Faster equilibrium algorithms:
Algorithm B

CE 392C

November 1, 2016
ANNOUNCEMENTS
Project abstracts due this week

HW 4 due today
REVIEW
Faster methods for finding equilibrium

1. What’s wrong with Frank-Wolfe on large networks
GRADIENT PROJECTION
Gradient projection is an example of a *path-based algorithm*, which tracks the path flows $h$ in addition to the link flows $x$.

Since there can be billions and billions of paths in a network, we don’t want to keep track of literally every path’s flow.

Instead, we define a set of working paths $\hat{\Pi}^{rs}$ for each OD pair, and only track the flows on these paths. We’ll let this set grow and shrink over successive iterations.
A general scheme for path-based algorithms is:

1. Initialize $\hat{\Pi}^{rs} \leftarrow \emptyset$ for all OD pairs.
2. Find the shortest path $\pi^*_{rs}$ for each OD pair. Add it to $\hat{\Pi}^{rs}$ if it’s not already used.
3. Within each OD pair, shift travelers among paths to get closer to equilibrium.
4. Update travel times; drop paths from $\hat{\Pi}^{rs}$ if they are no longer used; return to step 2.

This should remind you of the trial-and-error method, with a “relaxed” step 3 and the “try again” steps spelled out more clearly.
Step 3 is where path-based algorithms usually differ.

The gradient projection algorithm uses Newton’s method to try to move closer to an equilibrium.
NEWTON’S METHOD FOR GRADIENT PROJECTION
There are several ways to perform step 3. Here is one method.

For each OD pair, define the *basic path* to be the shortest path in $\hat{\Pi}^{rs}$. All other paths are *nonbasic*.

For each nonbasic path, perform one step of Newton’s method with the basic path to try to equalize their travel times.
Consider any two paths $\pi$ and $\pi^*$. Let $\Delta h$ be the amount of flow to shift away from $\pi$ and onto $\pi^*$.

Let $C_\pi(\Delta h)$ and $C_{\pi^*}(\Delta h)$ be the travel times on these paths after shifting $\Delta h$ flow.

Defining $g(\Delta h) = C_\pi(\Delta h) - C_{\pi^*}(\Delta h)$, a zero of $g$ corresponds to equal travel times on these two paths.
What is $g'(\Delta h)$?

$$g(\Delta h) = \sum_{(i,j) \in \mathcal{A}} (\delta_{ij}^{\pi} - \delta_{ij}^{\pi^*}) t_{ij}(x_{ij}(\Delta h))$$

so

$$g'(\Delta h) = \sum_{(i,j) \in \mathcal{A}} (\delta_{ij}^{\pi} - \delta_{ij}^{\pi^*}) \frac{dt_{ij}}{dx_{ij}} \frac{dx_{ij}}{d\Delta h}$$

For each link in the sum, there are four cases.
Case I : \( \delta_{ij}^{\pi} = \delta_{ij}^{\pi^*} = 0 \). Then \( (\delta_{ij}^{\pi} - \delta_{ij}^{\pi^*}) \frac{dt_{ij}}{dx_{ij}} \frac{dx_{ij}}{d\Delta h} = 0 \).

Case II : \( \delta_{ij}^{\pi} = \delta_{ij}^{\pi^*} = 1 \). Then \( (\delta_{ij}^{\pi} - \delta_{ij}^{\pi^*}) \frac{dt_{ij}}{dx_{ij}} \frac{dx_{ij}}{d\Delta h} = 0 \).

Case III : \( \delta_{ij}^{\pi} = 1 \) and \( \delta_{ij}^{\pi^*} = 0 \). Then \( (\delta_{ij}^{\pi} - \delta_{ij}^{\pi^*}) \frac{dt_{ij}}{dx_{ij}} \frac{dx_{ij}}{\Delta h} = -\frac{dt_{ij}}{dx_{ij}} \).

Case IV : \( \delta_{ij}^{\pi} = 0 \) and \( \delta_{ij}^{\pi^*} = 1 \). Then \( (\delta_{ij}^{\pi} - \delta_{ij}^{\pi^*}) \frac{dt_{ij}}{dx_{ij}} \frac{dx_{ij}}{\Delta h} = -\frac{dt_{ij}}{dx_{ij}} \).

In short, the only links contributing to the derivative \( g' \) are those which appear in either \( \pi \) or \( \pi^* \), but not both. These are the only links whose flow values will change when we shift travelers from \( \pi \) to \( \pi^* \).
The derivative can be written

\[ g'(\Delta h) = - \sum_{(i,j) \in A_3 \cup A_4} \frac{dt_{ij}}{dx_{ij}} \]

where \( A_3 \) and \( A_4 \) are the sets of links falling into Case III and Case IV on the previous slide. (This is the “gradient” part.)

Then, using the Newton’s method formula with an initial guess \( \Delta h_0 = 0 \), the amount of flow to shift is

\[ \Delta h = -\frac{g(0)}{g'(0)} = \frac{C_\pi - C_{\pi^*}}{\sum_{a \in A_3 \cup A_4} \frac{dt_{ij}}{dx_{ij}}} \]

We do need to make sure that the flow on \( \pi \) remains nonnegative:

\[ \Delta h = \min \left\{ h_\pi, \frac{C_\pi - C_{\pi^*}}{\sum_{a \in A_3 \cup A_4} \frac{dt_{ij}}{dx_{ij}}} \right\} \]

(This is the “projection.”)
Complete algorithm

A general scheme for path-based algorithms is:

1. Initialize $\hat{\Pi}^{rs} \leftarrow \emptyset$ for all OD pairs.
2. For each OD pair $(r, s)$:
   1. Find the shortest path $\pi_{rs}^*$. Add it to $\hat{\Pi}^{rs}$.
   2. If there is only one path $\pi_{rs}$ in $\hat{\Pi}^{rs}$, set $h_{rs}^{\pi} \leftarrow d_{rs}$. Otherwise, for each non-basic path $\pi_{rs} \neq \pi_{rs}^*$, adjust path flows with
      \[
      h_{\pi}^* \leftarrow h_{\pi}^* + \min \left\{ h_{\pi}, \frac{C_{\pi} - C_{\pi}^*}{\sum_{a \in A_3 \cup A_4} \frac{dt_{ij}}{dx_{ij}}} \right\}
      \]
      and
      \[
      h_{\pi} \leftarrow h_{\pi} - \min \left\{ h_{\pi}, \frac{C_{\pi} - C_{\pi}^*}{\sum_{a \in A_3 \cup A_4} \frac{dt_{ij}}{dx_{ij}}} \right\}
      \]
3. Update travel times
3. Update travel times; drop paths from $\hat{\Pi}^{rs}$ if they are no longer used; check convergence; return to step 2.

Algorithm B
Newton’s Method for Gradient Projection
Example

Algorithm B
Newton’s Method for Gradient Projection
If you recall from the user equilibrium notes, after three iterations the FW method found an AEC of 1.56 minutes. Three iterations of gradient projection produced AEC of 0.17 minutes.

Furthermore, if we were continue further, the relative advantage of gradient projection would only grow.
BUSH-BASED METHODS
Recall that *link-based methods* require very little memory, but are also slow to converge.

By contrast, *path-based methods* are much faster, but can potentially require a large amount of memory.

The Chicago regional network has roughly 90 million equilibrium paths; each path contains several dozen links on average.
Bush-based methods involve substantially less memory than path-based methods, and don’t seem to be any slower. Some evidence even suggests that they are faster.

Caveat: It is relatively difficult to compare the most advanced algorithms; much of their performance depends on subtle implementation details.
Remember that a bush is an acyclic subnetwork in which every node can be reached from the root.

At the equilibrium solution, the set of all paths used by a given origin forms a bush. (Why?)

In a bush-based algorithm, we maintain a set of bushes, one for each origin. We have two goals:

1. Find the *right bush* for each origin (containing all of the paths used at equilibrium).
2. Find the *link flows* on the bush links which satisfy the equilibrium principle.

Origins are only allowed to place flow on links in the bush.
All bush-based algorithms perform the following steps:

1. Create an initial bush for each origin (easy way: start with shortest paths using free-flow times)
2. Shift flows within each bush to move it closer to equilibrium.
3. Update the bushes to remove unused links, and to add links which provide shorter paths.

Step 2 is where most bush-based algorithms differ.
Some bush-based algorithms:

- Origin-based assignment (Bar-Gera, 1999–2000; Nie, 2009)
- Algorithm B (Dial, 1999–2006)
- LUCE (Gentile, 2009)
- TAPAS (Bar-Gera, 2011)
ALGORITHM B
Algorithm B adopts the following simple rule for step 2:

For each destination, find the *longest used path* from the origin as well as the *shortest path* on the bush. Use Newton’s method to shift flow between these paths to move towards equilibrium.

The Newton shift works in the same way as for gradient projection, but the bush structure makes it easier to find longest and shortest paths.
This relies on the acyclic property of bushes: longest and shortest paths can be found by simply going over the network in topological order.

By contrast, if there are cycles, shortest path is a bit harder, and finding longest paths is much harder.
In step 3, bushes are updated using the following rules:

- If a bush link has zero flow, it is removed (unless doing so would disconnect a node from the root)
- If a non-bush link “provides a shortcut”, it is added.

Specifically, if $L_i$ is the travel time on the shortest path to node $i$ only using bush links, a link $(i, j)$ is a shortcut if $L_i + t_{ij} < L_j$.

Will adding links according to this rule create a cycle?
\[ t_{ij} = 3 + \left(\frac{x_{ij}}{200}\right)^2 \] on thin links

\[ t_{ij} = 5 + \left(\frac{x_{ij}}{100}\right)^2 \] on thin links

OD matrix: \( d^{49} = 1000, d^{49} = 1000 \)
Free flow travel times

Algorithm B
Initial bushes

Algorithm B

Algorithm B
These bushes are at equilibrium because there is only one path from each origin to the destination.
Initial link flows
Travel times

Algorithm B
Bush 1
Add shortcuts
Add shortcuts to the bush for origin 1
The pair of alternate segments is \{[1, 4, 5, 6], [1, 2, 3, 6]\}.
Flow on [1, 4, 5, 6]: 0
Flow on [1, 2, 3, 4]: 1000
Travel time on [1, 4, 5, 6]: 13
Travel time on [1, 2, 3, 4]: 84
Derivative on [1, 4, 5, 6]: 0
Derivative on [1, 4, 5, 6]: 0.15

Newton shift is given by
\[
\frac{84 - 13}{0 + 0.15} = 473
\]
Newton shift is given by

\[
\frac{84 - 13}{0 + 0.15} = 473
\]
Flow on [1, 4, 5, 6]: 473
Flow on [1, 2, 3, 4]: 527
Travel time on [1, 4, 5, 6]: 63.4
Travel time on [1, 2, 3, 4]: 29.8
Derivative on [1, 4, 5, 6]: 0.213
Derivative on [1, 4, 5, 6]: 0.079

Newton shift is given by
\[
\frac{63.4 - 29.8}{0.079 + 0.213} = 115
\]
Newton shift is given by

\[
\frac{63.4 - 29.8}{0.079 + 0.213} = 115
\]
Flow on [1, 4, 5, 6]: 358
Flow on [1, 2, 3, 4]: 642
Travel time on [1, 4, 5, 6]: 41.9
Travel time on [1, 2, 3, 4]: 39.9
Derivative on [1, 4, 5, 6]: 0.161
Derivative on [1, 4, 5, 6]: 0.097

Newton shift is given by

\[
\frac{41.9 - 39.9}{0.161 + 0.097} = 7.71
\]
Newton shift is given by

\[
\frac{41.9 - 39.9}{0.161 + 0.097} = 7.71
\]
Flow on [1, 4, 5, 6]: 350
Flow on [1, 2, 3, 4]: 650
Travel time on [1, 4, 5, 6]: 40.64
Travel time on [1, 2, 3, 4]: 40.65
Derivative on [1, 4, 5, 6]: 0.158
Derivative on [1, 4, 5, 6]: 0.097

Newton shift is 0.034, close enough to equilibrium.