Data augmentation for graph based data

Improving representation of cycling trips with varying speed conditions using data augmentation

Lucas Petre
Supervisor(s): Dr. E. Isufi, MSc. T Gao
EEMCS, Delft University of Technology, The Netherlands

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 21, 2025

Name of the student: Lucas Petre
Final project course: CSE3000 Research Project
Thesis committee: Dr. E. Isufi, MSc. T. Gao, Dr. J. Sun

Abstract

Accurate estimation of bicycle trip travel times remains a challenge due to the limited availability of structured cycling data. This paper investigates how graph-based data augmentation can be used to address this limitation, specifically within the context of the DG4B model, a Graph Convolutional Neural Network for travel time estimation. We explore and evaluate three augmentation techniques: Random Walk (with and without node revisiting), Dijkstra Walk and Subgraph Stitching. These methods generate new trips by traversing or recombining paths within an existing road network graph, aiming to expand the training dataset while preserving realistic routing behavior. The augmented data is evaluated both statistically, using metrics like mean, variance and Frobenius norm, and in terms of model performance using RMSE, MAE and MAPE. Experimental results show that Subgraph Stitching and Dijkstra Walk yield the most effective improvements in model accuracy, with each method exhibiting strengths across different trip duration ranges. This work demonstrates that carefully designed graph-based data augmentation can improve GCNN-based travel time predictions in settings with limited cycling trip data.

1 Introduction

Contrary to most other popular travel options, there are not many accurate ways of estimating travel time for bicycle trips. In one of the few papers on this topic, there has been attempted to formulate a model for performing such estimations [2]. The model mentioned in this paper makes use of graph-based deep learning in the form of a Graph Convolutional Neural Network (GCNN), in order to estimate travel time given a certain route across a graph network of roads and intersections. A graph convolutional neural network is a type of artificial intelligence that can operate and learn from graph based data. One of the major problems faced when attempting to use aforementioned model for estimating the travel time, is the limited availability of structural cycling trip data. In order to tackle this problem, this paper will describe, propose and analyse multiple possible ways of extending the data by means of data augmentation. As such, the main focus of this paper will not be the bicycle trip time estimation itself, but rather the augmentation of graph-based data, which in our case is focused on generating new trip data based on the previously existing and classified map data (i.e. roads and intersections) from [2].

Whilst data augmentation is most often used for image based applications [4], it can also be used for graph-based data [1]. It is important to note that data augmentation is not to be confused with data synthetisation. While, by definition, data synthetisation focuses on generating new data, data augmentation applies transformations to existing data in order to generate new data.

To tackle the challenge of data augmentation mentioned above, we have formulated two subquestions:
1. What data augmentation techniques can be used to augment the available data, such that it can be used in a GCNN?
2. How do we measure the generated data and compare it to the previously existing data and how do we measure its performance when used for the model?

In order to answer these questions in a clear manner, the paper is organised as follows: We will start off by covering the related work, here we will introduce previous research to describe what methods already exist for graph data augmentation. Next the methodology will describe what augmentation methods we will be using and how they work. It will also describe the metrics we will use for measuring each augmentation method's performance on the DG4B model. After that, we will present the results, first by analysing and comparing the generated trip data through the different augmentation methods and afterwards by comparing the model performance of the augmented dataset. Finally, we will draw conclusion from these results and provide possible future work for this field of research.

2 Related work

As mentioned in the introduction, data augmentation is most often used for image based applications. Hence we will not be able to use and apply all data augmentation previously research [4].

Before getting started, we must also examine what type of graph data we are working with. The DG4B model makes use of 2 graphs, a road network graph, in which the infrastructure, that is roads, intersections, traffic lights, etc. is stored, and a trip graph, which stores data for individual cycling trips. In our research, we will augment trips by use of the road network graph. We will consider roads as edges and intersections as nodes, where we will traverse roads in order to simulate a traversal from a certain location A to a another location B in the road network graph. This location can either be a node (intersection) or a location along a road, meaning we can also start or finish on an edge. This traversal is referred to as a trip.

Existing studies in graph data augmentation focus on tasks such as node classification and graph classification [5] [1] [3], using techniques like:

- **Edge Perturbation**: We drop or add a random percentage of traversed road edges to or from a trip. This method is not fit for our the data we are trying to augment since for a trip to be valid, there needs to be a connected path in the road network graph from starting starting point A to finishing point B. If we drop a random edge (road), we will almost certainly lose this direct path and as such we would no longer have a connected trip from A to B. [1].

- **Attributes masking**: We hide certain features or attributes in the graph, in order to help the model understand not only the context in the graph, but also it's content [1]. This method is however beyond the scope of this research, as hiding attributes would more resemble feature engineering.

- **Subgraph extraction**: We generate a subgraph, in our case a trip. Generating such subgraphs can be done by for example: Random Walk, a method in which we start at a random node, then traverse random edges until we decide through some metric that the trip should end. The node in which we end is the final node. We now have a generated subgraph. We can also crop the graph. Cropping a graph for a trip would mean traversing along the trips edges from the starting node and then according to some metrics stopping before the final node. We consider the node where we stopped the finishing node. Now we have a new subgraph. Finally, we could mix up two subgraphs. This would mean swapping certain attributes from one edge or node to another edge or node, for example distance or average speed. This however would mean modifying the road network graph. As such it's beyond the scope of this research. Having both local and global information, shorter and longer trips in our case, which can be achieved through subgraph extraction, is advantageous for representation learning [1].

Although these methods can work on graph based data, most of them are not particularly suited for trip augmentation. In our scenario, the preservation of path continuity (that is, a trip needs to go from a starting point A to a finishing point B without interruption) and realistic travel behaviour are critical. Therefore, while we can draw inspiration from the aforementioned techniques, this research will modify them where possible to the context of trip augmentation by introducing custom approaches that retain meaningful data.

This work distinguishes itself by not only applying some of the graph augmentation techniques mentioned above, but also proposes some new approaches of augmentation, in order to enhance the model and reduce the overall error for trip time estimation. By using existing classified map data used in the DG4B model [2], we develop augmentations that simulate variations in travel behavior and routing, ultimately aiming to improve model performance through better training data.

3 Methodology

3.1 Choosing Augmentation Methods

As seen in the previous chapter, some data augmentation techniques would not work for our goal of graph based data augmentation. Thus, we propose the following methods, some of which are slightly modified versions of methods from the previous chapter, to augment the trip data:

- Random Walk: We traverse the road network graph from a random starting node A by arbitrarily choosing a new edge to follow until we decide, through some metric, that we must stop. The node where we stop is the end node of the trip. We have now performed a random walk. We will do random walks both with and without allowing revisiting of previous nodes. Random walks where we allow previous nodes to be revisited will from now be referred to as RWT, random walks in which we don't allow revisiting of previous nodes will be called RWF. The revisiting configuration allows us to explore cyclic or looping paths, which may reflect realistic human behavior in urban environments (e.g., missed turns or searching behavior), while the non-revisiting mode ensures more direct, more efficient traversals of the graph. The pseudocode for this method is described in A.1.

- Dijkstra Walk: Similarly to the random walk, we pick a random starting node in the road network graph. We also pick a random end node for our new trip. Instead of randomly traversing edges from the starting node, we now use Dijkstra's algorithm, a pathfinding algorithm that finds the shortest possible path between 2 nodes, to find a path along edges from the starting node to the finishing node that has the smallest possible travel time. We have now generated a new trip. For this method, the pseudocode can be found in A.2.

- Subgraph Stitching: We take 2 arbitrary trips, called T1 and T2, that overlap in at least one node in our road network graph. If these trips intersect in more than one node, we will pick a random node M from this set of intersecting nodes. We now take the edges from the starting node of T1 up to M and the edges from M to the end node of T2. We have now generated a new trip by levering 2 previously existing trips. A.3 describes the pseudocode for this method.

The reason for choosing these methods is that they allow us to use the road network graph from the DG4B model [2] to augment new trips. It is important to note that the methods may then feel like data synthetisation. We can argue however that since we are using the existing road data to generate new trips, we are still constrained by the road graph, thus our generated trips still fall under the category of augmented data and not synthesised data. We have now answered our first subquestion from the introduction, and can move onto answering the second one.

3.2 Experiment Setup

In order to test the previously techniques, we will implement them in the code and run the augmentations. We will then compare the augmented trips to the original trip data. Finally, we will compare the DG4B model performance on the un-augmented trip dataset vs the augmented trip dataset (by augmented dataset, we refer to the original trips **plus** the augmented trips)

To ensure reproducability, we apply random seed = 42 to all experiments. Similarly, for all experiments, we will use a learning rate of 0.01.

3.3 Data & Model Metrics

For us to be able to compare the augmented trips with the original trips, we can use multiple metrics in order to get a better understanding of what trips have exactly been generated. One example of these metrics is the Frobenius norm. This metric can be used to measure the change or spread of two separate datasets. The lower the returned value ($\|A\|_F$), the closer the metric values of the two datasets are. It is important to note that the Frobenius norm indicates how close the metric values are, not necessarily how good they are. We can calculate the norm using the following formula: $\|A\|_F = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n}|a_{ij}|^2}$. The full formula with the variables explained can be found in the appendix: B.1. We can also look at the mean of trips length/size or analyse the variance for these features. Furthermore, we can plot a graph of the distribution of trips with regards to their trip time by using a histogram as was done in [2].

The model also provides multiple performance metrics, RMSE (Root Mean Square Error), MAE (Mean Absolute Error) and MAPE (Mean Absolute Percentage Error), which are given based on the performance of the test/validation set of the DG4B model when trained. The test and validation set are taken from the original trip data, in order to avoid inconsistent test results due to possible badly augmented trip data. Section B.2 (RMSE), B.3 (MAE) and B.4 (MAPE) describe the way each error metric is calculation in more detail.

To compare the model performance for different augmentation methods, we can use the metrics mentioned above and use a table to compare and establish how each method performs.

Metric	**Original**	**RWF**	**RWT**	**Dijkstra Walk**	**Subgraph Stitching**
Travel Time Mean (s)	787.35	50.72	1121.90	386.43	**672.49**
Travel Time Variance (s)	452,222.43	**3294.39**	31276.38	36802.74	175214.47
Speed Mean (m/s)	5.54	7.50	7.45	8.29	**5.52**
Speed Variance (m/s)	0.84	18.81	4.68	13.19	**0.09**
Frobenius Norm ($\|A\|_F$)	N/a	1128.57	1873.04	1543.21	**1043.76**
Short Trips	7641	4997	74	3442	2146
Medium Trips	7051	3	524	1554	1410
Long Trips	7628	0	4402	4	1444

Table 1: Mean and variance for travel time and speed under different augmentation methods. Frobenius norm is computed over the standardized metrics of each method. Values in bold indicate best performance out of all 4 augmentation methods.

Metric	**Original**	**RWF**	**RWT**	**Dijkstra Walk**	**Subgraph Stitching**
Overall					
RMSE	314.04	314.62	317.96	**298.98**	364.97
MAE	126.80	127.09	126.47	**122.83**	125.05
MAPE	18.09	17.28	16.64	16.72	**16.24**
≤8 min					
RMSE	82.60	80.04	76.64	77.59	**73.35**
MAE	57.39	54.85	**51.40**	52.33	52.99
MAPE	26.08	23.11	**21.30**	22.53	21.85
8-16 min					
RMSE	145.37	152.72	149.52	144.30	**136.80**
MAE	105.85	111.18	109.24	104.76	**96.78**
MAPE	14.93	15.63	15.39	14.91	**13.83**
≥16 min					
RMSE	510.96	510.43	517.79	**485.04**	604.63
MAE	214.84	213.27	216.69	**209.29**	221.82
MAPE	13.11	13.05	13.19	**12.66**	13.05

Table 2: Model performance metrics (RMSE, MAE, MAPE) for different augmentation methods for varying trip travel times. Values in bold indicate best performance out of all 4 augmention methods.

4 Results

This section will present the results of our data augmentation, including any conclusion that can be drawn from these results.

4.1 Original Dataset vs Augmented Dataset

In order to be able to compare the model, we first have to establish what the data looks like.

Using every augmentation method, we augmented 5000 new trips. Since the original dataset contained roughly 23000 trips, this increase of approximately 22% in trips avoids overwhelming or thinning of the original data whilst still being a significant enough change such that the effect can be seen in model performance. For easier analysis, we will divide the dataset into 3 categories: trips shorter or equal to eight minutes will be called short trips, trips between 8 and 16 minutes will be referred to as medium trips, and trips equal to or longer than 16 minutes will be known as long trips.

Table 1 shows the mean and variance of both the speed and travel time of the data generated by each augmentation method for 5000 augmented trips. It also includes how the trips are distributed between the 3 trip categories. These categories, in combination with Figures 1-4, can be used to get a more complete overview of the travel time of the trips generated by each augmentation method. Additionally, Table 1 also gives these metrics for the original data. Here we can see that subgraph stitching produces trips most similar to the original dataset. The variance however is way lower. Figure 4 provides a better insight into why this happens: the data generated contains few to none outlier trips , resulting in an overall lower variance for the trip travel time. Still, the variance is way higher than for the RWF nodes. Figure 2 shows that the RWF generates a very skewed trip dataset with lots of very short trips. This correlates with the low variance and low mean. The cumulative distribution function (CDF) also reflects this, as for very short trip time its already higher than in the rest of the augmented datasets.

The mean travel time of Dijkstra walk is also lower than that of the original dataset. This can be explained due to the fact that Dijkstra walk assumes perfect trips, meaning no road is traversed more than once. This causes the mean travel time of the trips to be shorter, as there can be no repetition of road, thus limiting the maximum distance and in turn also the maximum travel time a trip can have within the road network graph.

RWT also shows an interesting distribution of trip travel time. In the algorithm, the trips were capped at a maximum travel time of 2500 seconds, in order to keep trips within realistic bounds as the trips in the original data very rarely exceed 2000 seconds. The limiting factor thus must be the max amount of roads traversed, which we set to 1000, which also explain why the data generated so little short trips.

Its also very noticeable that for all augmentation methods except subgraph stitching, the speed, mean and variance are significantly higher than for the original dataset. This can be caused by the fact that we don't take into account road types. Cyclist may for example be more wary of cycling on car roads, whilst our augmentation does not take such exemptions into account.

As can be expected looking at Figures 1 through 4, the trips generated through subgraph stitching also have the lowest Frobenius norm, indicating the most similarity in travel time and speed to the original dataset. It it however important to keep in mind that the Frobenius norm merely indicates how close the metric values are, not how good they are. It is hence only good use in combination with the figures and rest of the data we have shown.

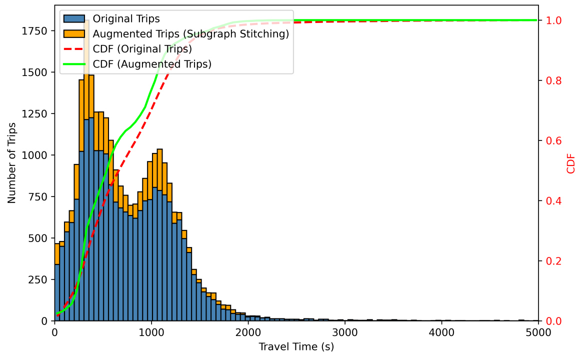

Figure 1: Data distribution of dataset with 5000 augmented trips via subgraph stitching

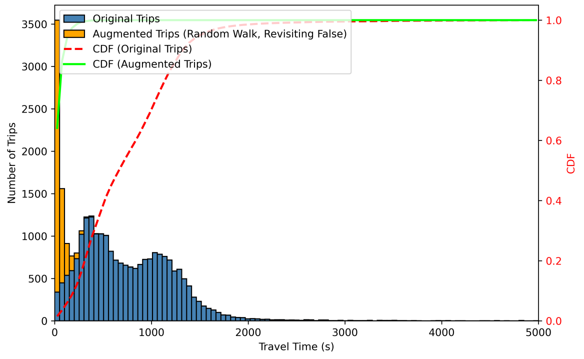

Figure 2: Data distribution of dataset with 5000 augmented trips via RWF

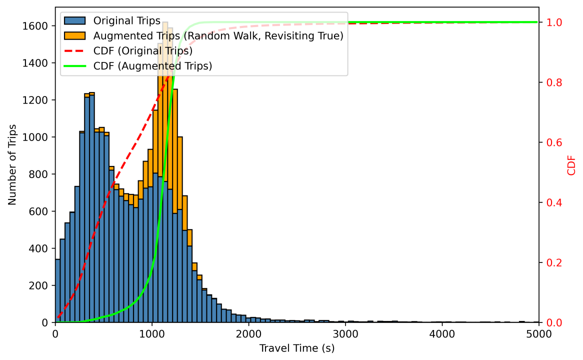

Figure 3: Data distribution of dataset with 5000 augmented trips via RWT

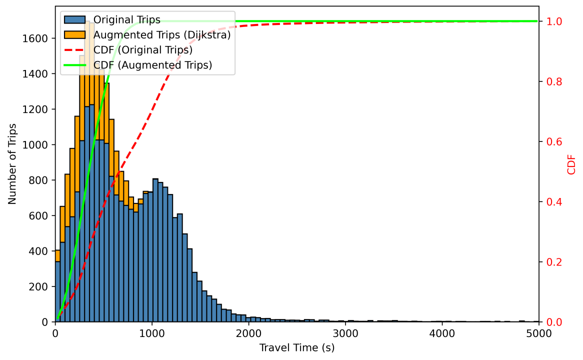

Figure 4: Data distribution of dataset with 5000 augmented trips via Dijkstra walk

4.2 Model performance

For evaluating the model performance, we ran the model on the augmented dataset (that is, the original trips + the augmented trips) for each augmentation method. The results can be seen in Table 2.

For overall performance, all augmented datasets (so original + augmented trips) perform better than the original dataset. Subgraph stitching obtains the lowest MAPE (mean average percentage error) score. This indicates that the average error regarding actual trip time is percentually the lowest with this method. Dijkstra Walk however has a lower MAE and MAPE. The reason for this can be found in the fact that since subgraph stitching uses previously existing data, it may generate new trips using outlier trips, which then bumps up the RMSE and MAE drastically. This can also be seen in figure 1, where we see that there are augmented trips with high travel times as well. The reason for the Dijkstra walk's MAPE being higher than for subgraph stitching is the fact that Dijkstra walk generates trips on the smaller end. Due the the inherent nature of MAPE calculation (we are dividing by the true value), smaller values will result in a higher MAPE.

Another interesting thing to note is the performance for short trips. Again, all augmented datasets perform better in the model, however the RWT performs best, although only marginally. As mentioned before, since we are working with smaller values here (shorter trip time), the MAPE will in general be more affected by smaller error margins.

Moreover, for the medium trips, subgraph stitching outperforms both the original data and the other augmentation methods by a lot. This can be attributed to the fact that subgraph stitching mostly retains the original travel time distribution, unlike the other methods. As such it augments the most trips in the medium trips range. This can be seen when comparing Figures 1-3 with Figure 4.

Finally, for long trips, Dijkstra walk performs best. This is counterintuitive however, as Dijkstra walk doesn't augment many trips in this travel time range. A possible explanation is that although the augmented trips are mostly short, they are structurally consistent and follow realistic shortest paths. This likely improves the model's understanding of local routing behavior and network structure, enabling it to generalize better to longer, unaugmented trips. Furthermore, by avoiding the addition of unrealistic long trips, Dijkstra walk helps the model maintain and even improve performance on long trip estimation.

Overall, the best performing methods from our experiments are Dijkstra walk and Subgraph stitching, both for short, medium and long trips. Although RWT performed best for short trips, it performs only marginally better than subgraph stitching, whilst losing out in all the other trip time categories.

5 Conclusion

This research addresses the insufficient data that can hold back accurate travel time estimation for cycling trips by proposing and evaluating graph-based data augmentation strategies focused on travel time estimation. In this study we investigated what data augmentation techniques can be used to augment data, such that it can be used in a GCNN.

All data augmentation techniques explored in this research, Dijkstra walk, RWF, RWT and Subgraph Stitching, are suitable for augmenting available cycling trip data for use in a GCNN model. However, their performance varies and their attributes may better suit some use cases than other. Using a combination of these methods is recommended, as this helps alleviate any biases in the generated data that may be caused the individual augmentation methods.

In conclusion, data augmentation can indeed be used to help solve data limitation problems in real-world, graph-based problems, thereby improving model performance for cycling trip travel time estimation. The choice of augmentation technique significantly impacts the quality of the augmented data and, in consequence, the model's performance.

5.1 Limitations and Future Work

Research limitations include the inherent complexity of GCNNs and the DG4B model's data, which constrained full exploration of all variable impacts. Future work should focus on optimizing the balance between original and augmented datasets to prevent overfitting, exploring hybrid augmentation techniques for greater data diversity and applying these strategies to other graph-based travel time estimation problems (e.g., pedestrian, public transport). Furthermore, future work could consider how to better improve on the methods proposed and how to better reflect the original data distribution.

References

[1] Michael Adjeisah, Xinzhong Zhu, Huiying Xu, and Tewodros Alemu Ayall. Towards data augmentation in graph neural network: An overview and evaluation. *Computer Science Review*, 47:100527, 2023.

[2] Ting Gao, Winnie Daamen, Elvin Isufi, and Serge Hoogendoorn. Bicycle Travel Time Estimation via Dual Graph-Based Neural Networks. *JOURNAL OF ISTEX CLASS FILES.*, 14(8), August 2021.

[3] Xin Juan, Xiao Liang, Haotian Xue, and Xin Wang. Multi-strategy adaptive data augmentation for Graph Neural Networks. *Expert Systems With Applications*, 258:125076, 2024.

[4] Alhassan Mumuni and Fuseini Mumuni. Data augmentation: A comprehensive survey of modern approaches. *Array*, 16:100258, 2022.

[5] Tong Zhao, Yozen Liu, Leonardo Neves, Oliver Woodford, Meng Jiang, and Neil Shah. Data Augmentation for Graph Neural Networks. In *The Thirty-Fifth AAAI Conference on Artificial Intelligence (AAAI-21)*. AAAI Press, 2021.

A Synthetic Trip Generation Methods

A.1 Random Walk Algorithm

Algorithm 1 Pseudocode of Random Walk algorithm

```
1: for each trip $i$ from 1 to $N$ do
2:     Randomly select a start node $v_0$ from graph $G$
3:     Initialize path $P \leftarrow [v_0]$, edge list $E \leftarrow []$, total distance $d \leftarrow 0$
4:     while step limit and distance limit not exceeded do
5:         Get neighbors $N(v)$ of current node
6:         if $N(v)$ is empty then
7:             Break
8:         end if
9:         Randomly select next node $v_{next} \in N(v)$
10:        Append edge $(v, v_{next})$ to $E$, update $d$
11:        Append $v_{next}$ to $P$
12:        $v \leftarrow v_{next}$
13:    end while
14:    Compute speed, duration, and metadata for trip
15:    Append trip data to dataset
16: end for=0
```

With:

- G: The input graph representing a road network or transit network.
- N: The total number of synthetic trips to generate.
- v_0: The randomly selected starting node for the trip.
- v: The current node being visited in the walk.
- v_{next}: The next node chosen from the neighbors of v.
- $N(v)$: The set of neighboring nodes adjacent to node v in graph G.
- P: The ordered list of nodes (path) visited in a trip.
- E: The list of edges traversed in the trip, where each edge is a pair (v, v_{next}).
- d: The total cumulative distance of the trip.
- $\text{dist}(v, v_{\text{next}})$: The distance between nodes v and v_{next}.
- metadata: Additional trip-related information, such as average speed, total duration, or time of day.

A.2 Dijkstra Walk Algorithm

Algorithm 2 Pseudocode of Dijkstra walk algorithm

```
1:  for each trip $i$ from 1 to $N$ do
2:      Randomly select source node $s$ and destination node $t$ ($s \neq t$)
3:      Run Dijkstra's algorithm from $s$ to $t$ to get node path $P$
4:      if no path exists then
5:          Continue
6:      end if
7:      Convert $P$ to ordered edge list $E$
8:      Retrieve edge attributes (length, speed, etc.)
9:      Compute cumulative distance, travel time, metadata
10:     Append trip data to dataset
11: end for=0
```

With

- G: The input graph representing the network (e.g., road or transit system).
- N: The total number of synthetic trips to generate.
- s: The randomly selected source node for a trip.
- t: The randomly selected destination node for a trip ($s \neq t$).
- P: The ordered list of nodes forming the shortest path from s to t obtained via Dijkstra's algorithm.
- E: The ordered list of edges derived from node path P, where each edge connects two consecutive nodes in P.
- length: The physical distance associated with each edge in the path.
- speed: The speed attribute (e.g., speed limit or average speed) associated with each edge.
- distance: The cumulative total distance of the trip, computed from the edge lengths.
- travel time: The total estimated time for the trip, computed using edge lengths and speeds.
- metadata: Additional information about the trip, such as travel time, route complexity, time of day, or mode of transport.

A.3 Subgraph Stitching Algorithm

Algorithm 3 Pseudocode of Subgraph stitching algorithm

```
1: Extract edge-to-routeID mappings from existing trips
2: Identify all trip pairs ($T_1, T_2$) that share at least one edge
3: for each overlapping pair ($T_1, T_2$) do
4:     Choose a common overlapping edge $e$
5:     Find index of $e$ in $T_1$ and $T_2$
6:     Take prefix of $T_1$ up to $e$, and suffix of $T_2$ after $e$
7:     Concatenate both parts to form stitched trip
8:     Compute speed, total distance, duration, metadata
9:     Append stitched trip to dataset
10:    if number of generated trips reached then
11:        Break
12:    end if
13: end for=0
```

With:

- T_1, T_2: Two existing trips (paths) that share at least one common edge.
- e: A shared edge between trips T_1 and T_2 used as the stitching point.
- prefix(T_1, e): The segment of trip T_1 from its start node up to and including edge e.
- suffix(T_2, e): The segment of trip T_2 starting just after edge e to the end of the trip.
- stitched trip: A new trip formed by concatenating the prefix of T_1 and the suffix of T_2, producing a synthetic but plausible path.
- edge-to-routeID mapping: A mapping that records which original trips each edge belongs to; used to find overlapping trips.
- speed: Average or segment-specific speeds computed over the new stitched trip.
- distance: The total length of the stitched trip, typically the sum of edge distances.
- duration: The estimated time taken for the stitched trip based on distance and speed.
- metadata: Additional attributes related to the stitched trip, such as trip type, mode, or time of day.
- dataset: The collection into which the newly generated (stitched) synthetic trips are stored.

B Formulas and Definitions

B.1 Frobenius Norm

$$\|A\|_F = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n} |a_{ij}|^2}$$

- A: An $m \times n$ matrix.
- a_{ij}: The element in the i-th row and j-th column of A.
- m: The number of rows in the matrix.
- n: The number of columns in the matrix.
- $|a_{ij}|^2$: The squared magnitude of the element a_{ij}; equal to a_{ij}^2 if the matrix has real entries.
- $\sum_{i=1}^{m}\sum_{j=1}^{n}$: A double summation that adds up $|a_{ij}|^2$ for all elements in the matrix.
- $\sqrt{\cdot}$: The square root of the total sum, yielding the Frobenius norm.

B.2 Root Mean Squared Error (RMSE)

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

- n: The number of data points.
- y_i: The actual (true) value for observation i.
- $\hat{y}_i$: The predicted value for observation i.
- $(y_i - \hat{y}_i)^2$: The squared error for observation i.
- $\sum_{i=1}^{n}$: Summation over all observations.
- $\frac{1}{n}$: The mean of the squared errors.
- $\sqrt{\cdot}$: The square root of the mean squared error.

B.3 Mean Absolute Error (MAE)

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

- n: The number of data points.
- y_i: The actual value for observation i.
- $\hat{y}_i$: The predicted value for observation i.
- $|y_i - \hat{y}_i|$: The absolute error for observation i.
- $\sum_{i=1}^{n}$: Summation over all observations.
- $\frac{1}{n}$: The average of the absolute errors.

B.4 Mean Absolute Percentage Error (MAPE)

$$\text{MAPE} = \frac{100\%}{n} \sum_{i=1}^{n} \left| \frac{y_i - \hat{y}_i}{y_i} \right|$$

- n: The number of data points.
- y_i: The actual value for observation i.
- $\hat{y}_i$: The predicted value for observation i.
- $\left|\frac{y_i - \hat{y}_i}{y_i}\right|$: The relative error for observation i.
- $\sum_{i=1}^{n}$: Summation over all observations.
- $\frac{100\%}{n}$: Computes the mean and expresses the result as a percentage.