# Dynamic Modeling and Analysis of Urban Transportation Networks Using Graph Neural Networks

1st Neel Agarwal
*Goergen Institute for Data Science*
*University of Rochester*
Rochester, USA
nagarwa9@ur.rochester.edu

2nd Shyam Shah
*Goergen Institute for Data Science*
*University of Rochester*
Rochester, USA
sshah77@ur.rochester.edu

***Abstract*—This paper presents a comprehensive study on the predictive modeling of urban transportation dynamics, focusing on the integration and interaction between CitiBike and Yellow Taxi networks in New York City. Utilizing a decade's worth of data, we employ Graph Neural Networks (GNNs) to capture the spatio-temporal relationships inherent in these transportation modes. The study highlights the application of dynamic and temporal graph models to predict future network states and transportation demands, addressing both methodological challenges and practical implications. Our findings reveal significant insights into the temporal dynamics of urban mobility and propose strategies for optimizing transportation infrastructure and policy decisions.**

***Index Terms*—citibike, yellow taxi, EDA, community detection, weights prediction**

## I. Introduction

URBAN transportation networks represent the lifeline of modern cities, facilitating the flow of people, goods, and services that sustain urban life. In megacities like New York, where the complexity and density of the population demand efficient and scalable transportation solutions, understanding and predicting the dynamics of these systems become crucial. This research focuses on two pivotal elements of New York City's transportation infrastructure: the CitiBike bike-sharing program and the Yellow Taxi service. Each system features unique challenges and plays a critical role in the daily commutes of millions of residents and visitors.

With the advent of big data and advanced computational techniques, the opportunity to analyze and forecast the behaviors of such networks has grown significantly. Graph Neural Networks (GNNs), in particular, have emerged as a powerful tool for modeling data with intricate relational patterns and dependencies, typical of transportation networks. This study leverages a decade of granular operational data from both CitiBike and Yellow Taxi networks to explore their spatio-temporal dynamics through the lens of dynamic and temporal graph theory.

Our methodology involves constructing and analyzing a series of dynamic graphs that encapsulate the interaction between these transportation modes over time. We investigate various facets of the networks, including station connectivity, demand fluctuations, and the impact of external factors such as weather, urban development, and policy changes. By applying a combination of graph convolutional and recurrent neural network models, we aim to not only understand the current state of these networks but also to predict future patterns and demands. The implications of this research are manifold. For urban planners and policymakers, the insights gleaned from our analysis could inform more effective strategies for infrastructure development, congestion management, and environmental sustainability. For commuters and the general public, improved forecasting could lead to better service, enhanced reliability, and a more seamless integration of different modes of transportation.

This paper is structured to first introduce the datasets used and the preprocessing steps involved in making them suitable for analysis. We then detail the modeling techniques employed, followed by a discussion of the results and their implications. Finally, we conclude with potential future research directions that could further enhance the predictiveness and applicability of our models in urban planning and smart city initiatives. Through this comprehensive study, we aim to contribute to the growing field of urban analytics by providing a model that not only predicts but also helps shape future transportation landscapes in major cities worldwide.

## II. Literature Survey - prior work

Saff et al. (2022) in their paper on bike demand evolution using dynamic graphs [1] attempted to apply graph neural networks to predict demand in order to moderate the number of bikes stored at each station in New York City. They used a Spatio-Temporal Graph Convolutional Network to predict the traffic of Citi Bikes (departures and arrivals). The model overfit the training data and produced an exploded MAPE value. However, on visual inspection, their predicted values on test data were close to ground truths. This paper can serve as an inspiration to compare the project's approach and outcomes. Safalidis and Valabueno (2022) in their article on predicting evolution on dynamic graphs [2] demonstrated how to apply deep learning techniques to dynamic graphs in PyG and PyTorch for the purpose of predicting traffic flow. It is a highly resourceful article explaining the mathematical aspect behind temporal and spatial convolutions as well as providing valuable code examples for dynamic visualizations. Li and Axhausen (2020) in their paper on short-term traffic demand prediction using graph convolutional neural networks [3] tried to address the issue for the taxi industry and Mobility- on-demand systems. They divided the study area into several non-overlapping sub-regions with each sub-region treated as a node then constructing a traffic demand graph. Further, they built three graph convolution networks based on three different weighted adjacency matrices, which represent three graph structures. Davis et al. (2018) proposed a HEDGE algorithm-based tes- sellation strategy for taxi demand forecasting [4]. Preliminarily they investigated Geohash and Voronoi tessellation strategies for partitioning city space, in the context of real-time taxi demand forecasting. For comparison, they employed classi- cal time-series tools to model the spatio-temporal demand. However, due to poor performance, they came up with a hybrid strategy using the HEDGE algorithm. They achieved an accuracy of 80% per squared kilometers for efficient spatial partitioning towards better location-based demand modeling and forecasting.

## III. Data Pre-processing

### A. Citibike

Our data[5] spans from Jun 2013 to Dec 2023. Out of the many columns in our data set we kept only the following columns:

1) start time: The timestamp when the bike ride started.
2) end time: The timestamp when the bike ride ended.
3) start station id: A unique id for locating the station .
4) end station id: A unique id for locating the station.
5) start latitude: Latitude of the start station.
6) start longitude: Longitude of the start station.
7) end latitude: Latitude of the end station.
8) end longitude: Longitude of the end station.
9) usertype: Tells if the user is a subscriber of citibike or a casual customer.

The data changed the schema(i.e, names of other features and the "usertype" attribute) and the change had to be accommodated for in the preprocessing to be able to map and handle the essential attributes together for the entire data across the years. After dropping the irrelevant columns, rows with missing or null values were also dropped. As per the work of R. Shekhar , we mapped our bike stations to one of the 265 official taxi zones of New York City using the latitude and longitudes of each station such that all the stations within a taxi zone would correspond to that zone/node in all further references and calculations . The features thus created were, 'start station zone id', and 'end station zone id'. All the citibike stations that geographically lies inside one zone, are considered as one node and the incoming/outgoing edges to any one of those nodes is considered as incoming/outgoing edge to that zone. The edges that are going from one station to another inside the same zone are considered as self loops and have been removed as part of the preprocessing. The start time and end time were given in the date_time format and were therefore converted to the standard UNIX epoch timestamp format (representing the seconds elapsed from January 1st 1970) for easier calculations in further processing and code. Further, we also removed all those bike trips that lasted more than 1 hour, to only keep those bike rides assuming that these trips would have been made mostly to reach from one point to another such that long rent-type usages or long cycling marathons were eliminated. As mentioned earlier, for user_member_type, due to the change in schema over the years, 'Subscriber' and 'member' were mapped together as 'Subscriber' and likewise for 'Customer' and 'casual' as 'Customer'.Lastly, the data was sorted by the 'unix_start_time' and all column data types were type casted to integer format to avoid any type mismatch errors.

### B. Yellow Taxi

The data pre-processing for the taxi data [6] was done in line with the pre-processing of the CitiBike data [5]. The final retained/processed columns were 'unix_start_time", "unix_end_time", "passenger_count", "PULocationID" and "DOLocationID". Since the taxi data over the years was exploding, the aggregations were exported in parquet format and processed in py-arrow table instead of conventional dataframes for faster computations. As done earlier in the CitiBike data, the pickup and drop off times in the date_time formats were converted to UNIX epoch timestamps. Also the taxi trips lasting more than an hour (3600 seconds) were dropped to maintain the consistency of data and remove outliers. The trips that had the pickup and drop off ids identical were dropped indicating self loops. Same was done for rows essentially have trip durations of zero where the pick up and drop off times were identical, indicating redundant trips. Also, rows containing data pertaining to the nodes/ zones with ids '264' and '265' were dropped as these zones corresponded to trips outside New York or unidentified trips. Lastly, the data was sorted by the 'unix_start_time'.

## IV. Methodology

### A. Assumptions

- Aggregating the data of a single day to create a graph will lead to a comparatively sparse graph with fewer edges and less edge weights for EDA.
- We kept only those bike rides where the trip duration was less than 1 hour. We are assuming that these will be the rides which are mainly used as a mode of public transport. Any ride longer than 1 hour is assumed to be those rides where, the people are going for a stroll or a hike. This assumption is supported by the fact that the average trip duration of all bike rides was around 16 minutes.
- We kept only those taxi rides where the trip duration was less than 1 hour for the same reason. This assumption is supported by the fact that the average trip duration of all Yellow taxi rides was around 15 minutes.

## B. *Graph Generation*

### 1) *Citibike:*

- Nodes are official taxi zones.
- Edge between two nodes exists only if there was at least one bike that went from any station in zone1 to any station in zone2 during the given time period.
- Edge weight is the number of bikes going from one zone to another during the given time period.

We created 3 different types of time windows. i) Weekly snapshots, ii) Monthly snapshots, and iii) Yearly snapshots, for the purpose of seeing trends of different granularity. The time windows are defined and moved forward in different ways as follows

1) Weekly snapshots have aggregated data of a week by using the frequency of bikes as edge weights and the snapshots are moved forward by one day.(i.e. Mon-Sun, Tue-Mon, Wed-Tue, . . . )
2) Monthly snapshots have aggregated data of a month by using the frequency of bikes as edge weights and the snapshots are moved forward by one week.(i.e. Week1 Jan-Week4 Jan, Week2 Jan-Week1 Feb, . . . )
3) Yearly snapshots have aggregated data of a year by using the frequency of bikes as edge weights and the snapshots are moved forward by one month.(i.e. Jan 2014-Dec 2014, Feb 2014-Jan 2015, March 2014-Feb 2015, . . . )

For every type of time window, 3 different graphs have been created, which are as follows,

1) All the data is considered for creating the graph
2) Only those bike rides are considered where a subscriber of Citibike used the bike.
3) Only those bike rides are considered where a customer(Anyone who is not a subscribe) used the bike.

### 2) *Yellow Taxi:*

- Nodes are official taxi zones.
- Edge between two nodes exists only if there was at least one taxi that went from any point in zone1 to any point in zone2 during the given time period.
- We are storing 2 edge attributes. i) number of taxis , and ii) number of passengers going from zone1 to zone2 during the given time period.

We created 3 different types of time windows. i) Weekly snapshots, ii) Monthly snapshots, and iii) Yearly snapshots, for the purpose of seeing trends of different granularity. The time windows are defined and moved forward in exactly same way they were moved for Citibike data.

## V. Exploratory Data Analysis of Citibike Data

For all kinds of time windows, and for total, customer, and subscriber different properties were measured for graphs.

## A. *Number of Arcs*

Number of arcs are measured by considering the graph as unweighted and directed. Through the temporal variations in number of arcs, we can see the evolution of connectivity patterns in the Citibike network. This analysis can be used to observe commute routes of people from different zones. Through analysing the evolution of number of arcs in customer graphs and subscriber graphs, strategies can be employed to cater both groups differently.

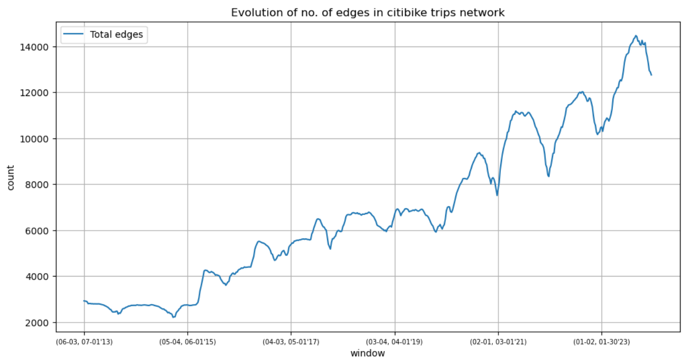
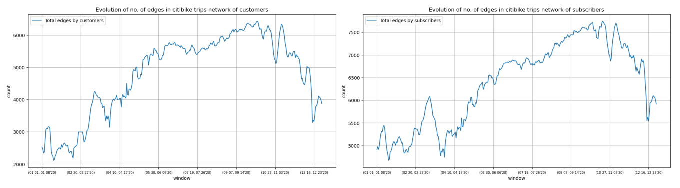


Fig. 1: Evolution of no. of edges in Citibike network across years on a sliding month time window



(a) Customers (b) Subscribers

Fig. 2: Evolution of no. of edges in Citibike network of year 2020 on a sliding week time window

From Fig.1 it is evident that there is a nice upward trend across years, that is the number of edges in the Citibike network kept increasing over the years, which is supported by Fig.3 as well where we can see a clear upward trend in number of nodes of the network. There is also a clear seasonal pattern in Fig.1, where no. of edges in the network decreases significantly during winter time. This suggests that many commute routes are not used during winter. From this analysis, you can devise strategies for allocating resources accordingly. Fig. 2 focuses on a more granular period of sliding weeks in the year 2020 only, which also shows clear seasonal pattern of peak connectivity during summer. Comparing Fig.2a) and Fig.2b), even though they have the same trend, the number of edges in customer network is significantly lower than that in subscriber network.

## B. *Number of Nodes*

Number of nodes is the number of active zones during the time period. Through the temporal variations in the number of nodes, we can understand the pattern of which zones are consistent and which are not as well as which zones show seasonal fluctuations. This analysis can help in visualizing the expansion of Citibike network.

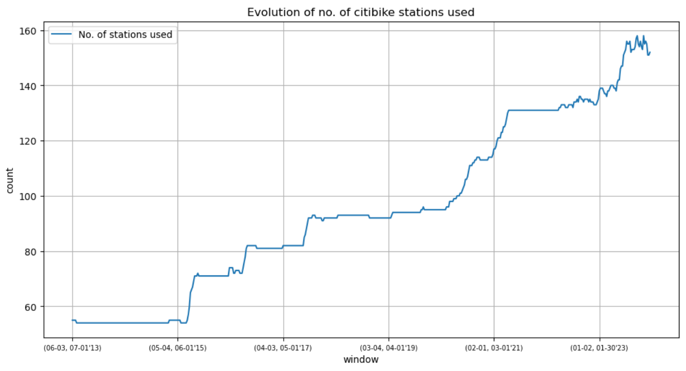
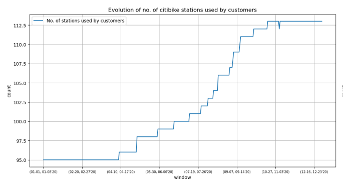
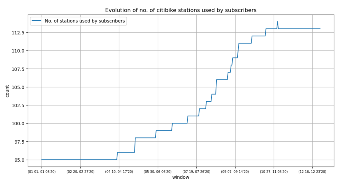


Fig. 3: Evolution of no. of active zones in Citibike network across the years on a sliding month time window



(a) Customers



(b) Subscribers

Fig. 4: Evolution of no. of active zones in Citibike network of year 2020 on a sliding week time window

Fig.3 shows a nice upward trend with some very minor fluctuations, which suggests that over the years many new stations were added, while a very few stations were closed. Fig.4a) and Fig.4b) shows an increase in the number of nodes over the year 2020. This shows that nearly 17-18 zones saw new stations opening during the year 2020.

### C. *Number of bike rides*

Number of bike rides is the sum of all edge weights in the graph at a given time period. This gives us the total no. of bike rides in that time window. Through the temporal variations in this, we can understand how the demand of Citibikes change overtime. This analysis on the customers' graph can give us a good idea of the seasonal fluctuations in casual users of Citibike network, which can help in employing various strategies to try and make them subscribers of Citibike.

Fig.5 enhances the effect of seasonal variation very well, where the drop in usage of bikes during winter is very clear. There is also a clear upward trend across years, which is expected with the increasing number of edges shown in Fig.1. Fig.6a) and Fig.6b) shows the trend in customers and subscribers network during 2020. The sudden drop in subscribers network around March coincides with the start of COVID-19 pandemic. The count is also consistent with other plots, where customers' usage is consistently lower than subscribers' usage.

### D. *Average Degree*

Measuring average degree of a graph can give us the idea of how densely connected a graph is. Temporal variations in

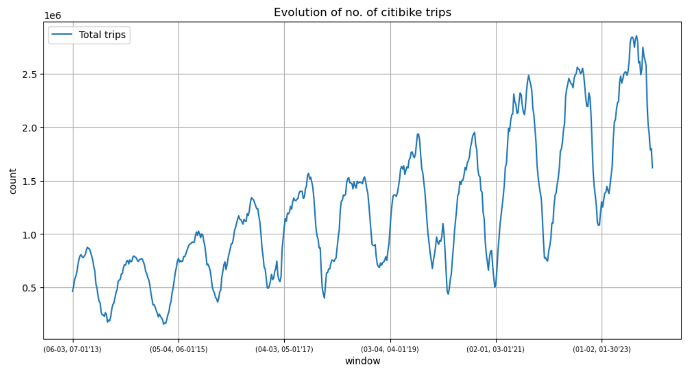
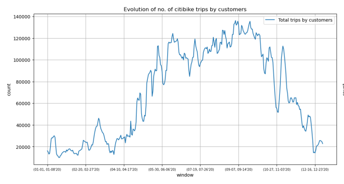
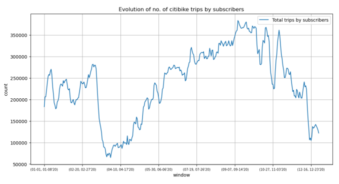


Fig. 5: Evolution of no. of total bike rides in Citibike network across the years on a sliding month time window



(a) Customers



(b) Subscribers

Fig. 6: Evolution of no. of total bike rides in Citibike network of year 2020 on a sliding week time window

average degree can tell us how the inter-connectivity of graph changes over time.

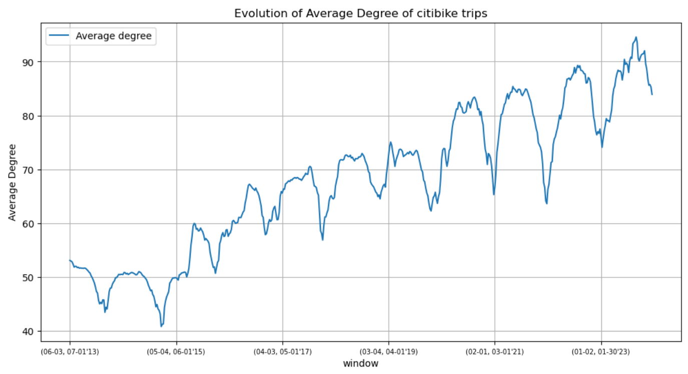
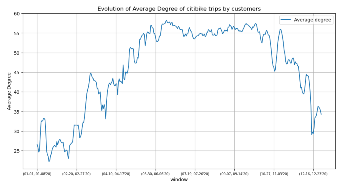


Fig. 7: Evolution of average degree of Citibike network across the years on a sliding month time window

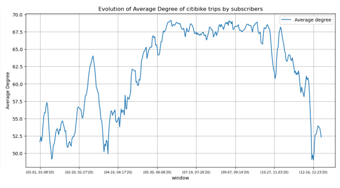
Fig.7, Fig.8a), and Fig.8b) are consistent with the trends seen above, where we can see the overall upward trend with seasonal variations.

### E. *Clustering Coefficient*

Measuring the clustering coefficient is another way of observing the denseness of a network.Specifically through clustering coefficient, we can understand the strength of local connectivity. Through its temporal variations, we can see the changes in local density across years or through seasonal fluctuations.



(a) Customers



(b) Subscribers

Fig. 8: Evolution of average degree in Citibike network of year 2020 on a sliding week time window

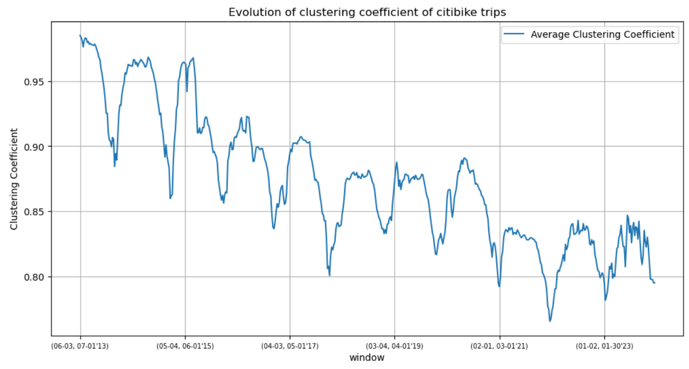
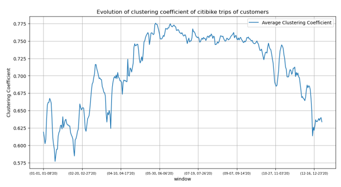


Fig. 9: Evolution of clustering coefficient of Citibike network across the years on a sliding month time window

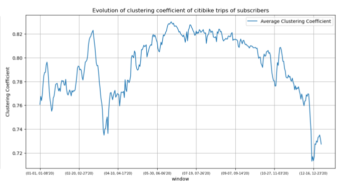
Fig.9 shows a downward trend with seasonal patterns as usual. This suggests that the local denseness of the network slowly decreases with the expansion of the Citibike network. The value of clustering coefficient in Fig.9 shows that even though the local denseness is decreasing, its absolute value drops to around 0.75 only, which suggests very high local denseness. Fig.10 is consistent with our observations from before, but the range of clustering coefficient is (0.575, 0.775) and (0.7, 0.83) for customers and subscribers respectively. This clearly describes the difference in local density of customers and subscribers networks.

## F. Similarity in Community Structure

The examination of community structure can reveal many interesting things. Especially if there are some communities which are very similar across the years, we can find out some physical relation to explain the fact, which in turn can be used for targeted marketing. Here, we used two approaches



(a) Customers



(b) Subscribers

Fig. 10: Evolution of clustering coefficient in Citibike network of year 2020 on a sliding week time window

two measure the similarity in community structure.

**Sequential Similarity Score :**

The Sequential Similarity Score measures the consistency in community structure between consecutive temporal snapshots of the network. This sequential analysis enables the observation of gradual changes in community memberships over time. By computing similarity scores iteratively between adjacent time frames, we unveil patterns of evolution or stability within the network's community structure.

**Partial Similarity Score :**

Contrary to the sequential approach, the Partial Similarity Score assesses the resemblance in community structure between each network snapshot and the initial state. This analysis allows us to discern whether subsequent configurations maintain similarities with the network's original organization. By examining partial correlations between the initial state and subsequent time frames, we gain insights into the persistent or divergent nature of community structures.

**Adjusted Mutual Information (AMI) and Normalized Mutual Information (NMI) :**

To quantify the similarity in community structures, we employ two widely used indices: Adjusted Mutual Information (AMI) and Normalized Mutual Information (NMI). These indices measure the agreement between two partitionings of the network into communities, accounting for the possibility of random chance.

**Adjusted Mutual Information (AMI)** is a corrected version of Mutual Information that adjusts for chance agreement. It accounts for both overestimation and underestimation of similarity, providing a balanced measure of agreement between two community structures.

**Normalized Mutual Information (NMI)**, on the other hand, normalizes the Mutual Information score by the entropy of the individual partitions. This normalization ensures that the index falls within a fixed range, facilitating comparison across different network sizes and community structures.

The formulas for AMI and NMI are as follows:
Given a set S of N elements $S = \{s_1, s_2, \ldots, s_N\}$, if there are two pairwise disjoint partitions $U = \{U_1, U_2, \ldots, U_R\}$, and $V = \{V_1, V_2, \ldots, V_C\}$. Then,

$$\mathrm{AMI}(U, V) = \frac{\mathrm{MI}(U, V) - E\{\mathrm{MI}(U, V)\}}{\max\{H(U), H(V)\} - E\{\mathrm{MI}(U, V)\}}$$

$$\mathrm{NMI}(U, V) = \frac{2 \times \mathrm{MI}(U, V)}{\mathrm{H}(U) + \mathrm{H}(V)}$$

where MI(U,V) is the mutual information between two partitions.
E{MI(U,V)} is the expected mutual information between two partitions considering random chance
H(U), and H(V) are the entropies of U and V respectively.

From Fig.11, we can see many different things.

- AMI score is consistently lower than NMI score for both Sequential and Partial measurement of similarity in

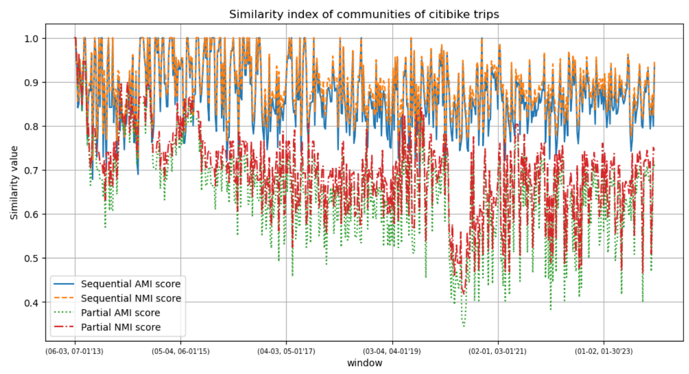
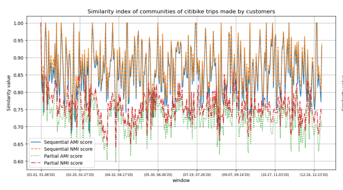
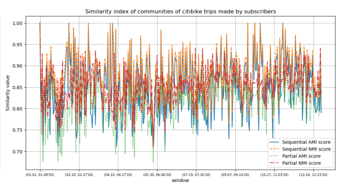


Fig. 11: Evolution of Similarity Score of Citibike network communities across the years on a sliding month time window



(a) Customers



(b) Subscribers

Fig. 12: Evolution of Similarity Score of Citibike network communities of year 2020 on a sliding week time window

community structures. This is to be expected as AMI takes random chances of being similar into account as well.

- Sequential score is similar to Partial score for both AMI and NMI, during the initial period, then the Partial scores are consistently lower than the Sequential scores.
- Consistently same Sequential score suggests that there is no sudden change in community structure. This shows that the core community structures may very a little but are not changing significantly even if the network is expanding, or the winter sees less usage.
- The trend in Partial score suggests that the initial core community structure is not changing in the initial period. After some point as the network starts expanding, the initial community structure changes a little and then becomes stable. The second sudden change in the community structure comes around the time when the network expands very quickly just after COVID-19.

From Fig.12a) and Fig.12b), we can see many obseravtions similar to that of Fig.11 with some differences.

- For customers, the Partial scores are consistently lower than Sequential scores for the year 2020, which is consistent with our observations from Fig.11
- For subscribers, the Partial scores are nearly the same as the Sequential scores for the year 2020, which suggets that the community structure at the start of the year stays the same throughout the year.

From Fig.11 and Fig.12, we can understand that the underlying community structures are very robust, and are immune to any kind of sudden changes or seasonal variations.

## G. Degree Sequence

**Percentage of Nodes with High In-Degree/Out-Degree:**

- In the context of Citibike data, calculating the percentage of nodes with high in-degree or out-degree can provide insights into the popularity or centrality of certain bike stations.
- For instance, if a high percentage of nodes have in-degree higher than 60% of the total number of nodes, it indicates that these stations are major destinations where a significant number of trips end.
- Similarly, a high percentage of nodes with out-degree higher than 60% suggests that these stations are major origins where a significant number of trips start.
- This information can be useful for urban planners and transportation authorities to identify key stations for infrastructure improvements or service optimizations.

**Percentage of Nodes with Low In-Degree/Out-Degree:**

- Conversely, calculating the percentage of nodes with low in-degree or out-degree sheds light on stations that are less frequently used or peripheral in the bike-sharing network.
- Nodes with in-degree less than 10% of the total number of nodes may indicate stations that are less popular as destinations.
- Similarly, nodes with out-degree less than 10% may represent stations with fewer trips originating from them.
- Understanding the distribution of low-degree nodes can help identify areas where bike-sharing services may be underutilized or where additional station placement may be beneficial to improve network coverage and accessibility.

Through temporal variations in this we can see the variations in business of nodes in Citibike network.

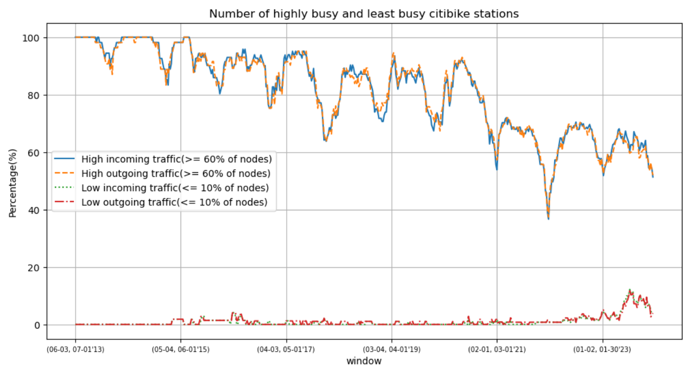
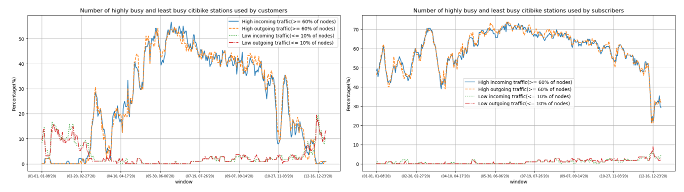


Fig. 13: Evolution of percentage of Highly busy and Least busy zones of Citibike network across the years on a sliding month time window

Fig.13 shows that the number of nodes which are connected to more than 60% nodes of the graph starts to decrease over the years as the network expands, along with the seasonal



(a) Customers (b) Subscribers

Fig. 14: Evolution of percentage of Highly busy and Least busy zones of Citibike network of year 2020 on a sliding week time window

fluctuations. On the other hand, nodes that are connected to less than 10% nodes of the graph are consistently nearly 0, with a very little increase in the year 2023. Fig.14 shows that the seasonal patterns are consistent in both customer and subscriber networks. The effect of seasonal fluctuations is much more significant in customers' network of the year 2020 as the percentage of highly busy stations shows more significant drops during winter than that in subscribers' network of the year 2020.

## H. *In-Degree Centrality*

In-Degree Centrality measures the number of direct connections (or incoming routes) a bike station has from other stations. A station with high in-degree centrality acts as a popular destination within the network. Analyzing this metric helps identify stations that are highly preferred by users, possibly due to their proximity to key attractions, employment centers, or transit hubs. This information is crucial for operational decisions such as deploying additional bikes and docks to meet high demand and planning for peak usage times.

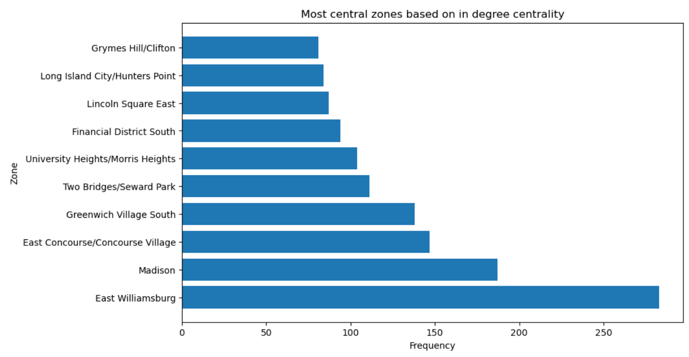
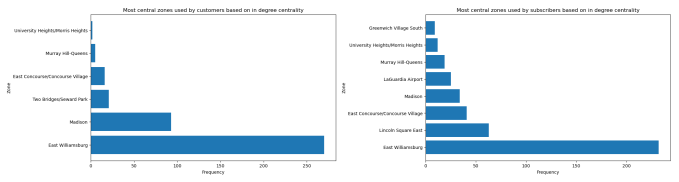


Fig. 15: The number of times each zone was the most central in Citibike network based on in-degree centrality on a sliding month time window

Fig.15 and Fig.16 shows that East Madison zone is most frequently the most central zone based on in-degree centrality over the years as well as during the year 2020 for both customer and subscriber network. Many other zones also remain consistently in the top for different granularity.



(a) Customers (b) Subscribers

Fig. 16: The number of times each zone was the most central in Citibike network of year 2020 based on in-degree centrality on a sliding week time window

## I. *Out-Degree Centrality*

Out-Degree Centrality measures the number of direct connections (or outgoing routes) a station has to other stations. A high out-degree centrality indicates that a station is a common starting point for trips within the network. Stations with high out-degree centrality might be located in residential areas or other starting points of daily commutes. Understanding this measure helps in managing bike availability to ensure users find available bikes when they need them, especially during morning hours or special events.

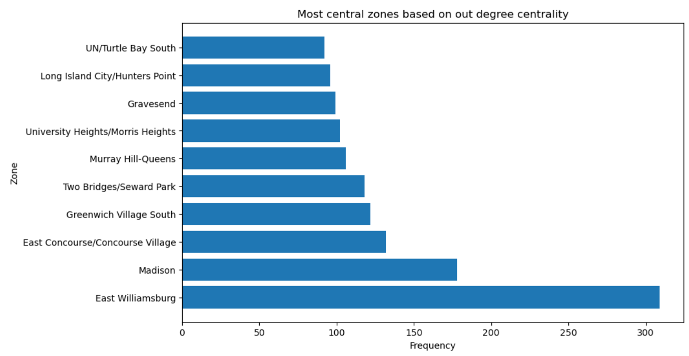
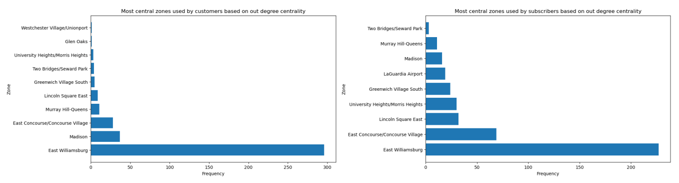


Fig. 17: The number of times each zone was the most central in Citibike network based on out-degree centrality on a sliding month time window



(a) Customers (b) Subscribers

Fig. 18: The number of times each zone was the most central in Citibike network of year 2020 based on out-degree centrality on a sliding week time window

Fig.17 and Fig.18 shows that East Williamsburg zone is most frequently the most central zone based on out-degree centrality over the years as well as during the year 2020 for both customer and subscriber network. Many other zones also remain consistently in the top for different granularity.

### J. Betweenness Centrality

Betweenness Centrality is particularly useful in understanding the role of stations as intermediaries along the shortest paths between other stations in the network. Stations with high betweenness centrality are those through which many of the shortest paths pass and are crucial in connecting different parts of the network. These stations are strategic for ensuring network connectivity and resilience. Enhancing infrastructure at these key nodes can improve the overall efficiency of the network, reduce travel times, and increase the robustness of the network against disruptions.

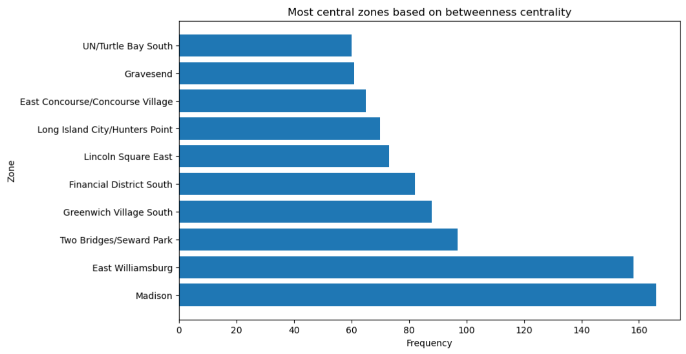
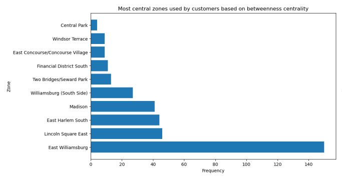
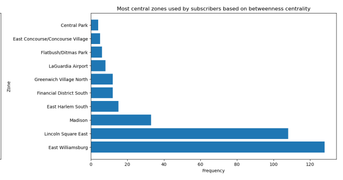


Fig. 19: The number of times each zone was the most central in Citibike network based on betweennesss degree centrality on a sliding month time window



(a) Customers



(b) Subscribers

Fig. 20: The number of times each zone was the most central in Citibike network of year 2020 based on betweenness centrality on a sliding week time window

Fig.19 and Fig.20 shows that East Williamsburg zone is most frequently the most central zone based on betweenness centrality for the year 2020 for both customer and subscriber network, while Madison is most frequently the most central zone based on betweenness centrality over the years. Many other zones also remain consistently in the top for different granularity.

### K. Eigenvector Centrality

Eigenvector Centrality identifies stations that are not only well-connected but also connected to other well-connected stations. This metric indicates influential stations in the network. A station with high eigenvector centrality suggests it is a part of a core group of stations that are extensively used and pivotal to the flow of the network. Enhancing these stations can significantly impact the overall network performance, influencing user patterns and promoting more balanced usage across the network.

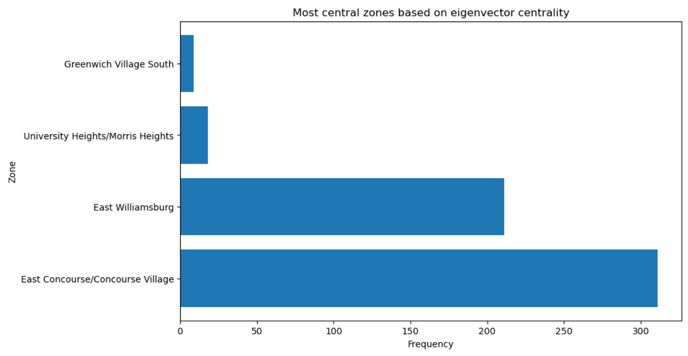
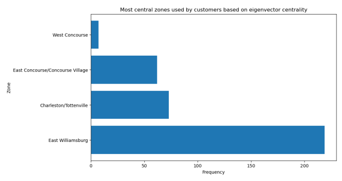
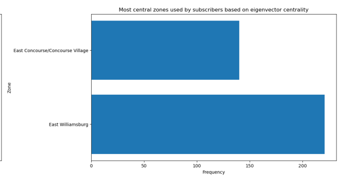


Fig. 21: The number of times each zone was the most central in Citibike network based on eigenvector degree centrality on a sliding month time window



(a) Customers



(b) Subscribers

Fig. 22: The number of times each zone was the most central in Citibike network of year 2020 based on eigenvector centrality on a sliding week time window

Fig.21 and Fig.22 shows that East Williamsburg zone is most frequently the most central zone based on eigenvector centrality for the year 2020 for both customer and subscriber network, while East Concourse/Concourse Village is most frequently the most central zone based on eigenvector centrality over the years. Many other zones also remain consistently in the top for different granularity.

## VI. Exploratory Data Analysis of Yellow Taxi Data

For all kinds of time windows, all the properties that were measured for Citibike network were measured for Yellow Taxi network as well.

### A. Number of Arcs

Fig.23 shows a gradual downward trend in the number of edges in the Yellow Taxi network over the years. This suggests that over time some commute routes between zones were replaced by something else. Additionally, there are 2 uncharacteristic dips in the plot in the end of 2014 and the start of 2015. These dips exist because all the Yellow taxi drivers of NYC went on hunger strikes during that time for months due to the introduction of Lyft and the popular adoption of Uber getting support from the government respectively. In general, the usage of Yellow taxis have dipped after 2011-2013 period

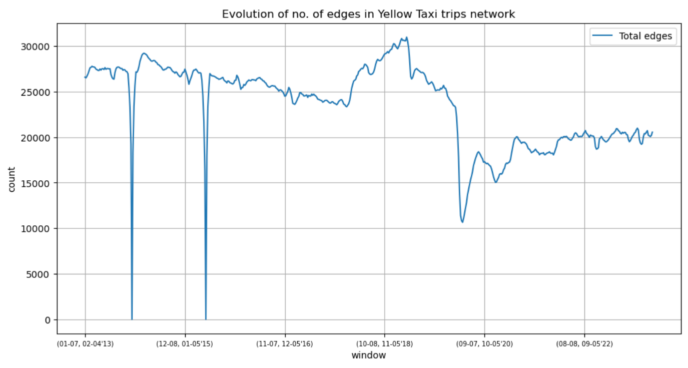


Fig. 23: Evolution of no. of edges in Yellow Taxi network across years on a sliding month time window

because of the introduction of many other public transport options such as Citbike, Green Taxi, Orange Taxi, Uber, Lyft, and many more.

### B. Number of Nodes

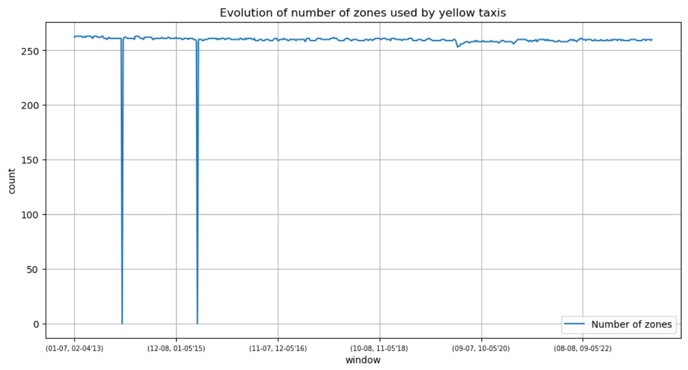


Fig. 24: Evolution of no. of nodes in Yellow Taxi network across years on a sliding month time window

Fig.24 shows a nearly constant plot of number of nodes, which suggests that all the zones are still seeing Yellow taxi activities. Even with the declining number of yellow taxi rides as seen in Fig.25, all the zones are still active. The two dips in the plot are consistent with previous observation.

### C. Number of Taxi Trips

Fig.25 shows a consistent decline in the number of passengers travelled by taxis as well as the total number of taxi trips with some seasonal variations. There is another sudden dip in the plot around early 2020, which coincides with the first wave of COVID-19. After that dip the number of trips have been increasing slowly, but is still significantly lower than the number of trips pre-COVID. This further enhances the fact that yellow taxi usage is declining due to the introduction of other means of public transport.

### D. Passengers to Taxi ratio

Fig.26 also shows a gradual decline in the ratio of number of passengers per taxi trip starting from 1.75 to nearly 1.3 at the end of 2023. This plot also sees the two hunger strike dips as well as the dip due to COVID-19.

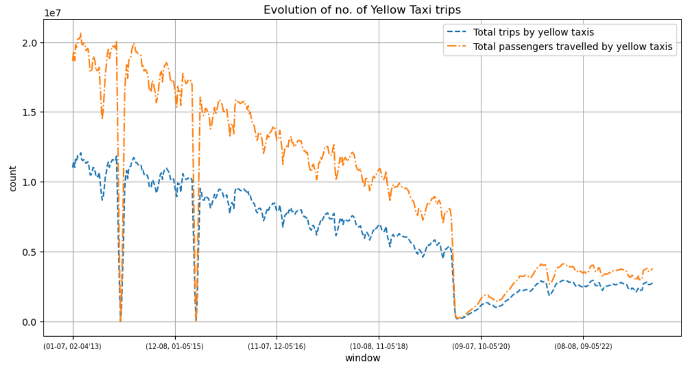


Fig. 25: Evolution of no. of taxi trips in Yellow Taxi network across years on a sliding month time window

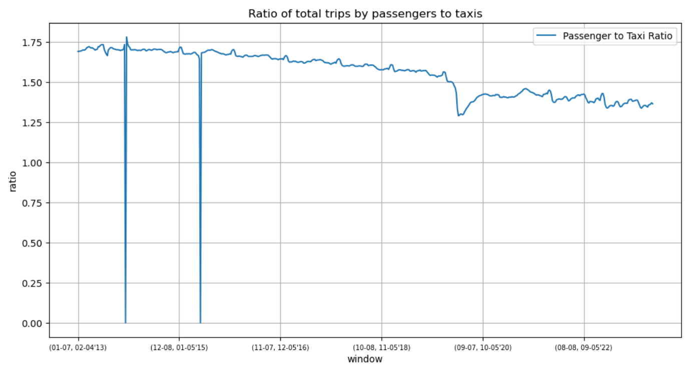


Fig. 26: Evolution of Passengers to taxi ratio in Yellow Taxi network across years on a sliding month time window

### E. Average Degree

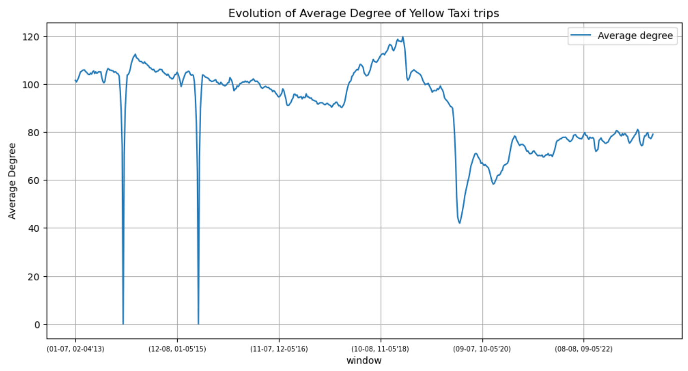


Fig. 27: Evolution of average degree in Yellow Taxi network across years on a sliding month time window

Fig.27 shows the evolution of average degree in yellow taxi network over the years, which is also consistent with our previous observations(The overall decline, two hunger strikes, and COVID-19 dip).

## F. Clustering Coefficient

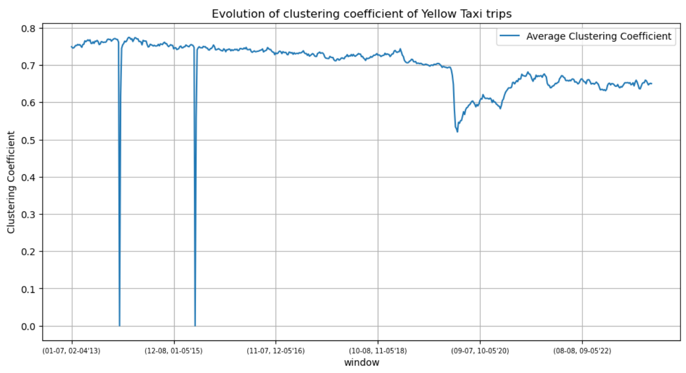


Fig. 28: Evolution of clustering coefficient in Yellow Taxi network across years on a sliding month time window

Fig.28 shows the evolution of clustering coefficient in yellow taxi network over the years, which is also consistent with our previous observations(The overall decline, two hunger strikes, and COVID-19 dip).

## G. Similarity in Community Structure

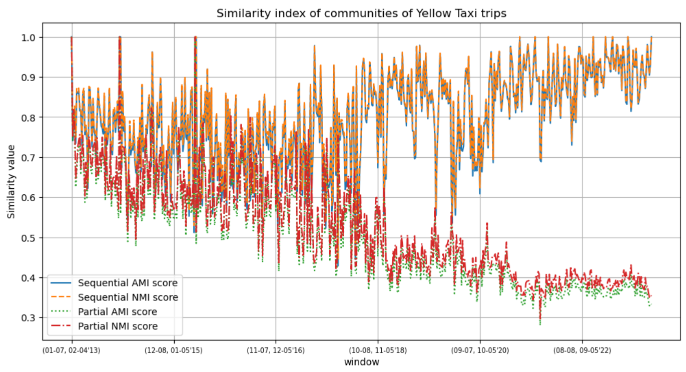


Fig. 29: Evolution of Similarity score in Yellow Taxi network across years on a sliding month time window

From the Sequential scores in Fig.29, we can see that the similarity in subsequent community structures is slowly increasing and it starts to stabilize after the end of 2021. This suggests that unlike the community structure in Citibike network, the community structure of yellow taxi network is not as robust and is affected significantly by sudden changes such as COVID-19. From the Partial scores in Fig.29, we can see that the community structure at the start of 2013 is slowly changing and keeps changing across the years in such a way that the community structure at the start of 2013 and the community structure at the end of 2023 are very dissimilar.

## H. Degree Sequence

From Fig.30, we can see that the percentage of highly busy zones steadily keeps decreasing over the years and are showing consistent dips for hunger strike and COVID-19. On the other hand, zones with low incoming traffic are steadily increasing.

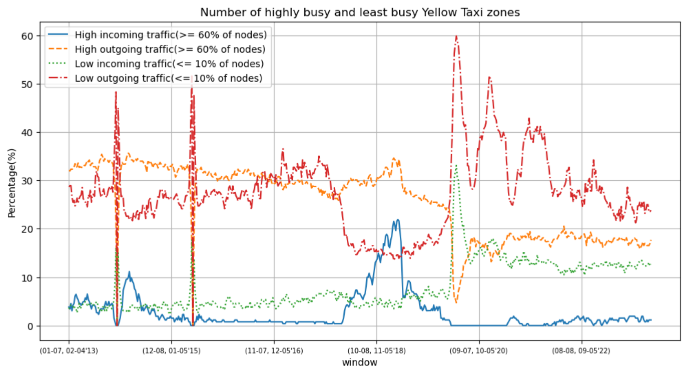


Fig. 30: Evolution of percentage of highly busy and least busy taxi zones in Yellow Taxi network across years on a sliding month time window

However, zones with low outgoing traffic doesn't show some predictable pattern and might give more insights on a more granular analysis, it is still consistent with the sudden changes of hunger strikes and COVID-19.

## I. In-Degree Centrality

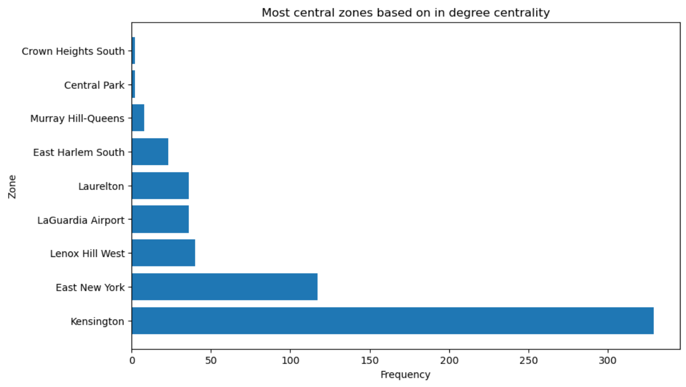


Fig. 31: The number of times each zone was the most central in Yellow Taxi network based on in-degree centrality on a sliding month time window

Fig.31 shows that Kensington zone has been most frequently the most central zone based on in-degree centrality.

## J. Out-Degree Centrality

Fig.32 shows that Kensington zone has been most frequently the most central zone based on out-degree centrality as well.

## K. Betweenness Centrality

Fig.33 shows that Kensington zone has been most frequently the most central zone based on betweenness centrality as well.

## L. Eigenvector Centrality

Fig.34 shows that Upper East Side South and Midtown East zones have been most frequently the most central zones based on eigenvector centrality.

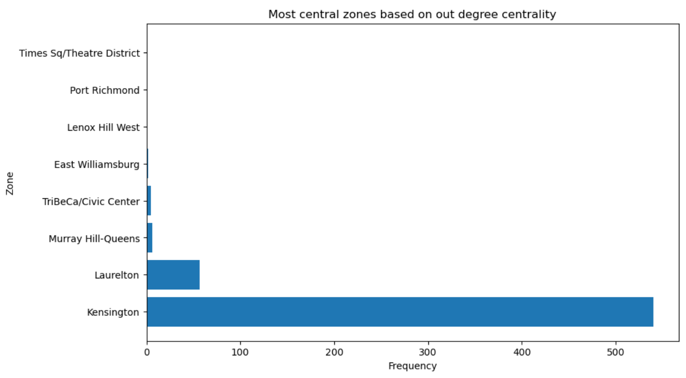


Fig. 32: The number of times each zone was the most central in Yellow Taxi network based on out-degree centrality on a sliding month time window

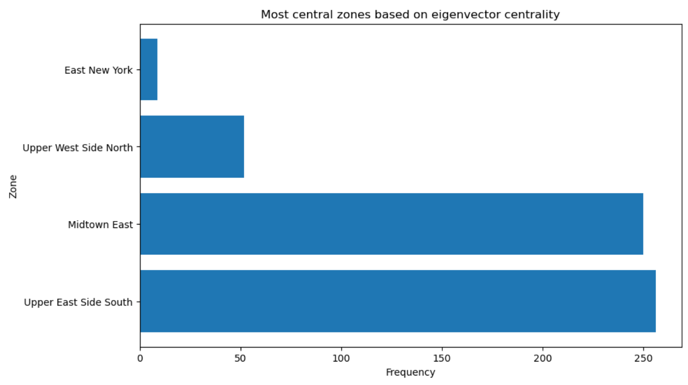


Fig. 34: The number of times each zone was the most central in Yellow Taxi network based on eigenvector centrality on a sliding month time window

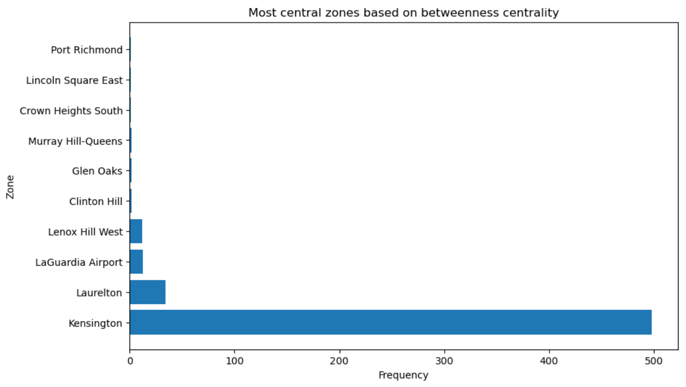


Fig. 33: The number of times each zone was the most central in Yellow Taxi network based on betweenness centrality on a sliding month time window

## VII. Predictive Modelling of Dynamic Graphs with Temporal Signals

The entire CitiBike Data from June 2013 to December 2023 was aggregated based on weekly tumbling windows, i.e., all the trips from one station(node) to another over one week were aggregated and added to make the edge weights between the respective nodes. Implying that we would have one graph for every week and therefore, approximately 52 graphs in a year $\times$ 10 and a half years. We have the data for a total of 552 graphs for the period as mentioned earlier.

After the initial Exploratory Data Analysis for the entire CitiBike Data, we wanted to build a machine learning/deep learning model that could take the graph structure over the years, node attributes and edge attributes as inputs and be able to predict those attributes for a future time (future week). This defines the problem as predictive modelling for Dynamic Graphs with Temporal Signals.

### A. Approach and Difficulties

Since the CitiBike data being handled here to predict future topologies and station-to-station trip frequency (edge weights) was unstructured and not conventional tabular data, it was challenging to find the right model appropriate for our problem statement. The problem at hand comprised two major components to be taken care of. One was, the changing nodes, implying new stations were being added and some old ones were being removed making the graphs dynamic. This comprised the spatial component. Secondly, the changing nodes, node neighbourhoods, and therefore evolving connections and topology over the years comprised the temporal part. One by one, we had to explore options that would suitably fit this dual component evolution and prediction as well as handle the computation efficiently. Another constraint for our problem was that we needed models that can not only predict edges but their weight as well.

Our first line of thought was to use classical GNN models used for traffic prediction such as STGCN or its variants. STGCN(Spatio-Temporal Graph Convolutional Network) had the limitation of taking Static Graph with temporal signals as input. That is why we tried to implement a variant of STGCN called D-STGCN(Dynamic Spatio-Temporal Graph Convolutional Network), which was able to handle dunamic graphs as well. It turned out that the purpose of D-STGCN was to predict node attributes instead of edge existence and edge weights.

The next attempt after exploring multiple model architectures, both built-in as well as open-source implementation examples from varying sources, we found an implementation example for TGN with Attention model [8] implemented from scratch for Twitter research [9] as mentioned above. The underlying algorithm used GNNs that could focus on handling dynamic graph data along with the temporal-attention mechanism which is used to learn the importance of different nodes and their connections at various time points. This helps the model to weigh the influence of different events or interactions based on when they occurred and their context within the graph. Typically, TGN with Temporal Attention works in the following manner: Temporal Attention: This mechanism adjusts the influence of different nodes based on the temporal context, such as how recent the interactions were. This helps in capturing the dynamic nature of the graph where more recent interactions might be more relevant than older ones. Memory Component: TGNs often include a memory module that stores state representations of each node, which get updated as new interactions occur. The attention mechanism can help in deciding how much of the new information should update the existing memory state. Message Passing: Similar to other GNNs, TGNs utilize a message-passing architecture where nodes communicate with each other. The attention mechanism can be employed to determine the relevance and weighting of these messages based on the temporal dynamics and past states. Aggregation: After computing the attention-weighted features from neighboring nodes, these features are aggregated to update the memory state of nodes, which can be used in conjunction with the spatial embedding of the graph at the current timestamp given by a graph convolutional layer for downstream tasks like prediction or classification. The TGN-attention mentioned above used the architecture in Fig.35.

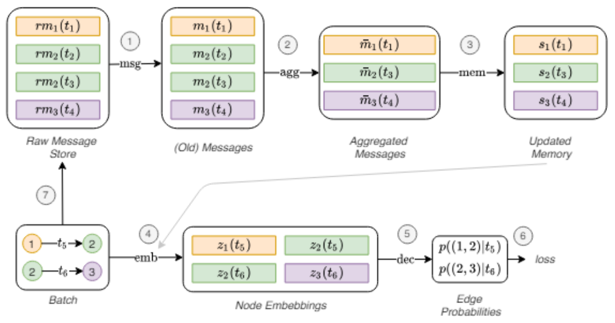


Fig. 35: Architecture used by TGN-attention

This implementation perfectly aligned with our problem at hand. Multiple attempts were made to comprehend the underlying structure and map the code to our problem and data. However, the complexity of implementation and lack of built-in libraries, made it extremely sophisticated for us to implement it in our project given the time constraints.

As mentioned in the section above, we explored multiple other models that could have possibly aligned with our problem but either had the recurrent (memory and time) component was not handled well or the spatial part was completely left out. If they seemed to fit, some of the underlying theory was incomprehensible and could not be mapped well from the code.

After these, we moved on to a simpler structure that seemed suitable to our needs. We explored the torch-geometric-temporal package [10] which is an extension of PyTorch Geometric, designed specifically to handle dynamic and temporal graphs.It supported several types of temporal graph neural networks (GNNs) that were specialized for handling changes in graph structure and node/edge attributes over time. Models like Temporal Graph Convolutional Networks (TGCN), Dynamic Graph Convolutional Neural Networks (DGCNN), and Recurrent Graph Convolutional Networks (RGCN), EvolveGCN, A3TGCN, TGCN, AGCRN, DCRNN were available within this library. The main difficulty with the torch-geometric-temporal package was to successfully install it in our local systems and the CIRC BlueHive cloud compute cluster. The package required some specific dependencies which were not aligning with the versions of other libraries already installed in the system as well as errors in ”wheel building”. After several trials in vain, we moved to Google’s Colab notebook in which the package could be imported successfully. However, in some of the required class implementaions of the library, the source code had the usage of several deprecated built-in functions that were no longer supported and had to be edited manually. After converting our data into the required format which was a ”DynamicGraphTemporalSignal” [11] object successfully which was challenging, we made the required changes in the source code manually and imported the code for entire classes being put to use. The DCRNN package was used along with an implementation of RecurrentGCN to model the spatio-temporal predictions. However, it was later discovered, that the libraries had its limitations on being able to predict node features only and could not predict edges (link prediction) and edge weights while handling the dynamic topology of the graph.

Finally we moved on to the raw implementation of Temporal Graph Neural Networks [12] in torch-geometric which has been explained in detail as under. The difficulties faced in its implementation while mapping the source code to our problem and data were primarily with multiple data-type casts and editing the model architecture such that it fits the weight prediction problem while handling dynamic topology along with link prediction that the original code aimed at. The architecture of Link predictor class before our manual modifications is as given in Fig.36.

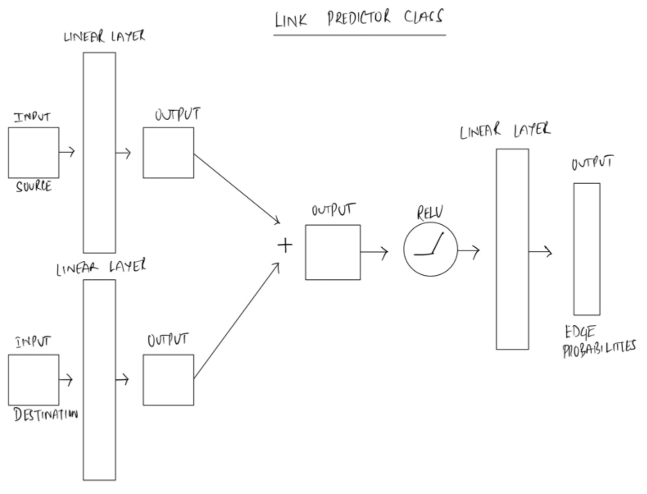


Fig. 36: Architecture of Link Predictor Class of TGN

We tried to create our own simple architectures that can be stacked on top of the existing code. The architectures we designed are as under:-

Architectures:-

- Alongside the Link predictor class of TGN, we created our own Weight predictor class, which took the actual weights from the data and the probability of edge existence (given by the Link predictor class) as input, which in turn will return the predicted weights. We created two different loss functions for this method. One being the BCElogitloss used by TGN for link existence prediction, and one being the MSE loss for weight prediction. We used two different optimizers and adjusted the gradients during the backpropagation accordingly to handle the gradients of our architecture. The architecture for this scenario is given in Fig.37.

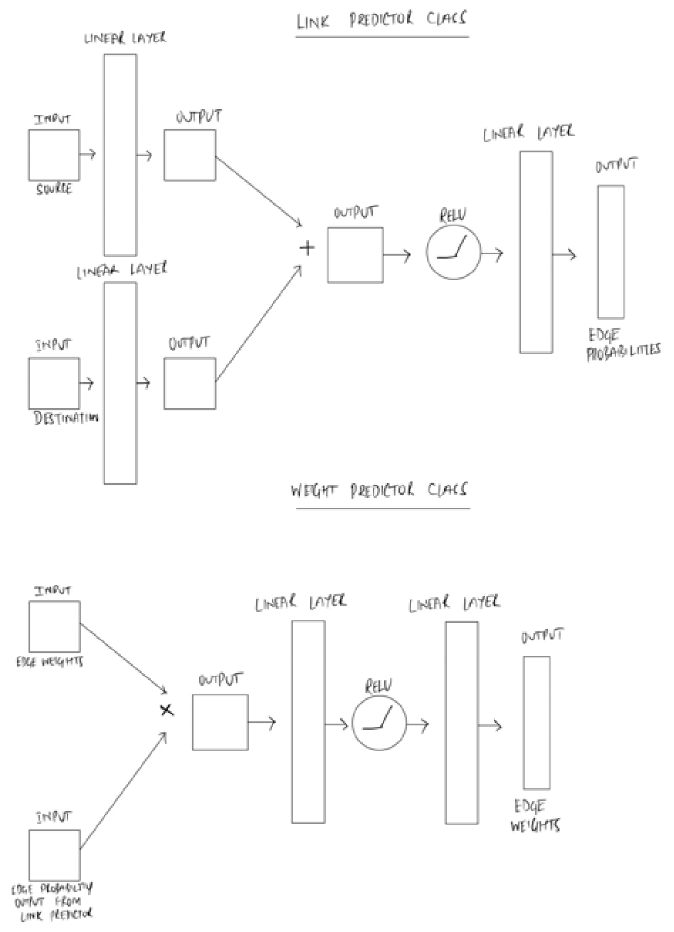


Fig. 37: Architecture for two different classes Link-Predictor and Weight-Predictor

- Instead of Link predictor, we created only one class - Weight predictor. Weight predictor was doing exactly the same thing as Link predictor. The only change was that we passed the actual weights as input alongside, source and destination. This architecture kept the calculation of edge probabilities exactly the same. After that the predicted probabilities were multiplied by the actual weights(so that the predicted links can have higher weights, and those links which have low probability of being present can have lower weights) before passing through two Linear layers with Relu as the activation function between the layers. In this case, there was only one loss function, which was MSE loss, as our model was returning only the predicted weights. The architecture for this scenario is given in Fig.38.

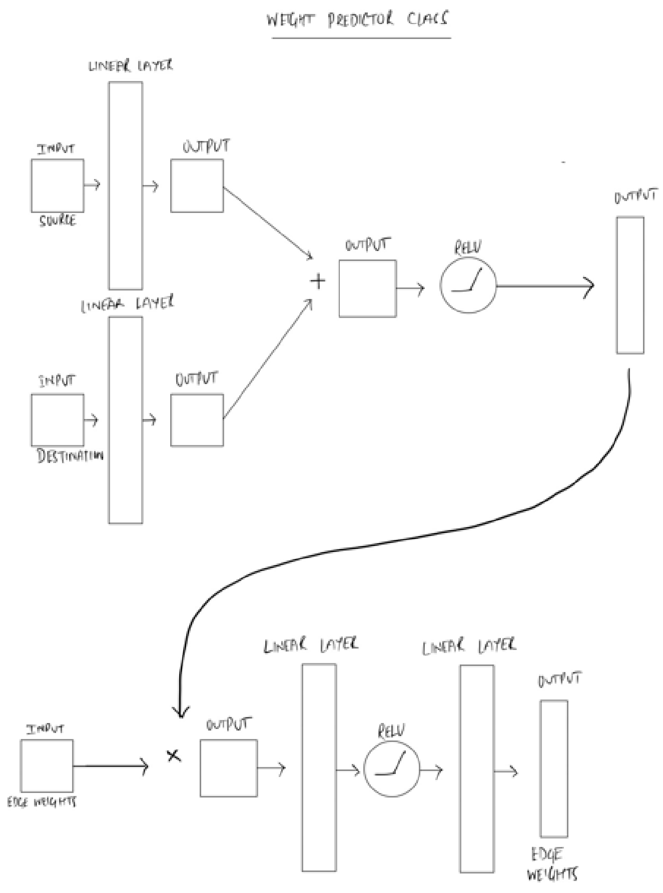


Fig. 38: Architecture for the class Weight-Predictor

Additionally, since the entire CitiBike data from June 2013 to December 2023 occupied almost 5-6 Gigabytes of memory, it was essential to bring it down to be able to process it with limited computational power. Hence the data was aggregated based on weekly tumbling windows making a total of 552 graphs as mentioned earlier. Finally, the currently working model (has been detailed in the following sections), was trained on a limited number of epochs (100), due to time constraint, leaving immense scope to improve model performance.

## VIII. Current Model

### A. Architecture

Our current architecture is using Graph Attention Embedding, which is internally using Transformer Convolutional

Network for spatial embedding of nodes. Then for the memory part, our model is using TGNmemory, which is storing the memory part which works on message passing method similar to that of TGN-attention and handles the temporal part. Then the combined output of past messages with current node embedding is called z. From this z, we take source and destination and pass it to Weight Predictor class which uses the architecture given in Fig.39. We modified the loss function of the source code of TGN and changed it to MSE loss by comapring this outputs to the actual weights. Then the optimizer step is used with learning rate = 0.001 to calculate the gradients. This neural network is given batches of size 200, and the dimensions of each layers of TGNmemory, Graph Attention Embedding, and Weight Predictor is set to 100.

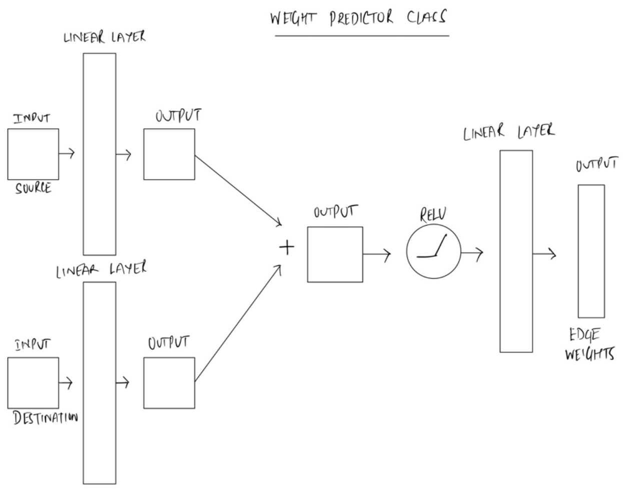


Fig. 39: Architecture of the class Weight-Predictor of our current model

## B. Evaluation

The model utilizes a memory module alongside a graph neural network, specifically tailored to manage and leverage the temporal aspects of the data. For testing, we initialize a deterministic seed to ensure reproducibility of results across different runs, then we process each batch of data, where each batch consists of nodes, their corresponding edges, and temporal features. The model generates predictions for edge weights based on the current state of nodes and their historical updates. We store these predictions, along with the source and destination node indices, for later analysis. Throughout the test, we maintain and update the state of the memory component with new interactions, ensuring that the temporal dynamics are accurately reflected in the node representations.

After processing all test batches, we compile all predicted edge weights and compare them against the actual weights from the test dataset using the Mean Squared Error (MSE) and R-squared metrics:

Mean Squared Error (MSE): This metric provides a straightforward measure of the average squared difference between the estimated values and the ground truth. It offers a clear quantitative measure of error magnitude, with lower values indicating better model performance.

R-squared: This metric indicates the proportion of the variance in the dependent variable that is predictable from the independent variable(s). An $R^2$ of 1 indicates that the regression predictions perfectly fit the data.

Interpretation of Results: The high MSE value of 13,385.15 suggests that the model's predictions are, on average, deviating significantly from the actual values in terms of squared differences. This high error magnitude could be indicative of model underfitting and inadequate learning due to limited computational power and time constraints. The R-squared value of 0.3979 indicates that approximately 39.79% of the variability in the true edge weights can be explained by the model. While not negligible, this suggests that the model has room for improvement in terms of capturing the variance in edge weights based on the graph's temporal dynamics.

The evaluation results highlight the model's current limitations in predicting edge weights accurately. To improve performance, we could explore various avenues such as hyperparameter tuning, enriching the dataset with more representative features of the temporal dynamics and most importantly, training on more iterations. Additionally, investigating the distribution of errors could provide insights into whether specific types of interactions or particular timespans are more challenging for the model, guiding targeted improvements.

## IX. Future Work

Due to time constraints, we were not able to achieve the originally intended simulation task, i.e., predicting how the introduction and removal of new bike stations and yellow taxis affect their respective networks as well as the other network. This has high utility and good potential as part of future work. Assessing how the introduction of new bike stations affects taxi usage - analyzing which locations are frequented more by taxis with no nearby bike stations. On the addition of a bike station in that location, prediction for change in bike network and its impact on the existing taxi network and vice versa for the introduction of new taxis. Additionally, another potential future work for this project is the inclusion of other various taxi networks like Green Taxi Network, Uber Taxi Network and other diverse categories. We could also add the subway network to this study and do an overall study of how these various transportation networks could potentially affect each other in the future using demand prediction and simulations.

## Acknowledgment

We are immensely grateful to our subject Professor Gonzalo Mateos Buckstein who helped us build foundation to be able to pursue this study as well as gave us this opportunity to explore details about concepts involving centralities, community detection, graph neural networks and more. We are also grateful to the creators and contributors of Pytorch Geometric who have left the code implementations of libraries in their package as open source due to which we were able to understand, edit

and implement existing and new architectures for our problem statement.

## References

[1] Saff, A., Bhandary, M. and Srivastava, S. (2022). Predicting Citi Bike Demand Evolution Using Dynamic Graphs. arXiv (Cornell University). doi:https://doi.org/10.48550/arxiv.2212.09175.

[2] Sapalidis, T. (2022). Predicting Evolution of Dynamic Graphs. [online] Stanford CS224W GraphML Tutorials. Available at: https://medium.com/stanford-cs224w/predicting-evolution-of-dynamic-graphs-7688eca1daf8 [Accessed 15 Mar. 2024].

[3] Li, A. and Axhausen, K.W. (2020). Short-term Traffic Demand Prediction using Graph Convolutional Neural Networks. AGILE: GIScience Series, 1, pp.1–14. doi:https://doi.org/10.5194/agile-giss-1-12-2020.

[4] Davis, N., Raina, G. and Jagannathan, K. (2018). Taxi Demand Forecasting: A HEDGE-Based Tessel- lation Strategy for Improved Accuracy. IEEE Transactions on Intelligent Transportation Systems , 19(11), pp.3686–3697. doi:https://doi.org/10.1109/tits.2018.2860925.

[5] citibikenyc.com. (n.d.). Citi Bike System Data — Citi Bike NYC.[online] Available at: https://citibikenyc.com/system-data.

[6] www.nyc.gov. (n.d.).TLC Trip Record Data - TLC. [online] Available at: https://www.nyc.gov/site/tlc/about/tlc-trip-record-data.page

[7] R. Shekhar, ”NYC-transport,” GitHub. https://github.com/r-shekhar/NYC-transport.

[8] Rossi, Emanuele, et al. TEMPORAL GRAPH NETWORKS for DEEP LEARNING on DYNAMIC GRAPHS.

[9] “Twitter-Research/Tgn.” GitHub, 10 May 2024, https://github.com/twitter-research/tgn. Accessed 10 May 2024.

[10] Rozemberczki, Benedek. “Benedekrozemberczki/Pytorch_geometric_temporal.” GitHub, 9 May 2024, https://github.com/benedekrozemberczki/pytorch_geometric_temporal. Accessed 10 May 2024.

[11] “Torch_geometric_temporal.signal .dynamic_graph_temporal_signal — PyTorch Geometric Temporal Documentation.” Pytorch-Geometric-Temporal.readthedocs.io, https://pytorch-geometrictemporal.readthedocs.io/en/latest/_modules/torch_geometric_temporal/signal/dynamic_graph_temporal_signal.html. Accessed 10 May 2024.

[12] pytorch-geometric. “Pytorch_geometric/Examples/Tgn.py at Master · Pyg-Team/Pytorch_geometric.” GitHub, https://github.com/pyg-team/pytorch_geometric/blob/master/examples/tgn.py. Accessed 10 May 2024.