Bilevel problems: OD matrix estimation, network design, and sensitivity analysis

CE 392C

OUTLINE

Outline

INPUT DATA

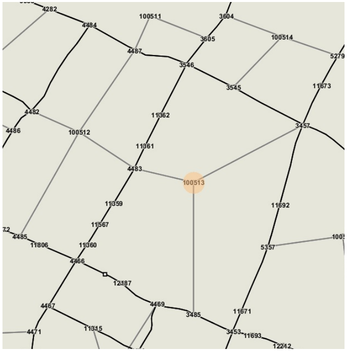
What do you need to run the basic traffic assignment model?

- The network itself
- Link performance functions
- OD matrix

The extensions of TAP require additional information, such as demand functions, destination attractiveness, logit parameters, value of time, etc.

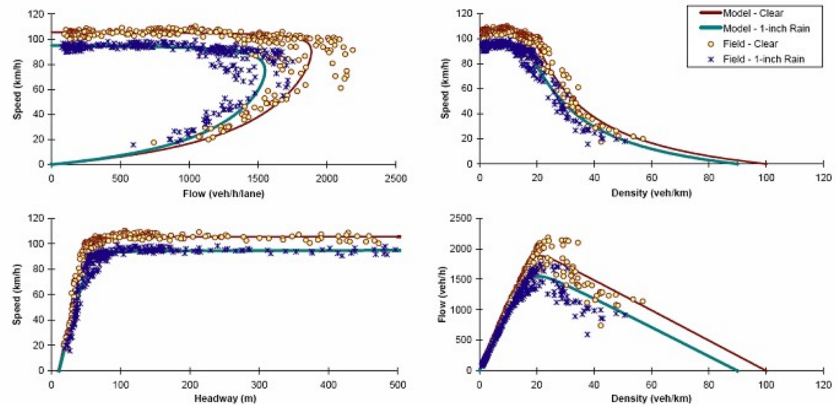
Deciding which streets to include in the network is a balance of accuracy and computation time/data collection requirements.

In practice, regional models typically include minor arterials and larger roads; neighborhood streets are typically abstracted into centroid connectors:



Neighborhood streets are typically uncongested, so there isn't a need to model them in great detail. (Or is there?)

Ideally, link performance functions are obtained through regression of field data. What are the complications?



Traffic engineering concepts (signal delay, etc.) can also be included.

When you have a network of tens of thousands of links, this is impractical.

The advantage of “standard functions” like the BPR relation are simpler data collection requirements:

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

Free-flow speeds and capacities are relatively easy to calculate. Typically $\alpha = 0.15$ and $\beta = 4$.

Caution: The BPR “capacity” is intended to be a “practical capacity” corresponding to LOS C, roughly 80 percent of true capacity. If true capacity is used, $\alpha = 0.84$ and $\beta = 5.5$ are recommended instead.

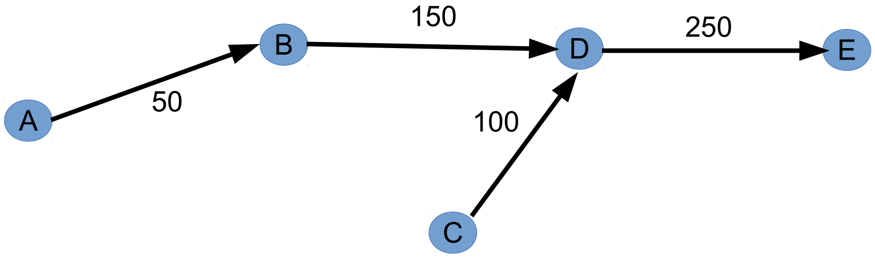
The OD matrix is often the most challenging input data to calibrate, for several reasons:

- There are many more OD matrix entries than links.
- The OD matrix can't be observed directly (unlike link speeds and flows).

Can we use direct observations (say, link flows) to try to estimate the OD matrix?

This is surprisingly difficult!

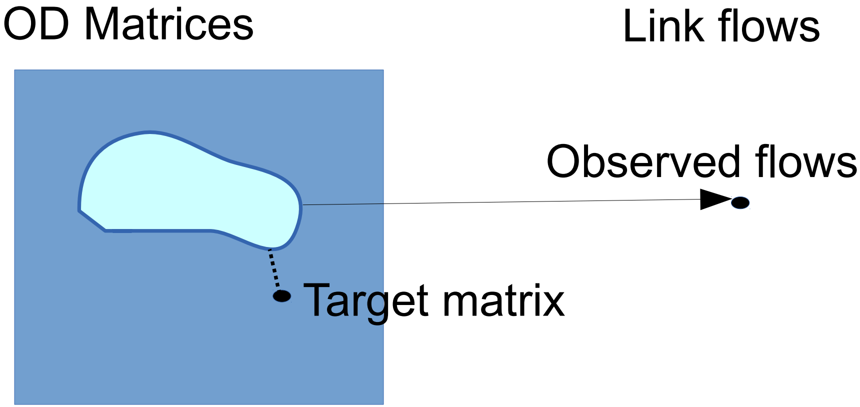
Because there are more OD matrix entries than links, the problem is highly underdetermined; the problem is not finding an OD matrix that matches the



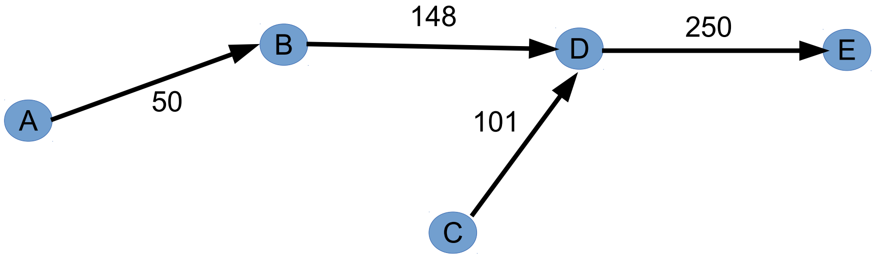
One trivial solution is for all trips to go from one node to a neighboring node.

(As an aside, the distinction between a good regression fit and a good model is absolutely critical here.)

We often have an OD matrix available from other parts of the planning process, say, a gravity model. Can we use this “target” OD matrix as a starting point which can be adjusted to conform to link flows?



Because field data contains some noise and error, however, all solutions which satisfy link flows exactly may have short trips:



Instead, we can try a least-squares approach where we try to match *both* the target OD matrix and the link flows:

$$\min_{\mathbf{d},\mathbf{x}} \Theta \sum_{rs} \left(d^{rs} - \overline{d}^{rs} \right)^2 + (1 - \Theta) \sum_{ij} (x_{ij} - \overline{x}_{ij})^2$$

where Θ reflects the importance put on matching the OD matrix relative to the link flows; the proper balance is a matter of judgment and depends on the level of trust in the accuracy of $\overline{\mathbf{d}}$ and $\overline{\mathbf{x}}$.

We have two constraints: nonnegativity of OD matrix entries $d^{rs} \geq 0$, and that the link flows $\mathbf{x}$ must be a user equilibrium solution given the OD matrix $\mathbf{d}$.

Since user equilibrium is itself an optimization problem, we have a bilevel program, where one of the constraints is itself an optimization problem:

$$\begin{aligned}
\min_{\mathbf{d},\mathbf{x},\mathbf{h}} \quad & \Theta \sum_{rs} \left(d^{rs} - \overline{d}^{rs} \right)^2 + (1-\Theta) \sum_{ij} (x_{ij} - \overline{x}_{ij})^2 & \\
\text{s.t.} \quad & d^{rs} \geq 0 & \forall (r,s) \in Z^2 \\
& \mathbf{x} \in \arg\min_{\mathbf{x}'} \sum_{ij} \int_0^{x'_{ij}} t_{ij}(x)\ dx & \\
& \sum_{\pi \in \Pi^{rs}} h^\pi = d^{rs} & \forall (r,s) \in Z^2 \\
& h^\pi \geq 0 & \forall \pi \in \Pi
\end{aligned}$$

The "upper level" is a least squares fit for the OD matrix, and the "lower level" is the user equilibrium condition.

Bilevel programs are generally difficult to solve, because the feasible region is nonconvex. Heuristic solution methods are often the best we can do.

An example of such a heuristic: write link flows as a function of demand $\mathbf{x}(\mathbf{d})$ with the understanding that this function contains the equilibrium mapping.

$$\begin{aligned}
\min_{\mathbf{d}} \quad & \Theta \sum_{rs} \left(d^{rs} - \overline{d}^{rs} \right)^2 + (1-\Theta) \sum_{ij} \left(x_{ij}(\mathbf{d}) - \overline{x}_{ij} \right)^2 \\
\text{s.t.} \quad & d^{rs} \geq 0 \qquad \forall (r,s) \in Z^2
\end{aligned}$$

A gradient projection algorithm can then be applied.

Given some OD matrix $\mathbf{d}$, identify the gradient $\nabla f(\mathbf{d})$ and take a step of size μ in the opposite direction, "projecting" onto the feasible region:

$\mathbf{d} \leftarrow [\mathbf{d} - \mu \nabla f(\mathbf{d})]^+$

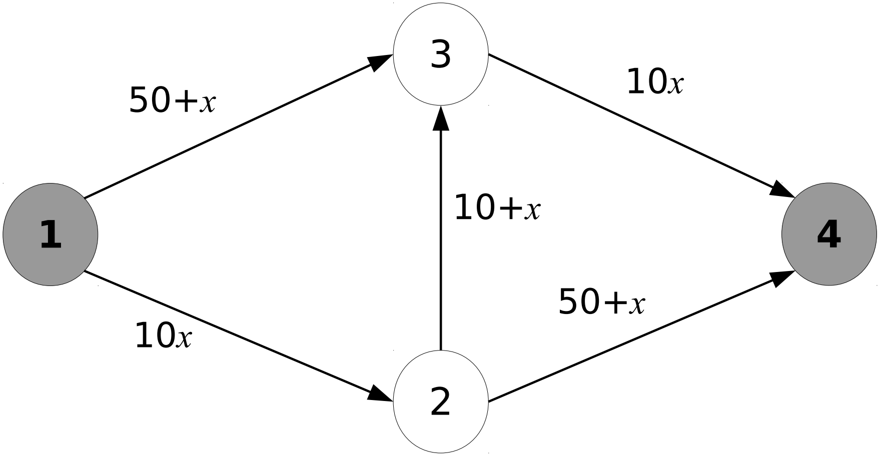
So, what is $\nabla f(\mathbf{d})$?

Its elements are

$$\frac{\partial f}{\partial d^{rs}} = 2\Theta(d^{rs} - \overline{d}^{rs}) + 2(1 - \Theta) \sum_{(i,j) \in A} (x_{ij}(\mathbf{d}) - \overline{x}_{ij}) \frac{\partial x_{ij}}{\partial d^{rs}}$$

so we need to know what $\frac{\partial x_{ij}}{\partial d^{rs}}$ is; that is, the sensitivity of flow on link (i, j) to demand from OD pair (r, s).

Example

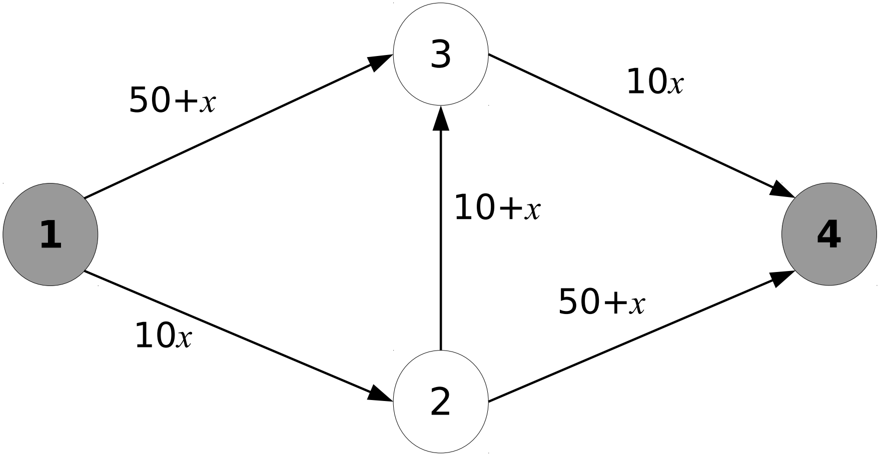


Because the link flows will adjust to restore equilibrium, the sensitivity analysis amounts to *solving a modified equilibrium problem with the derivatives of the link performance functions without non-negativity constraints*

Specifically:

- Replace each link performance function with $t'_{ij} x_{ij}$ where t'_{ij} is the derivative evaluated at the current equilibrium solution.
- Fix each origin's bush as the equilibrium bush in the original solution.
- The only demand is 1 vehicle going from r to s
- Solve the user equilibrium problem without non-negativity constraints. (Easy for gradient projection or Algorithm B)
- The resulting link flows are the values $\frac{\partial x_{ij}}{\partial d^{rs}}$

Example



Algorithm

So, our OD matrix estimation algorithm works as follows:

- Initialize OD matrix to target, $\mathbf{d} \leftarrow \overline{\mathbf{d}}$
- Solve for equilibrium link flows $\mathbf{x}$ corresponding to $\mathbf{d}$
- Calculate derivatives $\frac{\partial x_{ij}}{\partial d^{rs}}$ for each link and OD pair using current link travel time derivatives.
- Calculate the gradient $\nabla f(\mathbf{d})$.
- Update $\mathbf{d} \leftarrow [\mathbf{d} - \mu \nabla f(\mathbf{d})]^+$ for some μ which ensures descent of f
- Check convergence and return to step 2 if not converged.

You have to solve equilibrium many times in this algorithm; here it's especially important to have a fast algorithm.

BUSH-BASED SENSITIVITY ANALYSIS

In a bush-based framework, the sensitivity analysis is not too difficult to describe. At the equilibrium solution, the following equations must hold for each bush:

$$
\begin{aligned}
L_j^r - L_i^r - t_{ij}(\bar{\mathbf{x}}) &= 0 && \forall (i,j) \in \mathcal{B}^r \\
L_r^r &= 0 \\
\sum_{(h,i)\in\Gamma^{-1}(i)} \bar{x}_{hi}^r - \sum_{(i,j)\in\Gamma(i)} \bar{x}_{ij}^r &= d^{ri} && \forall i \in N \backslash r \\
\sum_{(h,r)\in\Gamma^{-1}(r)} \bar{x}_{hr}^r - \sum_{(r,j)\in\Gamma(r)} \bar{x}_{rj}^r &= -\sum_{s\in Z} d^{rs} \\
\bar{x}_{ij}^r &= 0 && \forall (i,j) \notin \mathcal{B}^r
\end{aligned}
$$

By differentiating each of these equations with respect to one entry in the OD matrix $d_{\hat{r}\hat{s}}$, we can see how the solution would change with the OD matrix.

The derivatives we are interested in are $\frac{\partial x_{ij}^{r}}{\partial d_{\hat{r}\hat{s}}}$ and $\frac{\partial L_{i}^{r}}{\partial d_{\hat{r}\hat{s}}}$.

For brevity denote these by ξ_{ij}^{r} and Θ_i.

(We will need to handle each origin's bush separately — this can be done in parallel.)

Taking derivatives, we have

$$\Theta_j^r - \Theta_i^r - \frac{dt_{ij}}{dx_{ij}} \sum_{r' \in Z} \xi_{ij}^{r'} = 0 \qquad \forall (i,j) \in \mathcal{B}^r$$

$$\Theta_r^r = 0$$

$$\sum_{(h,i) \in \Gamma^{-1}(i)} \xi_{hi}^r - \sum_{(i,j) \in \Gamma(i)} \xi_{ij}^r = \begin{cases} 1 & \text{if } r = \hat{r} \text{ and } i = \hat{s} \\ 0 & \text{otherwise} \end{cases} \qquad \forall i \in N \backslash r$$

$$\sum_{(h,r) \in \Gamma^{-1}(r)} \xi_{hr}^r - \sum_{(r,j) \in \Gamma(r)} \xi_{rj}^r = \begin{cases} -1 & \text{if } r = \hat{r} \\ 0 & \text{otherwise} \end{cases}$$

$$\xi_{ij}^r = 0 \qquad \forall (i,j) \notin \mathcal{B}^r$$

With some manipulations, these equations can be seen as the *optimality conditions to the following convex optimization problem*:

$$\begin{aligned}
\min_{\boldsymbol{\xi}^r} \quad & \int_0^{\xi_{ij}} \frac{dt_{ij}}{dx_{ij}} \xi \; d\xi \\
\text{s.t.} \quad & \sum_{(h,i) \in \Gamma^{-1}(i)} \xi^r_{hi} - \sum_{(i,j) \in \Gamma(i)} \xi^r_{ij} = \Delta_{ri} && \forall i \in N, r \in Z \\
& \xi^r_{ij} = 0 && \forall (i,j) \notin \mathcal{B}^r
\end{aligned}$$

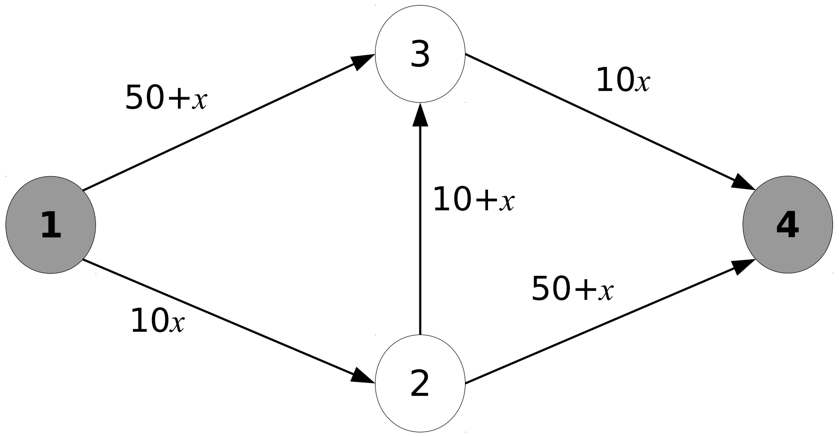
This is essentially a traffic assignment problem in modified form.

Sensitivity traffic assignment problem

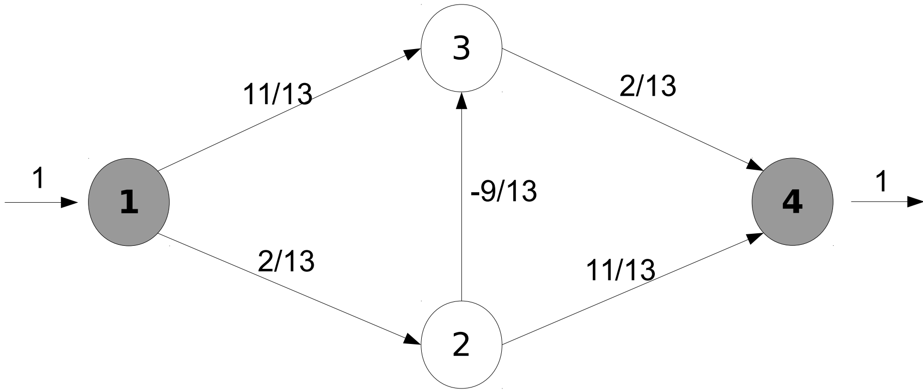
The relationships between the demand sensitivity problem and the original traffic assignment problem:

1. Link performance functions are now linear, with slope dt_{ij}/dx_{ij}
2. The bush is fixed.
3. The only entry in the OD matrix is one unit of flow from $\hat{r}$ to $\hat{s}$.
4. There are no non-negativity conditions.

Example



Example



NETWORK DESIGN

Another common bilevel program concerns optimal allocation of network improvements.

Assume that the link performance function is $t_{ij}(x_{ij}, y_{ij})$, where x_{ij} is the flow and y_{ij} is the amount of money spent improving this link (increasing capacity, increasing free-flow speed, etc.)

Further, assume that the cost of improving link (i, j) by y_{ij} units is given by a cost function $C_{ij}(y_{ij})$.

The objective is to identify y_{ij} values that minimize the *total cost*, including user costs (travel time) and capital improvement costs.

What makes this problem difficult is how x_{ij} is determined: after the network improvements are made, drivers will adjust their route choices to find a new equilibrium.

This gives another bilevel problem:

$$\begin{aligned}
\min_{\mathbf{y},\mathbf{x},\mathbf{h}} \quad & \sum_{(i,j)\in A} x_{ij} t_{ij}(x_{ij}, y_{ij}) + \sum_{(i,j)\in A} C_{ij}(y_{ij}) & \\
\text{s.t.} \quad & y^{ij} \geq 0 & \forall (i,j) \in A \\
& \mathbf{x} \in \arg\min_{\mathbf{x}'} \sum_{ij} \int_0^{x'_{ij}} t_{ij}(x, y_{ij})\ dx & \\
& \sum_{\pi \in \Pi^{rs}} h^\pi = d^{rs} & \forall (r,s) \in Z^2 \\
& h^\pi \geq 0 & \forall \pi \in \Pi
\end{aligned}$$

The problem is easy if we assume that route choices won't change, but this is completely unrealistic.

Let's apply the same approach as before:

Given some investment plan $\mathbf{y}$, identify the gradient $\nabla f(\mathbf{y})$ and take a step of size μ in the opposite direction, projecting onto the feasible region:

$\mathbf{y} \leftarrow [\mathbf{y} - \mu \nabla f(\mathbf{y})]^+$

So, what is $\nabla f(\mathbf{y})$?

Its elements are

$$\frac{\partial f}{\partial y_{ij}} = x_{ij} \frac{\partial t_{ij}}{\partial y_{ij}} + \sum_{(k,\ell) \in A} t_{k\ell} \frac{\partial x_{k\ell}}{\partial y_{ij}} + \frac{dC_{ij}}{dy_{ij}}$$

so we need to know what $\frac{\partial x_{k\ell}}{\partial y_{ij}}$ is; that is, the sensitivity of flow on link (k, ℓ) to investing an additional dollar in improving (i, j).

If y_{ij} is small enough, the set of equilibrium paths will usually remain the same.

This means that the network demand will redistribute among the existing equal-cost paths to maintain equilibrium.

In other words, each origin's bush remains the same, and the travel times on each path should remain equal.

As before, flows on paths can decrease as well as increase (this should be obvious since there is no change in total demand).

Changes in the link performance function

A similar sensitivity analysis can be done to chanegs in the link performance function.

$$t(x) = t^0 \left(1 + \alpha \left(\frac{x}{\gamma} \right)^\beta \right)$$

This function has four parameters t^0, α, γ, and β. If any of these change, the equilibrium solution will change as well.

Differentiating the same equations with respect to one of these parameters (call it y_{ij}) produces the following linear system:

$$
\begin{aligned}
\Theta_j^r - \Theta_i^r - \frac{dt_{ij}}{dx_{ij}} \sum_{r' \in Z} \xi_{ij}^{r'} - \frac{dt_{ij}}{dy_{ij}} &= 0 && \forall (i,j) \in \mathcal{B}^r \\
\Theta_r^r &= 0 \\
\sum_{(h,i) \in \Gamma^{-1}(i)} \xi_{hi}^r - \sum_{(i,j) \in \Gamma(i)} \xi_{ij}^r &= 0 && \forall i \in N \backslash r \\
\sum_{(h,r) \in \Gamma^{-1}(r)} \xi_{hr}^r - \sum_{(r,j) \in \Gamma(r)} \xi_{rj}^r &= 0 \\
\xi_{ij}^r &= 0 && \forall (i,j) \notin \mathcal{B}^r
\end{aligned}
$$

As before, all derivatives are evaluated **at the current equilibrium solution**.

These equations form the optimality conditions for the following convex program:

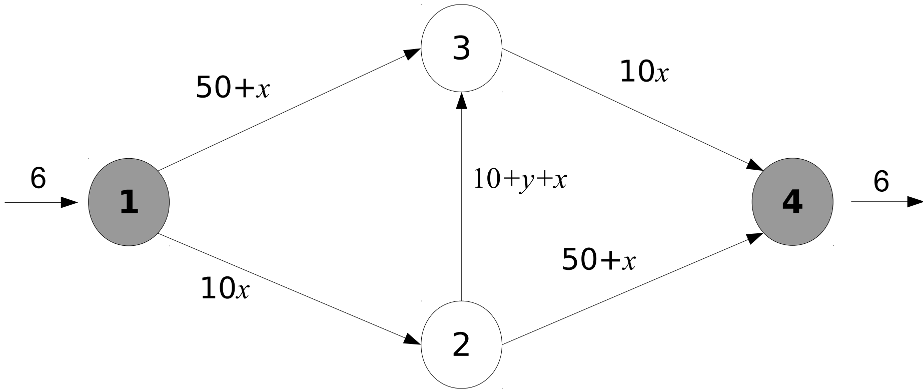
$$\begin{aligned}
\min_{\boldsymbol{\xi}^r} \quad & \int_0^{\xi_{ij}} \left(\frac{dt_{ij}}{dx_{ij}} \xi + \frac{dt_{ij}}{dy} \right) \, d\xi && && (1) \\
\text{s.t.} \quad & \sum_{(h,i) \in \Gamma^{-1}(i)} \xi^r_{hi} - \sum_{(i,j) \in \Gamma(i)} \xi^r_{ij} = 0 && \forall i \in N, r \in Z && (2) \\
& \xi^r_{ij} = 0 && \forall (i,j) \notin \mathcal{B}^r && (3)
\end{aligned}$$

Sensitivity traffic assignment problem

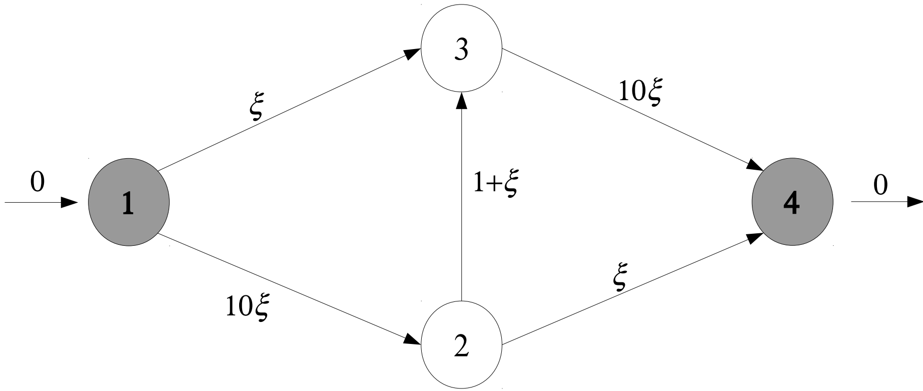
The relationships between the link performance function sensitivity problem and the original traffic assignment problem:

1. Link performance functions are now linear, with slope dt_{ij}/dx_{ij}; the modified link has a constant term dt_{ij}/dy_{ij} added to it.
2. The bush is fixed.
3. The OD matrix has zeros in all entries.
4. There are no non-negativity conditions.

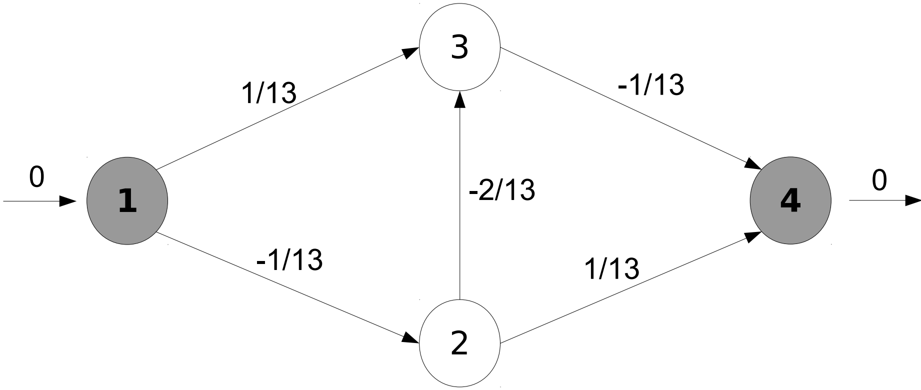
Example

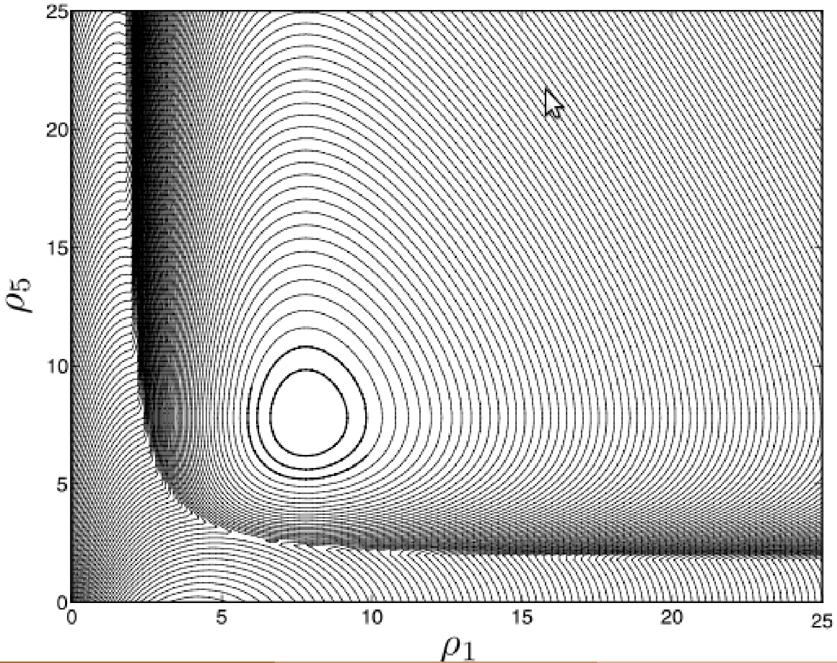


Example



Example





25
20
15
10
5
0
ρ_5
0
5
10
15
20
25
ρ_1

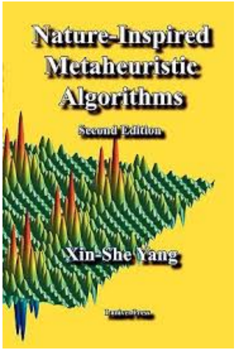
OTHER APPROACHES

These heuristics are problem-specific, that is, they used features of the equilibrium problem.

There is also a class of *metaheuristics* which can be applied to nearly any optimization problem, regardless of convexity, differentiability, etc.

MORE COMPLICATED PROBLEMS AND HEURISTICS

Some of the most broadly applicable methods are called *heuristics* (or sometimes *metaheuristics*).

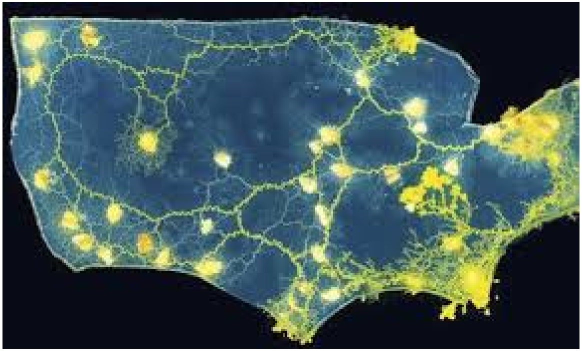


A heuristic is not guaranteed to find the global optimum, but often works well in practice.

In practice, heuristics are only used for very large problems, or problems where there is no known exact method which is efficient enough to use.

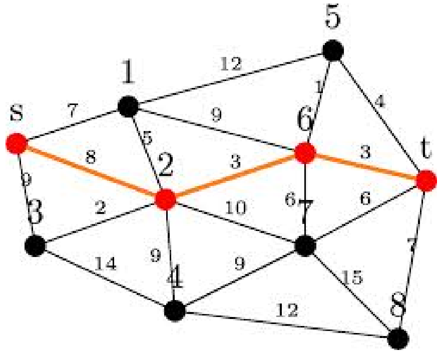
Sometimes, finding a “good enough” solution in a short amount of time is better than spending a lot of time to find the exact optimal solution. Why?

Many heuristic techniques are inspired by the natural world.

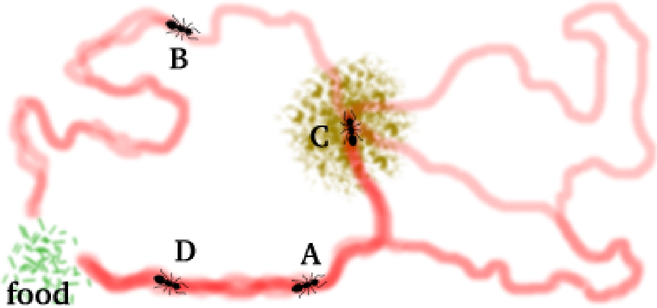


Very commonly living systems successfully (if not optimally) accomplish complicated tasks with relatively simple rules

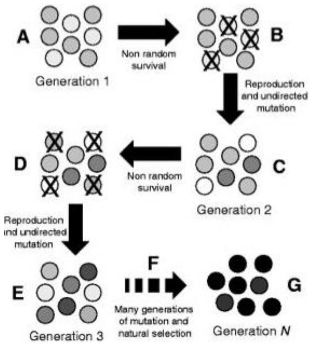
How do humans find shortest paths?



How does an ant colony find food sources?



How does natural selection work?


A
Generation 1
Non random survival
B
Reproduction and undirected mutation
C
Generation 2
Non random survival
D
Reproduction ınd undirected mutation
E
Generation 3
F
Many generations of mutation and natural selection
G
Generation N

Two common heuristics are:

- *Simulated annealing*, which searches through the feasible region in a half-systematic, half-random way.
- Genetic algorithms, which maintain a “population” of feasible solutions which are selectively “bred” with each other.

These algorithms make virtually *no* assumptions on the nature of the objective function or constraints.

SIMULATED ANNEALING

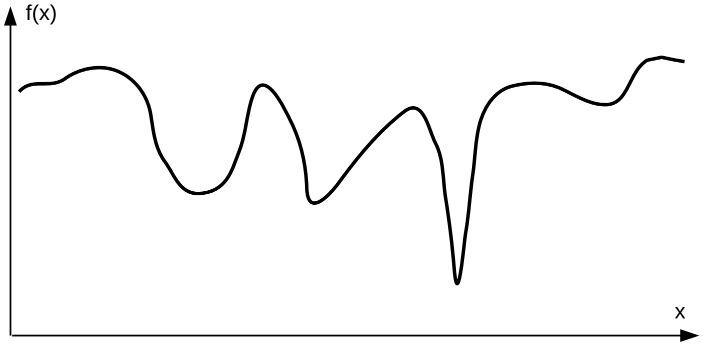
Imagine a hiker walking through a mountainous park area.

The hiker is trying to find the lowest elevation point in the park. (The park is the feasible region, the elevation profile is the objective function.)

However, there is a dense fog and the hiker has no map, so they have to rely on what they can see nearby.

What strategy should they use to try to find the lowest point?

One approach is to always walk in a downhill direction.



The downside of this “local search” strategy is that it is very easy to get trapped in a local minimum which is not global.

So, sometimes we need to walk uphill… but we can't do it all the time.

The *simulated annealing* algorithm incorporates randomization into local search.

A typical step of simulated annealing works as follows:

- Given a current feasible solution $\mathbf{x}$, generate a new feasible solution $\mathbf{x}'$ which is close to $\mathbf{x}$.
- Compare $f(\mathbf{x})$ and $f(\mathbf{x}')$. If $\mathbf{x}'$ is better, accept $\mathbf{x}'$ as the new current solution and repeat.
- If $\mathbf{x}'$ is *worse*, accept $\mathbf{x}'$ with some probability and repeat. Otherwise, generate another $\mathbf{x}'$.

How should this probability be chosen? We run into problems if the probability is too high or too low.

Simulated annealing uses a parameter T called the *temperature* to reflect the probability of moving uphill. When the temperature is high, uphill moves are more likely to be chosen. When the temperature is low, uphill moves are unlikely to be chosen. The algorithm uses a *cooling schedule* to gradually decrease the temperature over time.

The hope is that having an initially high temperature allows a large portion of the feasible space to be explored, but that lowering the temperature ensures that over time there is more and more preference for lower points.

One common formula (not the only choice) for the probability of moving uphill is $\exp(-[f(\mathbf{x}') - f(\mathbf{x})/T)$.

One common cooling schedule is to start with an initial temperature T_0, and multiply it by a common factor $k \in (0, 1)$ every n iterations.

Finally, since there is no guarantee that the algorithm will end up in the lowest point found during the search, you should keep track of the best solution found so far (call it $\mathbf{x}^*$).

This is like the hiker keeping a record of their journey, and at the end of their search returning to the lowest point found so far.

Then, the entire algorithm is:

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) Randomly 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 sone, 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 $\mathbf{x}^*$

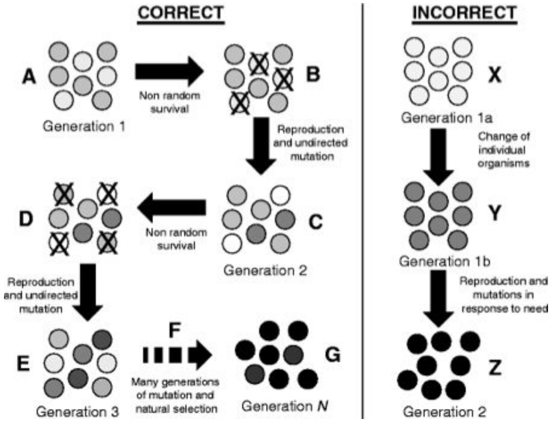
Some questions:

- **How should the cooling schedule be chosen?** Hard to give general guidance, very problem-specific. Trial and error is common.
- **How should the initial solution be chosen?** If you can come up with a good rule of thumb *quickly*, use it. Otherwise, just randomize.
- **What is a neighboring solution?** Again problem-specific; should be similar to the current solution, but should eventually be able to reach any other feasible point.
- **How can I perform an action with probability p?**

  Generate a uniform random number between 0 and 1, perform the action if that number is less than p.

GENETIC ALGORITHMS

Genetic algorithms attempt to find a good-quality solution by mimicing the process of natural selection.


CORRECT
INCORRECT
A
Generation 1
Non random survival
B
Reproduction and undirected mutation
C
Generation 2
Non random survival
D
Reproduction and undirected mutation
E
Generation 3
F
Many generations of mutation and natural selection
G
Generation N
X
Generation 1a
Change of individual organisms
Y
Generation 1b
Reproduction and mutations in response to need
Z
Generation 2

Like simulated annealing, it is just a heuristic by can often find high-quality solutions to complicated problems.

In a nutshell, natural selection in most animals and plants involves the following principles:

- "Survival of the fittest": organisms better suited to their environment are more likely to reproduce.
- Sexual reproduction: two organisms produce a new organism by combining DNA from the parents.
- Mutation: rare, random changes in a gene due to damaged DNA or a copying error.

We want to search for good solutions to an optimization problem by making an analogy to these principles.

In contrast to simulated annealing, which had just one current solution, in genetic algorithms we maintain a *population* of many feasible solutions.

Furthermore, there will be multiple *generations*, each with their own population of solutions.

Starting with an initial generation, we want to create a generation of "offspring" which ideally have lower objective function values.

We can adapt the principles of natural selection in the following way:

- "Survival of the fittest": solutions with lower objective function values are more likely to "reproduce."
- Sexual reproduction: a new feasible solution is created by combining aspects of two "parents."
- Mutation: rare, random changes in a solution.

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.

How to generate an initial population?

It is more important for the initial population to be *highly diverse* than for the initial solutions to have low objective function values.

(Genetic algorithms get most of their power from breeding solutions together, mutation plays a secondary role.)

There is nothing wrong with randomly generating all of the initial solutions (just make sure they are feasible).

How to select parent solutions?

The *tournament selection* rule works like this: pick a tournament size t.

From the current generation, randomly select t solutions. The one with the lowest objective function value is the first parent.

Repeat the tournament by selecting t more solutions randomly. The one with the lowest objective function value is the second parent.

How to “breed” solutions together?

This part can be trickier, and varies from one problem to the next. We need to combine two feasible solutions in a way that results in a new feasible solution that resembles its parents in some way.

For OD matrix estimation or network design, this could be the average of the two parents.

How to mutate solutions?

The same way as neighbors were generated in simulated annealing: randomly change the solution in a minor way.