**Implementation of the Frank-Wolfe Algorithm**

Wei Zou

3/31/2016

# Background

The purpose this project is to implement the Frank-Wolfe Algorithm for transportation network analysis. The next section summarizes the key steps involved in the Python coding process, followed by two traffic assignment applications. The report is concluded with a discussion of findings and future plans.

# Implementation Process

The Frank-Wolfe Algorithm is a popular method used for user equilibrium transportation network analysis. There are two key steps involved in the algorithm:

1. Descent direction search: maximize the \*drop\* (product of the rate of descent in a given direction and the length of feasible move in that direction); popular methods include linear programming and Dijstra’s shortest path algorithm;
2. Line search: since the bounding of the move size is naturally accomplished in the descent direction search process, simple interval reduction methods could be used to find the optimal step size.

The above steps are implemented repeatedly until a convergence is reached.

Python is used to replicate the process described above in the following steps (for more details, please refer to the comments in the code):

1. Use array data structures to represent the network and given information;

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Input Datasets | Python Variable | Definition | Data Structure | Size |
| Network Information: Link & Node | n, k | n: # of links  k: # of nodes | Python number | 1 |
| Network Information: Link & Node | LinkNode | Link-node incidence matrix (adjacency) | Python array | nparray(n, k) |
| OD flow demand (q\_ij) | Q | Travel flow demand between each OD pair | Python array | nparray(n,n) |
| Link travel time information: free flow time, parameters for the BPR function, link capacity | coeff | Includes two columns, column 1 is for free flow travel time on each link (t0); column 2 is for link capacity (ca), other parameters used in the BPR function (e.g., alpha, beta) are constants | Python array | nparray(n,2) |

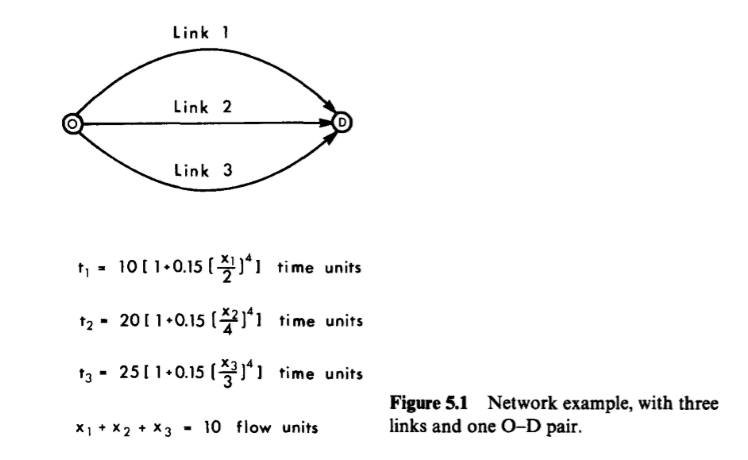
1. Conduct the descent direction search by forming a Linear Programming problem and solve it;

* Initialization: carry out the All or Nothing flow assignment (find xa) based on the free flow travel time on each link (t0), then update the travel time on each link (ta)
* Iteration (direction search): formulate the objective function (Z) and constraints for the LP program, then use the \*optimize.linprog\* function in Python package to solve the problem and obtain updated link flow (ya)

1. Find the optimal step size using the Golden section interval reduction method for the one-dimensional search;
   * Iteration (move): formulate the line search function and use the \*minimize\_scalar\* function in Python package to solve the step size alpha
2. Update the network and check for convergence.
   * Iteration (update): update link flow (xa) using ya and alpha, then use xa to update ta
   * Iteration (check convergence): calculate the norm of ta(iteration n) and ta(iteration n-1), when Norm < 0.1\*n (on average the link flow changes 0.1), the iteration stops and the algorithm is converged

# Applications

Two applications are carried using the python code discussed in the previous section.

Application 1: Simple case with one pair of OD 

(Ref: Sheffi, Yosef. "Urban transportation networks." (1985), pg114)