Progress Report: Min-Transfers Algorithm Implementation

Ghulam Mustafa and syeda Wania hussain

April 20, 2025

1 Implementation Summary

In this project, we implemented the 3 out of the 5 algorithms given in the paper, and they were running on Temporal Paths Preservation (TPP) graph model. The objective of the algorithms is to minimize the number of vehicle transfers when traveling across a public transportation network, using a temporal graph representation.

- First the The TpP graph is constructed by converting the temporal transportation data into a directed acyclic graph (DAG), where each edge represents a trip between two stations.
- Then We implemented three variants of the algorithm: SQ (Single Queue), NQ (No Queue), and MQ (Multiple Queues). These algorithms calculate the minimum number of transfers required to reach the destination station.
- We used randomly generated datasets for testing.At first we intended to follow the approach of the authors of the research paper,where they got real world data and then performed some filtering on it to get the relevent data to be used to construct the TPP graph,but we had issues understanding the data,and due to the time constraint,we decided to go with the approach of randomly generating the data.

2 Correctness Testing

The correctness of the implementation was tested using both small and large datasets. We verified the results by comparing them with expected outputs, ensuring that the algorithm produces the correct number of transfers for various source and destination stations.

We used the following test cases:

- Small Sample Dataset: A dataset with 5 trips was used to verify basic functionality and edge cases.
- Medium and Large Datasets: Datasets with 50 and 500 trips were used to evaluate the algorithm's performance and correctness on larger inputs.

- For the small sample, the timing results for all three algorithms were nearly identical, as expected. The dataset was small enough that the overhead of managing queues was negligible, leading to almost no difference in performance. The times were in milliseconds

```
====== Source = 2 on Small Sample ======
SQ took 0.02ms → {3: 0, 9: 1}
NQ took 0.02ms → {3: 0, 9: 1}
MQ took 0.02ms → {3: 0, 9: 1}
```

- For the medium sample, NQ outperformed both SQ and MQ, taking the least amount of time (around 0.04ms). SQ and MQ were similar in performance, with MQ taking slightly more time due to the added complexity of managing multiple queues. These results align with the theoretical expectations, where NQ is expected to be faster than SQ and MQ due to the simpler queueing mechanism. The timing differences between the algorithms remained small due to the relatively small size of the dataset.

```
===== Source = 4 on Medium Sample =====
SQ took 0.05ms → {9: 0, 10: 0, 7: 0, 1: 0, 8: 0, 6: 1, 4: 1, 2: 1, 5: 1, 3: 1}
NQ took 0.04ms → {9: 0, 10: 0, 7: 0, 1: 0, 8: 0, 6: 1, 4: 1, 2: 1, 5: 1}
MQ took 0.06ms → {9: 0, 10: 0, 7: 0, 1: 0, 8: 0, 6: 1, 4: 1, 2: 1, 5: 1, 3: 1}
```

- The large sample (500 trips) exhibited more noticeable differences in runtime. NQ continued to perform the best, taking only 0.23ms. SQ took the longest (0.91ms), followed by MQ (0.64ms). While MQ was slightly faster than SQ, the additional overhead of managing multiple queues in MQ was evident when compared to the simplicity of NQ. These results confirm that for larger datasets, NQ provides the most efficient solution, while SQ remains the least efficient, particularly as the number of trips increases.

```
===== Source = 1 on Large Sample =====
SQ took 0.91ms → {10: 0, 6: 0, 4: 0, 9: 0, 5: 0, 2: 0, 7: 0, 8: 0, 3: 0, 1: 0}
NQ took 0.23ms → {10: 0, 6: 0, 4: 0, 9: 0, 5: 0, 2: 0, 7: 0, 8: 0, 3: 0, 1: 0}
MQ took 0.64ms → {10: 0, 6: 0, 4: 0, 9: 0, 5: 0, 2: 0, 7: 0, 8: 0, 3: 0, 1: 0}
```

3 Complexity & Runtime Analysis

The theoretical complexity of the algorithms is:

- SQ (Single Queue): The time complexity is $O(n \cdot \Delta)$, where n is the number of stations and Δ is the average number of outgoing edges per station.
- NQ (No Queue): The time complexity is $O(n + m)$, where n is the number of stations and m is the number of edges (trips).
- MQ (Multiple Queues): The time complexity is $O(r + L)$, where r is the number of levels and L is the number of trips.

The runtime of the algorithms was empirically tested on small, medium, and large datasets. The results showed that the algorithms run efficiently on datasets with up to 500 trips, with times ranging from a few milliseconds to around 1 second on large datasets.

4 Challenges & Solutions

The challenge we faced was the acquiring of the dataset that the authors have used in the paper.We singed up on the website from where they got the raw data,but we were unable to figuer out how to get it into the required format so that we can further proceed by constructing a TPP graph from it and then running our implemented algorithms.To counter this.for now we generated sample data to atleast get something to run our algorithms on.

5 Enhancements

In the future, we plan to:

- we will try to implement this on the real world GTFS dataset that you have used.if we are unable to get that dataset within time,then we will try to run these algorithm on even larger datasets,so that we can mimick the size of the datasets that were used in the paper and get a comparision.
- We implemented 3 versions of this algorithm out of the 5 given in the paper,since our instructor said that we should implement atleast 3 of them.However,we will try our best to implement the remaining 2 algorithms aswell.