University of California, Irvine

CS 169/268 : Final Project Report

Solving Traffic Assignment Problem with Frank-Wolfe Algorithm

Team Members
Siwei Hu
Tyler Ryan Staton
Miguel Angel Meza-Ponce

Course Instructor
Eric D. Mjolsness

December 18, 2020

Abstract

In this report, the Frank-Wolfe algorithm is applied to solve traffic assignment problem. The traffic assignment problem is first introduced in Chapter 1 with its basic principle, *User Equilibrium*, where all used paths have the same lowest travel time. Besides, traffic assignment problem can be mathematically formulated as a minimization programs and reviewed in section 1.2. Following that, an example is introduced to help readers further understand the mathematical formulation of traffic assignment problem in section 1.3.

To solve the traffic assignment problem, the heuristic and deterministic methods are proposed by researchers, and they are reviewed and compared in Chapter 2. Since heuristic methods can neither guarantee convergence, nor *User Equilibrium* solutions, Frank-Wolfe algorithm, one of deterministic methods is chosen to implemented in this project. Frank-Wolfe algorithm, an iterative linear approximation optimization algorithm for constrained optimization, is first reviewed and analyzed in Chapter 3. Chapter 4 is about the project decomposition and group member responsibilities, while Chapter 5 is about group member's experience when they coded this project.

Frank-Wolfe algorithm is first implemented in a 4-node small network and then in a 24-node large Sioux-Falls network in Chapter 6. And the result both shows that Frank-Wolfe algorithm can reach a convergence and guarantee a *User Equilibrium* solution quickly.

Contents

List of Figures

List of Tables

1 Introduction

When it comes to transportation planning and transportation system analysis, it is necessary for researchers to understand the flow pattern in the transportation network, given certain travelers demand patterns. After predicting the travel demand of residents, researchers and transportation agencies might want to know how these travelers are distributed in the transportation network, in order to predict the congestion level caused by travelers. After understanding how travelers are distributed in the network, researchers will know how congested the network is and they can do some congestion control and management accordingly.

Traffic Assignment Problem is to anwser such kind of problems - given certain demand patterns, how are travelers distributed in the transportation network? The basic elements of Traffic Assignment Problem include demand part - how many travelers are "assigned" into the road network, and supply part - how many travelers can the network "support".

Since congestion increases as the traffic flow increases, while the travel demand can be discouraged by the congestion, this process can be modeled as a process of reaching **equilibrium** between congestion and travel decisions [1].

1.1 Problem statement

Traffic assignment problems deal with the problem below:

Given a certain origin-destination (OD) demand and the whole transporation network, how is the traffic flow distributed in the whole transportation network? For example, from Origin Irvine to Destination LA, there are 2000 trips per hour, how are these trips distributed in the network connecting Irvine and LA, shown in Figure 1.1 [1]? That is the question we could answer by solving traffic assignment problems.

To follow the discussion above, main elements of traffic assignment problem will be introduced below:

1. Network, $G(v, e)$, a graph G with vertices (nodes) v and edges (links) e;

[1]Source: $https://en.wikipedia.org/wiki/Southern\_California\_freeways$

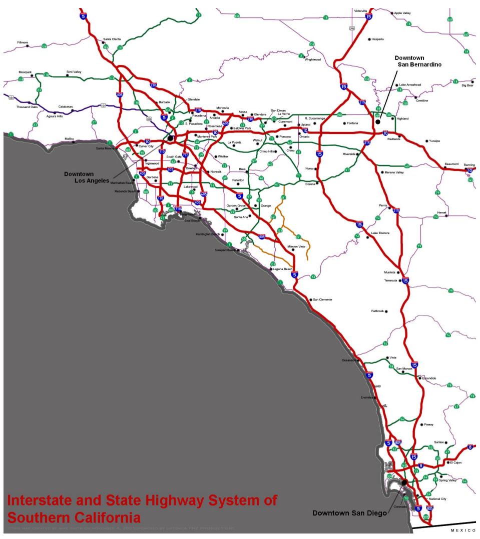


Figure 1.1: How are these trips distributed in the network?

2. Origin-Destination (OD) demand matrix, which reflects how many trips will go from a certain origin to a certain destination;

3. Link performance function, $t_a(x_a)$, which measures the relationship between the traffic flow, x_a, (in vehicle/hour) and the congestion level, link/edge travel time, $t_a(x_a)$, (in minutes), for a cetain edge a;

4. Behavior assumption: User Equilibrium. As mention above, the interactions between congestion levels and travel demand can reach an equilibrium, which transportation researcher called "*User Equilibrium*". *User Equilibrium* is first proposed by Wardrop [2], which is equivalent to Nash Equilibrium, and includes 2 major patterns:

   a) For a certain origin-destination pair, all the utilized paths have the same travel times

   b) Travel times for all the utilized paths are less than or equal to those for the unutilized paths

   Namely in mathematics, $c_k \leq c_{k'}$, where c_k and is the path travel cost for path k. And k represents the utilized paths, k' represents the unutilized paths.

1.2 Mathematical formulation

The mathematical formulation of traffic assignment problem was first proposed by Beckmann, McGuire, and WinstenUser in 1955 [3]. Traffic assignment problem with **User**

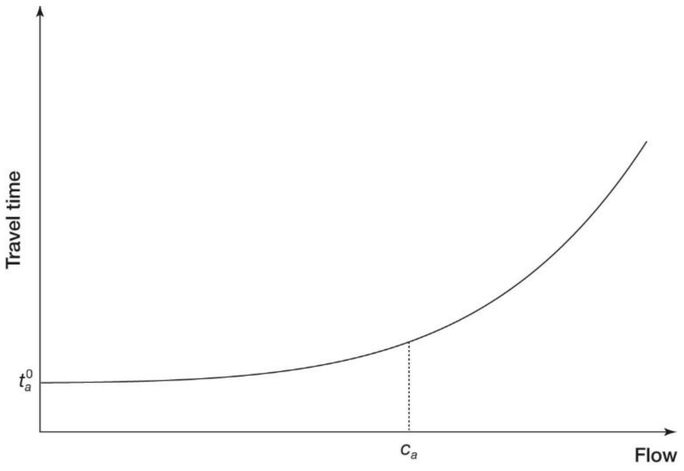


Figure 1.2: An example of link performance function

Equilibirum principles can be formulated as a minimization problem as following:

$$min \sum_a \int_0^{x_a} t_a(y)dy \tag{1.1}$$

$$s.t. \sum_k f_k^{rs} = q^{rs} \quad \forall v, s \tag{1.2}$$

$$f_k^{rs} \geq 0 \quad \forall k, v, s \tag{1.3}$$

$$x_a = \sum_r \sum_s \sum_k f_k^{rs} \delta_{a,k}^{rs} \quad \forall a \tag{1.4}$$

, where a represent a certain link in the network, and $t_a(\cdot)$ ($t_a(y)$ in Equation (1.2)) is the link performance function. It can be usually written as:

$$t_a(x_a) = t_0(1 + \alpha(\frac{x_a}{C_a})^\beta) \quad \forall a \tag{1.5}$$

, and it usually has the shape as shown in Figure 1.2. In Equation (1.5), x_a is the link a's traffic flow/volume (in vehicle/hour), C_a is the capacity of a certain link a (in vehicle/hour), the maximum of x_a. And t_0 is the free-flow travel time, which is the travel time for link a when there is no traffic on link a. α and β are parameters. Usually $\alpha = 0.15$ and $\beta = 4$.

Some other variables are described below:

1. In Equation (1.2), f_k^{rs} is the path flow of the kth path connecting origin r and destination s. q^{rs} is the origin-destination demand for origin-destination pair r and s (e.g. 2,000 trips/hour)

2. In Equation (1.4), $\delta^{rs}_{a,k}$ is the link-path incidence matrix, describing the incidental relationship for links a and path k given a origin-destination pair r and s. Given a certain origin-destination (OD) pair r and s, the link path incidence matrix, $\delta^{rs}_{a,k}$, is defined as:

$$\delta^{rs}_{a,k} = \begin{cases} 1, & \text{if link } a \text{ is incident on path } k \\ 0, & \text{otherwise} \end{cases} \tag{1.6}$$

1.3 An example of the mathematical formulation

In this section, a 4-node network example, shown in Figure 1.3, will be used to elaborate the parameters and variable of the mathematical formulation. From Figure 1.3, we could see this network contains 4 nodes and 5 directed links.

Suppose the The origin-destination (OD) demand flows are:

1. $1 \to 2$, 2 units of flow
2. $3 \to 2$, 2 units of flow

And the capacity of every link, C_a, are $1, 2, 3, 4, 5$ for link $1, 2, 3, 4, 5$, where $C_1 = 1$, $C_2 = 2$ and so on. The free flow travel time is assumed to be $t_0 = 1$. For this network, the link-path incidence matrix for origin-destination (OD) pair 1 and 2 will be:

$$\delta^{1,2}_{a,k} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix} \tag{1.7}$$

, where rows of this matrix represent links, and columns of it represent paths. For example, within OD pair $1, 2$, we have 3 different paths, $1 \to 2$, $1 \to 3 \to 2$, $1 \to 3 \to 4 \to 2$, so we

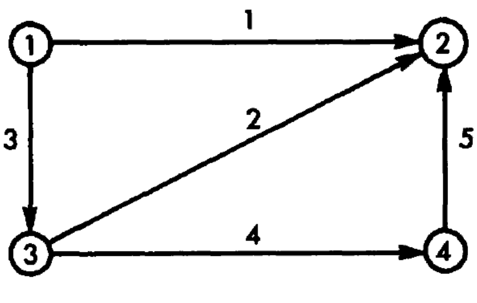


Figure 1.3: A 4 node 5 link small network

have 3 columns. For path 1 $1 \to 2$, there is only one link incident on this path, which is link 1. So in the column 1 of the link-path incidence matrix $\delta_{a,k}^{1,2}$, there is only one "1", which is in position $(1,1)$. And the link-path incidence matrix for origin-destination (OD) pair 3 and 2 will be:

$$\delta_{a,k}^{3,2} = \begin{pmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 0 \\ 0 & 1 \\ 0 & 1 \end{pmatrix} \tag{1.8}$$

Thus the mathematical formulation for this 4-node small network will become:

$$min \sum_{a=1}^{5} \int_0^{x_a} t_a(y)dy \tag{1.9}$$

$$s.t. \sum_k f_k^{12} = q^{12} \tag{1.10}$$

$$\sum_k f_k^{32} = q^{32} \tag{1.11}$$

$$f_k^{rs} \geq 0 \tag{1.12}$$

$$x_a = \sum_k f_k^{12} \delta_{a,k}^{12} + \sum_k f_k^{32} \delta_{a,k}^{32} \tag{1.13}$$

And this 4 node network traffic assignment problem with *user equilibrium* principles will be solved in section 6.1.

2 Related work on solving traffic assignment problem

In this chapter, several method for solving traffic assignment problem will be discussed, including heuristic approach and deterministic approach. The basic idea behind solving traffic assignment problem is network loading - assigning "traffic flows" onto the transportation network. Heuristic method "load" flows onto the network heuristically, deterministic methods "load" flows onto the network accurately.

2.1 Heuristic methods

In this section, two heuristic methods will be discussed, including capacity restraint method and incremental assignment method. Before introducing these two method, a more fundamental network loading method will be introduced: all-or-nothing assignment.

The basic idea behind all or nothing assignment is: load all the traffic flow onto the shortest path, load nothing onto non-shortest paths. As discussed in Chapter 1, congestion increases as flow increases, and discourage travel demand at the same time. Thus all-or-nothing assignment does not consider the effect of congestion caused by network loading.

2.1.1 Capacity restraint methods

To refine all-or-nothing assignment, capacity restraint methods are proposed and developed in 1960s [4][5][6]. According to Sheffi [1], the basic idea behind capacity restraint methods is to smooth travel time while doing all-or-nothing assignment, and averaging last 4 flows that are treated as "closest to equilibrium". Steps are summarized as follows [1]:

Step 0: Initialization. Perform an all-or-nothing assignment based on $t_a^0 = t_a(0)$. Obtain $\{x_a^0\}$. Set $n := 1$.

Step 1: Update. Set $\tau_a^n = t_a(x_a^{n-1}), \quad \forall a$.

[1]Page 113-114 in Sheffi's [1]

Step 2: Smoothing. Set $t_a^n = \theta t_a^{n-1} + (1-\theta)\tau_a^n, \quad \forall a, \quad where \theta \in [0,1]$.

Step 3: Network loading. Perform all-or-nothing assignment based on travel times $\{t_a^n\}$. This yields $\{x_a^n\}$.

Step 4: Stopping rule. If $n = N$, go to step 5. Otherwise, set $n := n+1$ and go to step 1.

Step 5: Averaging. Set $x_a^* = \frac{1}{4}\sum_{l=0}^{3} x_a^{l-1} \quad \forall a$ and stop.

The step 5 is to average the last 4 steps when the maximum iteration, N, is reached. However, this method can neither guarantee the convergence of traffic flow x_a while the algorithm proceeds, not guarantee a *User Equilibrium* solution, namely, all used paths have the same travel time. In order to solve this shortcomings, researchers proposed Incremental assignment methods. They will be discussed in next subsection.

2.1.2 Incremental assignment methods

Incremental assignment methods are proposed and developed during 1970s [7]. According to Sheffi [1], the basic idea behind it is to divide the total flow into parts and assign them incrementally. Steps are summarized as follows [2]:

Step 0: Preliminaries. Divide each origin-destination entry into N equal portions (i.e. set $q_{rs}^n = q_{rs}/N$). Set $n := 1$ and $x_a^0 = 0, \quad \forall a$.

Step 1: Update. Set $t_a^n = t_a(x_a^{n-1}), \quad \forall a$.

Step 2: Incremental loading. Perform all-or-nothing assignment based on $\{t_a^n\}$, but using only the trip rates q_{rs}^n for each $O-D$ pair. This yields a flow pattern $\{w_a^n\}$.

Step 3: Flow summation. Set $x_a^n = x_a^{n-1} + w_a^n, \quad \forall a$

Step 4: Stopping rule. If $n = N$, stop (the current link flows is the solution); otherwise, set $n := n+1$ and go to step 1.

Like capacity constraint method, incremental assignment method does not guarantee a convergence solution, nor guarantee an equilibrium solution. These methods just "give" a solution.

2.2 Deterministic methods

The failure of the heuristic methods discussed above motivates researchers to develop deterministic methods to solve for traffic assignment problems. Frank-Wolfe algorithm will be discussed in this section. Frank-Wolfe algorithm is the benchmark algorithm of solving traffic assignment problems, because traffic assignment problem with user equilibrium was first solved in 1975 by Frank-Wolfe algorithm [8]. Therefore, any algorithm developed

[2]Page 115 in Sheffi's [1]

later will be compared to Frank-Wolfe algorithm.

Frank-Wolfe algorithm was proposed in 1955 [9], which is an iterative first-order optimization algorithm for constrained convex optimization. And in each iteration, the Frank–Wolfe algorithm considers a linear approximation of the objective function, and moves towards a minimizer of this linear function (taken over the same domain).

Suppose D is a compact convex set in a vector space and $f\colon \mathcal{D} \to R$ is a convex, differentiable real-valued function. The Frank–Wolfe algorithm solves the optimization problem

$$Minimize \quad f(\mathbf{x}) \tag{2.1}$$

$$subject \quad to \quad \mathbf{x} \in \mathcal{D} \tag{2.2}$$

The main steps of Frank-Wolfe algorithm can be summarized as follows [9]: [3]

Step 0: Initialization: Let $k := 0$, and let x_0 be any point in $\mathcal{D}$

Step 1: Direction-finding subproblem. Find S_k solving:

$$Minimize \quad s^T \nabla f(\mathbf{x}) \tag{2.3}$$

$$subject \quad to \quad \mathbf{s} \in \mathcal{D} \tag{2.4}$$

Step 2: Step size determination: Find *gamma* tha minimizes $f(x_k + \gamma(s_k - x_k))$ subject to $0 \leq \gamma \leq 1$.

Step 3: Update: Let $x_{k+1} := x_k + \gamma(s_k - x_k)$, let $k := k + 1$ and go to Step 1.

Frank-Wolfe is very useful when the objective function is highly non-linear, because the linear approximation helps reduce the complexity of the problem a lot. And this algorithm works well when the feasible region is convex and compact. In traffic assignment cases, its objective function is 4 power function, and the constraints are linear, which must be convex and compact. So Frank-Wolfe algorithm works pretty well in solving traffic assignment problem.

[3]Source: $https://en.wikipedia.org/wiki/Frank\%E2\%80\%93Wolfe\_algorithm$

3 Methodology

The general Frank-Wolfe is discussed above. However, in traffic assignment problems, every step in Frank-Wolfe algorithm can have physical transportation meaning. For example, in the step 1 - direction finding, solving the linear approximation program is equivalent to doing an all-or-nothing assignment in network loading. When Frank-Wolfe algorithm is applied to traffic assignment problem, it has more transportation meanings, and can be related all-or-nothing concept discussed in the previous sections. The adopted Frank-Wolfe algorithm, bisection line search and the stopping criteria used in this project will be discussed in Section 3.1, Section 3.2 and Section 3.3.

3.1 Adopted Frank-Wolfe Algorithm

According to Sheffi [1],In transportation, the adopted Frank-Wolfe algorithm is summarized as follows [1]:

Step 0: Initialization. Perform all-or-nothing assignment based on $t_a^0 = t_a(0), \quad \forall a$. This yields $\{x_a^1\}$. Set counter $n := 1$.

Step 1: Update. Set $t_a^n = t_a(x_a^n), \quad \forall a$.

Step 2: Direction finding. Perform all-or-nothing assignment based on travel times $\{t_a^n\}$ from step 1. This yields a set of auxiliary flows $\{y_a^n\}$.

Step 3: Line search. Find α_n that solves

$$\min_{0\leq\alpha\leq 1} \sum_a \int_0^{x_a^n+\alpha(y_a^n-x_a^n)} t_a(\omega)d\omega \tag{3.1}$$

on travel times $\{t_a^n\}$. This yields $\{x_a^n\}$.

Step 4: Move. Set $x_a^{n+1} = x_a^n + \alpha_n(y_a^n - x_a^n)$

Step 5: Convergence test. If a convergence criterion is met, stop (the current solution, $\{x_a^{n+1}\}$, is the set of equilibrium link flows); otherwise, set $n := n + 1$ and go to step 1.

[1]Page 119-120 in Sheffi's [1]

Contrary to two heuristic methods mentioned in section 2.1, which can neither guarantee a convergence nor a *User Equilibirum*, the adopted Frank-Wolfe algorithm can guarantee both convergence and a *User Equilibrium*.

In step 3, a line search is implemented and in step 5, the convergence of the algorithm should be tested, these two aspects will be discussed in Section and Section .

3.2 Bisection line search

In step 3, we need to find α_n that solves

$$\min_{0\leq\alpha\leq 1} \sum_a \int_0^{x_a^n+\alpha(y_a^n-x_a^n)} t_a(\omega)d\omega \tag{3.2}$$

However, the objective function in Equation (3.2) is an sum of integrals, which are difficult to implement in a numerical settings. In order to tackle this problem, a bisection line search is implemented, because bisection search is to find the roots (zeros) of the derivative of the objective function with respect to α. Thus, instead of calling the integral function in each iteration, we call the derivative of it, which is:

$$\frac{\partial}{\partial\alpha} \sum_a \int_0^{x_a^n+\alpha(y_a^n-x_a^n)} t_a(\omega)d\omega = \sum_a (y_a^n - x_a^n) t_a[\alpha(y_a^n - x_a^n)] \tag{3.3}$$

So we only need to find the roots (zeros) of Equation (3.3) instead of minimum of Equation (3.2), which greatly reduce the complexity of function call in the line search, because we are calling the objective function, rather than the integral of it. The bisection search code is adopted from Kochenderfer and Wheeler [2] [10] and adjusted to fit this problem.

3.3 Stopping criteria

In this final project, the stopping criteria is as follows:

$$||x_a^n - x_a^{n-1}||_2 \leq \epsilon, \tag{3.4}$$

where $\epsilon = 1e-7$ in this final project. Once the $2-norm$ of two consecutive flows, x_a^n and x_a^{n-1} is less than ϵ, then the algorithm stops.

[2]Kochenderfer and Wheeler [10] Page 50 Algorithm 3.6.

3.4 Strengths and weakness of Frank-Wolfe algorithm

Noticing that the Frank-Wolfe algorithm converges very fast. It has several **strengths**:

1. **Linear approximation reduce the complexity of the problem.** Frank-Wolfe algorithm does a linear approximation of the highly non-linear 4 power objective function of our traffic assignment problem. This approximation yields a linear programs, with linear objective function and linear constraints, which is easy to solve. And there are many sophisticated methods to solve linear problems in the literature [11]. This line approximation happens to be the all-or-nothing assignment in traffic assignment problem, thus an all-or-nothing is the solution of the approximated linear program;
2. **One dimensional exact line search helps reduce the computation.** In step 3, a bisection line search is implemented and reach minimum exponentially with speed of 2^n, and its function call saves time.

However, the Frank-Wolfe algorithm also has **shortcomings**. For example, when the algorithm gets close to the minimum, there will appear "zig-zag" effect, making the algorithm reaching optimum slowly at the end. An implementation of Frank-Wolfe algorithm solving traffic assignment problem with *User Equilibrium* will be illustrated in Chapter 6.

4 Project Decomposition and Responsibility

The group project decomposition and each group member's responsibility will be describe in this chapter.

4.1 Siwei Hu

I am the group leader and responsible for writing all of Abstract, Chapter 1, Chapter 2 and Chapter 3. Since I am a graudate student majoring in Transportation Engineering from Civil and Environmental Engineering Department, I am familiar with the concepts of traffic assignment, *User Equilibrium*, and algorithms solving traffic assignment problems, including Frank-Wolfe Algorithm. So it is natural for me to write all the introduction and related work review, as well as the methodology part.

Apart from writing, I am responsible for hard coding solving the traffic assignment problem with a 4-node small network. Since Tyler and Miguel are new to this problem, so I explained the background and methods in solving the traffic assignment problems. And I share my code solving 4-node network with Tyler and Miguel, so that they could generalized the code to be adjusted into a larger 24-nodes 76-links Sioux-Falls network. And I plotted the numerical traffic flow result, x_a, back to Sioux-Falls network.

And I am also responsible for revise and add the slides, based on Tyler's draft slides.

4.2 Tyler Ryan Staton

Miguel and I were in charge of scaling and applying Frank-Wolfe to the Sioux-falls graph, as well as applying our knowledge of graphs to generate the data for the algorithm that we needed, ie. using Depth First Search to collect all possible paths from $O \to D$.

I specifically made the methods that created our Link-Path matrices as well as Calculating the Path Weights and the initialization of and first few steps in the loop of the Frank-Wolfe Algorithm. Everyone aided in debugging, which we did several times in our breakout rooms and our zoom meetings outside of class time. I was also in charge of making much of the presentation and gathering our plotted results, though most of what we plotted all had the same shape.

4.3 Miguel Angel Meza-Ponce

The first step to solve the network problem was solving for tackling the graph. Tyler and I worked on solving the problem using python and performed a depth-first search on a given graph, represented by an adjacency matrix.

I worked on the graph traversal, and passed it off to Tyler where he computed the initialization of the solution by getting the shortest links and assigning a traffic flow to those paths and finding the link-path matrix. Tyler's computation for initialization allowed me to perform the bisection and line search to converge to a local minimum.

I received a lot of assistance from Siwei since he was most familiar with the problem. He did a lot of the explanations to allow Tyler and I a solid understanding of how to tackle the problem. He provided us with a textbook that explained the Frank-Wolf algorithm and also gave us an outline of how much traffic flow we should assign to certain paths (we compared with three different flows: low, medium and high).

5 Experience in Coding This Project

Each group member's experience in coding will be describe in this chapter.

5.1 Siwei Hu

When coding this project, I got familiar with the methods and contents covered in the class, including line search, and setting a proper stopping criteria.

One important thing is that I could really learn a lot from my fellow group members Tyler and Miguel. They are terrific undergraduate students and computer engineers. I learn a lot from them how to write the code in a neat way and how to enumerate the paths and do the network loading update efficiently. I appreciate their hard-working in this project.

This project really benefits my way of thinking, because I start to analyze why certain algorithm is more proper than the other. Before this project, I just implement whatever homework assigned, but now I start to analyze the efficiency of an algorithm and carefully selection one to implement.

5.2 Tyler Ryan Staton

Much of the coding I did for this project was very similar to how I approached the homeworks in the class, that is I spend a day familiarizing myself with the problem, get the facts straight on how I need to go about approaching it, and then spend the next day writing most of it. Now, this isn't the best method to go about writing massive programs, but the others were very gracious about commenting their code and explaining it if some of it was unclear, so the "leap of faith" programming methodology worked alright here.

As far as understanding this new algorithm in Frank-Wolfe, it took some time as at the core it is very similar to other methods we've seen in class, but it wasn't as clean and pretty of a problem to work with as the mathematical formulas we dealt with in class. Nevertheless after a few hurdles, we managed to reach something that I think we should be proud of. As far as time complexity, I think at one point we have $O(n^2)$ for calculating the Paths, but Siwei convinced us we needed all of them, so it wasn't really avoidable, plus it allowed us some nice $O(1)$ lookups later in the meat of the algorithm.

5.3 Miguel Angel Meza-Ponce

When coding this, I was able to apply previous knowledge from my ICS 46 class. I implemented the depth-first search algorithm and I thought it was fascinating that the courses intertwined. Graph traversals and algorithms are places I feel more comfortable in, so it was nice when Siwei presented the traffic-flow problem by introducing a four node graph.

I must admit, at first the class was a lot more math-based than I thought, but reading the textbook KW paid off in this project. I had to apply material that was presented in this class to solve the problem. Finding the shortest path was not as simple as applying Dijkstra's, we actually had to do a bisection and line search on the problem, and then converge to a local minimum. The previous homework helped significantly for this.

I would also like to add that working with Siwei and Tyler also was helpful. They are both really bright students, and although I was a little slower, they were patient with me and helped me through it. I am really glad I was able to work with them!

6 Result Anlysis and conclusion

6.1 4-node network result-Hu

The traffic assignment problem with 4-node network is solved and discussed in this section.

6.1.1 Input data

The input data includes demand part and supply part, including the **Origin-Destination (OD) demand**, shown in Table 6.2, and the **4 node network** is shown in Figure 6.1 again. The characteristic of the network is shown in Table.

As shown in Table 6.2, there are 2 units of flow from node 1 to node 2 and 2 units of flow from node 3 to node 2.

Table 6.1: Input Data: OD demand in the 4 node network

node-node	1	2	3	4
1	0	2	0	0
2	0	0	0	0
3	0	2	0	0
4	0	0	0	0

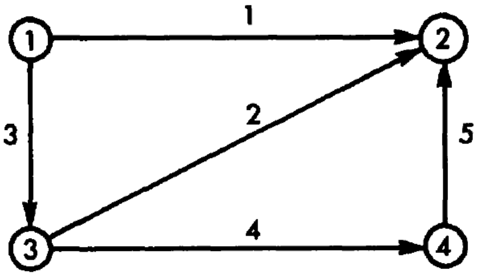


Figure 6.1: An example 4 node network

Table 6.2: Input data: link characteristics of the 4 node network

link/edges	1	2	3	4	5
free flow travel time t_0 (in min)	1	1	1	1	1
capacity C_a (units of flow)	1	2	3	4	5

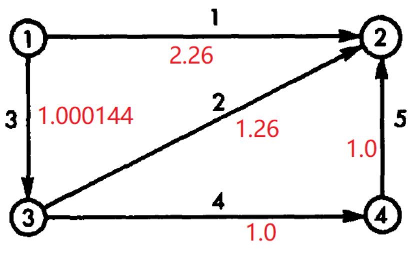


Figure 6.2: Travel time $t_a(min)$ for all links

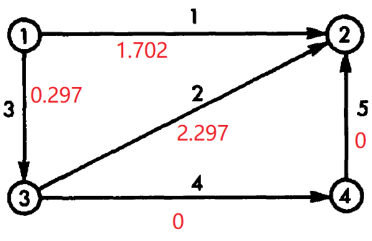


Figure 6.3: Link flows $x_a(unit)$ for all links

6.1.2 Experimental results

To test Frank-Wolfe algorithm in the four node network, path travel time, link travel time, path flow, wall clock running time are calculated. To analysis the trend of convergence, the 2-norm of all link flows and total travel cost are calculated.

The link travel time and link flow is shown in Table 6.3 and visualized in Figure 6.2 and 6.3.

Table 6.3: Results: Link travel time (t_a) and flow (x_a) of the 4 node network

link/edges	1	2	3	4	5
link travel time t_a (in min)	2.26107	1.26107	1.0000144	1.0	1.0
link traffic flow x_a (units of flow)	1.70279	2.29720	0.29720	0.0	0.0

Table 6.4: Results: Path travel cost, c_{rs} for origin r and destination s

Paths	1	2	3
Path travel time from node 1 to node 2 c_{12} (in min)	2.26107	2.26108	3.00001444
Path travel time from node 3 to node 2 c_{32} (in min)	1.26107	2.0	NaN

From the path level results, we could see that the *User Equilibrium* is reached. Below is *User Equilibrium* principles:

1. For a certain origin-destination pair, all the utilized paths have the same travel times. As shown in Table 6.4, for OD pair $1 \rightarrow 2$, path 1 and path 2 share the same travel time, so they are utilized paths.
2. Travel times for all the utilized paths are less than or equal to those for the unutilized paths. As shown in Table 6.4, for OD pair $1 \rightarrow 2$, path 1 and path 2's same travel times are less than travel time for path 3. Same logic can be applied for OD pair $3 \rightarrow 2$, because there is only one path utilized, which is path 1.

As shown from the result above, Frank-Wolfe algorithm can guarantee the *User Equilibrium*. And its ability to guarantee a convergence will be discussed in Section 6.2.

6.2 4-node network Discussion-Hu

In addition to the guarantee of a *User Equilibrium* solution, Frank-Wolfe algorithm can also guarantee a very quick convergence. As shown in Figure 6.4, the 2-norm of x_a converge to 2.874 within 2 iterations. After convergence, the $||x_a||_2$ will have some minor changes in value, but only in the scale of $10e - 5$.

As shown in Figure 6.5, the total travel cost of the whole network, $\sum_a x_a \cdot t_a(x_a)$ converge to 7.0443 within 2 iterations. After convergence, the $\sum_a x_a \cdot t_a(x_a)$ will have some minor changes in value, but only in the scale of $10e - 4$. So it converge very quickly in a small scale network. And the **Wall clock run time** for this 4-node small network is: **0.0003219** sec. The scalability of Frank-Wolfe algorithm in solving large scale traffic assignment problem will be discussed in Section 6.3 and Section 6.4.

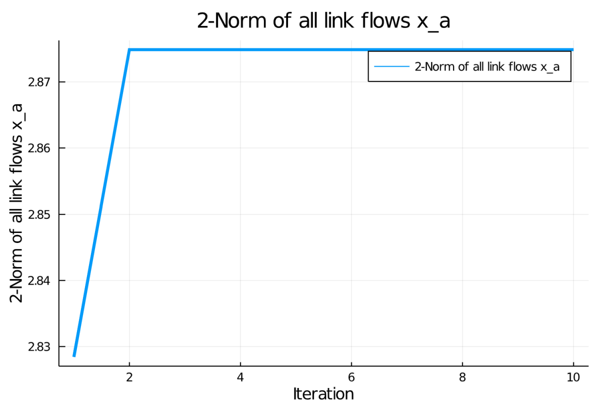


Figure 6.4: 2-norm of x_a, $||x_a||_2$, in every iteration

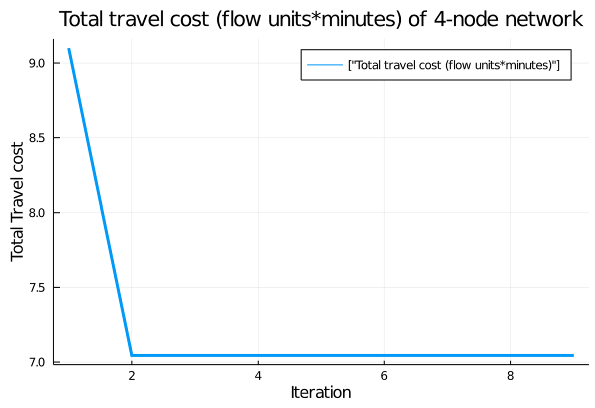


Figure 6.5: Total travel cost, $\sum_a x_a \cdot t_a(x_a)$ (unit-of-flows ·min) for all links

6.3 24 node Sioux-Falls results–Staton and Meza-Ponce

6.3.1 Input data

The traffic assignment problem with 24 nodes 76 links Sioux-Falls network is solved and discussed in this section. The network the topology and information characteristics are adopted from Ukkusur and Yushimito's article [12]. Network topology of the Sioux-Falls network is shown in Figure 6.6. And the supply data, the link characteristic data, are shown in Figure 6.7, including the length (in mile), number of lanes, capacity (veh/hr), speed (mph) limit of a certain link.

In addition to the data from the supply side (i.e. topology and link characteristics), we also have demand side data (i.e. OD demand), and they are shown in Table 6.5. There are only 4 origin nodes (i.e. node 1, 2, 3 and 13) and there are only 4 destination nodes (i.e. node 6,7,18 and 20).

Table 6.5: Input Data: OD demand Sioux-Falls Network

Origin-Destination	6	7	18	20
1	0	1,360	1,100	1,600
2	1,000	0	1,200	1,250
3	1,500	1,200	0	1,028
13	1,600	1,000	1,400	0

6.3.2 Experimental results

Given the experimental settings mentioned in Subsection 6.3.1, the assigned flow result is shown in Figure 6.8. From Figure 6.8, some links experience high demand travel demand,

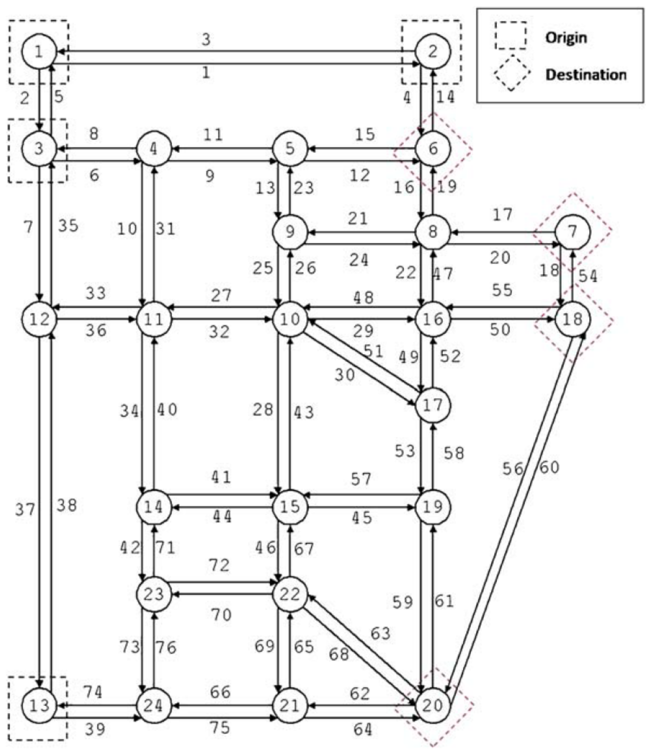


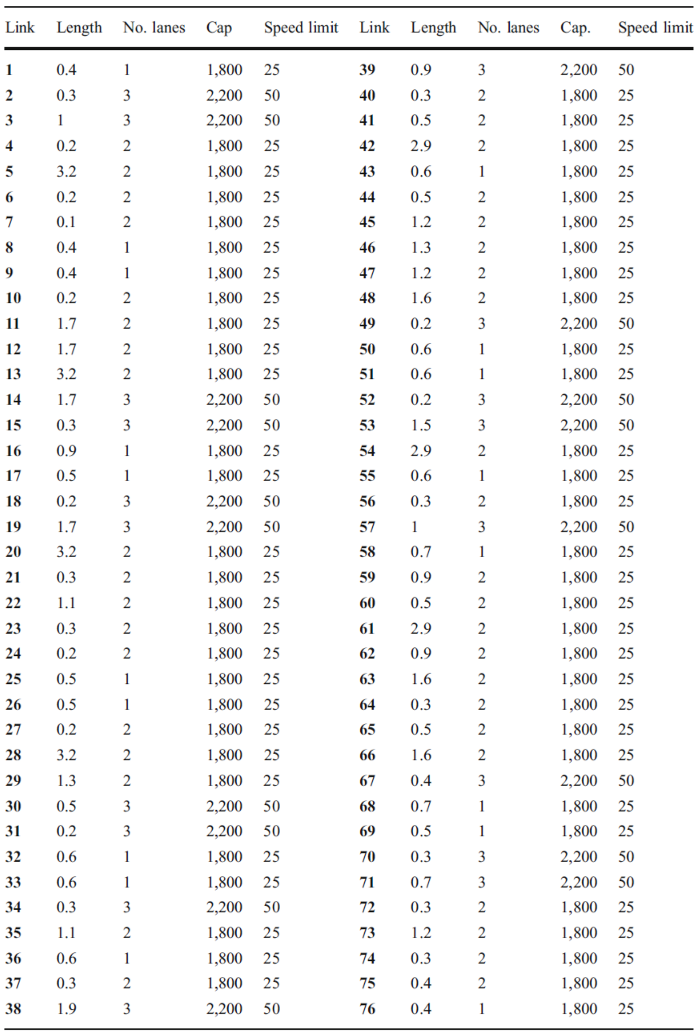
Figure 6.6: Topology of 24 nodes 76 links Sioux-Falls network

because they are critical in distributing the traffic onto different parts of the network. For example, link 2 has a flow of $4332.53 veh/hr$, link 6 has flow of $4178 veh/hr$, link 39 has flow of $7472 veh/hr$, link 64 has flow of $5865 veh/hr$, and link 75 has a flow of $5865 veh/hr$. And the link travel time results, t_a, is shown in Figure 6.9.

Table 6.6: Congestion illustration, x_a/C_a ratio, for some links

link/edges	2	6	39	64	75
link flow x_a (in veh/hr)	$4,332.53$	$4,178$	$7,472$	$5,865$	$5,865$
link capacity C_a (in veh/hr)	$6,600$	$3,600$	$6,600$	$3,600$	$3,600$
x_a/C_a ratio	0.6563	1.1605	1.1321	1.62916	1.62916

Based on our result, we select several links to shown the congestion level of these links, shown in Table 6.6. x_a/C_a ratio is a measurement of the congestion, the higher the ratio is, the more congestion the network is. We could see from Table 6.6, link 6, 39, 64 and link 75 are more congested, because their x_a/C_a ratio is greater than 1.0, meaning that the flow on those links are greater than the capacity. So transportation engineers might need to propose traffic management and control strategies to make those links less congested.



Link	Length	No. lanes	Cap	Speed limit	Link	Length	No. lanes	Cap.	Speed limit
1	0.4	1	1,800	25	**39**	0.9	3	2,200	50
2	0.3	3	2,200	50	**40**	0.3	2	1,800	25
3	1	3	2,200	50	**41**	0.5	2	1,800	25
4	0.2	2	1,800	25	**42**	2.9	2	1,800	25
5	3.2	2	1,800	25	**43**	0.6	1	1,800	25
6	0.2	2	1,800	25	**44**	0.5	2	1,800	25
7	0.1	2	1,800	25	**45**	1.2	2	1,800	25
8	0.4	1	1,800	25	**46**	1.3	2	1,800	25
9	0.4	1	1,800	25	**47**	1.2	2	1,800	25
10	0.2	2	1,800	25	**48**	1.6	2	1,800	25
11	1.7	2	1,800	25	**49**	0.2	3	2,200	50
12	1.7	2	1,800	25	**50**	0.6	1	1,800	25
13	3.2	2	1,800	25	**51**	0.6	1	1,800	25
14	1.7	3	2,200	50	**52**	0.2	3	2,200	50
15	0.3	3	2,200	50	**53**	1.5	3	2,200	50
16	0.9	1	1,800	25	**54**	2.9	2	1,800	25
17	0.5	1	1,800	25	**55**	0.6	1	1,800	25
18	0.2	3	2,200	50	**56**	0.3	2	1,800	25
19	1.7	3	2,200	50	**57**	1	3	2,200	50
20	3.2	2	1,800	25	**58**	0.7	1	1,800	25
21	0.3	2	1,800	25	**59**	0.9	2	1,800	25
22	1.1	2	1,800	25	**60**	0.5	2	1,800	25
23	0.3	2	1,800	25	**61**	2.9	2	1,800	25
24	0.2	2	1,800	25	**62**	0.9	2	1,800	25
25	0.5	1	1,800	25	**63**	1.6	2	1,800	25
26	0.5	1	1,800	25	**64**	0.3	2	1,800	25
27	0.2	2	1,800	25	**65**	0.5	2	1,800	25
28	3.2	2	1,800	25	**66**	1.6	2	1,800	25
29	1.3	2	1,800	25	**67**	0.4	3	2,200	50
30	0.5	3	2,200	50	**68**	0.7	1	1,800	25
31	0.2	3	2,200	50	**69**	0.5	1	1,800	25
32	0.6	1	1,800	25	**70**	0.3	3	2,200	50
33	0.6	1	1,800	25	**71**	0.7	3	2,200	50
34	0.3	3	2,200	50	**72**	0.3	2	1,800	25
35	1.1	2	1,800	25	**73**	1.2	2	1,800	25
36	0.6	1	1,800	25	**74**	0.3	2	1,800	25
37	0.3	2	1,800	25	**75**	0.4	2	1,800	25
38	1.9	3	2,200	50	**76**	0.4	1	1,800	25

Figure 6.7: Link Characteristics of the Sioux-Falls network

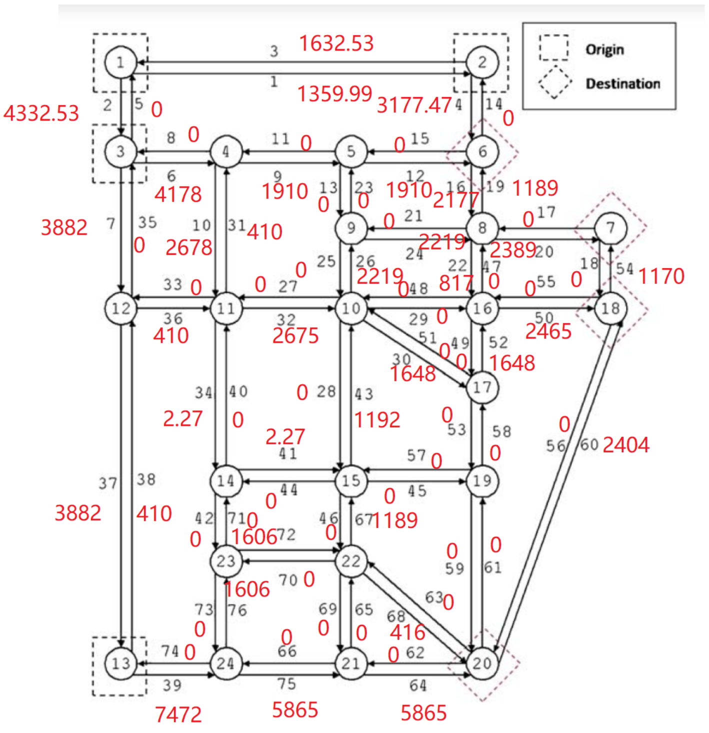


Figure 6.8: Assignment result of link flows, x_a (veh/hr), for Sioux-Falls network

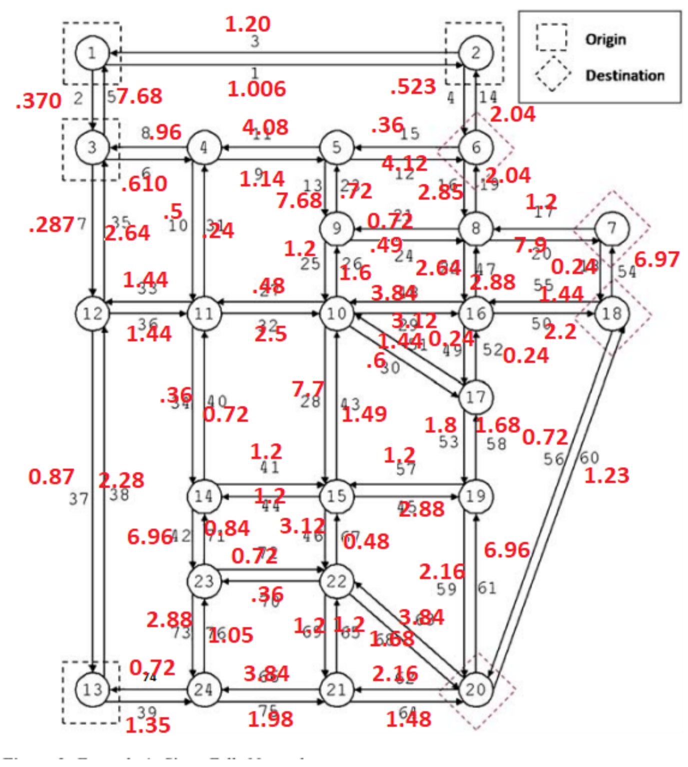


Figure 6.9: Assignment result of link travel time, t_a (mins), for Sioux-Falls network

6.4 24 node Sioux-Falls discussion-Staton and Meza-Ponce

As with all of our Optimization algorithms our results seem to converge to the point of very miniscule changes. In the graph to the top left we are comparing our x-axis(of Frank Wolfe Iterations) to our y-axis(norm of the flow through the network). The flow through the network is supposed to represent the of drivers being put onto our network with each iteration, and based on our graph, we converge to some 16xxx around 100 iterations, further supported by the graph below it with 100 iterations. Our table shows the total travel cost, the sum of the performances of the links times the flow through the network. We can see that as we add more iterations the total travel cost begins to stop changing dramatically after about 100 iterations and, based on what I observed, changed only by less than the 12th decimal place. This suggests that the flow being assigned to the network eventually reaches a state of equilibrium where no matter what path is assigned, the travel cost is optimal for the whole network as they all sum up to this amount.

6.5 Conclusion

...

...

Bibliography

[1] Y. Sheffi, *Urban transportation networks.* Prentice-Hall, Englewood Cliffs, NJ, 1985, vol. 6.

[2] J. G. Wardrop, "Road paper. some theoretical aspects of road traffic research.," *Proceedings of the institution of civil engineers*, vol. 1, no. 3, pp. 325–362, 1952.

[3] M. J. Beckmann, C. B. McGuire, and C. B. Winsten, "Studies in the economics of transportation," 1955.

[4] N. Irwin, N. Dodd, and H. Von Cube, "Capacity restraint in assignment programs," *Highway Research Board Bulletin*, no. 297, 1961.

[5] W. W. Mosher Jr, "A capacity-restraint algorithm for assigning flow to a transport network," *Highway Research Record*, no. 6, 1963.

[6] T. F. Humphrey, "A report on the accuracy of traffic assignment when using capacity restraint," *Highway Research Record*, no. 191, 1967.

[7] J. A. Ferland, M. Florian, and C. Achim, "On incremental methods for traffic assignment," *Transportation Research*, vol. 9, no. 4, pp. 237–239, 1975.

[8] L. J. LeBlanc, E. K. Morlok, and W. P. Pierskalla, "An efficient approach to solving the road network equilibrium traffic assignment problem," *Transportation research*, vol. 9, no. 5, pp. 309–318, 1975.

[9] M. Frank, P. Wolfe, *et al.*, "An algorithm for quadratic programming," *Naval research logistics quarterly*, vol. 3, no. 1-2, pp. 95–110, 1956.

[10] M. J. Kochenderfer and T. A. Wheeler, *Algorithms for optimization.* Mit Press, 2019.

[11] G. B. Dantzig, *Linear programming and extensions.* Princeton university press, 1998, vol. 48.

[12] S. V. Ukkusuri and W. F. Yushimito, "A methodology to assess the criticality of highway transportation networks," *Journal of Transportation Security*, vol. 2, no. 1-2, pp. 29–46, 2009.

[13] R. Smock, "An iterative assignment approach to capacity restraint on arterial networks," *Highway Research Board Bulletin*, no. 347, 1962.