Report on Dijkstra Vs. TNR

Raja Kantheti

March 3, 2025

1 Introduction

We defined two deliverables for this project. They are as follows:

1.1 Deliverable 1:

Given a graph, implement a working version of TNR with the Transit Node choosing method of our choice. See how the preprocessing time fares with the choice we made.

1.2 Deliverable 2:

The second deliverable was to check and learn what is local and to try and see if there exists a cutoff where utilizing local searches is more efficient than always choosing a global search. How transit nodes are selected to change the number of local vs global searches, and see how the speed between global and local changes.

2 Discussion about how it went:

The experimental setup is as follows. I took the Colorado Springs map to test my algorithm. The map has over 20,662 nodes and more than 50,000 edges. I have a graph in *resources/colorado_springs.graphml*.

The script *map_pre1.py* will do the preprocessing and output several JSON files. Some of the files are redundant to the goal of this project and are mostly used for debugging.

The two files which are used extensively are `centrality_between_nodes` and `final_transit_nodes`. The format of the files can be parquet for easier access to the data and for reducing the size of the preprocessing files.

After the preprocessing is done, the *algo.py* file will make a custom number of randomized runs comparing the TNR and Dijkstra algorithms.

2.1 Deliverable 1 (Implementation of TNR)

2.1.1 Selection of Transit Nodes

I followed the grid-based approach for finding the edges which cross the grid cell boundaries. The nodes that connect these edges are the initial set of transit nodes which are about 2000 nodes out of the 20,000 nodes available. This can be found in the *resources/grid.png* file after or within some seconds after preprocessing.

After the selection of this initial set of transit nodes, a parquet file will be generated for finding the betweenness centrality of the nodes in the graph. Betweenness centrality quantifies the importance of a node in terms of the shortest paths passing through it. The formula for the betweenness centrality of a node v is:

$$C_B(v) = \sum_{s \neq v \neq t} \frac{\sigma_{st}(v)}{\sigma_{st}}$$

where:

- σ_{st} is the total number of shortest paths from node s to node t.
- $\sigma_{st}(v)$ is the number of those shortest paths that pass through node v.

A node with high betweenness centrality plays a critical role in network connectivity and information flow.

The final transit nodes are selected from the initial set of transit nodes that have the top 5% of the betweenness centrality.

2.1.2 Preprocessing times:

It takes about 6 hours to complete the preprocessing stage the first time it sees the map. After that, it would only take about 10-20 minutes depending on how many transit nodes we are choosing.

2.2 TNR Implementation:

Along with the usual arguments, the method also takes d_{local} which defines the radius where Dijkstra should be used instead of TNR. In this case, the time would be the same for both algorithms.

The TNR design formula is as follows:

$$d(s,t) = \begin{cases} d_{\text{local}}(s,t), & \text{if } t \in N(s) \text{ or } s \in N(t) \\ \min\limits_{u \in T(s), v \in T(t)} \left(d_{\text{local}}(s,u) + d_{\text{table}}(u,v) + d_{\text{local}}(v,t) \right), & \text{otherwise} \end{cases}$$

where:

- $N(s)$ is the **local neighborhood** of node s.
- $T(s)$ is the set of **transit nodes** for s.
- $d_{\text{local}}(s,t)$ is the shortest path within the local neighborhood.
- $d_{\text{table}}(u,v)$ is the precomputed shortest distance between transit nodes u and v.

This concludes the implementation details.

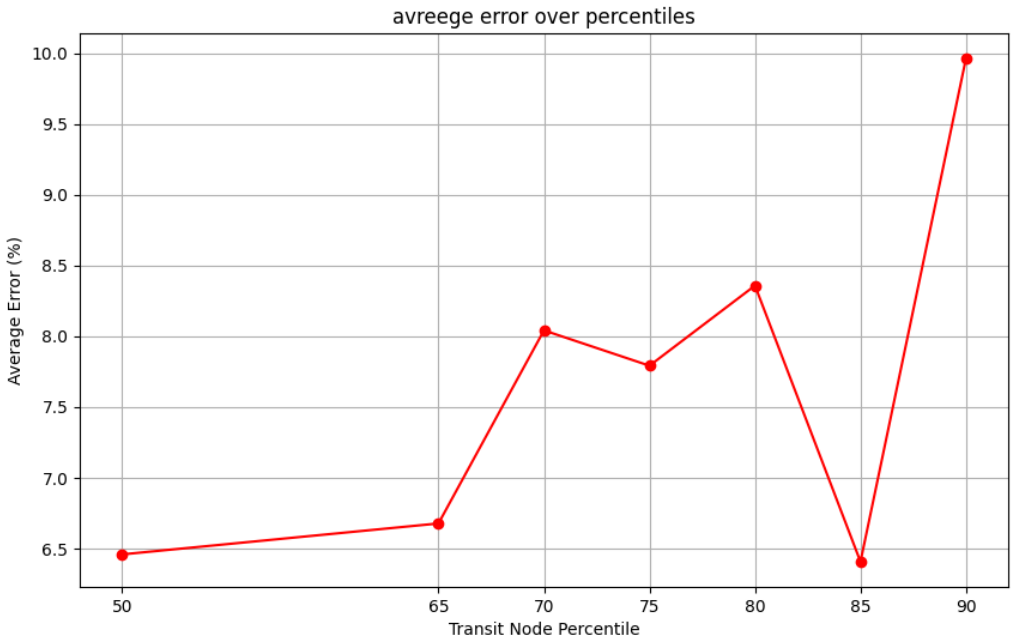

Figure 1: Average error percentage over different transit node percentiles.

2.3 Deliverable 2 (Comparing Dijkstra and TNR)

2.3.1 Error Rates Over Percentiles

The first graph (Figure 1) shows how the average error percentage varies with the transit node percentile. The error rate starts relatively low at around **6.5% at the 50th percentile** and gradually increases, with a significant spike at the **90th percentile reaching 10%**. This suggests that as we increase the number of transit nodes beyond a certain point, the accuracy of TNR compared to Dijkstra's exact results begins to degrade. The non-monotonic pattern indicates that some percentile choices perform better than others, suggesting an optimal range where error remains low while maintaining efficiency.

2.3.2 Execution Time Comparison

Figure 2 compares the **execution time** of Dijkstra's algorithm versus TNR across different transit node percentiles. Dijkstra's execution time remains significantly higher than TNR across all percentiles, consistently around **0.04 seconds**, whereas TNR's execution time remains negligible. This highlights the computational advantage of TNR, as it drastically reduces query time by leveraging precomputed transit node distances. The small fluctuations in Dijkstra's execution time suggest that network structure might slightly impact performance, but the key takeaway is that **TNR provides a substantial speedup while introducing some level of approximation error**.

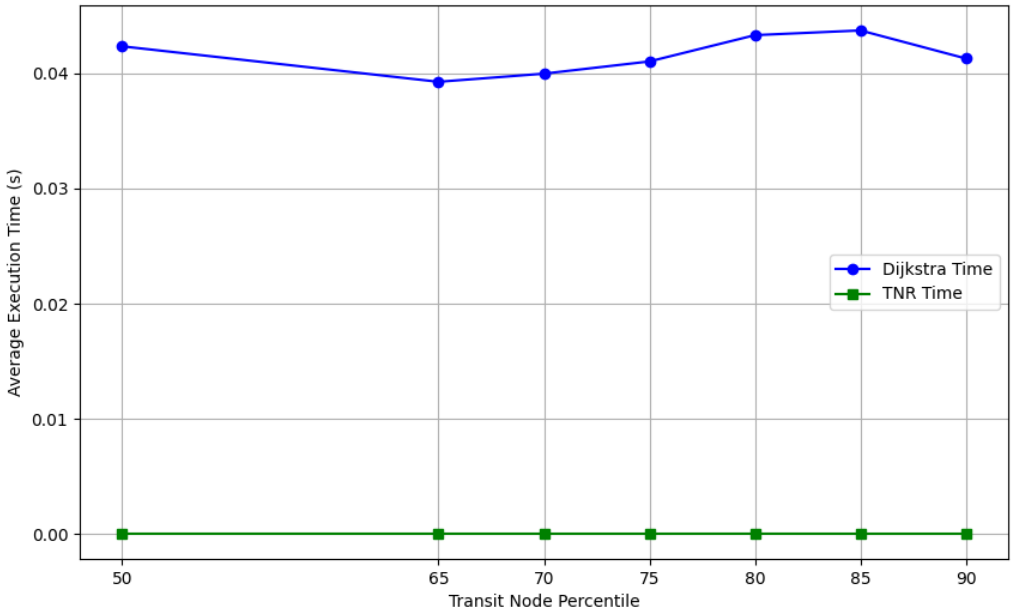

Figure 2: Comparison of execution time between Dijkstra's algorithm and TNR.

2.3.3 Inflection Point in Local Search

The third graph (Figure 3) examines the **inflection point in local search distance (d_{local})**, representing the threshold where local search is no longer efficient and global transit nodes must be used. The plot shows significant fluctuations, with the highest inflection point at **over 4000 meters at the 50th percentile**, dropping to around **1000 meters at the 65th percentile**, and then varying further. This suggests that the **inflection point is highly dependent on the choice of transit node percentile**, affecting when the algorithm transitions from local search to global transit nodes. The variations indicate that there is no single best percentile but rather a trade-off between early transit node selection and maintaining a balance between accuracy and efficiency.

3 Conclusion

From these results, we observe that **choosing the right transit node percentile is crucial**. While increasing the percentile reduces local search distances and speeds up queries, it also increases approximation errors. The **optimal balance** lies in identifying a percentile where the inflection point remains manageable, error rates are controlled, and execution time is minimized.

4 What I could've done better:

I think my understanding of the concept of local was primitive at best. Although the implementation is mostly correct, I believe it would've been perfect if I had implemented the d_{local} in a more sophisticated way. Also, I did not understand how many of the presentations in the class did not

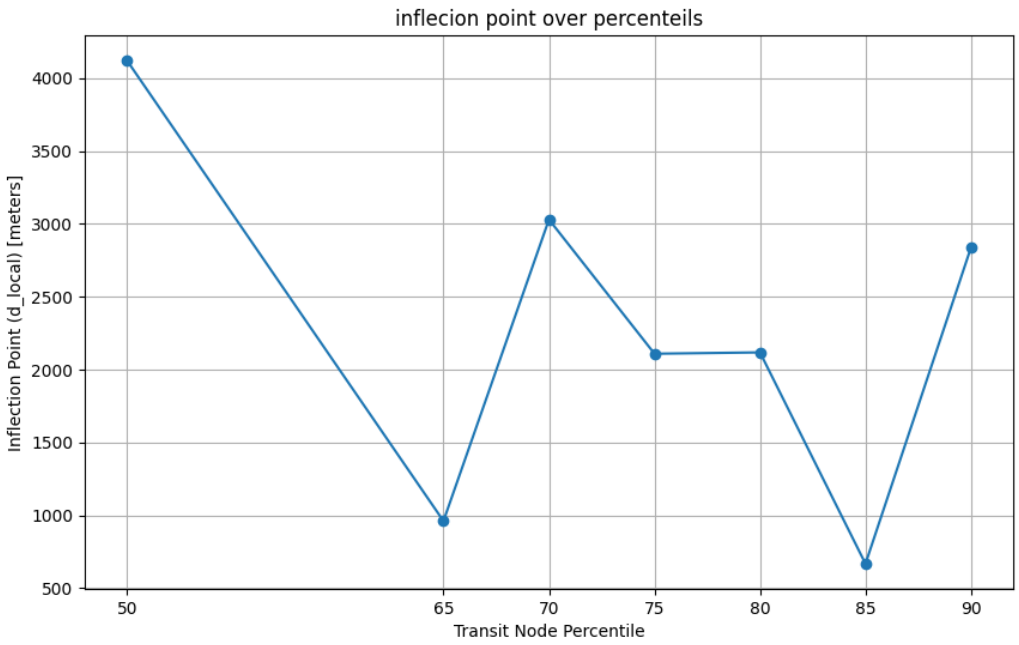

Figure 3: Inflection point in local search distance (d_{local}) over different transit node percentiles.

talk much about the error rates. I had to do a lot of debugging and preprocessing the data over and over to get to the error rates I have. I am not sure what I am missing here.

5 Attribution Statement:

I asked ChatGPT to make a checkpoint-based plan for the entire project and used it for minor debugging. It gave me the idea to use parquet files instead of using JSON files which worked to my advantage. Most of the logic and coding were my ideas and I used ChatGPT to reason with myself.

5.1 GitHub link:

here