CONSTRUCTING TEMPORAL STATIC GRAPH SIGNAL USING MA TABULAR TRAFFIC DATA

May 9, 2023

ABSTRACT

The main goal of this project is to construct a static temporal graph dataset(STGD) and applying a temporal graph network(TGN)[3] to verfiy the dataset's credibility and usability. This paper includes a detailed procedure for converting tabular data to temporal graph data through innovated data pre-processing/ cleansing and self-devised *Top-K* algorithms to connect graph vertices. The main learning problem for this project falls to node regression field in graphic neural network. The tabular excel data compiled from Boston Region Metropolitan Planning Organization could be found in this url: https://www.bostonmpo.org/. The source code of this project can be accessed in this github repository: https://github.com/yzw19990124/GCN-Application.git.

1 INTRODUCTION

According to INRIX 2022 Global Traffic Scoreboard[1], the city of Boston has experienced a total of 134 hours in traffic delays, resulting in its ranking as the fourth highest among global cities in terms of traffic delay. There are handful of negative effects regarding traffic congestion, namely additional fuel expenditure, delaying freight contractors, and environmental impact. Based on the INRIX report, for every additional 58 minutes trip duration, there would be a 5.45 dollars extra cost.

This paper presents a solution aimed at addressing traffic congestion in the city of Boston. Specifically, this paper includes a detailed procedure for converting tabular data to temporal graph data through innovated data pre-processing/ cleansing and self-devised *Top-K* algorithms to connect graph vertices. The final product, object of PyTorch static graph temporal signal, would be fed as the training/test input to a TGN[7] for verifying credibility of the product through mean-square-error(MSE) evaluation metric.

Due to the scarcity of free open-sourced STGD specifically the traffic data of Massachusetts area, the final product of this project would hopefully be an ideal tool for researchers to learn the underlying structure of MA traffic distribution and conduct detailed analysis for traffic forecasting. This paper also serves as a friendly tutorial for scholars that are interested in constructing temporal graph-like data from raw tabular data.

2 Related Work

Compared to the contemporary popular large language or computer visionary resources, the current amount of available spatio-temporal dataset does not have as many varieties. Currently the field of analyzing Small Molecules, Bioformatics and Social Networks contains the most graph databases. Some of the famous benchmark dataset are[4][6] namely "ChickenpoxDataset", "PROTEINS", "TWITTER-Real-Graph-Partial", and etc. This paper aims to construct an easier to access and fully customized temporal graph traffic database that hopefully contribute to increase the variety of temporal graph databases.

3 Background

Temporal data[2] refers to data that varies over time but not necessarily in space. Examples of temporal data include stock prices, sensor readings, and speech signals. The Graphic Neural Networks(GNN) for temporal data typically represent the data as a sequence of nodes or graphs, where each node or graph corresponds to a time step, which the GNNs can learn to capture the temporal dependencies between the nodes or graphs.
Spatio-temporal data[5] refers to data that varies both in space and time, such as climate data, traffic patterns, and social network activity. GNNs for spatio-temporal data typically represent the data as a spatio-temporal graph, where nodes correspond to spatial locations and edges represent the temporal relationship between the nodes. In this case, the GNN can learn to capture both the spatial and temporal patterns in the data.

4 Method

The pre-processing of the tabular data is rather crucial for constructing rational graph data. The tabular traffic data used for this project includes: i495-West, i495-South, i495-North, i90-Mass-Pike, i95-South, i93-North, and fresh-pond-Storrow. Each excel file contains roughly 230 traffic data entries from different road segments. Furthermore, each data entry in the original file has 11 attributes, which are the name of segment endpoint, segment length in miles, average for peak travel time/travel speed, and the other eight travel time duration/speed from 6:00 AM to 10:00 AM, and 3:00 PM to 7:00 PM. After conducted a general statistical analysis, the travel time is a better feature attribute overall than the speed attribute for it has less missing data and clearer data distribution. The general pre-processing steps includes: 1. Collapsing all sheets to a single Pandas dataframe; 2. Filtering out Speed, average for peak travel time/travel speed, segment length in miles; 3. Converting Road address to Latitude, Longitude locations through GeoPy and Google Map API; 4. Assigning each unique entry to an unique ID.

	Segment Endpoint	6:00 AM to 6:30 AM	6:30 AM to 7:00 AM	7:00 AM to 7:30 AM	7:30AM to 8:00 AM	8:00 AM to 8:30 AM	8:30 AM to 9:00 AM	9:00 AM to 9:30 AM	9:30 AM to 10:00 AM	3:00 PM to 3:30 PM	3:30 PM to 4:00 PM	4:00 PM to 4:30 PM	4:30 PM to 5:00 PM	5:00 PM to 5:30 PM	5:30 PM to 6:00 PM	6:00 PM to 6:30 PM	6:30 PM to 7:00 PM	Longitude	Latitude	UID
0	Cambridgepark Dr signal, Cambridge	:54	1:03	1:03	1:16	1:16	1:16	1:16	1:16	:42	:42	:42	:42	:42	:42	:42	:42	-71.144371	42.394678	0
1	Fresh Pond Mall signal, Cambridge	:52	1:10	1:30	1:45	1:45	1:45	1:45	1:30	1:03	1:03	1:03	1:10	1:10	1:10	1:03	:57	-71.141486	42.390025	1
2	West Concord Ave rotary, Cambridge	:26	:34	:43	1:26	1:26	1:26	:57	:49	:38	:38	:38	:38	:38	:38	:38	:38	-71.140808	42.386599	2

Figure 1: The first three entries of the dataframe after pre-processing

Next, since we are constructing graph data, we define the edge feature to be the euclidean distance of the two vertices:

$$w_{ij} := \textit{Euclidean Distance}(v_i, v_j) \forall v_{\textit{UID}} := (\textit{lat}, \textit{long})$$

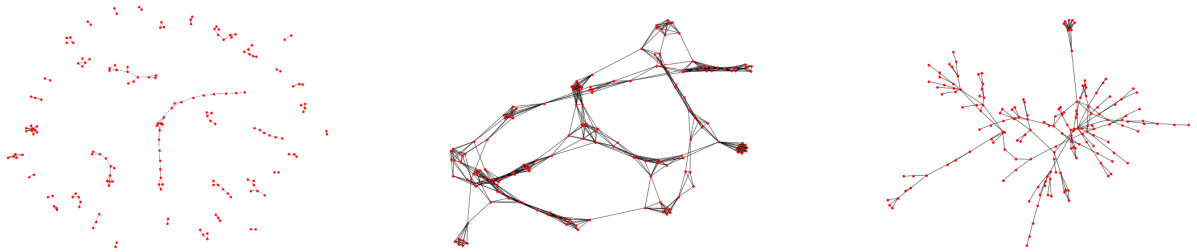
The vertex is the UID for each road segment. The reason for using the distance as the edge feature is under the assumption that two close road segments would more likely to share a similar traffic distribution than the pairs that are hundreds miles away. Once we obtain the distance calculated using the coordinate of vertices through the Google Map API, we would then need to devise an algorithm that decides the likelihood for a segment vertex to be connected with other vertices. The brute force solution would be manually connect the segments according to the real map, which is not feasible if the amount of data double or triple. Hence, an alternative solution would simply use the empirical mean distance to decide the connectivity. The proposed algorithm would perform an exhaustive calculations on all pairs of vertices, then decides K number of shortest edges to build the connection with the source vertex. In short, there exist an edge between the source and target vertices if and only if the pair satisfies:

$$e_{ij} := (v_i, v_j, w_{ij}) \iff w_{ij} \leq \frac{\textit{TOP-K}(\sum_{i=1}^{K} \sum_{j=1}^{K} w_{ij})}{K}$$

An algorithm description could be found in the next page. The algorithm could be invoked multiple times to generate a denser graph. However in this project, the Top-K only needs to be invoked once and in next section we will discuss how to tune the number of K and handle dis-connectivity issue of the generated graph.

Algorithm 1 Top-K Closest Segments

Require: Edge-DataFrame
Require: $K > 0$
  edge-Hashtable $\leftarrow$ *Empty*
  for *each source vertex* **do**
    Distance $\leftarrow 0.0$
    for each target vertex **do**
      Distance $\leftarrow$ *Distance* $+ w_{ij}$
    end for
    Calculate mean distance
    for each calculated edge candidate **do**
      if edge is smaller than the mean distance **then**
        edge-Hashtable add this edge
      end if
    end for
  end for



(a) $K = 5$ (b) $K = 15$ (c) $K = 5$ *After Adaption*

Figure 2: Handle Islands and Fine-tune K

5 Experiments

5.1 Handle Islands and Fine-tune K

Since the $Top - K$ algorithm only ensures that the target vertex finds its K closest neighbors, the generated adjacency matrix would become too sparse with a small K and results disconnected islands in the final graph. Figure 2a is the final graph generated using K equals to five. This is not ideal for a traffic mapping since at least in Massachusetts we can safely assume that every road segment from the original data is reachable following some routes otherwise the records for travel time/ speed makes no sense. Hence the graph would be intrinsically flawed with disconnected components. On the other hand, figure 2b is the graph generated using K equals to 15, which is also not ideal since the graph is too dense. One potential negative impact of using a over-dense graph for GNN model is that after a single message passing layer, almost all of the node features would contain every other node feature resulting over-smoothing. Hence, in order to improve a better edge connectivity for the $Top - K$ algorithm, we propose an additional step after the first iteration of the algorithm. That is, resolving the dis-connectivity issue through randomly choosing source and target nodes from two disconnected graphs. We could deduce that the newly constructed edge would not break the original K closest neighbor relationships from this lemma:
If $v_m \in G_1, v_n \in G_2$, *where* $G_1, G_2 \in \{G_1, G_2, G_3, \ldots\} \leftarrow G$(*Top-K First Iteration*), *Then* v_m, v_n *are not K nearest.*
By constructing edge in following this logic, we resolve the dis-connectivity issue due to small K value, while retaining the k-closest road segments relationships and avoiding generating a over-dense graph. Figure 2c is the graph generated using the same K value as 2a but using the adapted one iteration Top-K algorithm.

5.2 Verifying Graph Dataset Using TGN Model

The final output MA traffic dataset is a type of Torch Geometric Static Graph Temporal Signal[4]. The static graph contains 214 unique nodes which represents the 214 unique road segments. The node feature is dynamic and encapsulates the travel duration variables of 16 total time steps. Throughout the entire time period, the number of edge

is 223 which remains the same assuming no road is closed. Furthermore, the pre-defined lag value is three, which means that at each time step during training, node features at three consecutive time steps would be used for predicting the next time-step node labels.

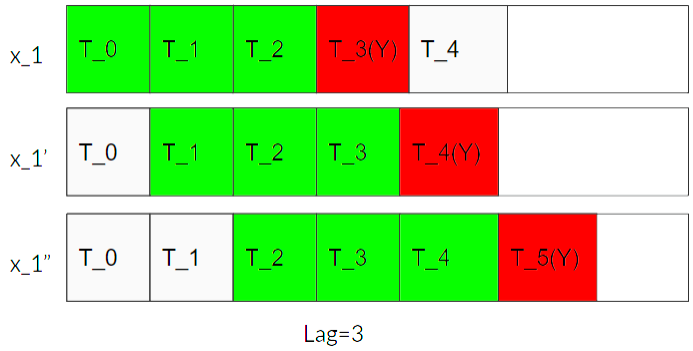


Figure 3: Three example snapshots for one node data entry

For benchmark purpose, the MA traffic dataset used as the train/test dataset for an attention temporal graph[7] and we use the mean square error to visualize the learning procedure. Figure 4 is the model training loss over 100 epochs, and we can see that the training loss is converging and the final model test square error is 2818.0022 seconds.

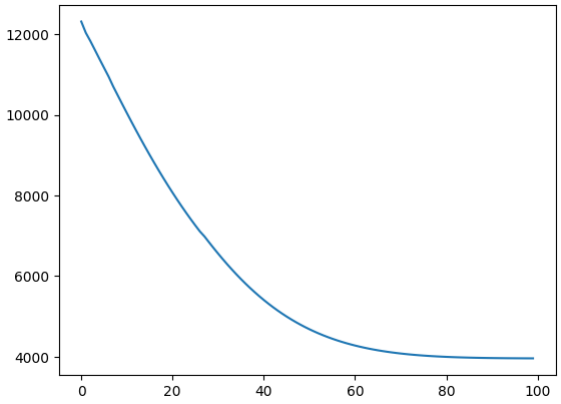


Figure 4: Mean Square Error vs. Training Epochs

6 Conclusion

This paper conducted a detailed exploration on constructing a temporal MA traffic graph database and applied a temporal graph network model as a benchmark example. The final product database is full customizable unlike many stat of art temporal dataset. For instance, users may adjust how dense or sparse their graph data to be through changing the value of K, how many training features for each time-step by tuning the lag value, and upload additional tabular dataset with the database auto-converting tabular to temporal.
As this paper previously mentioned, the database variety for temporal graphs is somewhat biased towards bio-molecular fields. Furthermore, the topic of transforming the tabular data to temporal graph data has received less attention and relatively scarce literature compared to benchmark and optimize graph network models. We hope in the future research, scholars might find this MA traffic database to be beneficial for their work.

References

[1] Inrix. Scorecard.

[2] Yaguang Li, Rose Yu, Cyrus Shahabi, and Yan Liu. Diffusion convolutional recurrent neural network: Data-driven traffic forecasting, 2018.

[3] Emanuele Rossi, Ben Chamberlain, Fabrizio Frasca, Davide Eynard, Federico Monti, and Michael M. Bronstein. Temporal graph networks for deep learning on dynamic graphs. *CoRR*, abs/2006.10637, 2020.

[4] Benedek Rozemberczki, Paul Scherer, Yixuan He, George Panagopoulos, Alexander Riedel, Maria Astefanoaei, Oliver Kiss, Ferenc Beres, Guzman Lopez, Nicolas Collignon, and Rik Sarkar. PyTorch Geometric Temporal: Spatiotemporal Signal Processing with Neural Machine Learning Models. In *Proceedings of the 30th ACM International Conference on Information and Knowledge Management*, page 4564–4573, 2021.

[5] Bing Yu, Haoteng Yin, and Zhanxing Zhu. Spatio-temporal graph convolutional neural network: A deep learning framework for traffic forecasting. *CoRR*, abs/1709.04875, 2017.

[6] Bing Yu, Haoteng Yin, and Zhanxing Zhu. Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, pages 3634–3640. International Joint Conferences on Artificial Intelligence Organization, 7 2018.

[7] Jiawei Zhu, Yujiao Song, Ling Zhao, and Haifeng Li. A3T-GCN: attention temporal graph convolutional network for traffic forecasting. *CoRR*, abs/2006.11583, 2020.