Debojjal Bagchi
Bachelor of Science (Research)
debojjalb@iisc.ac.in

Abridged Project Write Up
Method of Successive Averages
Guide: Prof. Tarun Rambha

Undergraduate (UG)
Dept. of Mathematics
Date: 18.06.21

1 The MSA Algorithm

The Method of Successive Averages (MSA) algorithm is similar to gradient descent as we start with a feasible solution and proceed towards the optimal solution by taking step sizes in such a way that the solution at each iteration stay within the feasible region.
The MSA Algorithm works with a very basic strategy involving the following 3 steps in cyclic order:

- Computation of shortest path.
- Shifting travellers to new paths.
- Updating Link costs & fixing them for next iteration.

2 Convergence

The convergence criteria is based on Wardrop principle. Two most popular gap measures for convergence are:

- Relative gap
- Average excess cost

We have Total System Travel Time (TSTT) and Shortest Path Travel Time (SPTT) as:

$$TSTT = \sum_{(i,j)\in A} x_{ij} * t_{ij}(x_{ij})$$

$$SPTT = \sum_{(i,j)\in A} \hat{x}_{ij} * t_{ij}(x_{ij})$$

Clearly $SPTT < TSTT$ in all other cases except User Equilibrium.
Thus the convergence measures have been defines as:

$$\text{Relative Gap } (RG) = \frac{TSTT}{SPTT} - 1$$

$$\text{Average Excess Cost } (AEC) = \frac{TSTT - SPTT}{\sum_{(r,s)\in Z^2} d_{rs}}$$

3 User Equiibrium & System Optimum

There are two possible ways of traffic assignment:

- **User equilibrium (UE)**: The objective is to find a feasible assignment in which all used paths have equal and minimal travel times.
- **System optimum (SO)**: The objective is to find a feasible assignment that would minimizes the total system travel time.

The **System optimum (SO)** Problem can be stated as:

$$\min \sum_{(i,j)\in A} x_{ij} t_{ij}(x_{ij})$$

Subject to:

$$\sum_{p\in P_{rs}} y_p = d_{rs} \ \forall (r,s) \in Z^2$$

$$x_{ij} = \sum_{p\in P} \delta_{ij}^p y_p \ \forall (i,j) \in A$$

$$y_p > 0 \ \forall p \in P$$

The **User equilibrium (UE)** Problem can be stated as:

$$\min \int_0^{x_{ij}} t_{ij}(w)dw$$

Subject to:

$$\sum_{p \in P_{rs}} y_p = d_{rs} \ \forall (r,s) \in Z^2$$

$$x_{ij} = \sum_{p \in P} \delta_{ij}^p y_p \ \forall (i,j) \in A$$

$$y_p > 0 \ \forall p \in P$$

4 Travel times for MSA Algorithm

$$time_{UE} = time_{FreeFlow} * (1 + B(\frac{flow}{capacity})^{power})$$

$$time_{SO} = time_{UE} + flow * time_{UE}^{'}$$

$$time_{SO} = time_{UE} + time_{FreeFlow} * B * power * ((\frac{flow}{capacity})^{power})$$

5 Details of the code

The code is divided into four sections:

- **Preliminaries:** This section dealt with creating appropriate classes and data cleanup from real transportation networks data available at this GitHub rep. Adjacency List & Adjacency Matrix were created.
- **One to all shortest path algorithms:** Algorithms 2 & 3 were used to get the shortest parth from origin to all nodes. Later results from the two were compared.
- **The MSA Algorithm:** Finally The MSA Algorithm (Algorithm 1) was coded using both Algorithm 2 & 3.
- **Analysis:** The final Total System Travel Time for User Equilibrium & System Optimum were calculated. The run times were compared. The final Relative Gap & Average Excess Coset were calculated.

6 Initialisation in MSA & other details

The codes are available in this GitHub Rep.

- The initial flow in MSA algorithm is initialised as follows: We find the times at 0 flow and then apply the shortest path algorithm to get initial flow.
- The $time_{SO}$ is commented in the code. The code runs on $time_{UE}$. To see SO Results please uncomment it in $time\_fn(flow)$ function
- Similarly the code uses Label Correcting algorithm label Setting algorithm is commented. Uncomment it in $shortest\_path(data)$ function to use it.

7 Summary of Results

Network	Nodes: Links: Zones:	TSTT (UE)	TSTT (SO)	% difference	Running Time (UE) MSA + LC	Running Time (UE) MSA + LS
Sioux Falls	Nodes: 24 Links: 76 Zones: 24	7497384.342333	7490919.22	0.086	8.17s	7.98s
Eastern-Massachusetts	Nodes: 74 Links: 258 Zones: 74	28225.81	27663.23	2.033	1.86s	2.31s
Anaheim	Nodes: 416 Links: 914 Zones: 38	1420110.10	1405632.089	1.03	1.76s	2.37s
Chicago-Sketch	Nodes: 933 Links: 2950 Zones: 387	18395240.49	18002791.86	2.17	387s	933s

8 Minimising time in large network like Chicago

- We see in all relatively shorter networks. MSA was able to give results in less than 10s. But as the number of zones increase the polynomial time comes to play and our run time increases. I suggest the taking following **approximations** to reduce run time.
- Approximation: In the demand matrix set $d[i][j]$ values to be 0 when $d[i][j] < 10^{-3} max(d[i][j])$
- Why this works?: Demand between an OD Pair affects the final flows. But note that if demand at an OD pair is very low less than 0.001 times the maximum flow it will barely affect the flow values. We assume the demands are well distributed in a real life network and not skewed for this approximation to work the assumption is pretty true!
- Does Doing this have any significant effect on Run Time?: Yes! Here are the results: (Also the equilibrium flow values are pretty close. As can be seen from the code)

		Without any approximation		Running time with Approximation of Order 10ˆ-3*	
Network	Algorithm	TSTT (UE)	Running Time (UE)	TSTT (UE)	Running Time (UE)
Chicago-Sketch	MSA + LC	18395240.49	387s	14223775.623	154s

9 Concluding Remarks

- It is difficult to predict in which situation MSA+LC is better and when MSA+LS is better. As LS terminates in n iterations but LC takes more iterations with less time per iteration. Still in most real case scenarios LC performed better.
- MSA is slow! As said, “Continental drift would beat it anyday!”. As Number of Zones & Nodes increase, MSA keeps taking more time as it runs on polynomial time. Also, convergence is slow based on inverse of iteration number as convex combination. Frank Wolfe Method would be much faster.
- While finding Shortest Paths, topological ordering could help if the network graph is acyclic & can be topologically ordered.

10 Pseudo-code

Please see next page.

Algorithm 1 MSA (Input: Graph G)

```
Set $k \leftarrow 1$
while $RelativeGap \geq threshold$ do
    $x \leftarrow 1/k * x + (1 - 1/k) * x$          ▷ Where $x$ is link flow vector
    Update $t(x)$          ▷ $t(x)$ refers to time for flow $x$
    Set $\hat{x} \leftarrow 0$
    for $r \in Z$ do
        shortest_path$(G, r)$
        for $s \in Z$, $(i, j) \in p_{rs}$ do          ▷ $p_{rs}$ refers to shortest path from $r$ to $s$
            $\hat{x_{ij}} \leftarrow \hat{x_{ij}} + d_{rs}$          ▷ $d_{rs}$ refers to demand from $r$ to $s$
        end for
    end for
    $RelatieGap = TSTT/SPTT - 1$
    $k \leftarrow k + 1$
end while
```

Algorithm 2 Label Correcting (Input: Graph G, Origin r)

```
$\mu_r = 0, \pi_r = r$
for $i \in N \setminus \{r\}$ do
    $\pi_i = \infty, \pi_i = -1$
end for
$SEL = r$
while $SEL \neq \emptyset$ do
    Remove $i$ from $SEL$
    for $j : (i, j) \in A$ do
        if $\mu_j > \mu_i + t_{ij}$ then
            $\mu_j = \mu_i + t_{ij}$
            $\pi_j = i$
            if $j$ not in $SEL$ then
                add $j$ in $SEL$
            end if
        end if
    end for
end while
```

Algorithm 3 Label Setting (Input: Graph G, Origin r)

```
$S = \phi, \bar{S} = N$
$\mu_r = 0, \pi_r = r$
for $i \in N \setminus \{r\}$ do
    $\pi_i = \infty, \pi_i = -1$
end for
while $\bar{S} \neq \emptyset$ do
    $i = argmin_{j \in \bar{S}} \mu_j$
    $S = S \cup i, \bar{S} = \bar{S} \setminus i$
    for $j : (i, j) \in A$ do
        if $\mu_j > \mu_i + t_{ij}$ then
            $\mu_j = \mu_i + t_{ij}$
            $\pi_j = i$
        end if
    end for
end while
```