations of FW, PG, GP, and MS in this subsection choose the step size which minimizes the objective function along the chosen direction. This indeed results in the greatest improvement possible at each iteration. However, this can be computationally demanding, and it might be more efficient to take a faster but less exact step. For instance, the algorithms in Chapter 6 often use a single iteration of Newton’s method in place of finding an exactly-optimal step size, since this tradeoff is usually worthwhile in traffic assignment. If you are comparing the presentations of the algorithms in that chapter which are specialized to traffic assignment to the more general presentations here, such differences should be noted, but they are matters of implementation and not essential algorithmic distinctions.

The standard form of the nonlinear optimization problem with linear constraints is as follows. The decision variables are $\mathbf{x} = [x_1 x_2 \dots x_n]$. The objective function f is a convex function of \mathbf{x} . The feasible region is given as $X = \{\mathbf{x} \in \mathbb{R}^n : \mathbf{A}\mathbf{x} - \mathbf{b} \leq \mathbf{0} \forall i \in \{1, 2, \dots, m\}; \mathbf{C}\mathbf{x} - \mathbf{d} = \mathbf{0} \forall i \in \{1, 2, \dots, \ell\}\}$. With linear constraints such as these, the feasible region is convex and closed. We further assume that it is bounded, as is the case in traffic assignment and many other applications.

Therefore, this is the form of the optimization problems we will consider in this section:

$$\begin{aligned} \min \quad & f(\mathbf{x}) \\ \text{s.t.} \quad & \mathbf{Ax} - \mathbf{b} \leq \mathbf{0} \quad \forall i \in \{1, 2, \dots, m\} \\ & \mathbf{Cx} - \mathbf{d} = \mathbf{0} \quad \forall i \in \{1, 2, \dots, \ell\} \end{aligned}$$

C.4.4 Convex combination methods

All convex combinations methods can be described by the following framework (see Algorithm 1). They have three main steps: (i) determining a feasible descent direction, (ii) determining the step size, and (iii) updating the solution. The convergence criterion for feasible direction methods can be chosen in a similar manner as for unconstrained optimization algorithms.

Initialize:

$k \leftarrow 1$: iteration counter

\mathbf{x}_1 : initial value

ϵ : tolerance

while *ConvergenceCriterion* $> \epsilon$ **do**

Determine feasible descent direction: \mathbf{p}^k

Determine step size: $\alpha_k \geq 0$ such that $f(\mathbf{x}_k + \alpha_k \mathbf{p}^k) < f(\mathbf{x}_k)$
 and $\mathbf{x}_k + \alpha_k \mathbf{p}^k \in X$

Update: $\mathbf{x}_{k+1} \leftarrow \mathbf{x}_k + \alpha_k \mathbf{p}^k$

$k \leftarrow k + 1$

end

return \mathbf{x}_k

Algorithm 1: Feasible directions method

At iteration k , when the current solution \mathbf{x}_k , the feasible descent direction \mathbf{p}^k is obtained by taking $\mathbf{p}^k = \mathbf{y}_k - \mathbf{x}_k$, where \mathbf{y}_k is a feasible solution obtained by solving an approximation of the original nonlinear problem. Both MSA and FW obtain the feasible direction by replacing the objective function with its first-order (linear) approximation at \mathbf{x}_k . Because the constraints are linear, the resulting first-order approximation will be the linear program shown below, which can be solved using the simplex algorithm. (In some applications, the first-order approximation can be solved more efficiently by exploiting a particular structure of the problem. For example, in traffic assignment, one can show that an all-or-nothing assignment to shortest paths is optimal to this subproblem, which can be found using one of the algorithms in Section 2.4.

$$\begin{aligned} \min \quad & \nabla f(\mathbf{x}_k)^T \mathbf{y} \\ \text{s.t.} \quad & \mathbf{Ax} - \mathbf{b} \leq \mathbf{0} \quad \forall i \in \{1, 2, \dots, m\} \\ & \mathbf{Cx} - \mathbf{d} = \mathbf{0} \quad \forall i \in \{1, 2, \dots, \ell\} \end{aligned}$$

The difference between MSA and FW is how the step size is chosen. In MSA, at each iteration k , the step size is commonly chosen to be $\alpha_k = 1/(k+1)$, although any sequence of step sizes satisfying $\sum \alpha_k = \infty$ and $\sum \alpha_k^2 < \infty$ will

work. In FW, the step size α_k is obtained by solving the following optimization problem in λ . This optimization problem can be solved using any of the line search methods from Sections 3.3.2 or C.1.

$$\begin{aligned} \min_{\alpha} \quad & f(\mathbf{x}_k + \alpha \mathbf{p}_k) \\ \text{s.t.} \quad & 0 \leq \alpha \leq 1 \end{aligned}$$

FW picks the step size in a more intelligent manner than MSA and typically converges faster. For this reason, MSA is rarely used in solving nonlinear optimization with linear constraints. However, in certain complex traffic assignment problems, evaluating the objective function f can be computationally expensive. In such cases, a line search method in FW can require large amounts of time, and taking faster MSA steps outweighs the greater precision that each FW step would have.

Example C.20. Determine the optimal solution of the following optimization problem using MSA and FW.

$$\begin{aligned} \min \quad & 2(x_1 - 1)^2 + (x_2 - 2)^2 \\ \text{s.t.} \quad & x_1 + 4x_2 - 2 \leq 0 \\ & -x_1 + x_2 \leq 0 \\ & x_1, x_2 \geq 0 \end{aligned}$$

Solution. The KKT conditions for this problem can be solved to yield the optimal solution $x_1^* = 0.7878$, $x_2^* = 0.3030$. The MSA and FW algorithms will converge to this solution over successive iterations; the advantage of these methods is that solving the KKT conditions for a large problem can be very difficult, whereas MSA and FW scale better with problem size. For this objective function, the gradient at any point (x_1, x_2) is

$$\nabla f(x_1, x_2) = \begin{bmatrix} 4(x_1 - 1) \\ 2(x_2 - 2) \end{bmatrix}.$$

MSA: Assume the initial solution is $x_1^1 = 0$, $x_2^1 = 0$. The search direction is obtained by solving the following linear program.

$$\begin{aligned} \min \quad & -4y_1 - 4y_2 \\ \text{s.t.} \quad & y_1 + 4y_2 - 2 \leq 0 \\ & -y_1 + y_2 \leq 0 \\ & y_1, y_2 \geq 0 \end{aligned}$$

The optimal solution to this problem is $y_1^1 = 2$, $y_2^1 = 0$. Therefore, the search direction is $\mathbf{p}^1 = \mathbf{y}^1 - \mathbf{x}^1 = \begin{bmatrix} 2 & 0 \end{bmatrix}$. Therefore

$$\begin{aligned} x_1^2 &= 0 + \frac{1}{2}(2) = 1 \\ x_2^2 &= 0 + \frac{1}{2}(0) = 0 \end{aligned}$$

Table C.5: Convergence of the method of successive averages.

k	x_1	x_2	Error
1	0	0	0.7124
2	1	0	0.1368
3	0.8	0.1333	0.0289
4	0.7	0.2	0.01831
5	0.96	0.16	0.0501
6	0.866	0.2	0.0168
7	0.800	0.2285	0.005
20	0.780	0.28	0.0005
53	0.7849	0.2943	8.33793×10^{-5}

At the start of the second iteration we have $x_1^2 = 1, x_2^2 = 0$. Therefore, the search direction is obtained by solving the following linear program.

$$\begin{aligned} \min \quad & -4y_2 \\ \text{s.t.} \quad & y_1 + 4y_2 - 2 \leq 0 \\ & -y_1 + y_2 \leq 0 \\ & y_1, y_2 \geq 0 \end{aligned}$$

The optimal solution to this linear program is $y_1^2 = 0.4, y_2^2 = 0.4$. In iteration 2, the step size is $\alpha_2 = \frac{1}{3}$. Therefore,

$$\begin{aligned} x_1^3 &= 1 + \frac{1}{3}(0.4 - 1) = 0.8 \\ x_2^3 &= 0 + \frac{1}{3}(0.4 - 0) = 0.1333 \end{aligned}$$

Table C.5 shows the progress of MSA over further iterations. The error in this table is calculated as the sum of square deviation from the optimal solution which we know to be $x_1^* = 0.7878, x_2^* = 0.3030$.

FW: Again assume the initial solution to $x_1^1 = 0, x_2^1 = 0$. The search direction is obtained by solving the same linear program as in MSA.

$$\begin{aligned} \min \quad & -4y_1 - 4y_2 \\ \text{s.t.} \quad & y_1 + 4y_2 - 2 \leq 0 \\ & -y_1 + y_2 \leq 0 \\ & y_1, y_2 \geq 0 \end{aligned}$$

The optimal solution to the LP is again $y_1^1 = 2, y_2^1 = 0$, so the direction is the same: $\mathbf{p}^1 = [2 \ 0]$. In FW, the step size is obtained by solving the following minimization problem:

$$\begin{aligned} \min_{\alpha_1} \quad & 2(0 + \alpha_1(2 - 0) - 1)^2 + (0 + \alpha_1(0 - 0) - 2)^2 \\ \text{s.t.} \quad & 0 \leq \alpha \leq 1 \end{aligned}$$

Table C.6: Convergence of the Frank-Wolfe algorithm.

k	x_1	x_2	Error
1	0	0	0.7124
2	1	0	0.1368
3	0.4545	0.3636	0.1147
4	0.7979	0.2828	0.0005
20	0.7919	0.2947	8.6×10^{-5}

The optimal solution is $\alpha_1 = 0.5$, and the solution is updated as

$$\begin{aligned}x_1^2 &= 1 + 0.5 \times (2 - 0) = 1 \\x_2^2 &= 0 + 0.5 \times (0 - 0) = 0\end{aligned}$$

To this point, FW is proceeding in the same way as MSA.

At the start of the second iteration we have $x_1^2 = 2, x_2^2 = 0$. Therefore, the search direction is obtained by solving the following linear program.

$$\begin{aligned}\min \quad & -4y_2 \\ \text{s.t.} \quad & y_1 + 4y_2 - 2 \leq 0 \\ & -y_1 + y_2 \leq 0 \\ & y_1, y_2 \geq 0\end{aligned}$$

The optimal solution to the linear program is $y_1^2 = 0.4, y_2^2 = 0.4$, as before. However, at this point FW chooses the step size differently than MSA. Solving the optimization problem

$$\begin{aligned}\min_{\alpha_2} \quad & 2(1 + \alpha_2(0.4 - 1) - 1)^2 + (0 + \alpha_2(0.4 - 0) - 2)^2 \\ \text{s.t.} \quad & 0 \leq \alpha_2 \leq 1\end{aligned}$$

gives the step size $\alpha_2 = \lambda = 0.9091$. Therefore, we obtain the new solution

$$\begin{aligned}x_1^3 &= 1 + 0.9091 \times (0.4 - 1) = 0.4545 \\ x_2^3 &= 0 + 0.9091 \times (0.4 - 0) = 0.3634.\end{aligned}$$

Table C.6 shows the convergence of FW over successive iterations. Comparing with Table C.5, we see that FW converges in fewer iterations than MSA. FW achieves an error of 0.0005 in 4 iterations; MSA required 20 iterations to achieve this level of precision. The tradeoff is that FW required more effort at each iteration to determine the step size α . In problems such as this, the tradeoff is clearly in favor of FW. ■

C.4.5 Gradient projection

The gradient projection method operates by taking a step in the direction of steepest descent (ignoring feasibility), and *then* jumping to the nearest feasible

point if the new solution is infeasible. Recall that $\text{proj}_X(\mathbf{x})$ means “the point in X closest to \mathbf{x} .” (If $\mathbf{x} \in X$ already, then $\text{proj}_X(\mathbf{x}) = \mathbf{x}$.) If the set X is convex, then the projection operation is uniquely defined, and a continuous function of \mathbf{x} . In general, projection cannot be easily computed. When the constraints are linear (as we assume in this section), it is considerably easier. For some problems, it can be exceptionally easy. For instance, in traffic assignment, we can reformulate all constraints to be simple non-negativity constraints of the form $h^\pi \geq 0$, and “projection” simply means “any negative path flow should be set to zero.”

The steps of gradient projection are shown in Algorithm 2.

Initialize:

$k \leftarrow 1$: iteration counter

\mathbf{x}_1 : initial value

ϵ : tolerance

while *ConvergenceCriterion* $> \epsilon$ **do**

$\mathbf{d}_k \leftarrow -\nabla f(\mathbf{x}_k)$
$\alpha_k \leftarrow \arg \min_{\alpha} \{f(\text{proj}_X(\mathbf{x}_k + \alpha \mathbf{d}_k)), 0 \leq \alpha \leq \bar{\alpha}\}$
$\mathbf{x}_{k+1} \leftarrow \mathbf{x}_k + \alpha_k \mathbf{d}$
$k \leftarrow k + 1$

end

return \mathbf{x}_k

Algorithm 2: Gradient projection method

Example C.21. *Re-solve the previous optimization problem using the gradient projection method, with $\bar{\alpha} = 1$.*

$$\begin{aligned} \min \quad & 2(x_1 - 1)^2 + (x_2 - 2)^2 \\ \text{s.t.} \quad & x_1 + 4x_2 - 2 \leq 0 \\ & -x_1 + x_2 \leq 0 \\ & x_1, x_2 \geq 0 \end{aligned}$$

Solution. We start from the same initial solution $x_1^1 = 0, x_2^1 = 0$. Here the search direction is $\mathbf{d}_1 = -\nabla f(\mathbf{x}_1) = [4 \ 4]$. To find the step size, we solve the optimization problem

$$\begin{aligned} \min_{\alpha} \quad & f(\text{proj}_X(\mathbf{x}_1 + \alpha \mathbf{d}_1)) \\ \text{s.t.} \quad & 0 \leq \alpha \leq 1 \end{aligned}$$

The projection operation can be more easily seen on a plot of the feasible region, as in Figure C.8. For a given value of α , $\mathbf{x}_1 + \alpha \mathbf{d}_1$ is the point $(4\alpha, 4\alpha)$. If $\alpha \leq 1/10$, this coincides with the line $x_1 = x_2$, which is part of the feasible region. As a result, projection does not change the point and $\text{proj}_X(4\alpha, 4\alpha) = (4\alpha, 4\alpha)$. Once $\alpha > 1/10$, the point $(4\alpha, 4\alpha)$ violates the constraint $x_1 + 4x_2 \leq 2$, and is infeasible. For these points, we have to project

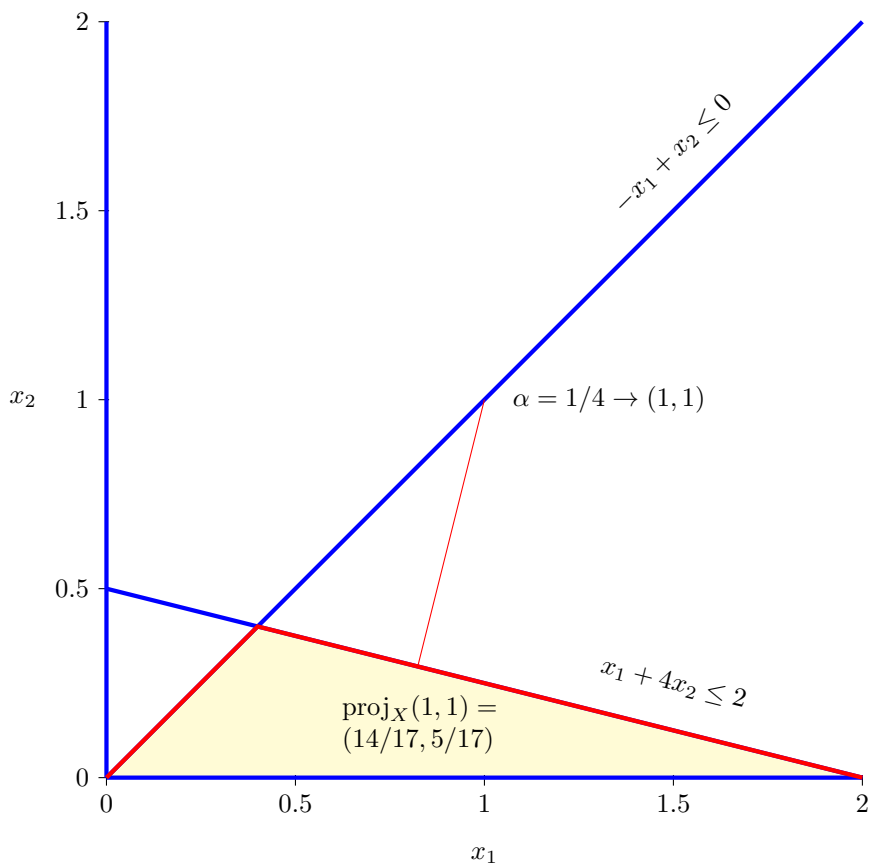


Figure C.8: Feasible region and projection.

them onto the nearest feasible point. As an example, if $\alpha = 1/4$, the point $(4\alpha, 4\alpha) = (1, 1)$ is infeasible. The closest feasible point is $(14/17, 5/17)$; this can be seen geometrically in Figure C.8 by drawing a line through $(1, 1)$ perpendicular to the line corresponding to that constraint. The expression mapping a value of α in this range to the closest feasible point can be found algebraically, and the formula is shown below. If $\alpha > \frac{2}{3}$, the point on the line $x_1 + 4x_2$ lies below the x_2 -axis, which is also infeasible. For α values this large, the closest feasible point is simply $(2, 0)$. Therefore we have

$$\text{proj}_X(\mathbf{x}_1 + \alpha \mathbf{d}_1) = \begin{cases} (4\alpha, 4\alpha) & \alpha \in [0, 1/10] \\ \frac{1}{17}(48\alpha + 2, -12\alpha + 8) & \alpha \in [1/10, 2/3] \\ (2, 0) & \alpha \geq 2/3 \end{cases} .$$

You can check that this formula is continuous.

■

Evaluating the objective function in terms of α , we thus have

$$f(\text{proj}_X(\mathbf{x}_1 + \alpha \mathbf{d}_1)) = \begin{cases} 2(4\alpha - 1)^2 + (4\alpha - 2)^2 & \alpha \in [0, 1/10] \\ 2\left(\frac{1}{17}(48\alpha + 2) - 1\right)^2 + \left(\frac{1}{17}(-12\alpha + 8) - 2\right)^2 & \alpha \in [1/10, 2/3] \\ 6 & \alpha \geq 2/3 \end{cases} .$$

Again, you can check that this function is continuous and convex. Using a line search method, the minimum is found for $\alpha = 0.236$, which corresponds to the point $(0.7878, 0.3030)$. As shown above, this is actually the optimal solution, found in a single iteration! Typically, GP only converges to the optimal solution in the limit, but as this example shows it can be extremely efficient in terms of iterations. The drawback is that each iteration requires a significant amount of work. But in problems where the projection can be calculated very efficiently (as in traffic assignment), GP can be an excellent algorithmic choice.

C.4.6 Manifold suboptimization

Manifold suboptimization works by modifying the direction of steepest descent (related to the gradient of the objective) to ensure that it remains within the feasible set. This method also involves projection, but unlike gradient projection, the projection is implicit through matrix multiplications which ensure that whichever constraints are exactly satisfied at the current solution remain satisfied. There are several names that are used to describe this method, which unfortunately can cause confusion. Some authors refer to this method as the *projected gradient* method (distinguishing it from gradient projection, because the feasibility is used to identify the improving direction, rather than moving in a direction and then checking feasibility), and others even call it *gradient projection* (for historical reasons). We believe that calling it *manifold suboptimization* avoids this confusion, and by doing so follow the convention of Bertsekas (2016).

Intuitively, the idea is to distinguish between “active” constraints (where the left-hand side is exactly equal to the right-hand side) and “inactive” ones (where the left-hand side is strictly less than the right-hand side). The gradient is modified so that any active constraints continue to hold with equality; inactive constraints do not impose any such restriction, since their left-hand sides can vary some without violating feasibility. Equality ($=$) constraints are always active. Inequality (\leq) constraints may or may not be active. As the algorithm proceeds, which constraints are active and which are inactive may vary, and some care must be taken to manage this set properly and to ensure feasibility of the step size (the left-hand side of an inactive constraint can vary some, but if it changes too much it may become violated). In this subsection, we present the steps of the algorithm. Readers wishing a derivation of these steps, or a proof that these steps indeed maintain feasibility and lead to an optimal solution, are referred to the books by Bazarra et al. (2006) and Bertsekas (2016).

We again consider the optimization problem

$$\begin{array}{ll} \min & f(\mathbf{x}) \\ \text{s.t.} & \mathbf{Ax} \leq \mathbf{b} \quad \forall i = 1, 2, \dots, m \\ & \mathbf{Cx} = \mathbf{d} \quad \forall i = 1, 2, \dots, l \end{array}$$

and write the feasible region X as

$$X = \{x : \mathbf{Ax} \leq \mathbf{b}, \mathbf{Cx} = \mathbf{d}\}.$$

For any feasible solution $\hat{\mathbf{x}}$, let \mathbf{A}_a be the matrix containing the coefficients of the set of active inequality constraints, that is,

$$\mathbf{A}_a \hat{\mathbf{x}} = \mathbf{b}_a.$$

Define \mathbf{E} as the matrix containing the coefficients of *all* active constraints:

$$\mathbf{E} = \begin{bmatrix} \mathbf{A}_a \\ \mathbf{C} \end{bmatrix}.$$

It may be that the rows of the matrix \mathbf{E} are not linearly independent; this indicates that one or more of the active constraints is redundant (they are implied by some of the others). These redundant constraints may be removed.

Algorithm 3 provides the steps for the manifold suboptimization method. Note that the convergence criterion is different from the MSA and FW methods. In this algorithm, $\bar{\alpha}$ is a parameter giving a “default” maximum step size (which

may be further restricted to ensure feasibility).

```

Initialize:
 $k \leftarrow 1$ : iteration counter
 $\mathbf{x}_1$ : initial value
 $OptimalityFlag \leftarrow FALSE$ 
while  $OptimalityFlag = FALSE$  do
   $\mathbf{P} \leftarrow \mathbf{I} - \mathbf{E}^T(\mathbf{E}\mathbf{E}^T)^{-1}\mathbf{E}$ 
   $\mathbf{d}_k \leftarrow -\mathbf{P}\nabla f(\mathbf{x}_k)$ 
  if  $\mathbf{d}_k \neq 0$  then
     $\beta \leftarrow \max\{\bar{\alpha}, \mathbf{x}_k + \alpha\mathbf{d}_k \in X\}$ 
     $\alpha_k \leftarrow \arg \min_{\alpha}\{f(\mathbf{x}_k + \alpha\mathbf{d}_k), 0 \leq \alpha \leq \beta\}$ 
     $\mathbf{x}_{k+1} \leftarrow \mathbf{x}_k + \alpha_k\mathbf{d}_k$ 
     $k \leftarrow k + 1$ 
  end
  if  $\mathbf{d} = 0$  then
     $\mathbf{Q} = -(\mathbf{E}\mathbf{E}^T)^{-1}\mathbf{E}\nabla f(\mathbf{x}_k)$ 
    if  $q_j \geq 0 \forall j$  corresponding to  $\mathbf{A}_a$  then
       $OptimalityFlag \leftarrow TRUE$ 
    end
    else
      Delete row from  $\mathbf{E}$  corresponding to most negative  $q_j$ 
    end
  end
end
return  $\mathbf{x}_k$ 

```

Algorithm 3: Manifold suboptimization

Example C.22. Re-solve the previous optimization problem using the manifold suboptimization method, using $\bar{\alpha} = 1$.

$$\begin{aligned}
 \min \quad & 2(x_1 - 1)^2 + (x_2 - 2)^2 \\
 \text{s.t.} \quad & x_1 + 4x_2 - 2 \leq 0 \\
 & -x_1 + x_2 \leq 0 \\
 & x_1, x_2 \geq 0
 \end{aligned}$$

Solution. We start from the same initial solution $x_1^1 = 0, x_2^1 = 0$. At this point, the active constraints are $-x_1 + x_2 \leq 0$, $x_1 \geq 0$, and $x_2 \geq 0$. We only use $-x_1 + x_2 \leq 0$ and $x_2 \geq 0$ as they together imply $x_1 \geq 0$. (Mathematically, the three vectors corresponding to the coefficients in these constraints are linearly dependent: $[-1 \ 1]$, $[0 \ 1]$, and $[1 \ 0]$.)

$$\mathbf{E} = \begin{bmatrix} -1 & 1 \\ 0 & -1 \end{bmatrix}$$

$$\mathbf{P} = \mathbf{I} - \mathbf{E}^T(\mathbf{E}\mathbf{E}^T)^{-1}\mathbf{E} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

$$\mathbf{d}_1 = -\mathbf{P}\nabla f(\mathbf{x}_1) = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Since $\mathbf{d} = 0$, we calculate \mathbf{Q} :

$$\mathbf{Q} = -(\mathbf{E}\mathbf{E}^T)^{-1}\mathbf{E}\nabla f(\mathbf{x}_1) = \begin{bmatrix} -4 \\ -8 \end{bmatrix}$$

Since the second row is the most negative, we delete the second row of the matrix \mathbf{E} .

$$\mathbf{E} = \begin{bmatrix} -1 & 1 \end{bmatrix}$$

$$\mathbf{P} = \mathbf{I} - \mathbf{E}^T(\mathbf{E}\mathbf{E}^T)^{-1}\mathbf{E} = \begin{bmatrix} 0.5 & 0.5 \\ 0.5 & 0.5 \end{bmatrix}$$

$$\mathbf{d}_1 = -\mathbf{P}\nabla f(\mathbf{x}_1) = \begin{bmatrix} 4 \\ 4 \end{bmatrix}$$

The new solution can be obtained as:

$$\begin{aligned} x_1^2 &= 0 + 4\alpha \\ x_2^2 &= 0 + 4\alpha \end{aligned}$$

We need to find the step size α which minimizes $2(0+4\alpha-1)^2 + (0+4\alpha-2)^2$ while retaining feasibility. Note that

$$x_1 + 4x_2 - 2 \leq 0 \implies \alpha \leq 0.1,$$

which is less than $\bar{\alpha} = 1$, so $\beta = 0.1$. Therefore, we obtain the step size by solving the optimization problem

$$\begin{aligned} \min_{\alpha} \quad & 2(0+4\alpha-1)^2 + (0+4\alpha-2)^2 \\ \text{s.t.} \quad & 0 \leq \alpha \leq 0.1 \end{aligned}$$

whose solution is $\alpha = 0.1$. Therefore, the new solution to the original problem is

$$\begin{aligned} x_1^2 &= 0 + 4\alpha = 0.4 \\ x_2^2 &= 0 + 4\alpha = 0.4 \end{aligned}$$

If we do one more iteration of MS, we get $x_1^3 = 0.7878, x_2^3 = 0.3030$, which is optimal. MS converges to the optimal solution much faster than MSA or FW (and about as fast as GP), because its steps are closer to the steepest descent direction. Section 6.3 has more discussion of these reasons in the specific context of traffic assignment. ■

C.5 Integer Programming

All of the optimization formulations presented above assume that the decision variables are continuous in nature. In some cases, this is not a tenable assumption. For example, when determining the location to establish a warehouse, a fractional solution splitting the warehouse location between two nodes is not meaningful. Integer programs are a special class of optimization problems, where some or all decision variables must take integer values. Integer programs have wide applications:

Vehicle routing : Determine routes for a specified number of trucks, starting from a depot location to service a given number of customers with demands, subject to truck capacity constraints at minimal cost.

Facility location : Determine where to locate facilities in a network, satisfying customer demand, with an objective such as minimizing transportation costs, distances, or maximum distance to facilities.

Transit network design : Determine the optimal routes and schedules of a bus fleet to satisfy customer demands at minimal costs.

As always, there is no such thing as a free lunch; for reasons discussed below, integer problems are significantly harder to solve, and often it is impractical to find a provably optimal solution.

There are several types of integer programs. An integer linear program (ILP) is a linear program where *all* the variables are restricted to be integers. A mixed integer linear program (MILP) is a linear program where only some of the decision variables are assumed to be integer. A MILP takes the form

$$\begin{array}{ll} \min_{\mathbf{x}, \mathbf{y}} & \mathbf{c} \cdot \mathbf{x} + \mathbf{d} \cdot \mathbf{y} \\ \text{s.t.} & \mathbf{A}\mathbf{x} + \mathbf{E}\mathbf{y} \leq \mathbf{b} \\ & \mathbf{x} \geq 0, \mathbf{y} \geq 0 \\ & \mathbf{y} \in \mathbb{Z}^p \end{array}$$

where $\mathbf{x}, \mathbf{c} \in \mathbb{R}^n$, $\mathbf{y}, \mathbf{d} \in \mathbb{R}^p$, $\mathbf{A} \in \mathbb{R}^{m \times n}$, $\mathbf{E} \in \mathbb{R}^{m \times p}$, and $\mathbf{b} \in \mathbb{R}^m$ with m , n , and p respectively denoting the number of constraints, continuous decision variables, and integer decision variables. In an ILP, $m = 0$ and the terms involving \mathbf{x} in this formulation disappear. A special case of an ILP is the binary integer linear program (BILP) where the decision variables are restricted to be either 0 or 1, i.e. $\mathbf{y} \in \{0, 1\}^p$.

Integer programs are tougher to solve than linear programs as optimal solutions need not occur at the corner points of a convex polyhedron. Indeed, the feasible region itself is not a convex set; $y_1 = 1$ and $y_2 = 2$ may be feasible solutions, but their average $\frac{1}{2}y_1 + \frac{1}{2}y_2 = 1.5$ is not. Solving the corresponding linear program and then rounding the solution may not produce a good solution either. (One notable exception is discussed in Section C.5.2.)

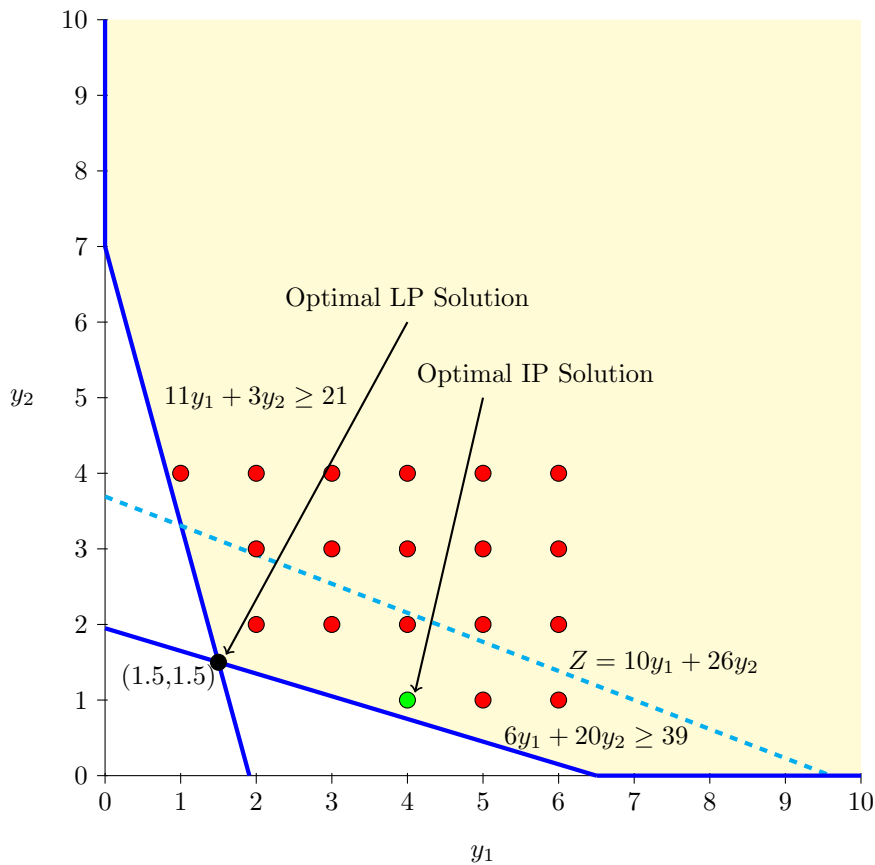


Figure C.9: Difference between LP and IP solution.

Consider the following ILP with two decision variables y_1 and y_2 :

$$\begin{aligned} \min_{y_1, y_2} \quad & 10y_1 + 26y_2 \\ \text{s.t.} \quad & 11y_1 + 3y_2 \geq 21 \\ & 6y_1 + 20y_2 \geq 39 \\ & y_1, y_2 \geq 0 \\ & y_1, y_2 \in \mathbb{Z} \end{aligned}$$

Without the integer restrictions on the variables, the optimal solution of the linear program would be $y_1 = 1.5$, $y_2 = 1.5$, with an optimal objective function value of 54 (see figure C.9). When the LP solutions are rounded up ($y_1 = 2, y_2 = 2$) or rounded down ($y_1 = 1, y_2 = 1$) the resulting points are sub-optimal or infeasible for the integer program. The optimal solution for the integer program is in fact $y_1 = 4, y_2 = 1$, with an optimal objective function value of 66.

Still, while ignoring the integrality constraints and solving the resulting linear program may not lead to an optimal solution, this *linear program relaxation* is still very useful, because it provides a lower bound on the optimal value of the objective function. In the above example, the optimal solution without integrality constraints had an objective function value of 54. Restricting the feasible set more by adding the integer constraints cannot improve this. Therefore, even if we don't know the optimal integer solution of the optimization problem, we know its objective function value cannot be lower than 54.

Furthermore, any feasible integer solution provides an upper bound on the optimal value for the objective. For instance, the rounded solution $y_1 = y_2 = 2$ is feasible, and has an objective function value of 72. Even if we don't know whether this solution is optimal or not, this solution tells us that the optimal objective function value can't be any higher than 72. We therefore have a corresponding pair of upper and lower bounds. Solving the LP relaxation, and calculating the objective function at the feasible solution (2, 2) tell us that the optimal objective function value (often denoted Z^*) must lie between 54 and 72.

A common framework for solving integer programs is to work to bring these bounds closer together. Upper bounds can be tightened by finding better feasible solutions (with lower objective function values that are closer to the optimal solution). Lower bounds can be tightened by solving "partial relaxations" where only some of the integrality constraints are enforced; these will have higher objective function values than a full relaxation, which are also closer to the optimal solution (but from the other side). Ultimately, we determine a decreasing sequence of upper bounds $Z_{UB}^1 \geq Z_{UB}^2 \geq \dots \geq Z^*$ and an increasing sequence of lower bounds $Z_{LB}^1 \leq Z_{LB}^2 \leq \dots \leq Z^*$. When the difference between the bounds is small enough (say, within ϵ), we know that the feasible solution corresponding to the best-known upper bound is within ϵ of being optimal to the original integer program.

The success of such a framework depends on the strength of the bounds. The tighter the upper and lower bounds are, the greater the guarantee we can provide for the solution we find. Effectively solving these problems often requires intelligently exploiting the problem structure, and expert domain knowledge, in order to provide good feasible solutions (for upper bounds), and good partial relaxations (for lower bounds). The branch and bound algorithm (described in Section C.5.1 is one framework for doing this. Before presenting this method, we will provide another concrete example of an integer program for facility location, and how it relates to the selection of good bounds.

Consider the case of locating a certain number of facilities in a region (say, warehouses, or fire stations). Let I denote the set of locations that demand what the facility provides, and J the set of potential locations where facilities can be built. Let c_{ij} denote the cost of meeting the demand at location $i \in I$ from facility $j \in J$. Let f_j represent the fixed cost of locating a facility at site $j \in J$. There are two sets of decision variables corresponding to location and assignment decisions. The location-related decision variable y_j takes the value 1 if a facility is located at site j and 0 otherwise. The decision variable x_{ij}

denotes the fraction of demand for center i met by facility j . The x_{ij} variables can be continuous (its demand can be provided by multiple facilities), whereas the y_j values must be integer (a facility cannot be “half-built”). The optimization model for the UFLP is shown below:

O-UFLP :

$$\min_{\mathbf{x}, \mathbf{y}} \sum_{i \in I} \sum_{j \in J} c_{ij} x_{ij} + \sum_{j \in J} f_j y_j \quad (\text{C.18})$$

$$\text{s.t.} \quad \sum_{j \in J} x_{ij} = 1 \quad \forall i \in I \quad (\text{C.19})$$

$$\sum_{i \in I} x_{ij} \leq |I| y_j \quad \forall j \in J \quad (\text{C.20})$$

$$y_j \in \{0, 1\} \quad \forall j \in J \quad (\text{C.21})$$

$$x_{ij} \geq 0 \quad \forall i \in I, \forall j \in J \quad (\text{C.22})$$

The objective of the UFLP is to locate facilities and assign demand points to facilities so that the overall facility location costs and transportation costs are minimized, as shown in equation (C.18). Constraint (C.19) ensures all demands are met. Constraint (C.20) ensures that the demand points are assigned to open facilities only. (If $y_j = 0$, no facility is located at site j , so no demand can be served from there; if $y_j = 1$, then potentially all demand could be served from there.)

Now consider a new optimization model consisting of the same objective function (C.18), along with constraints (C.19), (C.21), and (C.22), but with constraint (C.20) replaced by: $x_{ij} \leq y_j \forall i \in I, \forall j \in J$.

S-UFLP :

$$\min_{\mathbf{x}, \mathbf{y}} \sum_{i \in I} \sum_{j \in J} c_{ij} x_{ij} + \sum_{j \in J} f_j y_j \quad (\text{C.23})$$

$$\text{s.t.} \quad \sum_{j \in J} x_{ij} = 1 \quad \forall i \in I \quad (\text{C.24})$$

$$x_{ij} \leq y_j \quad \forall i \in I, \forall j \in J \quad (\text{C.25})$$

$$y_j \in \{0, 1\} \quad \forall j \in J \quad (\text{C.26})$$

$$x_{ij} \geq 0 \quad \forall i \in I, \forall j \in J \quad (\text{C.27})$$

Let P and Q denote the feasible regions of the linear relaxations for models O-UFLP and S-UFLP, respectively. We can show that $Q \subseteq P$, that is, any solution feasible to S-UFLP is also feasible to O-UFLP. Any point which satisfies $x_{ij} \leq y_j$ must also satisfy $x_{ij} \leq |I| y_j$, as can be seen by summing up x_{ij} along all values of i . Now, when restricted to integer values of \mathbf{y} , the set of feasible solutions is the same in both cases. However, when considering their

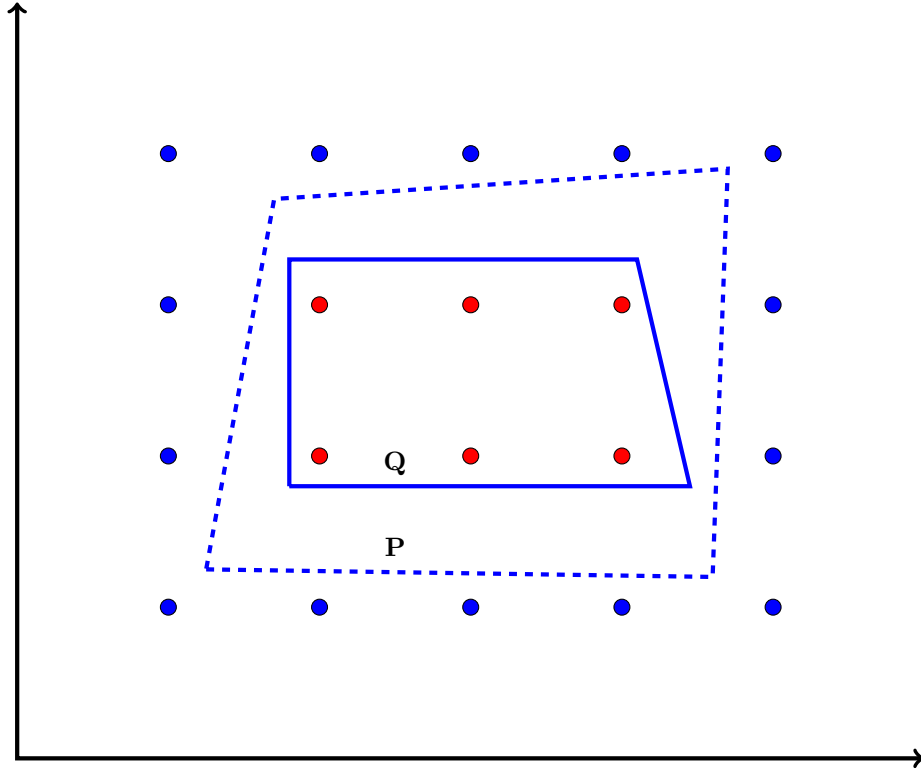


Figure C.10: Stronger and weaker IP formulations.

LP relaxations, there are solutions which are feasible to O-UFLP which are not feasible to S-UFLP. For example, if $y_1 = 1/2$, a solution of the form $x_{11} = 1$ and $x_{i1} = 0$ for $i = \{2, \dots, n\}$ satisfies $x_{i1} \leq |I|y_1$ but not $x_{11} \leq y_1$. As a result of this, the optimal solution to a relaxation based on S-UFLP will have a greater (or equal) objective function value than the same relaxation based on O-UFLP, producing a tighter (and more useful) lower bound on the original integer program. We therefore say that S-UFLP is a *stronger* (or *tighter*) formulation for UFLP. This discussion can be generalized in the following result:

Proposition C.1. Consider two feasible regions P and Q for the linear relaxation of the same minimization integer program $Z = \min\{\mathbf{c}^T \mathbf{x} : \mathbf{x} \in X \cap \mathbb{Z}^n\}$ with $Q \subseteq P$. If $z_{LP}^P = \min\{\mathbf{c}^T \mathbf{x} : \mathbf{x} \in P\}$ and $z_{LP}^Q = \min\{\mathbf{c}^T \mathbf{x} : \mathbf{x} \in Q\}$, then $z_{LP}^P \leq z_{LP}^Q$.

The intuition is shown in the figure C.10. Note that $Q \cap \mathbb{Z}^n = P \cap \mathbb{Z}^n$. However since $Q \subseteq P$, the corresponding LP solution will be larger. Stronger formulations leading to tighter linear programming bounds will result in faster solutions, as long as the time need to solve the tighter linear programming relaxation is not significantly higher.

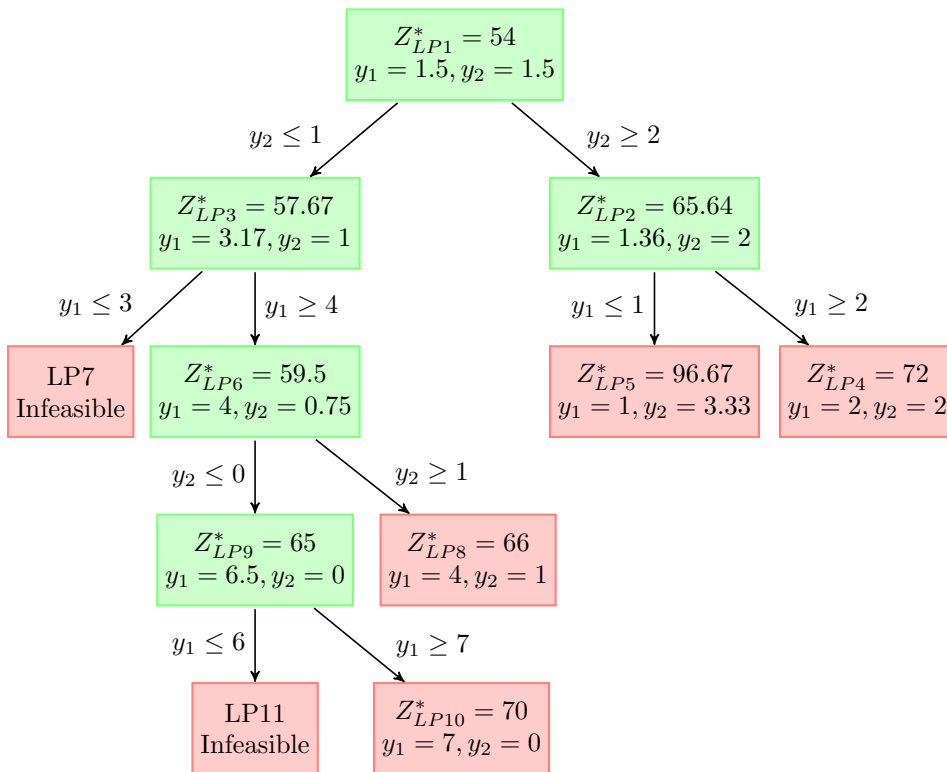


Figure C.11: Branch and bound example.

C.5.1 Branch and bound algorithm

Branch and bound is one of the most popular algorithms used to solve integer programs, and is the basis for many commercial integer programming solvers. It is essentially an intelligent enumeration algorithm, based on the principle of “divide and conquer.” The original integer program is replaced by a series of easier-to-solve relaxations, whose solutions are used to further guide the search toward the optimal solution of the integer program. The branch and bound algorithm is illustrated using the same integer programming example (see figure C.11) used before:

$$\begin{aligned}
 \min_{y_1, y_2} \quad & 10y_1 + 26y_2 \\
 \text{s.t.} \quad & 11y_1 + 3y_2 \geq 21 \\
 & 6y_1 + 20y_2 \geq 39 \\
 & y_1, y_2 \geq 0 \\
 & y_1, y_2 \in \mathbb{Z}
 \end{aligned}$$

We will first show how the algorithm works on this specific example, and then provide the general method below.

The branch and bound algorithm starts by initializing the upper bound

$Z_{UB} = \infty$ and lower bound $Z_{LB} = -\infty$, since we do not yet know any information about the optimal solution. The linear relaxation of the above integer program is solved. Let X_{LP1} denote the feasible region of the LP relaxation. Thus $X_{LP1} = \{\mathbf{y} \in \mathbb{R}_+^2 : 11y_1 + 3y_2 \geq 21, 6y_1 + 20y_2 \geq 39\}$. The optimal value of the linear programming objective function is $Z_{LP1}^* = 54$, and the solution is $y_1 = 1.5, y_2 = 1.5$. The best lower bound is now updated to 54, $Z_{LB} = 54$.

Now, the optimal solution either has y_2 as an integer greater than or equal to 2, or as an integer less than or equal to 1. We consider each option in turn. (We could have also branched on y_1 ; it would be instructive for you to solve the problem that way.) We generate two linear programs by adding the constraints $y_2 \geq 2$ to one, and $y_2 \leq 1$ to the other. The feasible region of the second linear program $X_{LP2} = X_{LP1} \cap \{y_2 \geq 2\}$ and the feasible region of the third linear program $X_{LP3} = X_{LP1} \cap \{y_2 \leq 1\}$. Solving the two linear programs we get $Z_{LP2}^* = 65.64$ with the solution $y_1 = 1.36, y_2 = 2$ and $Z_{LP3}^* = 57.67$ with the solution $y_1 = 3.17, y_2 = 1$.

Let us first examine LP2. Since y_2 is already an integer in its optimal solution, we don't need to branch again on y_2 . However, y_1 is not an integer, and in the optimal solution it must be either greater than or equal to 2, or less than or equal to 1. So the constraints $y_1 \geq 2$ and $y_2 \leq 1$ are added to the feasible regions of LP2 to generate new linear programs LP4 and LP5, with the respective feasible regions $X_{LP4} = X_{LP2} \cap \{y_1 \geq 2\}$ and $X_{LP5} = X_{LP2} \cap \{y_1 \leq 1\}$.

Solving LP4 we get the optimal objective value to be 74 with the solution $y_1 = 2, y_2 = 2$. The solution to this linear programming relaxation is an integer solution which is feasible for the original integer program. Therefore Z_{LP5}^* is an upper bound to the optimal integer solution. Moreover since $Z_{LP5}^* < Z_{UB}$ we update $Z_{UB} = 74$; this is the first upper bound we have the optimal objective function value.. Since this is an integer solution, we do not search further in this direction. This is called "fathoming by optimality."

Solving LP5 we get the optimal objective value to be 96.67 with the solution $y_1 = 1, y_2 = 3.33$. Since the LP objective function is greater than the current best upper bound 74, there is no point in further exploring in this direction also and the search is stopped here. This is called "fathoming by bound." Notice that this cuts off a considerable portion of the feasible region. We know we do not have to explore it any further, since the best possible objective function values there are at least 96.67, whereas we have already found a solution (2, 2) with an objective of 74.

Let us now look at LP3. Since y_2 has integer solution we don't need to branch on y_2 . But with respect to y_1 , the optimal solution is either an integer at least 4, or at most 3. So the constraints $y_1 \geq 4$ and $y_1 \leq 3$ is added to the feasible region of LP3 to get new linear programs LP6 and LP7, with respective feasible regions $X_{LP6} = X_{LP3} \cap \{y_1 \geq 4\}$ and $X_{LP7} = X_{LP3} \cap \{y_1 \leq 3\}$.

LP7 turns out to be infeasible, so the search is stopped along this direction. This is called "fathoming by infeasibility." LP6, on the other hand, has an optimal objective function value of 59.5 with $y_1 = 4, y_2 = 0.75$. We branch again on y_2 , adding the constraints $y_2 \geq 1$ and $y_2 \leq 0$ to the feasible region of LP6 to obtain new linear programs LP8 and LP9 with feasible regions $X_{LP8} =$

$X_{LP6} \cap \{y_2 \geq 1\}$ and $X_{LP9} = X_{LP3} \cap \{y_2 \leq 0\}$.

LP8 has an integer solution $y_1 = 4, y_2 = 1$, with an objective function value of 66. This solution is feasible for the original integer program. Moreover since $Z_{LP8}^* < Z_{UB}$, we have found a better feasible solution, and the upper bound is updated to 66: $Z_{UB} = 66$. We fathom the search in this direction by integrality.

Looking at the solution of LP9, we generate two linear programs, LP10 and LP11, by adding the constraints $y_1 \geq 7$ and $y_2 \leq 6$ to the feasible region of LP9. The feasible region of the new linear programs are $X_{LP10} = X_{LP9} \cap \{y_1 \geq 7\}$ and $X_{LP11} = X_{LP9} \cap \{y_1 \leq 6\}$. LP11 is infeasible, and therefore we fathom by infeasibility. LP10 has an integer solution, but the objective function value is greater than the current upper bound. Therefore we fathom by bound as well as integrality. There are no more directions left to search. Therefore the optimal solution corresponds to the current best upper bound of 66.

The general steps of branch and bound are shown in Algorithm 4.

Initialization: Set $Z_{UB} \leftarrow -\infty$ and $Z_{LB} \leftarrow \infty$. Add the LP relaxation of the original integer program to a queue \mathbf{Q} ;

while \mathbf{Q} is not empty **do**

- Select an element CurrLP from \mathbf{Q} and remove it;
- Attempt to solve CurrLP;
- if** CurrLP is infeasible **then**
 - Continue while loop (fathom by infeasibility);
- else**
 - Let \mathbf{x} be the optimal solution to CurrLP, and Z the objective value;
 - if** \mathbf{x} satisfies all of the original integrality constraints **then**
 - $Z_{UB} \leftarrow \min \{Z_{UB}, Z\}$;
 - If Z_{UB} changed, mark \mathbf{x} as the best known solution;
 - Continue while loop (fathom by optimality);
 - else**
 - if** $Z \geq Z_{UB}$ **then**
 - Continue while loop (fathom by bound);
 - else**
 - $Z_{LB} \leftarrow \max \{Z_{LB}, Z\}$;
 - Choose a fractional variable to branch on;
 - Generate two new LPs by adding \geq and \leq constraints to the feasible region of CurrLP;
 - Add the two new LPs to \mathbf{Q} ;

end

end

end

Algorithm 4: Branch and bound algorithm.

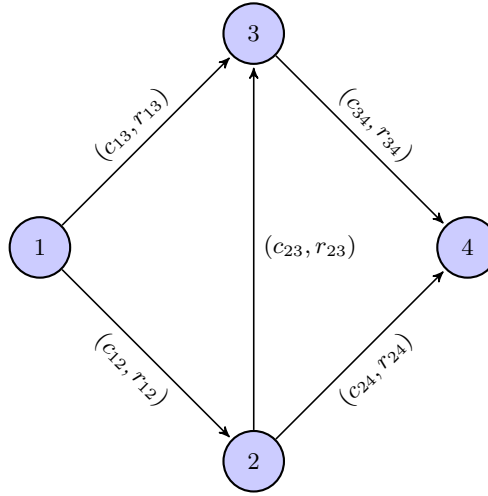


Figure C.12: Network for shortest path example

C.5.2 Total unimodularity

In general, we cannot exactly solve an integer program only by solving its linear programming relaxation (although we can use these LP relaxations in a method such as branch and bound). There is one notable exception, which shows up frequently enough in network optimization problems to be worth knowing.

Definition C.7. A matrix \mathbf{A} is totally unimodular if the determinant of every square submatrix of \mathbf{A} has the value -1 , 0 , or 1 .

Consider an integer linear program $Z = \min\{\mathbf{c}^T \mathbf{x} : \mathbf{x} \in X \cap \mathbb{Z}^n\}$ where $X = \{\mathbf{x} \in \mathbb{R}^n : \mathbf{A}\mathbf{x} \leq \mathbf{b}, \mathbf{x} \geq 0\}$, and the corresponding linear programming relaxation $Z_{LP} = \min\{\mathbf{c}^T \mathbf{x} : \mathbf{x} \in X\}$. If the matrix \mathbf{A} is totally unimodular, then the LP relaxation has integral solutions. The vertex points of X are integer points. That is, if the coefficients representing the constraints form a totally unimodular matrix, we can ignore the integer constraints altogether. The resulting linear program will have an integer optimal solution which is feasible to the original integer program. In such a case, we can simply apply a linear programming method to find the optimal solution.

The concept of total unimodularity is explained using a modification to the standard shortest path problem from Section 2.4. In this modification, each link is associated with *two* attributes: a cost for traversing it, and the amount of some other resource consumed when traveling it (time, battery charge, etc.) Consider the network shown in figure C.12. The network has four nodes and five arcs. Let c_{12} and r_{12} denote the cost and the amount of resource consumed by traversing arc $(1, 2)$. The goal of the shortest path problem is to determine the minimum cost path from 1 to 4.

Let $x_{12}, x_{13}, x_{23}, x_{24}$, and x_{34} denote the decision variables. These are equal

to 1 if the corresponding arc lies on the shortest path, and 0 otherwise. The shortest path formulation for the above network is:

$$\begin{aligned}
 \min \quad & c_{12}x_{12} + c_{13}x_{13} + c_{23}x_{23} + c_{24}x_{24} + c_{34}x_{34} \\
 \text{s.t.} \quad & x_{12} + x_{13} = 1 \\
 & -x_{12} + x_{23} + x_{24} = 0 \\
 & -x_{13} - x_{23} + x_{34} = 0 \\
 & -x_{24} - x_{34} = -1 \\
 & x_{12}, x_{13}, x_{23}, x_{24}, x_{34} \in \{0, 1\}
 \end{aligned}$$

The constraint matrix is

$$\mathbf{A} = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ -1 & 0 & 1 & 1 & 0 \\ 0 & -1 & -1 & 0 & 1 \\ 0 & 0 & 0 & -1 & -1 \end{bmatrix}$$

All the square submatrices of A have determinant 0, 1, or -1 . Therefore we can solve the shortest path problem by solving the corresponding linear programming relaxation. (The shortest path problem *could* be solved using branch and bound; but it is better to solve it as a linear program, and even better to solve it using a specialized method like those in Section 2.4.)

The resource-constrained shortest path problem is a variant where the shortest path is sought, but with an upper bound on the total amount of resources consumed. An example might be finding the fastest route in a network with toll roads, where the total monetary cost does not exceed a given budget. We represent this with an additional constraint in the shortest path formulation:

$$r_{12}x_{12} + r_{13}x_{13} + r_{23}x_{23} + r_{24}x_{24} + r_{34}x_{34} \leq R.$$

In this case the constraint matrix becomes

$$\mathbf{A} = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ -1 & 0 & 1 & 1 & 0 \\ 0 & -1 & -1 & 0 & 1 \\ 0 & 0 & 0 & -1 & -1 \\ r_{12} & r_{13} & r_{23} & r_{24} & r_{34} \end{bmatrix}$$

Depending on the values of the resources consumed, the constraint matrix need not be totally unimodular. In this case the linear programming relaxation does not always have an integer solution. For this reason, the resource-constrained shortest path problem is much harder to solve than the traditional shortest path problem.

C.6 Metaheuristics

In optimization, there is often a tradeoff between how widely applicable a solution method is, and how efficient or effective it is at solving specific problems.

For any one specific problem, a tailor-made solution process is likely much faster than a generally-applicable method, but generating such a method requires more effort and specialized knowledge, and is less “rewarding” in the sense that the method can only be more narrowly applied. Some of the most general techniques are *heuristics*, which are not guaranteed to find the global optimum solution, but tend to work reasonably well in practice. In practice, they only tend to be applied for very large or very complicated problems which cannot be solved exactly in a reasonable amount of time with our current knowledge.

Many engineers are initially uncomfortable with the idea of a heuristic. After all, the goal of an optimization problem is to find an optimal solution, so why should we settle for something which is only approximately “optimal,” often without any guarantees of how approximate the solution is? First, for very complicated problems a good heuristic can often return a reasonably good solution in much less time than it would take to find the exact, global optimal solution. For many practical problems, the cost improvement from a reasonably good solution to an exactly optimal one is not worth the extra expense (both time and computational hardware) needed, particularly if the heuristic gets you within the margin of error based on the input data.

Heuristics are also very, very common in psychology and nature. If I give someone a map and ask them to find the shortest-distance route between two points in a city by hand, they will almost certainly not formulate a mathematical model and solve it to provable optimality. Instead, they use mental heuristics (rules of thumb based on experience) and can find paths which are actually quite good. Many of the heuristics are inspired by things seen in nature.

An example is how ant colonies find food. When a lone wandering ant encounters a food source, it returns to the colony and lays down a chemical pheromone. Other ants who stumble across this pheromone begin to follow it to the food source, and lay down more pheromones, and so forth. Over time, more and more ants will travel to the food source, taking it back to the colony, until it is exhausted at which point the pheromones will evaporate. Is this method the optimal way to gather food? Perhaps not, but it performs well enough for ants to have survived for millions of years!

Another example is the process of evolution through natural selection. The human body, and many other organisms, function remarkably well in their habitats, even if their biology is not exactly “optimal.”² One of the most common heuristics in use today, and one described below, is based on applying principles of natural selection and mutation to a “population” of candidate solutions to an optimization problem, using an evolutionary process to identify better and better solutions over time. This volume describes two heuristics: *simulated annealing*, and *genetic algorithms*, both of which can be applied to many different optimization problems.

²For instance, in humans the retina is “backwards,” creating a blind spot; in giraffes, the laryngeal nerve takes an exceptionally long and roundabout path; and the descent of the testes makes men more vulnerable to hernias later in life.

C.6.1 Simulated annealing

Simulated annealing is a simple heuristic that makes an analogy to metallurgy, but this analogy is best understood after the heuristic is described.

As a better starting point for understanding simulated annealing, consider the following “local search” heuristic. (For now, descriptions will be a bit vague; more precise definitions will follow soon.) Local search proceeds as follows:

1. Choose some initial feasible solution $\mathbf{x} \in X$, and calculate the value of the objective function $f(\mathbf{x})$.
2. Generate a new feasible solution $\mathbf{x}' \in X$ which is close to the current solution \mathbf{x} .
3. Calculate $f(\mathbf{x}')$
4. If $f(\mathbf{x}') \leq f(\mathbf{x})$, the new solution is better than the old solution. So update the current solution by setting \mathbf{x} equal to \mathbf{x}' .
5. Otherwise, the old solution is better, so leave \mathbf{x} unchanged.
6. Return to step 2 and repeat until we are unable to make further progress.

One way to visualize local search is a hiker trying to find the lowest elevation point in a mountain range. In this analogy, the park boundaries are the feasible set, the elevation of any point is the objective function, and the location of the hiker is the decision variable. In local search, starting from his or her initial position, the hiker looks around and finds a nearby point which is lower than their current point. If such a point can be found, they move in that direction and repeat the process. If every neighboring point is higher, then they stop and conclude that they have found the lowest point in the park.

It is not hard to see why this strategy can fail; if there are multiple local optima, the hiker can easily get stuck in a point which is not the lowest (Figure C.13). Simulated annealing attempts to overcome this deficiency of local search by allowing a provision for occasionally moving in an uphill direction, in hopes of finding an even lower point later on. Of course, we don't always want to move in an uphill direction and have a preference for downhill directions, but this preference cannot be absolute if there is any hope of escaping local minima.

Simulated annealing accomplishes this by making the decision to move or not probabilistic, introducing a *temperature* parameter T which controls how likely you are to accept an “uphill” move. When the temperature is high, the probability of moving uphill is large, but when the temperature is low, the probability of moving uphill becomes small. In simulated annealing, the temperature is controlled with a *cooling schedule*. Initially, the temperature is kept high to encourage a broad exploration of the feasible region. As the temperature decreases slowly, the solution is drawn more and more to lower areas. The cooling schedule can be defined by an initial temperature T_0 , a final temperature T_f , the number of search iterations n at a given temperature level,

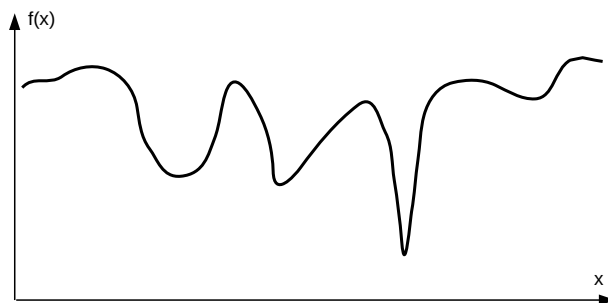


Figure C.13: Moving downhill from the current location may not lead to the global optimum.

and a scaling factor $k \in (0, 1)$ which is applied every n iterations to reduce the temperature.

Finally, because the search moves both uphill and downhill, there is no guarantee that the final point of the search is the best point found so far. So, it is worthwhile to keep track of the best solution \mathbf{x}^* encountered during the algorithm. (This is analogous to the hiker keeping a record of the lowest point observed, and returning to that point when done searching.)

So, the simulated annealing algorithm can be stated as follows:

1. Choose some initial feasible solution $\mathbf{x} \in X$, and calculate the value of the objective function $f(\mathbf{x})$.
2. Initialize the best solution to the initial one $\mathbf{x}^* \leftarrow \mathbf{x}$.
3. Set the temperature to the initial temperature: $T \leftarrow T_0$
4. Repeat the following steps n times:
 - (a) Generate a new feasible solution \mathbf{x}' which neighbors \mathbf{x} .
 - (b) If $f(\mathbf{x}') < f(\mathbf{x}^*)$, it is the best solution found so far, so update $\mathbf{x}^* \leftarrow \mathbf{x}'$.
 - (c) If $f(\mathbf{x}') \leq f(\mathbf{x})$, it is a better solution than the current one, so update $\mathbf{x} \leftarrow \mathbf{x}'$.
 - (d) Otherwise, update $\mathbf{x} \leftarrow \mathbf{x}'$ with probability $\exp(-[f(\mathbf{x}') - f(\mathbf{x})]/T)$
5. If $T > T_f$, then reduce the temperature ($T \leftarrow kT$) and return to step 4.
6. Report the best solution found \mathbf{x}^*

The key step is step 4d. Notice how the probability of “moving uphill” depends on two factors: the temperature, and how much the objective function will increase. The algorithm is more likely to accept an uphill move if it is only slightly uphill, or if the temperature is high. The exponential function captures

these effects while keeping the probability between 0 and 1. A few points require explanation.

How should the cooling schedule be chosen? Unfortunately, it is hard to give general guidance here. Heuristics often have to be “tuned” for a particular problem: some problems do better with higher temperatures and slower cooling (k values closer to 1, n larger), others work fine with faster cooling. When you use simulated annealing, you should try different variations of the cooling schedule to identify one that works well for your specific problem.

How should an initial solution be chosen? It is often helpful if the initial solution is relatively close to the optimal solution. For instance, if the optimization problem concerns business operations, the current operational plan can be used as the initial solution for further optimization. However, it’s easy to go overboard with this. You don’t want to spent so long coming up with a good initial solution that it would have been faster to simply run simulated annealing for longer starting from a worse solution. The ideal is to think of a good, quick rule of thumb for generating a reasonable initial solution; failing that, you can always choose the initial solution randomly.

How do I define a neighboring solution for step 4a? Again, this is problem-specific, and one of the decisions that must be made when applying simulated annealing. A good neighborhood definition should involve points which are “close” to the current solution in some way, but ensure that feasible solutions are connected in the sense that any two feasible solutions can be reached by a chain of neighboring solutions. For the examples in Section B.4, some possibilities (emphasis on *possibilities*, there are other choices) are:

Transit frequency setting problem: The decision variables \mathbf{n} are the number of buses assigned to each route. Given a current solution \mathbf{n} , a neighboring solution is one where exactly one bus has been assigned to a different route.

Scheduling maintenance: The decision variables are \mathbf{x} , indicating where and when maintenance is performed, and \mathbf{c} , indicating the condition of the facilities. In this problem, the constraints completely define \mathbf{c} in terms of \mathbf{x} , so for simulated annealing it is enough to simply choose a feasible solution \mathbf{x} , then calculate \mathbf{c} using the state evolution equations. In this problem, there is no reason to perform less maintenance than the budget allows each year. If the initial solution is chosen in this way, then a neighboring solution might be one where one maintenance activity on a facility is reassigned to another facility that same year. If the resulting solution is infeasible (because the resulting \mathbf{c} values fall below the minimum threshold), then another solution should be generated.

Facility location problem: The decision variables are the intersections that the three terminals are located at, L_1 , L_2 , and L_3 . Given current values for these, in a neighboring solution two of the three terminals are at the same location, but one of the three has been assigned to a different location.

Shortest path problem: The decision variables specify a path between the origin and destination. Given a current path, a neighboring path is one which differs in only one intersection. (Can you think of a way to express this mathematically?)

How do I perform a step with a given probability? Most programming languages have a way to generate a uniform random variable between 0 and 1. If we want to perform a step with probability p , generate a random number from the continuous uniform $(0, 1)$ distribution. If this number is less than p , perform the step; otherwise, do not.

Example with facility location

This section demonstrates simulated annealing, on an instance of the facility location problem from Section B.4. In this problem instance, the road grid consists of ten north-south and ten east-west streets. The cost of locating a terminal at each of the locations is shown in Figure C.14 (these numbers were randomly generated). There are 30 customers, randomly located throughout the grid, as shown in Figure C.15. In these figures, the coordinate system (x, y) is used to represent points, where x represents the number of blocks to the right of the upper-left corner, and y represents the number of blocks below the upper-left corner.

The cooling schedule is as follows: the initial temperature is $T_0 = 1000$, the final temperature is $T_f = 100$, $k = 0.75$, and the temperature is reduced every 8 iterations. (These values were chosen through trial-and-error, and work reasonably well for this problem.) The first several iterations are shown below.

The algorithm begins with an initial solution, generated randomly. Suppose this solution locates the three facilities at coordinates $(5,1)$, $(8,6)$, and $(6,1)$, respectively; the total cost associated with this solution is 158.0 cost units. A neighboring solution is generated by picking one of the facilities (randomly) and assigning it to another location (randomly). Suppose that the third facility is reassigned to location $(1,0)$, so the new candidate solution locates the facilities at $(5,1)$, $(8,6)$, and $(1,0)$. This solution has a cost of 136.4 units, which is lower than the current solution. So, simulated annealing replaces the current solution with the new one.

In the next iteration, suppose that the first facility is randomly chosen to be reassigned to $(8,2)$, so the candidate solution is $(8,2)$, $(8,6)$, and $(1,0)$. This solution has a cost of 149.1, which is higher than the current cost of 136.4. The algorithm will not always move to such a solution, but will only do so with probability calculated as in Step 4c:

$$p = \exp(-[f(\mathbf{x}') - f(\mathbf{x})]/T) = \exp(-[149.1 - 136.4]/100) = 0.881$$

This probability is high, because (being one of the early iterations) the temperature is set high. If the temperature were lower, the probability of accepting this move would be lower as well.

8.7	13.5	11.0	10.2	6.3	6.4	12.4	13.5	9.7	6.8
5.8	12.4	8.8	15.1	14.7	13.9	9.0	5.2	8.7	12.7
11.8	9.8	12.4	12.6	6.3	13.3	7.7	10.2	13.6	13.5
5.4	6.9	11.1	8.4	14.2	10.9	10.7	11.7	8.1	8.9
4.9	12.5	10.8	6.6	12.0	6.8	11.9	9.2	9.5	10.0
14.4	9.7	8.6	11.6	8.0	6.7	12.7	5.9	7.6	14.1
7.7	9.6	6.4	9.9	9.2	13.3	12.3	14.7	15.0	5.1
7.6	14.7	7.1	5.5	5.5	9.7	9.7	14.4	7.9	15.1
10.9	12.5	9.2	12.1	14.8	6.4	6.1	10.5	12.5	10.8
12.0	5.9	13.1	10.0	11.9	10.0	8.6	8.4	10.7	5.3

Figure C.14: Cost of locating facilities at each intersection.

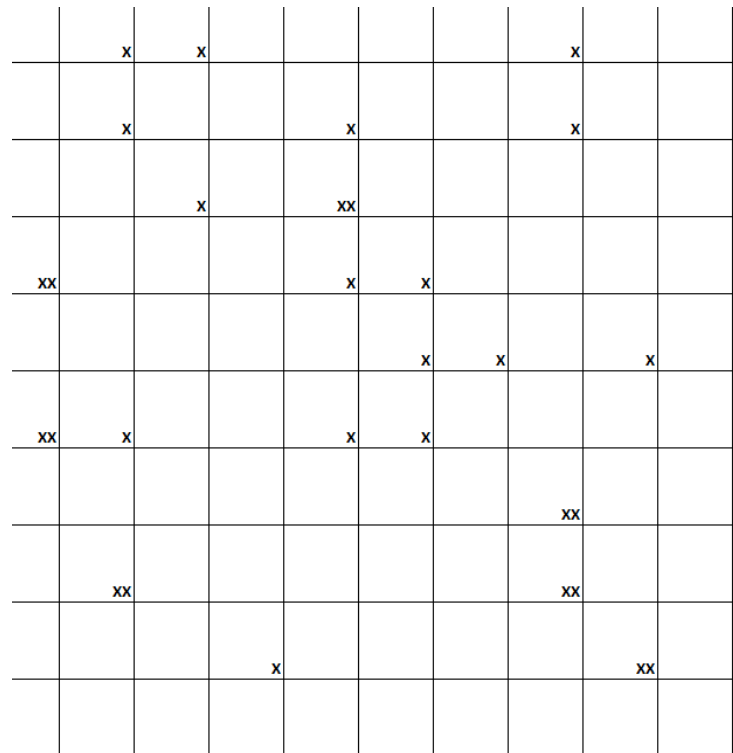


Figure C.15: Locations of customers.

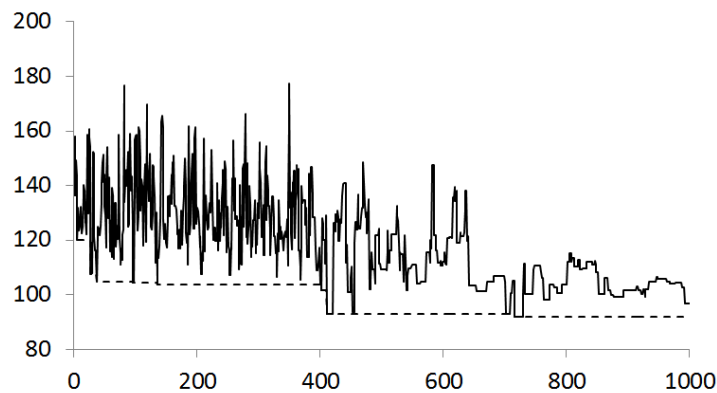


Figure C.16: Progress of simulated annealing. Solid line shows current solution cost, dashed line best cost so far.

Supposing that the move is accepted, the algorithm replaces the current solution with the candidate and continues as before. If, on the other hand, the move is rejected, the algorithm generates another candidate solution based on the same current solution $(5,1)$, $(8,6)$, and $(1,0)$. The algorithm continues in the same way, reducing the temperature by 25% every 8 iterations.

The progress of the algorithm until termination is shown in Figure C.16. The solid line shows the cost of the current solution, while the dashed line tracks the cost of the best solution found so far. A few observations are worth making. First, in the early iterations the cost is highly variable, but towards termination the cost becomes more stable. This is due to the reduction in temperature which occurs over successive iterations. When the temperature is high, nearly any move will be accepted so one expects large fluctuations in the cost. When the temperature is low, the algorithm is less likely to accept cost-increasing moves, so fewer fluctuations are seen. Also notice that the best solution was found shortly after iteration 700. The final solution is not the best, although it is close.

The best facility locations found by simulated annealing are shown in Figure C.17, with a total cost of 92.1.

C.6.2 Genetic algorithms

Another heuristic is the genetic algorithm. In contrast to simulated annealing, where there is a single search happening through the feasible region (recall the analogy with the hiker), genetic algorithms maintain a larger *population* of solutions which reflect a diverse set of points within the feasible region. Genetic algorithms work by attempting to improve the population from one iteration to the next.

The process of improvement is intended to mimic the processes of natural

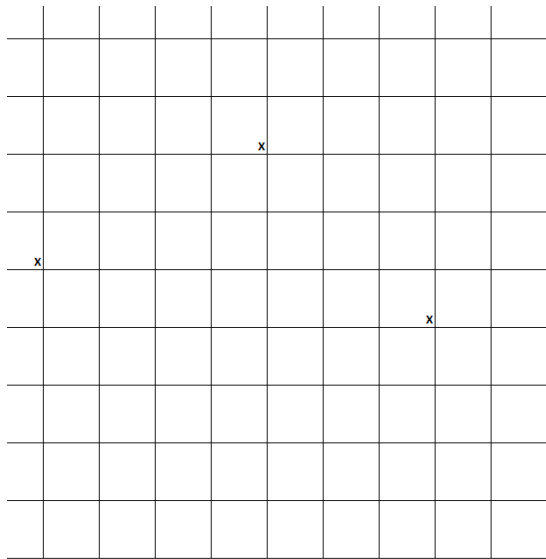


Figure C.17: Locations of facilities found by simulated annealing (cost 92.1).

selection, reproduction, and mutation which are observed in biology. For the sake of our purposes, *natural selection* means identifying solutions in the population which have good (low) values of the objective function. The hope is that there are certain aspects or patterns in these solutions which make them good, which can be maintained in future generations. Given an initial population as the first generation, subsequent generations are created by choosing good solutions from the previous generation, and “breeding” them with each other in a process called *crossover* which mimics sexual reproduction: new “child” solutions are created by mixing characteristics from two “parents.” Lastly, there is a random *mutation* element, where solutions are changed externally with a small probability. If all goes well, after multiple generations the population will tend towards better and better solutions to the optimization problem.

At a high level, the algorithm is thus very straightforward and presented below. However, at a lower level, there are a number of careful choices which need to be made about how to implement each step for a particular problem. Thus, each of the steps is described in more detail below. The high-level version of genetic algorithms is as follows:

1. Generate an initial population of N feasible solutions (generation 0), set generation counter $g \leftarrow 0$.
2. Create generation $g + 1$ in the following way, repeating each step N times:
 - (a) Choose two “parent” solutions from generation g .
 - (b) Combine the two parent solutions to create a new solution.

(c) With probability p , mutate the new solution.

3. Increase g by 1 and return to step 2 unless done.

Although not listed explicitly, it is a good idea to keep track at every stage of the best solution \mathbf{x}^* found so far. Just like in simulated annealing, there is no guarantee that the best solution will be found in the last generation. So, whenever a new solution is formed, calculate its objective function value, and record the solution if it is better than any found so far.

Here are explanations of each step in more detail. It is very important to realize that *there are many ways to do each of these steps* and that what is presented below is one specific example intended to give the general flavor while sparing unnecessary complications at this stage.

Generate an initial population of feasible solutions: In contrast with simulated annealing, where we had to generate an initial feasible solution, with genetic algorithms we must generate a larger population of initial feasible solutions. While it is still desirable to have these initial feasible solutions be reasonably good, it is also very important to have some diversity in the population. Genetic algorithms work by combining characteristics of different solutions together. If there is little diversity in the initial population the difference between successive generations will be small and progress will be slow.

Selection of parent solutions: Parent solutions should be chosen in a way that better solutions (lower objective function values) are more likely to be chosen. This is intended to mimic natural selection, where organisms better adapted for a particular environment are more likely to reproduce. One way to do this is through *tournament selection*, where a number of solutions from the old generation are selected randomly, and the one with the best objective function value is chosen as the first parent. Repeating the “tournament” again, randomly select another subset of solutions from the old generation, and choose the best one as the second parent. The number of entrants in the tournament is a parameter that you must choose.

Combining parent solutions: This is perhaps the trickiest part of genetic algorithms: how can we combine two feasible solutions to generate a new feasible solution which retains aspects of both parents? The exact process will differ from problem to problem. Here are some ideas, based on the example problems from Section B.4:

Transit frequency setting problem: The decision variables are the number of buses on each route. Another way of “encoding” this decision is to make a list assigning each bus in the fleet to a corresponding route. (Clearly, given such a list, we can construct the n_r values by counting how many times a route appears.) Then, to generate a new list from two parent lists, we can assign each bus to either its route in one parent, or its route in the other parent, choosing randomly for each bus.

Scheduling maintenance: As described above, optimal solutions must always exhaust the budget each year. So, along the lines of the transit

frequency setting problem, make a list of the number of maintenance actions which will be performed in each year, noting the facility assigned to each maintenance action in the budget. To create a new solution, for each maintenance action in the budget choose one of the facilities assigned to this action from the two parents, selecting randomly. Once this list is obtained, it is straightforward to calculate \mathbf{x} and \mathbf{c} .

Facility location problem: For each facility in the child solution, make its location the same as the location of that facility in one of the two parents (chosen randomly).

Mutating solutions: Mutation is intended to provide additional diversity to the populations, avoiding stagnation and local optimal solutions. Mutation can be accomplished in the same way that neighbors are generated in simulated annealing. The probability of mutation should not be too high — as in nature, most mutations are harmful — but enough to escape from local optima. The mutation probability p can be selected by trial and error, determining what works well for your problem.

Example with facility location

This section demonstrates genetic algorithms on the same facility location problem used to demonstrate simulated annealing. The genetic algorithm is implemented with a population size of 100, over ten generations. Tournaments of size 3 are used to identify parent solutions, and the mutation probability is 0.05. The initial population is generated by locating all the facilities are completely at random.

To form the next generation, 100 new solutions need to be created, each based on combining two solutions from the previous generation. These two parents are chosen using the tournament selection rule, as shown in Figure C.18. The two winners of the tournament locate the three facilities at (2,3), (1,7), (8,7); and (1,4), (7,6), (5,0) respectively. These combine in the following way:

1. The first facility is located either at (2,3) or (1,4); randomly choose one of them, say, (2,3).
2. The second facility is located either at (1,7) or (7,6); randomly choose one of them, say, (1,7).
3. The third facility is located either at (8,7) or (5,0); randomly choose one of them, say, (5,0).

This gives a new solution (2,3), (1,7), (5,0) in the next generation, as shown in Figure C.18. With 5% probability, this solution will be “mutated.” If this solution is selected for mutation, one of the three facility is randomly reassigned to another location. For instance, the facility at (1,7) may be reassigned to (0,4). This process is repeated until all 100 solutions in the next generation have been created.

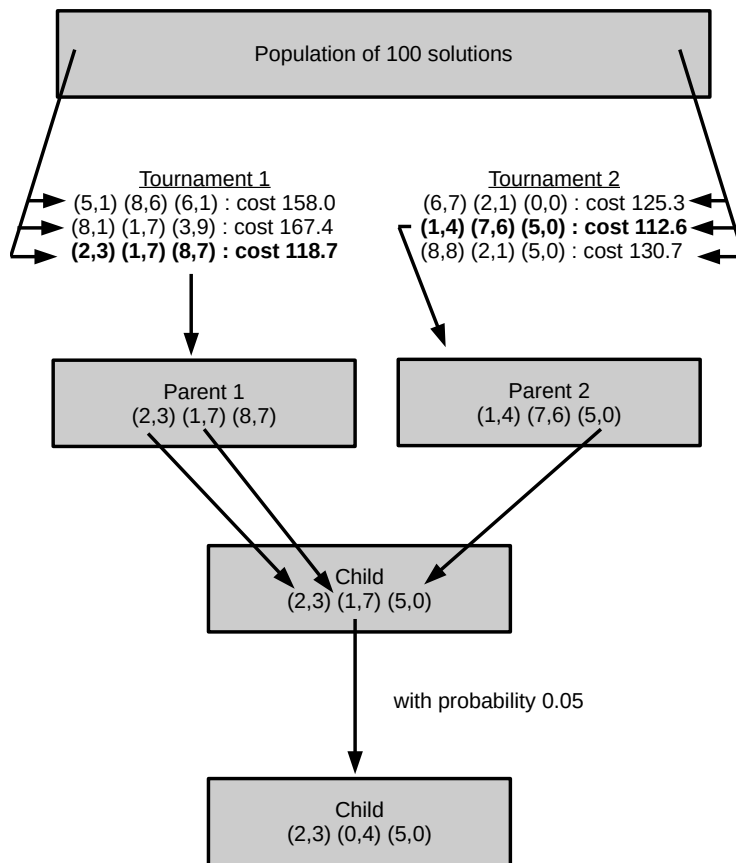


Figure C.18: Generation of a new solution: reproduction and mutation.

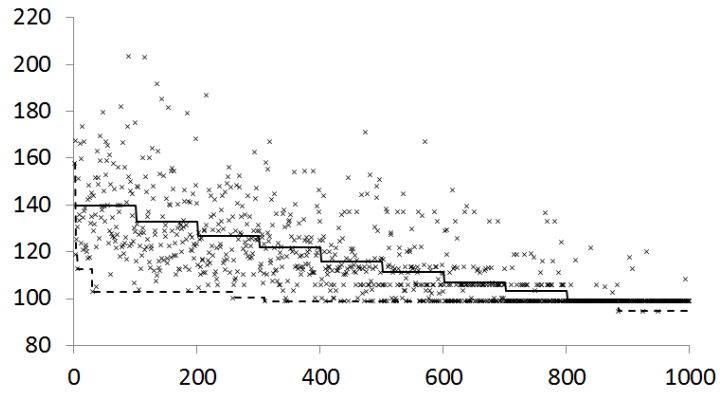


Figure C.19: Progress of genetic algorithm. Solid line shows average generation cost, dashed line best cost so far, crosses cost of individual solutions.

Figure C.19 shows the progress of the algorithm over ten generations. The solid line shows the average cost in each generation. The dashed line shows the cost of the best solution found so far, and the crosses show the cost of each of the solutions comprising the generations. Since lower-cost alternatives are more likely to win the tournaments and be selected as parents, the average cost of each generation decreases. Figure C.20 shows the locations of the terminals in the best solution found.

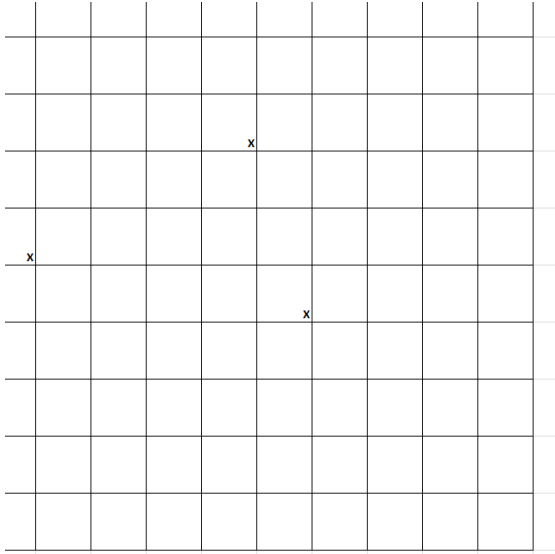


Figure C.20: Locations of facilities found by genetic algorithm (cost 94.7).

Appendix D

Algorithms and Complexity

Appendices B and C introduced the concepts of formulating optimization problem, provided examples of different types and categories of formulations, and presented algorithms for solving them. This appendix focuses more on what specifically an algorithm is, and how complexity analysis can be used to compare the performance of alternative algorithms for solving a problem.

D.1 Algorithms

The word *algorithm* appears often in applied mathematics and computer science to denote a concrete procedure for accomplishing a task, specified in enough detail that it can be implemented by a computer (often in a programming language like C or Python). We adopt Knuth's (1997) criteria for operationalizing this definition: an algorithm must be *finite* (always stopping in a finite number of steps), *definite* (each step is unambiguous), *effective* (each operation is simple enough to be done in a finite amount of time), and map *inputs* to at least one *output*. An algorithm can have any finite number of inputs, even zero: an algorithm with no inputs would compute the same thing every time it is run, but that still might be useful (e.g., to answer a difficult question). However, it must have at least one output; an algorithm must report *something* when it terminates.

For example, a step such as “add integers x and y ” is definite and effective. A step such as “multiply π by 2” is definite, but not effective since π is an irrational number with an infinite number of digits (there is no way to multiply *all* the digits by 2). Such a step can be made effective by specifying “round π to 15 digits and multiply it by 2.” A step such as “pick an integer between 1 and 10” is effective, but it is not definite (*how* exactly should I pick such an integer?) One way to make it definite is to specify “randomly pick an integer between 1 and 10, with each possibility equally likely.”

Another example of a “definite but not effective” step is “add 1 to x if there are one million consecutive zeros in the decimal expansion of π .” It is definite:

either there are that many consecutive zeros or there are not. However, we don't already know whether that is true, and there is no simple way to find out. We could spend a lot of time computing digits of π and maybe we will find this many zeros; but maybe we won't, and we don't know if it's because there aren't any, or if we just haven't computed far enough. With our current knowledge of mathematics, there is no way to answer that question in a finite amount of time, so this step is not effective.

Many algorithms contain three main components:

Initialization: These are the first steps performed. The initialization steps prepare for the main algorithm, assembling the inputs, and providing initial or default values to variables which will be used later on.

Body: The steps in the body form the main core of the algorithm. Many algorithms are *iterative*, and the steps in the body are repeated multiple times.

Stopping criterion: An algorithm must be finite, and not run forever. However, we often don't know how many iterations are required in advance. Therefore, we periodically check a stopping criterion; once it is satisfied, we can stop. There may also be a few final processing and "clean up" steps needed at the end of the algorithm.

You were probably taught an algorithm for adding two positive integers when you were a child (although that word was probably not used). One way to do this is to work from right to left, adding the digits one at a time. If the sum of the digits is 9 or less, then that is the corresponding digit in the sum; if it is 10 or more, subtract 10 to get the digit in the sum, and "carry" by adding 1 to the *next* digit to the left. For example, to add 158 and 371, you would start by adding $8 + 1 = 9$ for the rightmost digit; $5 + 7 = 12$, so enter 2 for the next digit and carry a 1 for the next digit: $1 + 3 + 1 = 5$, so the sum is 529.

This can be specified as an algorithm in the following way. For simplicity, we assume that the two integers we are adding (a and b) have the same number of digits n , and they are specified as $a = a_n a_{n-1} \dots a_2 a_1$ and $b = b_n b_{n-1} \dots b_2 b_1$. (In the previous example, $a = 158$, and $a_3 = 1$, $a_2 = 5$, and $a_1 = 8$; and likewise for $b = 371$.) We also assume that we know how to add two single digits, perhaps with an "addition table" telling us the value of adding $0 + 0$, $0 + 1$, and so on, up through $9 + 9$. The steps below describe the addition algorithm, calculating the digits in the sum s .

Note the use of the \leftarrow operator to mean *assignment*: calculate the expression on the right-hand side, and store the result in the variable on the left-hand side. For instance, $c \leftarrow 0$ sets the variable c to zero. Many programming languages use the $=$ sign to denote this (you would type $c = 0$). In this text, we will reserve $=$ to refer to *mathematical equality*, the statement that the left-hand side and right-hand side are currently equal to each other. So a statement like $x = x + 1$ can never be true (no number is equal to one plus itself). However, a statement like $x \leftarrow x + 1$ is perfectly fine; it means add one to the current value

in x , and then store that new value in x . Some programming languages use context to determine whether $=$ is meant in the sense of assignment or equality, while others use different operators for the two (for instance, C and Python use $=$ for assignment, and $==$ for equality). To avoid confusion, we will use the different characters \leftarrow for assignment, and $=$ for equality.

1. Initialization: set the carry digit $c \leftarrow 0$, and the current digit $i \leftarrow 1$.
2. Compute $s_i \leftarrow c + a_i + b_i$.
3. If $s_i \geq 10$, set $c \leftarrow 1$ and $s_i \leftarrow s_i - 10$; otherwise, set $c \leftarrow 0$.
4. If $i < n$, set $i \leftarrow i + 1$ and return to step 2.
5. If $c = 1$, set $s_{n+1} = 1$.
6. Return the integer s and terminate.

In this algorithm, step 1 is the initialization, and steps 2–3 are the body. Step 4 contains the stopping criterion, expressed in a “negative” way (“repeat as long as $i < n$ ” rather than “stop repeating if $i \geq n$.”) Step 5 contains a final step before we terminate (if there is a carry digit after adding the n -th place, we need to add a 1 at the left-hand side of the sum, which will have one more digit than the original numbers.)

In general, when reading an algorithm, *you should work out a small example on a separate piece of paper to see how the steps work*. Trying to mentally think through all the steps adds unnecessary difficulty, especially for complex algorithms. It is much easier to see *how* it works on a small example or two, and then study the specific steps to see how they fit together.¹ So, let’s see how this algorithm works for the numbers $a = 158$ and $b = 371$. These are three-digit numbers, so $n = 3$. After the initialization (step 1), $c = 0$ and $i = 1$, meaning we are working with the first (right-most) digit and there is nothing to carry so far. Step 2 adds the two right-most digits: $s_1 = c + a_1 + b_1 = 9$, giving the right-most digit in the sum. This is less than 10, so the carry digit is set to zero (it is already zero, so nothing changes). The current digit is $i = 1$, which is less than 3; so in step 4 we increase it and return to step 2 with $i = 2$. Repeating with the second digit, we compute $s_2 = c + a_2 + b_2 = 12$. This is greater than 10, so in step 3, set the carry digit c to 1, and reduce the sum digit by 10, so $s_2 = 2$, the second digit from the right in the sum. In step 4, the current digit is still less than the number of digits, so we increase i and return to step 2 with $i = 3$. With the third digit, we compute $s_3 = c + a_3 + b_3 = 5$; since this is less than 10, we set the carry digit to zero. Now $i = n$, so we meet the stopping criterion and move to step 5. Since the carry digit is zero, we are done and return the sum $s = 529$ (collecting the individual digits computed above). Step 5 would have handled the case when there is a carry digit in the end, meaning

¹This is similar to studying a play or movie. The best way to do this is to first *watch* a performance, and then turn to the script to study it in more detail. Starting with the script is much harder, and not really how a movie is meant to be experienced. So too with algorithms.

the sum has more digits than the addends; for instance, the sum of 999 and 111 is 1110, and this step would have added the ‘1’ at the very left.

This algorithm satisfies all the criteria. It is finite, since we repeat the body once for every digit, and any integer has a finite number of digits. It is definite, since every step is spelled out precisely without any room for ambiguity. It is effective, since every step can be done in a finite amount of time. It takes two inputs (the numbers a and b), and produces one output (the sum s).

An analogy is often made between algorithms, and cooking recipes. For example, cooking pasta can be expressed by the following recipe:

1. Bring a large pot of water to a rolling boil.
2. Add the pasta, and stir until the water returns to a boil.
3. Wait 9–11 minutes (or until pasta has desired consistency).
4. Drain pasta in a colander.

Which of the criteria of an algorithm are satisfied by a recipe like this? Which of them are violated, and are they violated in a “weak” way (where you could satisfy our definition by adding some more explanation to the step) or a “strong” way (a fundamental violation that can’t be easily fixed)?

D.2 Pseudocode

The previous section gave the formal definition of an algorithm. Sometimes, specifying an algorithm formally — either writing out all the steps in full detail in English (as in the addition example above) or with source code in a programming language — may be unnecessarily verbose or confusing. *Pseudocode* is a less formal way of specifying the steps of an algorithm. Pseudocode can be thought of as a midway point between intuitive English descriptions (which express the general idea, but are usually ambiguous) and computer code (unambiguous, but harder to read and understand). The goal of pseudocode is to express the main ideas of an algorithm while avoiding unnecessary pedantry. It communicates the algorithm in enough detail that a programmer can write actual code in a specific language, but without the “fiddly bits” inherent in specific languages (declaring variables, calling library functions, details of data structure implementations, and so forth). Algorithm 5 is an example of pseudocode, and illustrates some common conventions.

This pseudocode is for an algorithm that finds the maximum of n numbers. As an example, let us find the maximum of five numbers: 7, 3, 5, 9, and 1. In the initialization, the five numbers are assigned to the list or set or array A , so that $A[1] = 7, A[2] = 3, \dots, A[5] = 1$.² The variable *max_value* is initialized to a large negative number.

²Note that some programming languages index arrays from 0, rather than 1; using pseudocode lets us abstract from this. The assumption is that a programmer working in such a language would make the necessary changes.

Initialize:

$A = \{A[1], A[2], \dots, A[n]\}$: Set, list, or array containing the n numbers

$max_value \leftarrow -\infty$

for $i \leftarrow 1$ **to** n **do**

if $A[i] > max_value$ **then**

$max_value \leftarrow A[i]$

end

end

Algorithm 5: Finding the maximum of n numbers.

The *for loop* is a control structure which repeatedly executes a set of instructions a fixed number of times, as specified by an iterator variable. In this case, i is the iterator variable which takes values from 1 to n . The variable i takes the values 1, 2, 3, 4, and 5 in turn; for each of these values, the set of instructions indented under the for loop is executed. Inside the for loop is an *if* block. This is another control structure which executes the indented instructions when (and only when) a certain condition is satisfied.

Each *iteration* corresponds to one time that the instructions in the for loop are executed. We now illustrate a few iterations. At first, the iterator i takes the value 1. The if statement then checks whether $A[1] = 7 > max_value = -\infty$. Since $7 > -\infty$, the instructions under the if statement are executed, and max_value now takes the value 7. Next, the iterator variable i takes the value 2. The if statement checks whether $A[2] = 3 > max_value = 7$. This is not true, so the instructions under the if statement are not executed, and max_value remains at 7. After five iterations, we will find that max_value is equal to 9, which is indeed the maximum of the given five numbers. The stopping criterion for this algorithm is when the for loop has been executed for all the numbers in the array.

As another example, the pseudocode in Algorithm 6 represents the algorithm to find the root of a differentiable function $f(x)$ using Newton's method. This pseudocode indicates some conventions commonly found in optimization algorithms. Like the other algorithms seen so far, Newton's method repeats a "body" of steps iteratively until the stopping criterion is met. The variable k keeps track of the number of iterations, and x_k is the solution at iteration k .

Initialize:

x_0 : initial guess

ϵ : tolerance

$x_1 \leftarrow x_0 - \frac{f(x_0)}{f'(x_0)}$

$k \leftarrow 1$

while $|x_k - x_{k-1}| > \epsilon$ **do**

$x_{k+1} \leftarrow x_k - \frac{f(x_k)}{f'(x_k)}$

$k \leftarrow k + 1$

end

Algorithm 6: Finding the root of a function $f(x)$.

In this implementation, the *while loop* control structure is used to control the number of repetitions. In the while loop, a set of instructions is repeated as long a condition is met. The condition in the while loop corresponds to the stopping criterion, expressed in a “negative” way (repeat as long as the stopping criterion is false). In Newton’s method, the algorithm stops when the change in solution across consecutive iterations $|x_k - x_{k-1}|$ is less than a pre-specified tolerance level ϵ . The tolerance level is chosen depending on the level of accuracy desired.

We illustrate a few iterations of this algorithm for the function $f(x) = x^2 - 3$, with the initial guess $x_0 = 1$. The tolerance ϵ is set at 0.001, which represents the desired level of accuracy in the output. x_1 is now calculated using the formula to be equal to 2 and iteration counter k is set to 1. The condition in the while loop $|x_1 - x_0|$ is equal to 1 which is greater than *epsilon*. Therefore, we now calculate x_3 to be equal to 1.75 and increase the value of k to 2. This process is repeated until the difference between x_k and x_{k-1} is less than 0.001. The stopping criterion we use ($|x_k - x_{k-1}| \leq \epsilon$) is one of several widely used stopping criteria. Another popular criterion is to stop the iterations when the iteration counter hits a pre-specified maximum number N .

An attentive reader will notice that to be a valid algorithm, it must be finite, meaning that the convergence criterion must eventually be true. If the stopping criterion is a maximum number of iterations N , this will clearly happen at some point. For the stopping criterion $|x_k - x_{k-1}| \leq \epsilon$, it is not obvious that it will eventually be satisfied (and indeed, for some functions f and initial guesses it may never be). *Analysis of algorithms* is the sub-field of computer science which studies algorithms to identify conditions for finiteness (will it eventually stop?), proofs of correctness (are the outputs always the ones we want?), and the time and computer memory they need before stopping. For the most part, this book will not dwell on these details except where relevant for applying transportation network models.

In this case, the advantage of the $|x_k - x_{k-1}| \leq \epsilon$ stopping criterion is that it is directly related to the precision in the solution; after a fixed number of iterations N we may or may not be close to the actual root of the function; or we may have found the solution many iterations ago but kept on going unnecessarily until we hit N iterations. However, the $k = N$ stopping criterion has the advantage of guaranteeing that it will eventually be satisfied. We can combine the two stopping criteria to gain the advantages of both, by changing the condition in the while loop, so it reads “while $|x_k - x_{k-1}| > \epsilon$ and $k < N$.” This lets us stop as soon as we have found the solution, and also guarantees the algorithm will not run forever.

D.3 Algorithmic Efficiency

Many different algorithms can be devised to solve a particular problem. For example, there are several methods available in the literature to sort a list of numbers. How can we choose the “best” one for a particular application? Algorithm efficiency can be characterized in several ways: run time, memory

use, and so on. In this section, we focusing solely on characterizing efficiency in terms of run time; similar methods can be applied to characterize memory use and other resources.

One way to do that would be to implement each algorithm in a programming language, and measure how long each one takes. A difficulty with this approach is that this kind of performance comparison depends on the operating system, programming language, and the hardware specification of the computer used to test the algorithms. You would also need to decide what kinds of tests to run: how large should the list of numbers be, how many tests are needed, and so forth. This section presents an alternative approach based on *worst-case asymptotic analysis*, identifying the critical bottlenecks in the algorithm as the problem becomes harder and harder. By “worst-case,” we mean that our intention is to find a conservative performance guarantee; for particular inputs the algorithm may perform better than our analysis shows, but it will never perform worse. An alternative is to examine “average-case” performance under a typical range of inputs, but this is trickier because what is “typical” for one person’s application of an algorithm may be completely different than another’s. Worst-case performance lets us focus on the algorithm itself, and not one individual use case or implementation.

A small example illustrates what we mean by worst-case analysis. Imagine you are the owner of a casino, and you have developed a new game in which a gambler wagers a certain amount of money. A 6-sided die is then rolled. If the value is 1 or 2, the gambler loses 50% of the amount wagered; if the outcome is 3 or 4, the entire wager is lost; and if the outcome is 5 or 6, the gambler wins an amount of money equal to the wager (so their initial stake is doubled). The process of determining the amount won can be represented by Algorithm 7.

Initialize:

```
Initialize money_bet to the wager;
Obtain outcome by rolling a 6-sided die.
if outcome=1 or outcome=2 then
  | money_returned  $\leftarrow$  money_bet/2
end
else if outcome=3 or outcome=4 then
  | money_returned  $\leftarrow$  money_bet
end
else if outcome=5 or outcome=6 then
  | money_returned  $\leftarrow$  money_bet  $\times$  2
end
return money_returned
```

Algorithm 7: Gambling algorithm for demonstrating worst-case analysis.

We want to measure how long it will take for this algorithm to run. We can either measure this in terms of “clock time” (imagine using a stopwatch to time how long it takes from start to finish), or by counting the number of operations that need to be performed. Algorithmic analysis usually uses the latter

approach, because it keeps the focus on the algorithm rather than a specific hardware. The exact same algorithm will run much faster on a supercomputer than on your phone. As computers continue to advance, it is more useful to say “this algorithm requires X steps” than “it takes Y seconds to run.”

This requires defining what we mean by an “operation.” Some computations take much more time than others; computing the square root of a number, or the cosine of a number, takes significantly longer than changing the sign of a number from positive to negative, for instance. We will define a basic operation to be (i) a simple arithmetic computation (add, subtract, multiply, divide); (ii) testing whether one value is equal to, greater than, or less than another; and (iii) storing a value in a variable using the assignment operator \leftarrow .³

In Algorithm 7, there are two basic operations associated with initialization. Depending on the value of *outcome*, it takes a different number of basic operations to determine which if block is executed. If *outcome* = 1, we only require one comparison to see that the money returned should be half the bet (we don’t have to check whether *outcome* = 2, because the first if statement has an “or” in it; as soon as we know *outcome* = 1 we can jump to the statement in the block.) If *outcome* = 2, we need two comparisons: the first checks whether *outcome* is 1 (false), the second checks whether *outcome* is 2 (true), and only then we execute the statement. If *outcome* = 3, we need three comparisons; two failed comparisons testing whether *outcome* is 1 or 2, and a third successful one. In the worst case, *outcome* = 6, and we need six comparisons to determine which block of code to execute. Within the if blocks, there are either zero arithmetic operations (if the *outcome* is 3 or 4) or one arithmetic operation (all other *outcomes*) multiplying or dividing the wager by 2. There is always one assignment operation in each block.

Putting this together, how many basic operations does the algorithm need? In the best case, *outcome* = 1, and a total of five basic operations are needed (two initialization, one comparison, one arithmetic, and one assignment). In the worst case, *outcome* = 6, and ten operations are needed (two initialization, six comparison, one arithmetic, one assignment). The average case can be treated by representing the operation count as a random variable. Its expected value can be calculated by identifying all possible operation counts under every *outcome*, multiplying by the probability of that *outcome*, and adding. The resulting number is 7.2, representing the long-run average of the number of operations needed.

Which one of these is most useful? The average case might seem to be best, but its calculation was more involved, and it would have to be recomputed if a different kind of die were used where all six outcomes were not equally likely. By contrast, the best case and worst case were much simpler to compute, and they would stay the same even if the die were to be weighted more to some outcomes than others. Between these two, the worst case is more useful for planning purposes. The best case is optimistic to a fault; all it tells you is that

³By “value” in this section, we mean an integer, or a floating-point number rounded to a finite number of digits.

you'll need at least five basic operations (but nothing about how bad it could get). This is akin to a shady investment pitch which only tells you how much you could earn, and nothing about how much you stand to lose. The worst case, by contrast, is a more solid basis for planning and allocating resources. You will never need more than ten operations (and often will need less), so if you plan for this you will never exceed your resource budget.

In this case, the performance of the algorithm requires a constant number of steps. What if it is repeated n times (either the same gambler playing the game multiple times, or several gamblers playing)? The worst case is that ten operations are needed each time; in this case $10n$ basic operations are needed to run the game. Again, this is a conservative estimate. The only time $10n$ operations are needed is if a 6 is rolled all n times the game is played, which is unlikely. However, we can absolutely guarantee that you will never need more than this.

As another example, return to Algorithm 5 for finding the largest of n numbers. In the for loop, for each value of i we perform at most two basic operations: one comparison operation (testing whether $A[i] > max_value$) and one assignment operation ($max_value \leftarrow A[i]$). Since i takes values from 1 to n , these two operations will be repeated n times, giving a total of $2n$ operations. There is also one basic operation associated with initialization, for a total of $2n + 1$ basic operations in the worst case. (The best case has fewer than this; for example, if the $A[1]$ is the largest of the n numbers, then only $n + 2$ operations are needed, because max_value is only assigned once.)

To further simplify the worst-case analysis, we introduce an *asymptotic notation* which highlights the bottleneck as the problem grows harder. For finding the maximum of n numbers, the one operation needed for initialization becomes insignificant if n is large. If n is equal to a million, say, what really matters are the (up to) two million operations needed in the body of the algorithm, not the one needed for initialization. So we would like to focus on the $2n$ part and not the $+1$ part. Even here, we can simplify further. The most important part is that it grows with n (and not n^2 , say): doubling n will also double the number of steps needed. The factor of 2 is less important.

To see why, imagine that someone proposes an alternative algorithm for finding the maximum of n numbers, and worst case analysis shows that it may require up to $n^2 + 4n + 5$ operations. As n grows large, the n^2 part will dominate the remaining terms. Furthermore, if n were to double, n^2 would increase by a factor of four. This is evidence that Algorithm 5 is a better choice. As n gets larger and larger, the number of operations needed by Algorithm 5 increases in direct proportion, whereas that needed by the hypothetical alternative grows with the square of n . The fact that the first algorithm is $2n$, as opposed to n is *much less important* than that it is linear in n and not quadratic.

Another reason to focus on the n and not the 2 is to abstract certain implementation details. In Algorithm 5, do we count an initialization step associated with the first line, setting the values in the array A ? Maybe it is passed directly as an address in computer memory, to values that already exist; in this case there is no new operation. On the other hand, if we have to assign them

from other values in memory, we need an additional n assignment operations, for a total of $3n + 1$. Or maybe we have to read them from a file, requiring n operations to read them from the drive and n assignment operations to store them in A , for a total of $4n + 1$. In all of these cases, the most important part is that the computational requirements scale proportionally to n , since this will tell you how the run time will change if you give it a larger or smaller problem instance. Whether the coefficient is 2, 3, or 4 is of secondary importance.

We will introduce an asymptotic notation that lets us simplify in this way; we want a concise way to express that as n grows large, the number of steps required by Algorithm 5 grows in direct proportion to n , without fiddling around with the constants in the full expression $2n + 1$. The notation will say that the number of steps required by Algorithm 5 is $O(n)$ (read “big-oh of n ”). Technically, we say that a function $f(n)$ is $O(g(n))$ if there are constants A and n_0 such that $|f(n)| \leq Ag(n)$ whenever $n > n_0$. Practically, we can identify such a function $g(n)$ by selecting the dominant term in $f(n)$ as $n \rightarrow \infty$, and removing any constant coefficient it has. For example, $n^2 + 4n + 5$ is $O(n^2)$, and $4 \cdot 2^n + 6n^3 - 8 \log n$ is $O(2^n)$. The initial version of the gambling algorithm (Algorithm 7) required at most 10 steps; we can say this is $O(1)$.

Some common asymptotic notations that arise are:

- $O(1)$, for algorithms that run in *constant time* (independent of input size);
- $O(\log n)$, sometimes called *logarithmic time* or *sublinear time* (the run time increases more slowly than the problem size);
- $O(n)$, or *linear time* (run time scales in direct proportion to the problem size);
- $O(n^2)$, or *quadratic time*;
- $O(n^3)$, $O(n^4)$, and other *polynomial time* algorithms of the form $O(n^c)$ for some constant c .
- $O(2^n)$, or *exponential time*;
- $O(n!)$, or *factorial time*.

These are listed in increasing order of difficulty. Generally, algorithms up to polynomial time are considered to be reasonably efficient. Algorithms with exponential time complexity are difficult to solve for large instances in reasonable amount of time.

Some people think that this classification is not very useful, because computers continue to get faster over time. However, the difference between complexity classes is absolutely critical, and will remain so for the foreseeable future. Imagine you have two algorithms for a network optimization problem with n nodes; one runs in quadratic time, $O(n^2)$, and the other runs in exponential time, $O(2^n)$. Also imagine that using either algorithm, on your current computer you can just barely solve a problem with 1000 nodes in an acceptable amount of

time. After some time, you upgrade your computer so that it is twice as powerful as the one you had before. Roughly speaking, the quadratic-time algorithm on this better machine can handle a network which is larger by a factor of $\sqrt{2}$, so roughly 1400 nodes. The exponential-time algorithm, on the other hand, can only handle a network with one more node, since 2^{1001} is twice as large as 2^{1000} . Faster hardware will not overcome the disadvantages of an exponential-time algorithm, at least for large problems.

However, just because an algorithm is exponential doesn't mean it is useless. For one, the $O(\cdot)$ notation is asymptotic, and tells you what will happen as n tends to infinity. An exponential time algorithm may be perfectly fine working with the problem sizes you are dealing with; it just means that it does not scale well, and for large enough problems you will run into difficulties. It is also worst case, and there are algorithms which may have exponential worst case complexity, yet behave more like polynomial time in practice. The simplex method for solving linear programs (Section C.2.3), and the branch and bound method for solving mixed-integer linear programs (Section C.5.1) are examples of algorithms which are exponential in the worst case, yet offer better performance in practice, especially when implemented well and tailored to a specific application.

For one more example of complexity calculation, Algorithm 8 presents the "bubble sort" algorithm for sorting an array of n numbers into increasing order. The algorithm has two loops: an outer loop using the iterator i , and an inner loop with the iterator j . When $i = 1$, there are $4(n - 1)$ operations associated with the j loop. When $i = 2$, there are $4(n - 2)$ operations, and so on. Therefore in total there are $4(n - 1) + 4(n - 2) + \dots + 1 = 2n(n - 1)$ operations. The bubble sort is therefore $O(n^2)$ in complexity, and has quadratic time. This implies that if the input size doubles, the running time quadruples. There are faster sorting methods with $O(n \log n)$ complexity; if you have to sort a large array of numbers this will perform much better than bubble sort. However, bubble sort is much easier to implement, and may perform acceptably if you only have to sort a small array.

Initialize:

A : Set, list, or array containing the n numbers

```

for  $i \leftarrow 1$  to  $n - 1$  do
  | for  $j \leftarrow 1$  to  $n - i$  do
  | | if  $A[j] > A[j + 1]$  then
  | | |  $temp \leftarrow A[j]$ 
  | | |  $A[j] \leftarrow A[j + 1]$ 
  | | |  $A[j + 1] \leftarrow temp$ 
  | | end
  | end
end

```

Algorithm 8: Bubble sort

D.4 Complexity Classes

The previous section introduced asymptotic notation, and how it is used to characterize the worst-case run time of an algorithm, measured by counting the number of basic operations needed. This is useful for comparing two algorithms for the same problem. As the input size of a problem becomes larger and larger, any $O(1)$ algorithm will eventually run faster than any $O(n)$ algorithm, which will run faster than any $O(n^2)$ algorithm, which will run faster than any $O(2^n)$ algorithm, and so forth. But can we say anything about how hard a *problem* is? In other words, I might prefer a $O(n^2)$ algorithm to an $O(2^n)$ one. But is $O(n^2)$ the best I can do? Or with some more ingenuity and work, could we come up with something even better?

Complexity classes aim to understand hardness in terms of the underlying problems. As explained in more detail below, one such class of problems is called P ; these are problems for which a polynomial-time algorithm exists to solve it. A larger class of problems is called NP ; for some of these problems, we do not know of any polynomial-time algorithm that will always solve them. Some NP problems are called NP -complete, which means that any polynomial-time algorithm solving them would *also* solve any other problem in NP . Such an algorithm would be extremely valuable, and decades of work has gone into trying to find one. This search has thus far proved fruitless.

This is the core of the famous $P \stackrel{?}{=} NP$ question, arguably the most important open problem in theoretical computer science, and one carrying a \$1 million bounty from the Clay Mathematics Institute. Answering this question either way (providing such an algorithm, or definitively proving that none exists) would be a tremendous accomplishment. Indeed, even though this core question has as yet gone unanswered, the search for its answer has yielded many other important findings in computer science. Aaronson (2016) provides an excellent survey of the history of this question, attempts to answer it, and the value this search has provided

Many students learning about complexity classes initially find them an esoteric theoretical conceit. However, there is great practical significance! If your supervisor asks you to solve a particular problem, they may not be happy if the best you can come up with is an exponential time algorithm, since it will scale badly for large-scale problems. However, if you can show this problem is NP -complete, then you can point out that it's not just you who can't find a better algorithm — the brightest minds in computer science have also been unable to find a better algorithm for this problem! In other words, it's not you, it's the problem that's inherently difficult. If you can suggest a simplification of the problem that brings it into P , then this directly suggests a faster way to solve it.

This section provides a brief overview of complexity classes, emphasizing applications to transportation network optimization problems.

D.4.1 Decision problems vs. optimization

In computer science, complexity classes are usually defined in terms of *decision problems*, where the answer is either “yes” or “no.” This book primarily discusses *optimization problems*, where the answer involves a solution in terms of decision variables and an objective function. However, optimization problems can be transformed into decision problems. Here are a few examples of how to do this:

Shortest path problem: Consider a directed network $G = (N, A)$, where N is the set of nodes, and A the set of directed links, with non-negative integer costs c_{ij} for each $(i, j) \in A$. Let r and s denote the origin and destination node.

Optimization problem: Find the path π between r and s minimizing its cost $c^\pi = \sum_{(i,j) \in \pi} c_{ij}$.

Decision problem: For a given value C , is there a path π between r and s whose cost c^π does not exceed C ?

Traveling salesperson: Consider a complete network $G = (N, A)$, where N is the set of nodes and A the set of links, with integer costs c_{ij} .

Optimization problem: Determine a minimum cost tour which visits each node exactly once. (In other words, find a permutation of nodes (p_1, p_2, \dots, p_n) which minimizes $\sum_{i=1}^{n-1} c_{p_i p_{i+1}} + c_{p_n p_1}$.)

Decision problem: For a given value C , is there a tour visiting each node exactly once whose cost does not exceed C ? (In other words, is there a permutation of nodes such that $\sum_{i=1}^{n-1} c_{p_i p_{i+1}} + c_{p_n p_1} \leq C$?)

Knapsack problem: Consider a set of objects I with positive integer sizes a_i and profits p_i for each $i \in I$, and a positive integer U denoting the capacity.

Optimization problem: Find a subset of objects $J_I \subset I$ maximizing total profit $\sum_{j \in J_I} p_j$ such that $\sum_{j \in J_I} a_j \leq U$.

Decision problem: For a given profit level P , is there a set of objects $J_I \subset I$ such that $\sum_{j \in J_I} p_j \geq P$ and $\sum_{j \in J_I} a_j \leq U$.

Resource-constrained shortest path problem: Consider a directed network $G = (N, A)$, where N is the set of nodes, and A the set of directed links, with positive integer costs c_{ij} and resources r_{ij} for each $(i, j) \in A$. Let r and s denote the origin and destination node, and R the resource budget.

Optimization problem: Find the path π between origin node s and destination node t with minimum cost $c_\pi = \sum_{(i,j) \in \pi} c_{ij}$, subject to the resource usage constraint $\sum_{(i,j) \in \pi} r_{ij} \leq R$.

Decision version: For a given value C , is there a path π between s and t such that $c_\pi \leq C$ and $\sum_{(i,j) \in \pi} r_{ij} \leq R$?

Longest acyclic path problem: Consider a directed network $G = (N, A)$, where N is the set of nodes, and A the set of directed links, with positive integer costs c_{ij} for each $(i, j) \in A$. Let r and s denote the origin and destination node.

Optimization problem: Find the acyclic path π between r and s with maximal cost $c^\pi = \sum_{(i,j) \in \pi} c_{ij}$.

Decision problem: For a given value C , is there an acyclic path π between r and s whose cost c^π is at least C ?

From the standpoint of solution, a polynomial time algorithm for the optimization problem also leads to a polynomial time algorithm for the decision problem, and vice versa. For example, if we have a polynomial time algorithm for the optimization version of the shortest path problem, we can use it to solve the decision problem: just find the shortest path using the optimization version; if its cost is less than or equal to C , then the answer to the decision problem is “yes.” Otherwise it is “no.” Or, if we have a polynomial time algorithm for the decision version of the problem and want to find the exact cost of the shortest path (not just whether it is less than C or not), we can repeatedly solve the decision version problem for different versions of C using a bisection-like approach (cf. Section 3.3.2). This requires solving the decision problem multiple times, but a polynomial number of times; substituting one polynomial into another still results in a polynomial, so the overall method is still polynomial time (even if the polynomial has a higher degree).

D.4.2 The classes P , NP , NP -hard, and NP -complete

This subsection introduces four classes of problems that arise often in optimization and elsewhere in computer science: P , NP , NP -hard, and NP -complete. They are formally defined in terms of decision problems like the ones in the previous subsection, where the answer to a posed question is either “yes” or “no.” These four classes are not mutually exclusive, so a given problem may exist in more than one of these classes at the same time — for example, the standard shortest path problem is in both P and NP . They are also not exhaustive; there are decision problems which do not lie in any of these four classes.

We start by defining the classes P (“polynomial”) and NP (“nondeterministic polynomial,” and *not* “non-polynomial” as is often guessed by people seeing the acronym for the first time). The definitions given here are not fully rigorous, since we should also state exactly what our model of computation is, what we count as a basic operation, the exact format of the problem input, and so on. Readers interested in these details should consult Sipser (2012) or another standard text in the theory of computation. The definitions below will suffice if you are using common hardware, programming languages, and data formats.

Definition D.1. *A decision problem belongs to the class P if, given an instance of size n , the answer can be obtained using an algorithm whose worst-case running time is polynomial, that is, $O(n^k)$ for some constant k .*

Definition D.2. *A decision problem belongs to the class NP if, given an instance of size n whose answer is “yes,” there is a polynomial-time algorithm to confirm that the answer is in fact “yes.”*

Worded intuitively, a problem in P is easy to solve; a problem in NP is easy to *check*, in the sense that someone claiming the answer is “yes” can provide

evidence that you can verify in polynomial time. A few examples are shown below; keep in mind that all of these refer to the *decision* versions of these problems described in the previous subsection, not the optimization versions.

The decision version of the shortest path problem with non-negative link costs is in P , since Dijkstra's algorithm (and others) can find the shortest path in polynomial time. So we simply compute the shortest path; if its cost is less than or equal to C , the answer to the decision problem is "yes" (there is a path of cost at most C), otherwise the answer is "no" (no such path exists).

The decision version of the traveling salesperson problem is in NP . If the answer is "yes" (there is a tour visiting each node exactly once whose cost is less than or equal to C), someone can convince you of this by providing the tour itself as evidence. You can check in polynomial time whether this tour is valid (it visits every node once and only once) and whether its total cost does not exceed C . We have said nothing about how someone might find such a tour in the first place; simply that given such a tour as supposed evidence that the answer is "yes," you can check this evidence in polynomial time. Likewise, we have not said anything about "no" instances of the problem, in which all tours have a cost greater than C . Indeed, there does not seem to be an easy way to demonstrate that a given instance of the traveling salesperson problem is "no." Just providing a tour whose cost is more than C does not prove that there isn't some other tour with a lower cost. But this doesn't matter; the class NP is only concerned with "yes" instances of decision problems.⁴

Similarly, the decision versions for the knapsack problem, longest path problem, and resource-constrained shortest path problems are in NP , since in any case where the answer is "yes" you can provide a feasible knapsack assignment or path as easily-checked evidence that the answer really is "yes."

Every decision problem in P is also in NP . Given a "yes" instance of the problem, I can check whether the answer is indeed "yes" by simply answering the question myself using a polynomial-time algorithm (which exists because the problem is in P). So the decision version of the shortest path problem is in both P and NP .

We do not know if the reverse is true, that every problem in NP is also in P . For instance, we know that the traveling salesperson problem is in NP , but we do not know if it is in P or not. We do not currently know of any polynomial time algorithm for solving it, but this does not mean that none exists. Perhaps there is such an algorithm, but if so nobody has been clever enough to find it yet. Most computer scientists believe that no such algorithm exists, but nobody has been able to prove that either. Answering the question either way would be a tremendous advance in computer science.

The reason is that algorithms used to solve one kind of problem can often be used to solve other kinds of problems as well. Many important problems (even problems with no obvious "network" in them) can be reduced to the traveling salesperson problem in the sense that an algorithm for the latter can also be used to solve the former, by constructing a network, choosing costs in the right way,

⁴The complexity class $\text{co-}NP$ is the analogous class for "no" instances.

and then translating the resulting tour back to the original problem. So, finding a good way to solve the traveling salesperson problem would directly provide good ways to solve many other problems as well. This idea is formalized in the following definition:

Definition D.3. *A decision problem A can be polynomially reduced to decision problem B if (a) there is a procedure that can transform an instance of A to an instance of B in polynomial time; and (b) the answer for an instance of A is “yes” if and only if the answer for the corresponding instance of “ B ” under this transformation is also “yes.”*

The implication is that if we know a polynomial time algorithm for B , then we can use it to solve A . The number of steps needed to do this is the sum of the number of steps needed to construct an instance of problem B from the given instance of A , and the number of steps needed to answer that instance of B . If there is a polynomial reduction from A to B , and if B is solvable in polynomial time, then the total number of steps required is also polynomial. Intuitively, the problem A is no harder than B , since any method for B can also be used to solve A . (The problem A could be easier than B , in the sense that a specialized method for A might be a faster approach than translating the problem to B and solving that; we just know it can’t be harder.)

We give a few examples of these kinds of transformations for decision problems in the next subsection. The main reason for introducing polynomial reductions is to define two additional complexity classes: NP -hard, and NP -complete.

Definition D.4. *A decision problem X belongs to the class NP -hard if every problem in the class NP has a polynomial reduction to X .*

In other words, any problem in class NP -hard is at least as difficult to solve as any problem in NP , since any problem in class NP can be converted to an NP -hard problem. NP -hard problems are not necessarily in NP ; they might be strictly more difficult in the sense that we might not be able to verify a “yes” answer to an NP -hard problem in polynomial time.

Definition D.5. *A decision problem belongs to the class NP -complete if it is both in classes NP and NP -hard.*

The problems in NP -complete can be thought of as the most difficult problems in NP , because any other NP problem can be converted to an NP -complete problem which will have the same answer. The consequence of this is that if someone can develop a polynomial time algorithm for *any* problem in NP -complete, then we can answer all problems in NP in polynomial time as well; therefore the classes P and NP will be the same: $P = NP$. On the other hand, proving that no polynomial time algorithm exists to answer an NP -complete problem would show that there are some problems in NP which are not in P , so $P \neq NP$. This makes the class NP -complete very important. The decision version of the traveling salesperson problem is an example of an

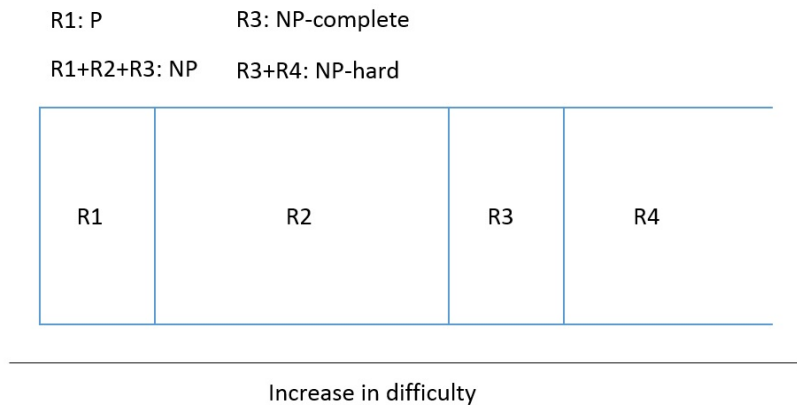


Figure D.1: P vs NP vs NP-complete vs NP-hard

NP-complete problem. The proof of this statement is difficult and beyond the scope of this book.

We summarize the discussion in this subsection in Figure D.1, which illustrates the relationships between the four complexity classes discussed. The left panel shows how the classes overlap if $P \neq NP$; the right panel shows what it would look like if $P = NP$. Again, determining whether P and NP are in fact equal is perhaps the most important open question in computer science. Most experts believe that $P \neq NP$ is the likelier answer to this question, but this is not a universally-held opinion, and in any case belief is not proof (there are plenty of true mathematical results whose discovery came as a surprise to expert mathematicians).

As a final word, we comment on some differences between decision problems (the kind used to formally define the complexity classes in this section) and optimization problems (the kind arising more when doing transportation network analysis in practice). For instance, the decision version of the traveling salesperson problem is in *NP* (a given tour can be verified to be feasible and have cost less than C in polynomial time), but the optimization version is not (there is no easy way to prove that a given tour is in fact minimal across all feasible tours). Sometimes, researchers in operations research use this terminology a bit loosely, and refer to a problem as being “*NP*-complete” (or similarly) without explicitly saying whether they are referring to the decision or optimization versions of these problems. This is unfortunate, and if you are conducting research involving the complexity of transportation network problems we recommend that you become familiar with the distinction between the two.

D.4.3 Examples demonstrating NP -completeness

One of the primary uses of complexity classes is to establish that a given problem is *intrinsically* hard. In particular, if you can show a problem is NP -complete, then expecting to develop a polynomial time algorithm is more than simply an ambitious goal; it would be a groundbreaking advance in computer science. For practical application, we are therefore content with heuristics that provide reasonably good solutions in a reasonable amount of time, rather than exact methods, since we do not know of any that scale well with problem size (at least in the worst case). This subsection provides some examples of how you can show a decision problem is NP -complete, using the idea of polynomial reductions introduced above.

The typical method for showing a decision problem X is NP -complete has two steps:

- Showing that X is in NP . That is, given an instance of X where the answer is “yes,” it is possible to demonstrate this in polynomial time.
- Providing a polynomial reduction from *any other* NP -complete problem to X . This shows that X is at least as difficult as any problem in NP (and therefore NP -hard); together with the proof that X is also in NP , this establishes NP -completeness.

The second of these is more challenging and can require more creativity than the first. Notice that this method requires that we already know some problems are NP -complete. Currently, hundreds of decision problems are known to be NP -complete, and you can use any of them as the starting point for the second step in a proof of NP -completeness. (The very first proof of NP -completeness for a decision problem was much harder, since the above scheme could not be followed. Instead, one had to give a general argument that any NP problem could be reduced to that problem!) Garey and Johnson (1979) is a good reference listing a variety of problems known to be NP -complete.

For example, the following decision problems are known to be NP -complete:

Hamiltonian cycle problem: Given a directed network $G = (N, A)$, is there a cycle which visits every node exactly once?

Partition problem: Given a set S of positive integers (possibly with multiple copies of each number), can the set be partitioned into two subsets S_1 and S_2 such that the sum of the integers in S_1 equals the sum of the integers in S_2 ?

Using these, we can prove that the traveling salesperson problem, knapsack problem, and resource-constrained shortest path decision problems introduced above are all NP -complete. Let us look at the NP -completeness proof of the decision problems discussed above.

Example D.1. *Show that the decision version of the traveling salesperson problem (given in Section D.4.1) is NP -complete.*

Solution. The first step is to show that this problem is in NP ; we have already shown in the previous section that we can check that a tour is valid, and that its total cost is at most C , in linear time, so we can verify a “yes” instance in polynomial (in fact, linear) time.

The second step involves a polynomial reduction from a known NP -complete problem. In this case, we select the Hamiltonian cycle problem, and produce a reduction that would allow us to answer the Hamiltonian cycle problem by solving a suitable traveling salesperson problem. (The order of the reduction is critical; showing that we could answer the traveling salesperson problem by solving a suitable Hamiltonian cycle problem would not be helpful. That would show that the Hamiltonian cycle problem is at least as hard as the traveling salesperson problem; but we already know that the Hamiltonian cycle problem is NP -complete. Our goal is to say something about the traveling salesperson problem, so we have to proceed in the other direction and show that we can use it to answer a Hamiltonian cycle question.)

So, assume we are given an instance of the Hamiltonian cycle problem with a network $G = N(A, A)$, and let $n = |N|$ denote the number of nodes in the network. A Hamiltonian cycle visits every node exactly once before returning to the starting node, so such a cycle will contain n links. Construct a new network $G' = (N', A')$ in the following manner: the node sets are identical ($N' = N$), but G' is a complete graph, meaning there is a link (i, j) between *every* pair of nodes. The costs in G' are chosen in the following way: $c_{ij} = 1$ if $(i, j) \in A$, and $c_{ij} = 2$ if $(i, j) \notin A$. That is, if the link exists in the original network, it has a cost of 1; if the link did not, it has a cost of 2. This transformation can be accomplished in polynomial time, since the number of links in G' is $n(n - 1)$.

Now, consider the decision version of the traveling salesperson problem in G' , with $C = n$. If the answer to this problem is “yes,” then there is a tour visiting every node and involving exactly n links, all with a cost of 1. All of these links therefore exist in the original network G , and correspond to a Hamiltonian cycle, so the answer to the Hamiltonian cycle problem is also “yes.”

Or, if the answer to the traveling salesperson problem is “no,” then every tour in G' has at least one link with a cost of 2. No such links exist in the original network G , so there is no Hamiltonian cycle either: the answer to that problem is also “no.”

This completes the reduction; we have already shown that it can be accomplished in polynomial time, and that the traveling salesperson problem is in NP . Therefore it is NP -complete. ■

Essentially, we showed that any procedure which can solve the traveling salesperson problem could *also* solve the Hamiltonian path problem, and therefore it cannot be any easier.

Example D.2. Show that the decision version of the knapsack problem (given in Section D.4.1) is NP -complete.

Solution. We start by showing that a “yes” instance can be verified in polynomial time to establish that the problem is in NP . If the answer is “yes,” one can prove this by furnishing a subset J_I whose total profit is at least P ,

and whose total size is no greater than U . The total profit and total size can be calculated in linear time by simply summing the profit and size of each object in J_I .

The remainder of the argument establishes a polynomial reduction from an NP -complete problem. For this example, we will reduce from the partition decision problem, with a given set of positive integers S . From this, we will construct an instance of the knapsack decision problem whose answer (“yes” or “no”) will always be the same as that for the given partition problem. Let $S = \{b_1, b_2, \dots, b_n\}$ (possibly with repetitions). Create an instance of the knapsack problem where the set of objects is the set of integers $I = \{1, 2, \dots, |S|\}$. The size of each object, and its profit, are both set equal to the corresponding integer in S , so $a_i = p_i = b_i$. The available capacity is half of the sum of the integers in S , so $U = \frac{1}{2} \sum_{i \in S} b_i$. The minimum profit P is set to this same amount as well: $P = \frac{1}{2} \sum_{i \in S} b_i$. This reduction can be done in linear time.

Assume that the answer to the knapsack problem is “yes.” Then there is a subset of objects J_I such that $\sum_{j \in J_I} a_j \leq U$ and $\sum_{j \in J_I} p_j \geq P$. But $U = P = \frac{1}{2} \sum_{i \in S} b_i$, so this implies $\sum_{j \in J_I} a_j = \frac{1}{2} \sum_{i \in S} b_i$. Furthermore, each $a_j = b_j$, so $\sum_{j \in J_I} b_j = \frac{1}{2} \sum_{i \in S} b_i$. That is, the integers represented by J_I form exactly half of the total sum of the integers in S . Therefore, the remaining integers not in J_I must also form exactly half the total sum of the integers in S , so the sets J_I and $I - J_I$ have equal sums, and the answer to the partition problem is “yes.”

Now assume that the answer to the knapsack problem is “no.” Then every subset of objects J_I either has $\sum_{j \in J_I} a_j > U$ (total size exceeds capacity) or $\sum_{j \in J_I} p_j < P$ (total profit falls short of target). Since $U = P = \frac{1}{2} \sum_{i \in S} b_i$ and $a_i = p_i = b_i$, this means that no subset J_I has the property that its integers add up to exactly half of the total of the integers in S . This means that the answer to the partition problem must also be “no,” since if there were sets S_1 and S_2 whose sums were equal, they would both have to equal $\frac{1}{2} \sum_S b_i$, and $J_I = S_1$ would be a solution to the knapsack problem.

We can therefore answer the partition problem (known to be NP -complete) by solving the knapsack problem, so the knapsack problem must be at least as hard. Since we have shown that the knapsack problem is in NP , it is therefore NP -complete as well. ■

Example D.3. *Show that the decision version of the resource-constrained shortest path problem (given in Section D.4.1) is NP -complete.*

Solution. This problem is in NP , since a “yes” instance can be verified in linear time once an appropriate path is provided as evidence; simply check that the path is valid, and compute two sums over its links to verify that its total cost is at most C , and that its total resource consumption is at most R .

To show NP -completeness, we reduce the partition decision problem to the resource-constrained shortest path problem. Given the set $S = \{b_1, b_2, \dots, b_n\}$, we construct a network $G = (N, A)$ with node and link sets defined as follows. Let $U = \sum_{b_i \in S} b_i$, and $V = 2nU$. We create $n + 1$ nodes; and between each

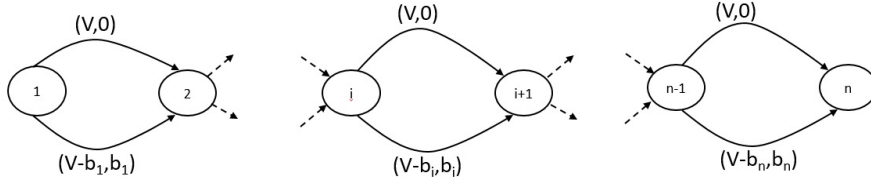


Figure D.2: Network transformation to reduce the partition problem to resource-constrained shortest paths.

consecutive pair of nodes i and $i + 1$, we create two parallel links. In each pair of parallel links, the top link has a cost of V , and a resource consumption of 0. The bottom link has a cost of $V - b_i$, and a resource consumption of b_i . (See Figure D.2.) Finally, for this instance of the resource-constrained shortest path decision problem, set the cost target $C = nV - (U/2)$, the resource limit to $U/2$, and select nodes 1 and $n + 1$ as the origin and destination. The number of links and nodes is linear in the size of the original set S , so this transformation can be done in polynomial time.

Assume that the resulting resource-constrained shortest path problem is a “yes” instance. Let π be a path whose cost is at most C , and whose resource consumption is at most R . This path consists of n links, each of which is either the top or bottom link in the n pairs of parallel links in the network. Let S_1 contain the values of S corresponding to “bottom” links, and S_2 the values corresponding to “top” links. Then the total cost of this path is $nV - \sum_{b_i \in S_1} b_i$, and its total resource consumption is $\sum_{b_i \in S_1} b_i$. Since this is a “yes” instance, the cost target is achieved and $nV - \sum_{b_i \in S_1} b_i \leq C = nV - (U/2)$, so $\sum_{b_i \in S_1} b_i \geq U/2$. Likewise, the resource constraint is satisfied, so $\sum_{b_i \in S_1} b_i \leq R = U/2$. Therefore $\sum_{b_i \in S_1} b_i$ is exactly $U/2$, that is, the sum of the entries in set S_1 is exactly half the total of the sum of the entries in S . We can thus conclude that the sum of the values in sets S_1 and S_2 are equal, and this is also a “yes” instance of the partition problem.

Or, if this is a “no” instance of the resource-constrained shortest path problem, then every path either exceeds the cost target or the resource limitation. Following the same construction as in the previous paragraph, this means that for every set S_1 we have either $\sum_{b_i \in S_1} b_i < U/2$ or $\sum_{b_i \in S_1} b_i > U/2$. Therefore, there is no subset of the values in S whose sum is exactly half of the total sum of the values in S ; so it is not possible to partition S into two sets S_1 and S_2 whose sums are exactly equal. Therefore this is a “no” instance of the partition problem as well. ■

You may have noticed that this proof was fairly similar to that used to show that the knapsack problem is NP -complete. In fact, once Example D.2 was done, and we established that the knapsack problem was NP -complete, we could have done Example D.3 in a faster way by reducing the knapsack problem to the resource-constrained shortest path problem (rather than starting from the

partition problem anew). Verifying this would be a very instructive exercise.

Example D.4. *Show that the decision version of the longest acyclic path problem (given in Section D.4.1) is NP-complete.*

Solution. For a “yes” instance, verifying that a given path is feasible and has cost at least C can be done in linear time, so this problem is in *NP*.

To show *NP*-completeness, we reduce the Hamiltonian cycle problem to the longest acyclic path problem. The transformation proceeds as follows: let $G = (N, A)$ be the given network for the Hamiltonian cycle problem (where n is the number of nodes). Create a network G' with $n + 1$ nodes in the following way: select any node i in N , and split it into two nodes i^+ and i^- . Any link (i, j) in the forward star of node i appears in G' as a link from i^+ to j . Any link (h, i) in the reverse star of node i appears in G' as a link from h to i^- . All other links in G appear in G' as they are, connecting the same nodes. Select i^+ as the origin, and i^- as the destination, and finally select the cost threshold $C = n$ to create an instance of the longest acyclic path problem in G' .

If this is a “yes” instance for the longest acyclic path problem, then there is a path π whose cost is at least n . Since the path is acyclic, it cannot revisit any node; and since there are $n + 1$ nodes in G' the path π must pass through all of them. This acyclic path can be translated to a Hamiltonian cycle on G by replacing i^+ and i^- in π by the single node i .

If this is a “no” instance for the longest acyclic path problem, then every path π in G' has cost less than n , meaning it does not pass through every node. In this case, the original Hamiltonian cycle problem must also be a “no” instance, since any Hamiltonian cycle can be translated to a path in G' from i^+ to i^- passing through every node. ■

Two final comments on this subject. First, pay attention to the problem specifics. Minor changes can have a significant impact on the complexity of a problem. For example, the Hamiltonian cycle problem (determining whether there is a path visiting each *node* exactly once) is *NP*-complete; but the Eulerian cycle problem (is there a path visiting each *link* exactly once) can actually be solved in linear time. The shortest path problem (with positive link costs) can be solved in polynomial time; the shortest path problem with negative link costs and the longest (acyclic) path problem are both *NP*-complete.

Second, just because a problem is *NP*-hard or *NP*-complete does not mean the problems are impossible to solve, or completely impractical. Such problems are often solved either using heuristics (which are inexact, but may be good enough), exact methods which are usually fast enough in practice (even if in the worst case the number of steps is exponential), or for small problem instances where an exponential time algorithm may still be acceptable. As some concrete examples, integer optimization problems (Section C.5) are *NP*-hard. But several algorithms have been developed that often solve real world instances efficiently. Knapsack problems can be solved by pseudo-polynomial algorithms, a complexity class where the number of operations is bounded by the *value* of the input, and not its absolute size. These problems are also commonly solvable in practice.

Bibliography

- Aaronson, S. (2016). $P \stackrel{?}{=} NP$. In *Open Problems in Mathematics*, pp. 1–122. Springer.
- Aashtiani, H. Z. and T. L. Magnanti (1981). Equilibria on a congested transportation network. *SIAM Journal on Algebraic and Discrete Models* 2, 213–226.
- Ahuja, R. K., T. Magnanti, and J. Orlin (1993). *Network Flows*. Englewood Cliffs, NJ: Prentice-Hall.
- Akamatsu, T. (1996). Cyclic flows, Markov process and stochastic traffic assignment. *Transportation Research Part B* 30, 369–386.
- Akamatsu, T. (1997). Decomposition of path choice entropy in general transport networks. *Transportation Science* 31, 349–362.
- Andreatta, G. and L. Romeo (1988). Stochastic shortest paths with recourse. *Networks* 18, 193–204.
- Ban, X. J., J.-S. Pang, H. X. Liu, and R. Ma (2012). Modeling and solving continuous-time instantaneous dynamic user equilibria: A differential complementarity systems approach. *Transportation Research Part B* 46(3), 389–408.
- Bar-Gera, H. (2002). Origin-based algorithm for the traffic assignment problem. *Transportation Science* 36(4), 398–417.
- Bar-Gera, H. (2005). Continuous and discrete trajectory models for dynamic traffic assignment. *Networks and Spatial Economics* 5, 41–70.
- Bar-Gera, H. (2006). Primal method for determining the most likely route flows in large road networks. *Transportation Science* 40(3), 269–286.
- Bar-Gera, H. (2010). Traffic assignment by paired alternative segments. *Transportation Research Part B* 44(8–9), 1022–1046.
- Bar-Gera, H. and D. Boyce (1999). Route flow entropy maximization in origin-based traffic assignment. In *Proceedings of the 14th International Symposium on Transportation and Traffic Theory*, Jerusalem, Israel, pp. 397–415.

- Barcelo, J. (1998). Parallelization of microscopic traffic simulation for ATT systems analysis. In P. Marcotte and S. Nguyen (Eds.), *Equilibrium and Advanced Transportation Modelling*. Kluwer Academic Publishers.
- Bazaraa, M. S., H. D. Sherali, and C. M. Shetty (2006). *Nonlinear Programming: Theory and Algorithms*. Hoboken, NJ: Wiley-Interscience.
- Beckmann, M. J., C. B. McGuire, and C. B. Winston (1956). *Studies in the Economics of Transportation*. New Haven, CT: Yale University Press.
- Bell, M. G. H. (1995). Alternatives to Dial's logit assignment algorithm. *Transportation Research Part B* 29B, 287–295.
- Bell, M. G. H. and Y. Iida (1997). *Transportation Network Analysis*. West Sussex, United Kingdom: John Wiley & Sons.
- Bellei, G., G. Gentile, and N. Papola (2005). A within-day dynamic traffic assignment model for urban road networks. *Transportation Research Part B* 39, 1–29.
- Bellman, R. (1957). *Dynamic Programming*. Princeton, NJ: Princeton University Press.
- Bellman, R. (1958). On a routing problem. *Quarterly of Applied Mathematics* 16(1), 87–90.
- Ben-Akiva, M., M. Bierlaire, H. Koutsopoulos, and R. Mishalani (1998). DynaMIT: a simulation-based system for traffic prediction. In *DACCORS Short Term Forecasting Workshop, The Netherlands*.
- Bertrand, J. (1883). Théorie mathématique de la richesse sociale. *Journal des Savants* 67, 499–508.
- Bertsekas, D. P. (2003). *Convex Analysis and Optimization*. Cambridge, MA: Athena Scientific.
- Bertsekas, D. P. (2016). *Nonlinear Programming*. Cambridge, MA: Athena Scientific.
- Bertsekas, D. P., E. M. Gafni, and R. G. Gallager (1984). Second derivative algorithms for minimum delay distributed routing in networks. In *IEEE Transactions on Communication, COM-32*, pp. 911–919.
- Bertsimas, D. and J. N. Tsitsiklis (1997). *Introduction to Linear Optimization*. Cambridge, MA: Athena Scientific.
- Bhat, C. R. and F. S. Koppelman (1999). Activity-based modeling of travel demand. In R. W. Hall (Ed.), *Handbook of Transportation Science*, Volume 23 of *International Series in Operations Research & Management Science*, pp. 35–61. Boston, MA: Springer.

- Birge, J. R. and F. Louveaux (1997). *Introduction to Stochastic Programming*. New York, NY: Springer Verlag.
- Boyce, D., B. Ralevic-Dekic, and H. Bar-Gera (2004). Convergence of traffic assignments: how much is enough? *Journal of Transportation Engineering* 130(1), 49–55.
- Boyles, S. D. (2006). Reliable routing with recourse in stochastic, time-dependent transportation networks. Master’s thesis, The University of Texas at Austin.
- Boyles, S. D. (2009). *Operational, Supply-Side Uncertainty in Transportation Networks: Causes, Effects, and Mitigation Strategies*. Ph. D. thesis, The University of Texas at Austin.
- Boyles, S. D. (2012). Bush-based sensitivity analysis for approximating subnetwork diversion. *Transportation Research Part B* 46, 139–155.
- Boyles, S. D. (2013). Statewide mesoscopic simulation for Wyoming. Technical Report FHWA-WY-13/05F, Wyoming Department of Transportation.
- Boyles, S. D., C. L. Melson, J. C. Duthie, and T. Rambha (2013). Diverge models and dynamic traffic equilibria. Presented at the Annual Meeting of the Institute for Operations Research and the Management Sciences, Minneapolis, MN.
- Boyles, S. D. and T. Rambha (2016). A note on detecting unbounded instances of the online shortest path problem. *Networks* 31, 270–276.
- Boyles, S. D., T. Rambha, and C. Xie (2015). Equilibrium analysis of low-conflict network designs. *Transportation Research Record* 2467, 129–139.
- Boyles, S. D. and N. Ruiz Juri (2019). Queue spillback and demand uncertainty in dynamic network loading. *Transportation Research Record* 2673, 38–48.
- Braess, D. (1969). Über ein Paradoxon aus der Verkehrsplanung. *Unternehmensforschung* 12, 258–268.
- Brouwer, L. E. J. (1910). Über eineindeutige, stetige transformationen von Flächen in sich. *Mathematische Annalen* 69, 176–180.
- Carey, M. (1992). Nonconvexity of the dynamic assignment problem. *Transportation Research Part B* 26(2), 127–133.
- Cayley, A. On the theory of the analytical forms called trees. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 13(85), 172–176.
- Chabini, I. (1999). Discrete dynamic shortest path problems in transportation applications: complexity and algorithms with optimal run time. *Transportation Research Record* 1645, 170–175.

- Chen, H.-K. and C. F. Hsueh (1998). A model and an algorithm for the dynamic user-optimal route choice problem. *Transportation Research Part B* 32B, 219–234.
- Chiu, Y.-C., J. Bottom, M. Mahut, A. Paz, R. Balakrishna, T. Waller, and J. Hicks (2010). A Primer for Dynamic Traffic Assignment. Prepared by the Transportation Network Modeling Committee of the Transportation Research Board (ADB30).
- Cho, H.-J., T. E. Smith, and T. L. Friesz (2000). A reduction method for local sensitivity analyses of network equilibrium arc flow. *Transportation Research Part B* 34(1), 31–51.
- Claudel, C. G. and A. M. Bayen (2010). Lax-Hopf based incorporation of internal boundary conditions into Hamilton-Jacobi equation. part I: theory. *IEEE Transactions on Automatic Control* 55, 1142–1157.
- Cooke, L. and E. Halsey (1969). The shortest route through a network with time-dependent internodal transit times. *Journal of Mathematical Analysis and Applications* 14, 492–498.
- Correa, J. R., A. S. Schulz, and N. E. Stier-Moses (2004). Selfish routing in capacitated networks. *Mathematics of Operations Research* 29, 961–976.
- Corthout, R., G. Flötteröd, F. Viti, and C. M. J. Tampère (2012). Non-unique flows in macroscopic first-order intersection models. *Transportation Research Part B* 46, 343–359.
- Cournot, A. (1838). *Recherches sur les Principes Mathematiques de la Theorie des Richesses*. Paris, France: L. Hachette.
- Dafermos, S. C. (1980). Traffic equilibrium and variational inequalities. *Transportation Science* 14, 42–54.
- Dafermos, S. C. (1982). Relaxation algorithms for the general asymmetric traffic equilibrium problem. *Transportation Science* 16, 231–240.
- Daganzo, C. F. (1982). Unconstrained extremal formulation of some transportation equilibrium problems. *Transportation Science* 16, 332–360.
- Daganzo, C. F. (1994). The cell transmission model: a dynamic representation of highway traffic consistent with the hydrodynamic theory. *Transportation Research Part B* 28B(4), 269–287.
- Daganzo, C. F. (1995a). The cell transmission model, part II: network traffic. *Transportation Research Part B* 29B(2), 79–93.
- Daganzo, C. F. (1995b). A finite difference approximation of the kinematic wave model of traffic flow. *Transportation Research Part B* 29B, 261–276.

- Daganzo, C. F. (1998). Queue spillbacks in transportation networks with a route choice. *Transportation Science* 32(1), 3–11.
- Daganzo, C. F. (2005a). A variational formulation of kinematic waves: basic theory and complex boundary conditions. *Transportation Research Part B* 39(2), 187–196.
- Daganzo, C. F. (2005b). A variational formulation of kinematic waves: solution methods. *Transportation Research Part B* 39(10), 934–950.
- Daganzo, C. F. and Y. Sheffi (1977). On stochastic models of traffic assignment. *Transportation Science* 11(3), 253–274.
- Dantzig, G. (1963). *Linear Programming and Extensions*. Princeton, NJ: Princeton University Press.
- Deo, N. and C. Pang (1984). Shortest path algorithms: taxonomy and annotation. *Networks* 14, 275–323.
- Di, X., H. X. Liu, J.-S. Pang, and X. Ban (2013). Boundedly rational user equilibria (BRUE): Mathematical formulation and solution sets. *Transportation Research Part B* 57, 300–313.
- Dial, R. B. (1971). A probabilistic multipath traffic assignment model which obviates path enumeration. *Transportation Research* 5, 83–111.
- Dial, R. B. (2006). A path-based user-equilibrium traffic assignment algorithm that obviates path storage and enumeration. *Transportation Research Part B* 40(10), 917–936.
- Diestel, R. (2016). *Graph Theory*. Heidelberg, Germany: Springer-Verlag.
- Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik* 1, 269–271.
- Domencich, T. A. and D. McFadden (1975). *Urban Travel Demand: A Behavioral Analysis*. Oxford, United Kingdom: North-Holland. Vol. 93 of Contributions to Economic Analysis.
- Dorigo, M. and T. Stützle (2004). *Ant Colony Optimization*. Cambridge, MA: MIT Press.
- Dowling, R., W. Kittelson, J. Zegeer, and A. Skabardonis (1997). Planning techniques to estimate speeds and service volumes for planning applications. Technical Report 387, National Cooperative Highway Research Program, Transportation Research Board.
- Downs, A. (1962). The law of peak-hour expressway congestion. *Traffic Quarterly* 16, 393–409.

- Dreyfus, S. (1969). An appraisal of some shortest-path algorithms. *Operations Research* 17, 395–412.
- Edgeworth, F. (1897). La teoria pura del monopolio. *Giornale degli Economisti* 15, 13–31.
- Eiger, A., P. B. Mirchandani, and H. Soroush (1985). Path preferences and optimal paths in probabilistic networks. *Transportation Science* 19(1), 75–84.
- Euler, L. (1736). Solutio problematis ad geometriam situs pertinentis. *Commentarii Academiae Scientiarum Imperialis Petropolitanae* 8, 128–140.
- Evans, L. C. (1998). *Partial Differential Equations*. Providence, RI: Graduate Studies in Mathematics, American Mathematical Society.
- Evans, S. P. (1976). Derivation and analysis of some models for combining trip distribution and assignment. *Transportation Research* 10(1), 37–57.
- Facchinei, F. and J. S. Pang (2003). *Finite-dimensional variational inequalities and complementarity problems*. Springer Verlag.
- Fan, Y., R. Kalaba, and J. Moore (2005). Arriving on time. *Journal of Optimization Theory and Applications* 127(3), 497–513.
- Fellendorf, M. (1994). VISSIM: A microscopic simulation tool to evaluate actuated signal control including bus priority. Presented at the 64th Annual Meeting of the Institute of Transportation Engineers, Dallas, TX.
- Feo, T. A. and M. G. C. Resende (1995). Greedy randomized adaptive search procedures. *Journal of Global Optimization* 6, 109–133.
- Fisk, C. (1980). Some developments in equilibrium traffic assignment. *Transportation Research* 14B, 243–255.
- Fisk, C. and S. Nguyen (1982). Solution algorithms for network equilibrium models with asymmetric user costs. *Transportation Science* 16, 361–381.
- Florian, M., I. Constantin, and D. Florian (2009). A new look at projected gradient method for equilibrium assignment. *Transportation Research Record* 2090, 10–16.
- Florian, M. and D. Hearn (1995). Network equilibrium models and algorithms. In M. O. Ball, T. L. Magnanti, C. L. Monma, and G. L. Nemhauser (Eds.), *Network Routing, Handbooks in Operations Research and Management Science*, Volume 8, pp. 485–549. Oxford, United Kingdom: Elsevier Science.
- Ford, L. R. (1956). Network flow theory. Technical report, Report P-923, Rand Corporation.

- Frank, M. and P. Wolfe (1956). An algorithm for quadratic programming. *Naval Research Logistics Quarterly* 3, 95–110.
- Friesz, T., D. Bernstein, T. E. Smith, R. L. Tobin, and B. W. Wie (1993). A variational inequality formulation of the dynamic network user equilibrium problem. *Operations Research* 41(1), 179–191.
- Friesz, T., J. Luque, R. Tobin, and B. Wie (1989). Dynamic network traffic assignment considered as a continuous time optimal control problem. *Operations Research* 37, 893–901.
- Fu, L. and L. R. Rilett (1998). Expected shortest paths in dynamic and stochastic traffic networks. *Transportation Research Part B* 32(7), 499–516.
- Fudenberg, D. and J. Tirole (1991). *Game Theory*. Cambridge, MA: MIT Press.
- Gallager, R. G. (1977). A minimum delay routing algorithm using distributed computation. In *IEEE Transactions on Communication, COM-25*, pp. 73–85.
- Gao, S. (2005). *Optimal Adaptive Routing and Traffic Assignment in Stochastic Time-Dependent Networks*. Ph. D. thesis, Massachusetts Institute of Technology, Cambridge, MA.
- Garey, M. R. and D. S. Johnson (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company.
- Gartner, N. H. (1980). Optimal traffic assignment with elastic demands: a review. Part II: algorithmic approaches. *Transportation Science* 14(2), 192–208.
- Gentile, G. (2010). The general link transmission model for dynamic network loading and a comparison with the DUE algorithm. In L. G. H. Immers, C. M. J. Tampère, and F. Viti (Eds.), *New Developments in Transport Planning: Advances in Dynamic Traffic Assignment*. Edward Elgar Publishing.
- Gentile, G. (2014). Local user cost equilibrium: a bush-based algorithm for traffic assignment. *Transportmetrica Part A* 10, 15–54.
- Gentile, G. (2016). Solving a dynamic user equilibrium model based on splitting rates with gradient projection algorithm. *Transportation Research Part B* 92, 120–147.
- Gibbons, R. S. (1992). *Game Theory for Applied Economists*. Princeton, NJ: Princeton University Press.
- Glover, F. (1989). Tabu search – part 1. *ORSA Journal on Computing* 1, 190–206.
- Glover, F. (1990). Tabu search – part 2. *ORSA Journal on Computing* 2, 4–32.

- Godunov, S. K. (1959). A difference scheme for numerical solution of discontinuous solution of hydrodynamic equations. *Matematicheskii Sbornik* 47, 271–306.
- Hall, R. W. (1986). The fastest path through a network with random time-dependent travel times. *Transportation Science* 20(3), 182–188.
- Hart, P. E., N. J. Nilsson, and B. Raphael (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics SSC4* 4(2), 100–107.
- Hopf, E. (1970). On the right weak solution of the Cauchy problem for a quasilinear equation of first order. *Indiana University Mathematics Journal* 19, 483–487.
- Horowitz, A. J. (1991). Delay volume relations for travel forecasting based on the 1985 highway capacity manual. Technical Report FHWA-PD-92-015, Federal Highway Administration.
- Institute of Transportation Engineers (2021). *Trip Generation Manual* (11 ed.). Washington, DC: Institute of Transportation Engineers.
- Jafari, E. and S. D. Boyles (2016). Improved bush-based methods for network contraction. *Transportation Research Part B* 83, 298–313.
- Jayakrishnan, R., W. T. Tsai, J. N. Prashker, and S. Rajadhyaksha (1994). A faster path-based algorithm for traffic assignment. *Transportation Research Record* 1443, 75–83.
- Jin, W. L. and H. M. Zhang (2003). On the distribution schemes for determining flows through a merge. *Transportation Research Part B* 37(6), 521–540.
- Josefsson, M. and M. Patriksson (2007). Sensitivity analysis of separable traffic equilibrium equilibria with application to bilevel optimization in network design. *Transportation Research Part B* 41, 4–31.
- Kakutani, S. (1941). A generalization of Brouwer’s fixed point theorem. *Duke Mathematical Journal* 8(3), 457–459.
- Kassoff, H. and H. D. Deutschman (1969). Trip generation: a critical appraisal. *Highway Research Record* 297, 15–30.
- Khani, A. and S. D. Boyles (2015). An exact algorithm for the mean-standard deviation shortest path problem. *Transportation Research Part B* 81, 252–266.
- Kirkpatrick, S., C. D. Gelatt, and M. P. Vecchi (1983). Optimization by simulated annealing. *Science* 220(4598), 671–680.
- Knight, F. H. (1924). Some fallacies in the interpretation of social cost. *Quarterly Journal of Economics* 38, 582–606.

- Knuth, D. E. (1997). *The Art of Computer Programming, Volume 1: Fundamental Algorithms* (3rd ed.). Reading, MA: Addison-Wesley.
- Larsson, T., J. T. Lundgren, C. Rydergren, and M. Patriksson (2001). Most likely traffic equilibrium route flows analysis and computation. In F. Giannessi, A. Maugeri, and P. M. Pardalos (Eds.), *Equilibrium Problems: Nonsmooth Optimization and Variational Inequality Models, Nonconvex Optimization and its Applications*, Volume 58, pp. 129–159. Boston, MA: Springer.
- Larsson, T. and M. Patriksson (1992). Simplicial decomposition with disaggregated representation for the traffic assignment problem. *Transportation Science* 26, 4–17.
- Laval, J. A. and L. Leclercq (2013). The Hamilton-Jacobi partial differential equation and the three representations of traffic flow. *Transportation Research Part B* 52, 17–30.
- Lawphongpanich, S. and D. W. Hearn (1984). Simplicial decomposition of the asymmetric traffic assignment problem. *Transportation Research Part B* 18(2), 123–133.
- Lax, P. D. (1957). Hyperbolic systems of conservation laws II. In P. Sarnak and A. Majda (Eds.), *Communications on Pure and Applied Mathematics*, pp. 537–566. Wiley Periodicals.
- Lebacque, J. P. (1996). The Godunov scheme and what it means for first order traffic flow models. In *Proceedings of the 13th International Symposium on Transportation and Traffic Theory*, London, United Kingdom, pp. 647–678.
- Lebacque, J. P. and M. M. Khoshyaran (2005). First-order macroscopic traffic flow models: intersection modeling, network modeling. In *Proceedings of the 16th International Symposium on Transportation and Traffic Theory*, College Park, MD, pp. 365–386.
- LeVeque, R. (1992). *Numerical Methods for Conservation Laws*. Basel, Switzerland: Birkhäuser.
- Lighthill, M. and G. Whitham (1955). On kinematic waves. II. A theory of traffic flow on long crowded roads. *Proceedings of the Royal Society of London. Series A, Mathematical and Physical Sciences* 229, 317–345.
- Long, J., W. Y. Szeto, H.-J. Huang, and Z. Gao (2012). An intersection-movement-based dynamic traffic assignment problem. Presented at the 4th International Symposium on Dynamic Traffic Assignment, Martha's Vineyard, MA.
- Lou, Y., Y. Yin, and S. Lawphongpanich (2010). Robust congestion pricing under boundedly rational user equilibrium. *Transportation Research Part B* 44, 15–28.

- Loui, R. P. (1983). Optimal paths in graphs with stochastic or multidimensional weights. *Communications of the ACM* 26(9), 670–676.
- Lu, S. (2008). Sensitivity of static traffic user equilibria with perturbations in arc cost function and travel demand. *Transportation Science* 42(1), 105–123.
- Luke, J. C. (1972). Mathematical models for landform evolution. *Journal of Geophysical Research* 77, 2460–2464.
- Mahmassani, H. S. (2000). DYNASMART-P, volume ii, user guide. Technical report, STO67-85-PII, Center for Transportation Research, The University of Texas at Austin.
- Mahmassani, H. S. and G.-L. Chen (1987). On boundedly rational user equilibrium in transportation systems. *Transportation Science* 21, 89–99.
- Mahut, M., M. Florian, and N. Tremblay (2003). Space-time queues and dynamic traffic assignment: a model, algorithm and applications. Presented at the 82nd Annual Meeting of the Transportation Research Board, Washington, DC.
- Marcotte, P. and L. Wynter (2004). A new look at the multiclass network equilibrium problem. *Transportation Science* 38(3), 282–292.
- McFadden, D. (1974a). Conditional logit analysis of qualitative choice behavior. In P. Zarempka (Ed.), *Frontiers in Econometrics*, pp. 105–142. New York, NY: Academic Press.
- McFadden, D. (1974b). The measurement of urban travel demand. *Journal of Public Economics* 3, 303–328.
- Merchant, D. and G. Nemhauser (1978a). A model and an algorithm for the dynamic traffic assignment problems. *Transportation Science* 12, 183–199.
- Merchant, D. and G. Nemhauser (1978b). Optimality conditions for a dynamic traffic assignment model. *Transportation Science* 12, 200–207.
- Metropolis, N., A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller (1953). Equation of state calculations by fast computing machines. *Journal of Chemical Physics* 21, 1087.
- Meyer, M. D. and E. J. Miller (2000). *Urban Transportation Planning: A Decision-Oriented Approach*. New York, NY: McGraw-Hill.
- Miller-Hooks, E. D. (2001). Adaptive least-expected time paths in stochastic, time-varying transportation and data networks. *Networks* 37(1), 35–52.
- Miller-Hooks, E. D. and H. S. Mahmassani (2000). Least expected time paths in stochastic, time-varying transportation networks. *Transportation Science* 34(2), 198–215.

- Mitradjieva, M. and P. O. Lindberg (2013). The stiff is moving — conjugate direction Frank-Wolfe methods with application to traffic assignment. *Transportation Science* 47(2), 280–293.
- Montemanni, R. and L. Gambardella (2004). An exact algorithm for the robust shortest path problem with interval data. *Computers and Operations Research* 31(10), 1667–1680.
- Murchland, J. D. (1970). Braess’s paradox of traffic flow. *Transportation Research* 4, 391–394.
- Murthy, I. and S. Sarker (1996). A relaxation-based pruning technique for a class of stochastic shortest path problems. *Transportation Science* 30(3), 220–236.
- Nash, J. (1950). Equilibrium points in n -person games. *Proceedings of the National Academy of Sciences* 36, 48–49.
- Netter, M. (1972). Affectations de trafic et tarification au coût marginal social: critique de quelques idées admises. *Transportation Research* 6, 411–429.
- Newell, G. F. (1993a). A simplified theory of kinematic waves in highway traffic, part I: General theory. *Transportation Research Part B* 27(4), 281–287.
- Newell, G. F. (1993b). A simplified theory of kinematic waves in highway traffic, part II: Queueing at freeway bottlenecks. *Transportation Research Part B* 27(4), 289–303.
- Newell, G. F. (1993c). A simplified theory of kinematic waves in highway traffic, part III: Multi-destination flows. *Transportation Research Part B* 27(4), 305–313.
- Newton, I. (1687). *Philosophiæ Naturalis Principia Mathematica*. London, United Kingdom.
- Nezamuddin and S. D. Boyles (2015). A continuous DUE algorithm using the link transmission model. *Networks and Spatial Economics* 15, 465–483.
- Nie, Y. and X. Wu (2009). Shortest path problem considering on-time arrival probability. *Transportation Research Part B* 43(6), 597–613.
- Nie, Y. M. (2010a). A class of bush-based algorithms for the traffic assignment problem. *Transportation Research Part B* 44, 73–89.
- Nie, Y. M. (2010b). Equilibrium analysis of macroscopic traffic oscillations. *Transportation Research Part B* 44, 62–72.
- Nie, Y. M., J. Ma, and H. M. Zhang (2008). A polymorphic dynamic network loading model. *Computer-Aided Civil and Infrastructure Engineering* 23(2), 86–103.

- Ortúzar, J. d. and L. G. Willumsen (2011). *Modelling Transport*. Hoboken, NJ: Wiley.
- Osorio, C. and M. Bierlaire (2009). An analytic finite capacity queueing network model capturing the propagation of congestion and blocking. *European Journal of Operational Research* 196, 996–1007.
- Osorio, C., G. Flötteröd, and M. Bierlaire (2011). Dynamic network loading: a stochastic differentiable model that derives link state distributions. *Transportation Research Part B* 45, 1410–1423.
- Pape, U. (1974). Implementation and efficiency of Moore-algorithms for the shortest route problem. *Mathematical Programming* 7, 212–222.
- Patriksson, M. (1994). *The Traffic Assignment Problem — Models and Methods*. Utrecht, Netherlands: VSP.
- Patriksson, M. (2004). Sensitivity analysis of traffic equilibria. *Transportation Science* 38, 258–281.
- Peeta, S. and A. K. Ziliaskopoulos (2001). Foundations of dynamic traffic assignment: the past, the present, and the future. *Networks and Spatial Economics* 1, 233–265.
- Pham, D. T., A. Ghanbarzadeh, E. Koc, S. Otri, S. Rahim, and M. Zaidi (2005). The bees algorithm. Technical report, Manufacturing Engineering Centre, Cardiff University.
- Pigou, A. C. (1920). *The Economics of Welfare*. London, United Kingdom: Macmillan and Co.
- Polychronopoulos, G. H. and J. N. Tsitsiklis (1996). Stochastic shortest path problems with recourse. *Networks* 27(2), 133–143.
- Powell, W. B. and Y. Sheffi (1982). The convergence of equilibrium algorithms with predetermined step sizes. *Transportation Science* 16, 45–55.
- Provan, J. S. (2003). A polynomial-time algorithm to find shortest paths with recourse. *Networks* 41(2), 115–125.
- Psaraftis, H. N. and J. N. Tsitsiklis (1993). Dynamic shortest paths in acyclic networks with Markovian arc costs. *Operations Research* 41(1), 91–101.
- Ran, B., D. E. Boyce, and L. J. LeBlanc (1993). A new class of instantaneous dynamic user-optimal traffic assignment models. *Operations Research* 41(1), 192–202.
- Rasmussen, E. (2006). *Games and Information: An Introduction to Game Theory*. Hoboken, NJ: Wiley-Blackwell.

- Rechenberg, I. (1973). *Evolutionstrategie*. Stuttgart, Germany: Holzmann-Froboog.
- Richards, P. (1956). Shock waves on the highway. *Operations Research* 4(1), 42–51.
- Ritzberger, K. (2002). *Foundations of Non-Cooperative Game Theory*. Oxford, United Kingdom: Oxford University Press.
- Rockafellar, R. T. (1997). *Convex Analysis*. Princeton, NJ: Princeton University Press.
- Rossi, T. F., S. McNeil, and C. Hendrickson (1989). Entropy model for consistent impact fee assessment. *Journal of Urban Planning and Development* 115, 51–63.
- Roughgarden, T. A. (2002). *Selfish Routing*. Ph. D. thesis, Cornell University.
- Schwefel, H.-P. (1977). *Numerische Optimierung von Computer-Modellen mittels der Evolutionstrategie: mit einer vergleichenden Einführung in die Hill-Climbing- und Zufallsstrategie*. Basel, Switzerland; Stuttgart, Germany: Birkhäuser.
- Sen, S., R. Pillai, S. Joshi, and A. K. Rathi (2001). A mean-variance model for route guidance in advanced traveler information systems. *Transportation Science* 35(1), 37–49.
- Shahabi, M., A. Unnikrishnan, and S. D. Boyles (2013). An outer approximation algorithm for the robust shortest path problem. *Transportation Research Part E* 58, 52–66.
- Shahabi, M., A. Unnikrishnan, and S. D. Boyles (2015). Robust optimization strategy for the shortest path problem under uncertain link travel cost distribution. *Computer-Aided Civil and Infrastructure Engineering* 30, 433–448.
- Sheffi, Y. (1985). *Urban Transportation Networks*. Englewood Cliffs, NJ: Prentice-Hall.
- Sipser, M. (2012). *Introduction to the Theory of Computation* (3rd ed.). Cengage Learning.
- Sivakumar, R. A. and R. Batta (1994). The variance-constrained shortest path problem. *Transportation Science* 28(4), 309–316.
- Smith, M. J. (1979a). The existence, uniqueness, and stability of traffic equilibria. *Transportation Research* 13B, 295–304.
- Smith, M. J. (1979b). Traffic control and route-choice; a simple example. *Transportation Research* 13B, 289–294.

- Smith, M. J. (1983). The existence and calculation of traffic equilibria. *Transportation Research Part B* 17B(4), 291–303.
- Tampère, C. M. J., R. Corthout, D. Cattrysse, and L. H. Immers (2011). A generic class of first order node models for dynamic macroscopic simulation of traffic flows. *Transportation Research Part B* 45, 289–309.
- Tobin, R. L. and T. L. Friesz (1988). Sensitivity analysis for equilibrium network flow. *Transportation Science* 22(4), 242–250.
- Vandermonde, A. T. (1774). Remarques sur des problèmes de situation. *Histoire de l'Académie royale des sciences, avec les mémoires de mathématique et de physique 1771*, 566–574.
- Vickrey, W. S. (1969). Congestion theory and transport investment. *American Economic Review* 59(2), 251–261.
- von Neumann, J. and O. Morgenstern (1944). *Theory of games and economic behavior*. Princeton, NJ: Princeton University Press.
- Waller, S. T. (2006). Personal communication.
- Waller, S. T. and A. K. Ziliaskopoulos (2002). On the online shortest path problem with limited arc cost dependencies. *Networks* 40(4), 216–227.
- Wardrop, J. (1952). Some theoretical aspects of road traffic research. *Proceedings of the Institute of Civil Engineers, Part II* 1, 325–378.
- Wie, B. W., R. L. Tobin, T. L. Friesz, and D. Bernstein (1995). A discrete time, nested cost operator approach to the dynamic network user equilibrium problem. *Transportation Science* 29, 79–92.
- Wilson, A. G. (1970). *Entropy in Urban and Regional Modeling*. London, United Kingdom: Pion.
- Wilson, A. G., J. D. Coelho, S. M. MacGill, and H. C. W. L. Williams (1981). *Optimization in Locational and Transport Analysis*. Chichester, United Kingdom: Wiley.
- Wolsey, L. A. (1998). *Integer Programming*. Hoboken, NJ: Wiley-Interscience.
- Xie, J., Y. Nie, and X. Yang (2013). Quadratic approximation and convergence of some bush-based algorithms for the traffic assignment problem. *Transportation Research Part B* 56, 15–30.
- Xing, T. and X. Zhou (2011). Finding the most reliable path with and without link travel time correlation: a Lagrangian substitution based approach. *Transportation Research Part B* 59, 22–44.
- Yang, H. and M. G. H. Bell (2007). Sensitivity analysis of network traffic equilibria revisited: the corrected approach. In B. Heydecker (Ed.), *Mathematics in Transport*, pp. 373–411. Elsevier.

- Yperman, I. (2007). *The Link Transmission Model for Dynamic Network Loading*. Ph. D. thesis, Katholieke Universiteit Leuven, Belgium.
- Yu, G. and J. Yang (1998). On the robust shortest path problem. *Computers and Operations Research* 25(6), 457–468.
- Zhang, H. M. and Y. Nie (2006). A hybrid traffic simulation model. Presented at the 85th Annual Meeting of the Transportation Research Board, Washington, DC.
- Zhang, Y., Z.-J. M. Shen, and S. Song (2016). Parametric search for the bi-attribute concave shortest path problem. *Transportation Research Part B* 94, 150–168.
- Ziegelmann, M. (2001). *Constrained Shortest Paths and Related Problems*. Ph. D. thesis, Universität des Saarlandes.
- Ziliaskopoulos, A. K. (2000). A linear programming model for the single destination system optimum dynamic traffic assignment problem. *Transportation Science* 34(1), 37–49.
- Ziliaskopoulos, A. K. and H. S. Mahmassani (1994). Time-dependent, shortest-path algorithm for real-time intelligent vehicle highway system applications. *Transportation Research Record* 1408, 94–104.
- Ziliaskopoulos, A. K. and S. T. Waller (2000). An Internet-based geographic information system that integrates data, models and users for transportation applications. *Transportation Research Part C* 8, 427–444.