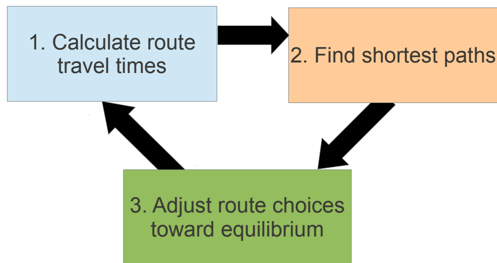
Solving for dynamic user equilibrium

CE 392D

Overall DTA problem



We can envision each of these steps as a "black box." Chapters 10 and 11 covered boxes 1 and 2, what do we need to do to close the loop?

This course will go over three methods for switching drivers between routes:

- Convex combinations
- Simplicial decomposition
- Gradient projection

In contrast to static assignment, all of these are *heuristics*. They function well enough for practice, but convergence guarantees are few and far between and involve severe restrictions on the network loading or network structure.

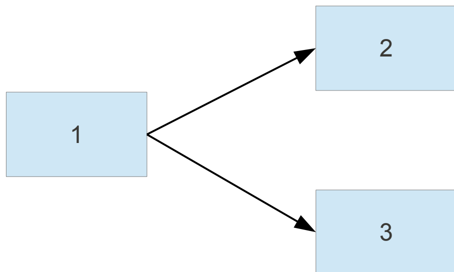
Recall the principle of dynamic user equilibrium:

All routes used by travelers leaving the same origin **at the same time** for the same destination have *equal* and *minimal* travel time.

If we have departure time choice, then all travelers leaving the same origin for the same destination have equal and minimal generalized cost, regardless of their departure time and path.

CONVEX COMBINATIONS

Let H be a matrix of time-dependent path flows



Cell data:	1	2	3
Capacity	20	5	5
Max. vehs.	30	10	10
w/v	1	1	1

Inflow to cell 1:	pi1	pi2
Interval 0	10	0
Interval 1	5	5
Interval 2	0	10

$$H = \begin{bmatrix} 10 & 0 \\ 5 & 5 \\ 0 & 10 \end{bmatrix}$$

One approach

- Start with path flows H_0
- Use a flow model (PQ, CTM, LTM) to simulate traffic flow
- Calculate travel time matrix T (showing travel time on each path and departure time)
- Get target path flow matrix H^* placing everybody on the shortest path available at their departure time
- Get updated path flow matrix by taking a weighted average of H_0 and H^*
- Repeat flow model, etc.

For instance, after simulating path flows $H = \begin{bmatrix} 10 & 0 \\ 5 & 5 \\ 0 & 10 \end{bmatrix}$, let's say we get travel times $T = \begin{bmatrix} 14 & 13 \\ 17 & 18 \\ 22 & 24 \end{bmatrix}$

Then $H^* = \begin{bmatrix} 0 & 10 \\ 10 & 0 \\ 10 & 0 \end{bmatrix}$

We now update the path flow matrix $H \leftarrow \lambda H^* + (1 - \lambda)H$ where $\lambda \in [0, 1]$ is the weight or "step size"

If $\lambda = 1/2$ then $H = \begin{bmatrix} 5 & 5 \\ 7.5 & 2.5 \\ 5 & 5 \end{bmatrix}$

If $\lambda = 1/5$ then $H = \begin{bmatrix} 8 & 2 \\ 6 & 4 \\ 2 & 8 \end{bmatrix}$

How do we choose λ?

There are two ways to go wrong. If λ is “too big”, then we are overcorrecting.

If λ is “too small”, then it will take a very long time to finish (if at all).

The *method of successive averages* tries to prevent both problems by starting with large λ values and moving to smaller ones.

If λ_i is the step size for the *i*-th iteration, $\{\lambda_i\} = \{1/2, 1/3, 1/4, 1/5, \ldots\}$.

We can choose other patterns for the step sizes, but will require $\sum \lambda_i = \infty$ and $\sum \lambda_i^2 < \infty$. (Why?)

We can also try to choose λ more intelligently: what value of λ brings us closest to equilibrium (lowest AEC)?

Unfortunately, there is no easy way to find this λ value. (This is different than in static assignment.)

A line search is usually the only option (trial and error on λ).

It is also possible to use different λ for different OD pairs and different times:

1. Use higher λ for earlier departure times than later.
2. Use higher λ values when travel times differ more.
3. etc.

Sometimes these help, sometimes they don't. Welcome to the world of dynamic traffic assignment.

SIMPLICIAL DECOMPOSITION

Simplicial decomposition is more sophisticated than convex combination algorithms: rather than "forgetting" the H^* matrices from previous iterations, we will save them

The set $\mathcal{H} = \{H_1^*, H_2^*, H_k^*\}$ stores the H^* matrices from each iteration (ignoring duplicates).

Why the name simplicial decomposition?

The key step in the algorithm is finding a "restricted equilibrium" only using the set $\mathcal{H}$ (rather than the set of *all* possible H matrices)

We say a matrix H is feasible if its entries are all nonnegative, and the sum of each row is the total demand leaving at each time interval.

If $H_1, H_2, \ldots, H_k$ are feasible, then the *convex combination* $\lambda_1 H_1 + \lambda_2 H_2 + \ldots + \lambda_k H_k$ is feasible if $\sum_{i=1}^k \lambda_i = 1$ and $\lambda_i > 0$.

We can frame the problem this way: Find $\lambda_1 \ldots \lambda_k$ such that $\sum_{i=1}^k \lambda_i = 1$, $\lambda_i > 0$, and $\lambda_1 H_1^* + \lambda_2 H_2^* + \ldots + \lambda_k H_k^*$ is a restricted equilibrium..

The set of λ_i satisfying the conditions $\sum_{i=1}^{k} \lambda_i = 1$ and $\lambda_i > 0$ is a $(k-1)$-dimensional *simplex*.

A simplex extends the concept of a triangle to higher dimensions.

Simplicial Decomposition Algorithm

This algorithm has two components: the **master algorithm** and a **subproblem**

Master algorithm:

1. Initialize the set $\mathcal{H} \leftarrow \emptyset$
2. Find shortest paths at all departure times
3. Form the all-or-nothing assignment H^* based on shortest paths
4. If H^* is already in $\mathcal{H}$, stop.
5. Add H^* to $\mathcal{H}$.
6. **Subproblem:** Find a restricted equilibrium H using only the matrices in $\mathcal{H}$.
7. Return to step 2.

Assuming we can solve the subproblem, this algorithm is guaranteed to converge. Why?

There are only finitely many all-or-nothing assignments.

After each iteration, we add another matrix to $\mathcal{H}$.

Eventually, we'll have them all and the algorithm will terminate.

Furthermore, when the algorithm terminates, we have found the equilibrium. Why?

If the shortest paths with respect to H are already in $\mathcal{H}$, then the restricted equilibrium is also an unrestricted equilibrium.

So, everything hinges on the subproblem: Find $\lambda_1 \ldots \lambda_k$ such that $\sum_{i=1}^{k} \lambda_i = 1$, $\lambda_i > 0$, and $\lambda_1 H_1^* + \lambda_2 H_2^* + \ldots + \lambda_k H_k^*$ is a restricted equilibrium.

We can use Smith's algorithm to solve the subproblem.

Note: Smith's algorithm assumes that $T(H)$ is continuously differentiable and monotone. In general, DTA models do not satisfy these properties, so Smith's algorithm is not guaranteed to find a restricted equilibrium. However, it usually works well in practice.

Smith's algorithm

Given a solution H, improve it in the following manner:

1. Find an improvement direction ΔH
2. Update $H \leftarrow H + \mu\Delta H$, where μ is chosen such that AEC is smaller after the update.

If μ is small enough, the new H will be feasible and AEC will be lower. Often we test different values of μ until we find a "successful" one, e.g., 1, $1/2$, $1/4$, etc.

How do we find the improvement direction?

Define the matrix dot product in the same way as the vector dot product, that is, $H \cdot T = \sum_\pi \sum_\tau h_\tau^\pi t_\tau^\pi$.

Let $[\cdot]^+ = \max\{\cdot, 0\}$ be the "positive" operator.

Then

$$\Delta H = \frac{\sum_{H_i^* \in \mathcal{H}} [T(H) \cdot (H - H_i^*)]^+ (H_i^* - H)}{\sum_{H_i^* \in \mathcal{H}} \left[T(H) \cdot (H - H_i^*)\right]^+}$$

is an improvement direction.

Proof: Smith, 1983.

How many improvement directions do we take?

At some point, we stop finding improvement directions with the current set $\mathcal{H}$, then return to the master algorithm to add another vector to $\mathcal{H}$.

We'll stop when H is "close enough" to being a restricted equilibrium (based on a *restricted AEC*, defined next).

Deciding when to stop the subproblem is a bit of an art. Often it doesn't pay to solve the subproblem to a high level of accuracy when $\mathcal{H}$ is still small

The *restricted AEC* is defined relative to $\mathcal{H}$.

$$AEC' = \frac{H \cdot T(H) - \min_{H_i^* \in \mathcal{H}}\{H_i^* \cdot T(H)\}}{||H||}$$

where $||H||$ is the total demand (sum of all the elements in H)

Why is this similar to the regular definition of AEC?

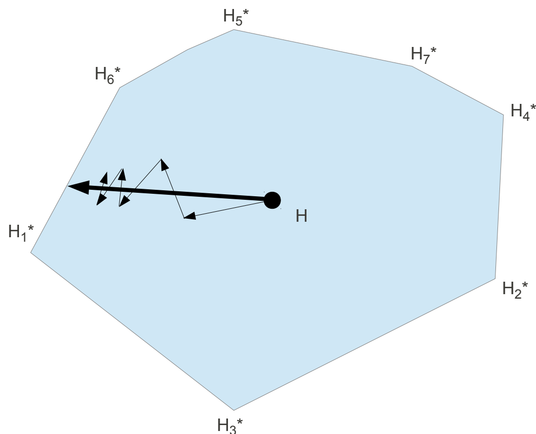
Master algorithm:

1. Initialize the set $\mathcal{H} \leftarrow \emptyset$
2. Find shortest paths at all departure times
3. Form the all-or-nothing assignment H^* based on shortest paths
4. If H^* is already in $\mathcal{H}$ (or AEC small enough), stop.
5. Add H^* to $\mathcal{H}$.
6. **Subproblem:** Find a restricted equilibrium H using only the matrices in $\mathcal{H}$.
    1. Find the improvement direction ΔH
    2. Update $H \leftarrow H + \mu \Delta H$, with μ sufficiently small (to reduce AEC').
    3. Return to step 1 of subproblem unless AEC' is small enough.
7. Return to step 2.

Review: does this algorithm always terminate? with the correct answer?

Example

Comparing simplicial decomposition and convex combinations



By combining multiple directions, simplicial decomposition can reach equilibrium faster.

GRADIENT PROJECTION

This method is based directly on the equilibrium principle

We want to choose H such that all of the used paths have equal and minimal costs.

Let's deal with a simpler case first: there are only two paths, the paths don't overlap, and there is just one departure time.

Then T_1 is a function of H_1 alone, and T_2 is a function of H_2 alone.

We want to choose H_1 and H_2 such that $T_1(H_1) = T_2(H_2)$, and $H_1 + H_2 = D$ where D is the travel demand.

Eliminating H_2, we need to solve

$$T_1(H_1) = T_2(D - H_1)$$

We use Newton's method to solve this equation numerically.

Newton's method finds the zero of a function f iteratively, with $x \leftarrow x - f(x)/f'(x)$

What are f and f' for our case?

We have $f(H_1) = T_1(H_1) - T_2(D - H_1)$, so
$f'(H_1) = dT_1/dH_1 + dT_2/dH_2$ (why?)

Intuitively: when we shift a unit of flow from H_1 to H_2, T_1 *decreases* by dT_1/dH_1, and T_2 *increases* by dT_2/dH_2.

Therefore, the *difference* in travel times changes by $dT_1/dH_1 + dT_2/dH_2$. This is the *gradient* part.

We also need to make sure that H_1 and H_2 are nonnegative.

If H_1 or H_2 is negative after a Newton shift, set its flow equal to zero, and assign all demand to the other. This is the *projection* part.

You should be asking the following questions:

- What if there are more than two paths?
- What if there is more than one departure time?
- How can we calculate derivatives in DTA?
- What if the paths overlap?

MULTIPLE PATHS

Assume there are k non-overlapping paths, with travel time functions $T_i(H_i)$.

Then we can eliminate one path flow variable using the constraint $\sum_i H_i = D$, as before. WLOG assume that path k is the shortest path; we choose this path to eliminate.

The system of equations is then

$$T_1(H_1) = T_k(D - H_1 - \cdots - H_{k-1}) \quad (1)$$

$$\vdots \quad (2)$$

$$T_{k-1}(H_{k-1}) = T_k(D - H_1 - \cdots - H_{k-1}) \quad (3)$$

Newton's method can be applied to multidimensional functions.

We want to find a zero of $f : \mathbb{R}^n \to \mathbb{R}^n$. Let $x \in \mathbb{R}^n$ be an n-dimensional vector.

The Newton iteration is now

$$x \leftarrow x - (Jf(x))^{-1} f(x)$$

where $Jf(x)$ is the Jacobian of f evaluated at x

Calculating the inverse Jacobian is computationally expensive. In multidimensional cases, we often use a *quasi-Newton* method based on an approximation to Jf or its inverse.

In our case, $f(H) = \begin{bmatrix} T_1 - T_k \\ T_2 - T_k \\ \vdots \\ T_{k-1} - T_k \end{bmatrix}$.

$$Jf = \begin{bmatrix} \frac{dT_1}{dH_1} + \frac{dT_k}{dH_k} & \frac{dT_k}{dH_k} & \cdots & \frac{dT_k}{dH_k} \\ \frac{dT_k}{dH_k} & \frac{dT_2}{dH_2} + \frac{dT_k}{dH_k} & \cdots & \frac{dT_k}{dH_k} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{dT_k}{dH_k} & \frac{dT_k}{dH_k} & \cdots & \frac{dT_{k-1}}{dH_{k-1}} + \frac{dT_k}{dH_k} \end{bmatrix}$$

.

We take the approximation

$$Jf = \begin{bmatrix} \frac{dT_1}{dH_1} + \frac{dT_k}{dH_k} & 0 & \cdots & 0 \\ 0 & \frac{dT_2}{dH_2} + \frac{dT_k}{dH_k} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \frac{dT_{k-1}}{dH_{k-1}} + \frac{dT_k}{dH_k} \end{bmatrix}$$

This matrix is easy to invert, so

$$(Jf)^{-1} \approx \begin{bmatrix} \left(\frac{dT_1}{dH_1} + \frac{dT_k}{dH_k}\right)^{-1} & 0 & \cdots & 0 \\ 0 & \left(\frac{dT_1}{dH_1} + \frac{dT_k}{dH_k}\right)^{-1} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \left(\frac{dT_{k-1}}{dH_{k-1}} + \frac{dT_k}{dH_k}\right)^{-1} \end{bmatrix}$$

.

Then the Newton update $x \leftarrow x - (Jf(x))^{-1} f(x)$ is simply the following:

$$H_i \leftarrow \left[H_i - \frac{T_i - T_k}{dT_i/dH_i + dT_k/dH_k} \right]^+ \qquad \forall i \neq k$$

$$H_k \leftarrow D - H_1 - \cdots - H_{k-1}$$

(Essentially, perform the two-path Newton update between each path and the shortest path without updating the travel times in between.)

WHAT ABOUT DIFFERENT DEPARTURE TIMES?

The major complication with different departure times is the same as with overlapping paths: T_k is not just a function of H_k alone, but also depends on other entries in the path flow matrix.

Simple solution: Pretend like this problem doesn't exist, and apply the multi-path quasi-Newton method for each departure time.

Slightly more complex solution: Introduce a step size μ and update as follows:

$$H_i \leftarrow \left[H_i - \mu \frac{T_i - T_k}{dT_i/dH_i + dT_k/dH_k} \right]^+ \qquad \forall i \neq k$$

$$H_k \leftarrow D - H_1 - \cdots - H_{k-1}$$

with $0 < \mu <= 1$. Typically start with $\mu = 1$ and reduce it if AEC gets stuck at a positive value.

HOW DO WE CALCULATE DERIVATIVES IN DTA?

How to calculate dT_i/dH_i depends on the traffic flow model.

Consider a path π and departure time τ, and track the flow when it reaches the downstream end of the link.

If there is no queue when the path trajectory reaches the downstream end of a link, we can add additional vehicles without causing one... the derivative of the travel time on that link is zero.

If there is a queue, adding an extra vehicle to the link will increase the travel time by $1/q^{\downarrow}$, where $q^{\downarrow}$ is the link outflow rate.

Therefore, with the PQ model, $dT_{\tau,\pi}/dH_{\tau,\pi} = \sum 1/q_a^{\downarrow}$, where the sum is over all links a on π which have a queue when a vehicle departing at time τ reaches the downstream end of a.

WHAT ABOUT OVERLAPPING PATHS?

One special case of overlapping paths is easier to treat. If π_1 and π_2 start with the same set of links before diverging, we know that switching flow between these paths *will not affect travel times upstream of the diverge*, so we only need to calculate derivatives after π_1 and π_2 diverge.