Graph neural networks for traffic modeling
Project Laboratory 2.

Benedek Máth, Ádám Nemes
Supervisor: Dr. Varga Balázs

***Abstract*—The present study experimentally investigated the efficiency of Graph Neural Networks (GNNs) in the context of traffic flow prediction. We generated a dataset using a microscopic traffic simulation tool - SUMO on the traffic network of Győr. Our focus was on car counting data, with partial coverage of the graph's edges. The Graph Neural Network was trained to estimate the values on the covered edges using data from the remaining edges, simulating the presence of car-counting sensors (e.g.:inductive loop detectors, cameras) in a real-world scenario. We used graph convolutional layers to effectively capture the structure of the graph in our model, allowing us to make accurate predictions based on the connections within the traffic network and we worked out an algorithm to choose which roads should be covered with sensors. We successfully trained the relatively simple model to estimate the traffic volume on the covered edges based on the data from the other edges.**

I. Introduction

Graph Neural Networks are becoming increasingly popular across several fields because they are better at capturing certain features of data structures described by graphs than the usual dense or even convolutional networks. Such fields involve anything where graphs are a good choice of describing a phenomena for example: chemistry, for instance, the authors in [1] want to classify the role of a protein within a biological interaction graph, and physics [11] analyzes interactions in a physical system, in a social network the authors of [6] want to predict an individual's role in a collaboration network, and recommend new friends to a user in a social network. Our aim is to analyse road traffic with graph neural network and determine if a graph neural network can capture the flow of traffic from a limited amount of information, in our case traffic flow data from a few road links.

The Traffic Sensor Location Problem (TSLP) is a widely researched topic since the question of how to measure the traffic flow effectively keeping the costs at a sensible level is very important in the field of traffic control. The current state-of-the-art method consists of creating an Origin-Destination matrix and deriving an Integer-Linear-Programming problem and solving it to find the best places for the sensors with some constraints, like all O-D pairs should be covered and the number of sensors are given. Since to the best of our knowledge, our work has not been done before, meaning that Graph Neural Networks have not been used for building a traffic estimation model in a city. The contribution of this work is to solve the TSLP using GNNs and a heuristic for partitioning the city road network. [8], [12], [10], [13]

This work is structured as follows, first we present what SUMO is, and how we used it to generate a data set. Then we give an introduction to GNNs, their advantages and basic properties. In the third section we describe how we built our model. In the fourth section we present our zone-based algorithm for solving the TSLP. In the fifth section we describe how we trained our model, and present our results and in the last section we summarize our work and results.

II. Methodologies

A. Introduction to SUMO

SUMO, short for "Simulation of Urban Mobility," is an open-source traffic simulation package designed to handle large road networks. Developed by the German Aerospace Center (DLR), SUMO is widely used in both academic research and industry to simulate and analyze various transportation scenarios. It provides a comprehensive set of tools to model the behavior of vehicles, pedestrians, and public transportation systems within urban environments.

SUMO has many features: It is Open-source and extensible, it is a Multi-Modal traffic simulator thus it supports many forms of transport such as cars, bicyles, public transport, and pedestrians. SUMO is scalable thus it can handle complex road networks and thousands of cars. SUMO includes tools for visualizing vehicle movement and traffic flow in 2D.

SUMO provides a wide range of data output options, allowing users to perform in-depth analysis of traffic dynamics and performance metrics. This detailed data collection capability is essential for understanding traffic patterns, evaluating the impact of traffic management strategies, and conducting various transportation research studies.

SUMO can create a traffic road layout with road junctions traffic lights that can be configured by the user. SUMO is a microscopic traffic simulator, meaning that each vehicle is modeled as individual agents with their own -simplified-dynamics, called the car following model. The movements of vehicles on a lane is continous in space and discrete in time. Then when the simulation start it slowly fills the roads with vehicles up to the set traffic scale. SUMO simulates every car individually with a route to go along obeying traffic laws, and saves numerous data from the simulation such as emission data, incident data, vehicle-specific data for example speed and acceleration, travel times, and traffic flow data. To train our model we used vehicle count data, specifically the count of cars that left a road segment.

We used the road map and traffic light system of the downtown of Győr to train our model. For the sensor placement strategy we also used the roads' data, the travelling average speed of the roads and their length. The required files to create the road map were provided by our supervisor. One simulation

generates data for a four hour long traffic flow. We set the period of car counting to fifteen minutes thus creating sixteen data point with one run. The data from the initial two time periods were excluded from our analysis. Since in the first half an hour the simulation loads the road network with cars, we are only interested in the states where the traffic flow is relatively stable.

Fig. 1. A section of Győr's map in SUMO simulation.

To create the training data we ran several simulations on Győr's roadmap with multiple modifications to acquire a diverse training data set. We mainly modified the global seed of the simulation and the traffic's scale. On Figure 1. Győr's part around coordinates $47.688059, 17.644442$ is shown.

In SUMO seeds are used to initialize the random number generators that control various stochastic processes within the simulation e.g.: veichle insertion, driver imperfection. Global seed is the primary seed that influences all random processes within the simulation. By setting the global seed, users ensure that the same sequence of random events occurs each time the simulation is run, which is crucial for reproducibility. In our case it was necessary to change the seed in every run to obtain a variety of data. We used 25 different traffic seed from 38-62.

The traffic scale parameter uniformly scales vehicle inflows to the network. As we set a traffic scale higher the number of cars are increasing in the simulations, more traffic jams occur the speed of the movement slows down.

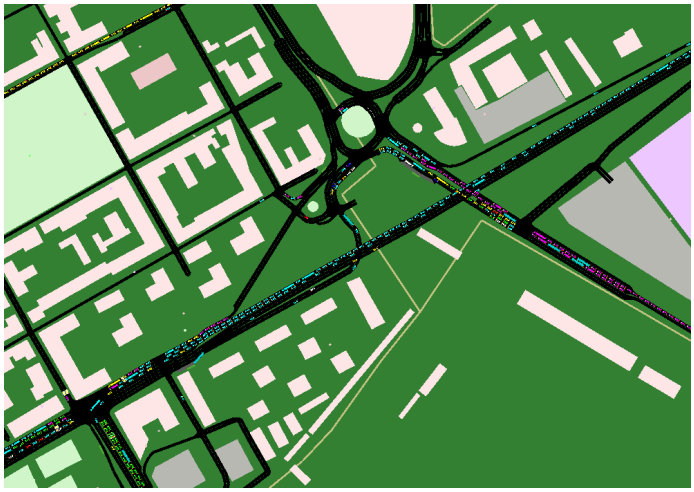

Fig. 2. SUMO simulation with 1.8 traffic scale.

As depicted in Figure 2, a notable increase in traffic volume is observed on the road network simulated using SUMO, leading to significant congestion patterns. We used four distinct traffic scale parameters to generate our data $\{1, 1.1, 1.2, 1.4\}$ thus we obtained in total a hundred different simulations with four types of traffic cale and 25 global seeds. [5]

B. Introduction to Graph Neural Networks

Graph Neural Networks(GNN) are a type of neural network designed to perform inference on data described by graphs. GNN's input are X the matrix of node feature vectors X_i. $G = (V, E)$ with $v_i \in V$ edges, N nodes, $(v_i, v_j) \in E$, an adjacency matrix $A \in \mathbb{R}^{N \times N}$.

In our case the nodes of the graph are road intersection and the edges are roads connecting them, also there are longer roads that we broke into multiple sections of edges and nodes. GNNs are using the adjacency matrix of the graph to capture the properties of the system. Nodes that are connected by edges and edges connected by nodes have greater impact on each other than those which are not. GNNs are using graph convulotional layers to pass information through the graph. In the first layer the edge/node gets information from its direct neighborhood that what we call it's first neighbor, in the second layer it gets information from the first neighbors' direct neighborhood and so on. From k'th layer it will get information from edges/nodes which can reach it in exactly k steps.

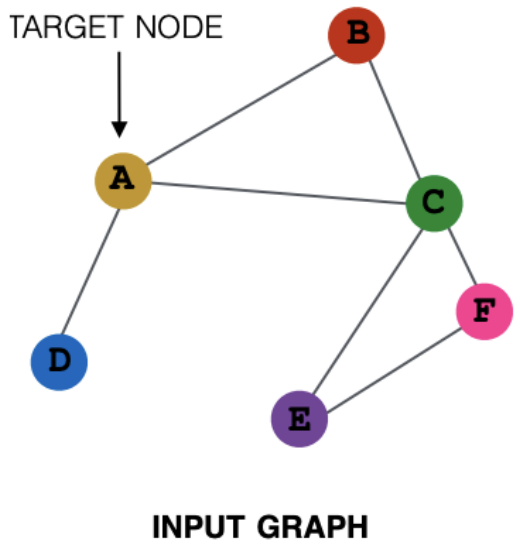

Fig. 3. Example graph (source: https://neptune.ai/blog/graph-neural-network-and-some-of-gnn-applications)

In this case shown in Figure 3 the target node A gets information from nodes: {B, C, D} in the first layer, in the second layer from B, C, D nodes neigborhoods: from {A (itself), C} these are the neighbors of B, from {A, B, E, F} C's neighbors and from the only neighbor of D {A}. This information passing method can be seen on Figure 4

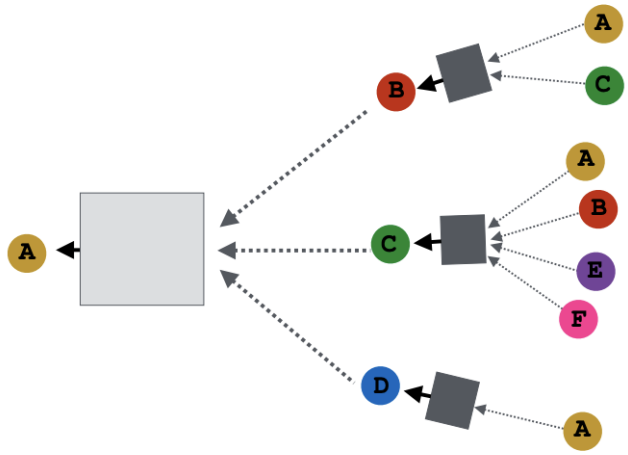

Fig. 4. Example graph (source: https://neptune.ai/blog/graph-neural-network-and-some-of-gnn-applications)

We used a multi-layer Graph Convolutional Network (GCN) layers with the following layer-wise propagation rule:

$$H^{(l+1)} = \sigma\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}\right). \quad (1)$$

Here,I_N is the identity matrix, $\tilde{A} = A + I_N$ is the adjacency matrix of the undirected graph G with added self-connections. $\tilde{D}$ is the degree matrix defined as $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$. $\sigma(\cdot)$ denotes an activation function, such as ReLU ($\mathrm{ReLU}(x) = \max(0, x)$), and $W^{(l)}$ is a layer-specific trainable weight matrix. $H^{(l)} \in \mathbb{R}^{N\times D}$ is the matrix of activations in the l-th layer; $H^{(0)} = X$. [4]

III. The Model

Our model is fairly simple, we used 8 Graph Convolutional Network Layers (GCN), and a dense layer. The latter is the one, that gives the final output, this is common practice. The reason for the number of GCN layers, is that we had to make sure, that every unknown road is within the reach of at least one known road, otherwise, we couldn't make a prediction on the given road. Since the edge values cannot be negative, after each GCN layer we used a ReLu activation function on the output. One interesting technical detail we ran into was, that if we put a ReLu function on the output of the final, dense layer, the model fails to learn. Therefore, we had to put the ReLu function by hand on the output of the model. Our results showed, that this fairly simple model already performed pretty well, therefore we did not experiment with more complex layers. [9]

In our project we used Adam optimiser. Adam is a popular optimizer in deep learning due to its adaptive learning rate approach, which helps in faster convergence and better handling of complex models and datasets compared to simpler optimization methods. It achieves this by maintaining per-parameter learning rates that are dynamically adjusted based on the first and second moments of the gradients. This adaptive behavior is particularly beneficial for models with high-dimensional parameter spaces or non-linear activation functions, where traditional SGD may struggle to find optimal solutions efficiently. We set the learning rate to 0.005, which is dictating the magnitude of parameter updates at each iteration. [2]

In the training of the model we used the data generated using SUMO. In this subsection we will call data points the members of the generated data set, that describes what was the traffic volume on a given road, in the last 15 minutes. For the training of the model, we used all 1358 data points. During the data preparation, we normed all the traffic volumes with a normalizing factor, as neural networks generally work better if the data is between certain bounds, in our case these are numbers between 0 and 1. We made an engineering judgement to find the normalizing factor. As a rule of thumb an arterial road (with no traffic lights) has a peak capacity of 1800veh/hour/lane. We set our normalizing factor to $\frac{1800\cdot 2}{4} = 900$ since we have quarterly hour data and two lane roads we divided by four and multiplied by two. [7] We wanted to make a clear distinction between the known and unknown roads. We had 3 options for this:

1) Set the traffic volume of the unknown roads to some non-negative number
2) Set the the traffic volume of the unknown roads to NaN
3) Set the traffic volume of the known roads to some negative number

The first option does not work, as the positive numbers have real meaning for our problem. The second option also does not work, as the GCN does not work well with NaN values, therefore we settled with option 3, and set the traffic volumes (edge attributes) to -1.

For the training of the model we created a training dataset, where we hid the traffic volumes of the unknown roads in the way written above, and a corresponding validation dataset, where we preserved all the edge features. Then we generated an output based on the input coming from the training dataset, and compared it with the corresponding element of the validation dataset. It is important to note, that the training and validation data should come from the same data point, as in our current goal it does not make sense to make an estimation for time t_2 based on the observations of t_1. We used Mean Squared Error as loss function and backpropagation for correcting the model.

IV. Zone-based algorithm for sensor placement

In this section we are going to go into the details of the algorithm we worked out for choosing the roads with sensors. We will explain our method using a small graph, and we will present figures of Győr at the key points.

We will present our zone-based algorithm for sensor placement using the figure 5 and we will place 5 sensors on 5 roads in this case. This is just a 'dummy' graph for better explanation. The weights on the edges in this case represents the number of cars that left the given road in the given timeframe. These numbers were generated randomly using Python.

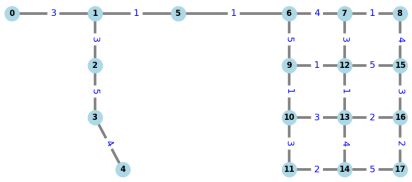

Fig. 5. We will present our methodology using this example. The numbers on the edges are the traffic volume that used the given road. This figure is the first of the series of figures based on this graph explaining our zone-based algorithm. The traffic volume on the edges were generated randomly using Python.

Based on the relevant literature we got the idea of dividing the city into 'districts' and a district leader with a size that suits the model we are using. A district is a set of roads located closely to each other, creating one neighbourhood of roads. The district leader is either the center of this set of roads or a road identified by an expert that represents well that set of roads. Once created the district leader is assumed to be the origin and destination point of the corresponding roads. Our goal with this is to lower the number of origin and destination points from 504 (the number of roads in our case) to a manageable number (20-30). We believe that the GNN will learn the information of a district based on this approach if we choose the district to have around the same diameter as the number of layers the model has. The general idea is to have the districts' diameter slightly smaller than the model's number of layers.

While this is a good idea in general, it could be the case (and it is), that some previous expert knowledge comes into the picture and identifies some of the roads as major origin or destination points, because many veichles would either enter or leave the system through those roads. If we have this, then it is worth it to choose these roads as a district leader, and based on their position in the graph we might choose a different district size for these. Let's say expert knowledge tells us, that the $(0, 1)$ edge is such that it should be chosen as a district leader. Looking at our graph structure it also makes sense, to increase the size of the neighbourhood we will assign to this district. In this case let's choose this kth neighbourhood in terms of edges to be 2. This figure is on 6. Now we ran out of edges chosen by experts, therefore we have to work out the rest by ourselves. In order to do so, now we choose a road randomly to be a district leader. Let's say the random choosing ended up on edge (3,4), the red one on 6. In this case it does not make sense to create a district for this edge only. The way we check this in our algorithm is that we introduce a hyperparameter called lower bound. We calculate the proportion of the roads not-yet ordered into any district of the kth neighbourhood of the now-selected edge, and if it is higher than the lower bound, then we accept this as a district leader, if it is lower, then we check which districts have a road in the kth neighbourhood of the edge, and select randomly from those districts to assign this would-be-district-leader to one of the already existing districts. In this example graph case the red edge is assigned to district 1. We had to implement this logic in order to reduce the number of district leaders to the desired 20-30 range instead of 40+. Note, that the number of districts is also a hyperparameter of the model, the 20-30 range in case of the 504 roads made sense to us, however we did not perform cross-validation to find the best number of districts. In our real case the lower bound was chosen to be 0.3. We found that with this value the number of districts will be somewhere between 20 and 30 as we wanted.

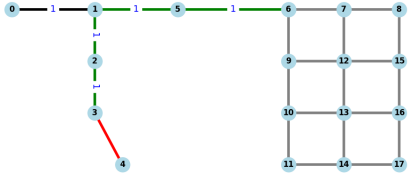

Fig. 6. Let's say that the random choosing ended up on the red edge. In this case the algorithm checks, what is the percentage of its 1st-step neighbour edges that is not in any district. If it is lower than the lower bound the would-be district leader is assigned to an already existing district. In this case the red edge gets assigned to district number 1.

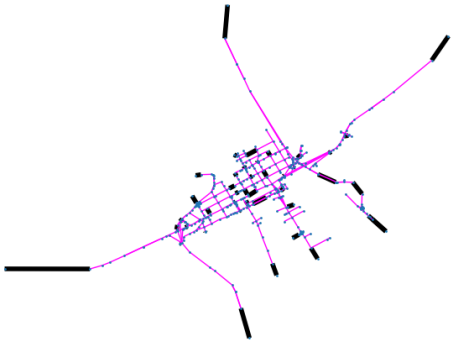

Fig. 7. On this figure we show how would Győr look like in terms of district leader location without the logic to blend this fragmentedness into a smoother result. On this figure the black edges are district leaders, the magenta edges are not. As we can see, some of the district leaders are very close to each other

Now let's say, the next road that we choose randomly is the (9,12). Now we get to the more road-dense part of the graph, therefore we will only check the 1st-step neighbourhood. Since none of its 1st-step neighbours are assigned to any district, all of them are assigned to it. This figure is on 8. Now let's assume the next road that was selected randomly is the (13,14). For this road, one of its 1st-step neighbours are already assigned to a district, but the others are not. Therefore we accept this road as a district leader, and assign the roads to it accordingly 9. Now we explained every detail of the district creation methodology, so let us assume, that the random choosing was such that it resulted in the districts on 10.

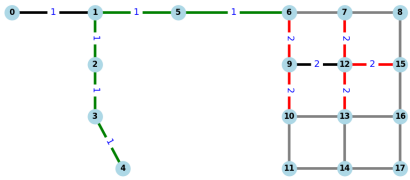

Fig. 8. Second step of creating the districts, the (9,12) edge is a new district leader, and every 1st neighbour of it is assigned to its district

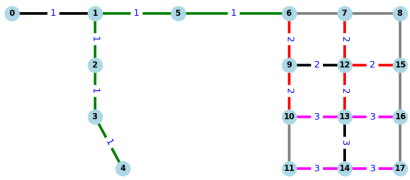

Fig. 9. Third step of creating the districts, the (13,14) edge is a new district leader and with the 1 exception of the (12,13) edge that is already assigned to district number 2, we assign the 1st-step neighbours to it.

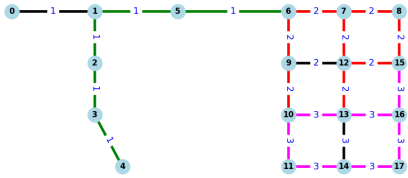

Fig. 10. Let us assume, that we got these districts. The labels on the edges are the number of district that they belong to. The black edges are the district leaders.

Now we successfully created the districts. The result of this process is on figure 11. We introduce a weight system that we will use in the future. The district leaders get the weight of the total traffic volume that is present in their district. In our case it means, that we sum up the number of veichles that left the roads. This is necessary for the road selection process.

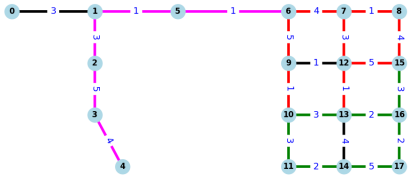

Fig. 11. The original data with the districts. The black edges are the district leaders, the differently coloured edges belong to different districts.

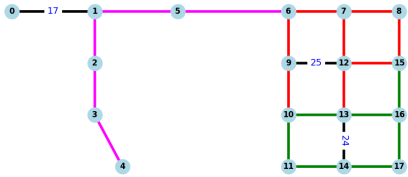

Fig. 12. The district leaders get the weights based on the district's traffic volume.

Before moving on with the explanation of how we use these districts in the choosing process we show a figure of Győr where the same-colored edges belong to the same district. The black edges are the district leaders. In our case the expert knowledge is that the collector roads are the origins or destinations for many veichles. Since these are far from the denser areas we decided to use 7th-neighbourhood in initializing these districts. We get the kth-neighbourhood of a road -let's call it AB by collecting every road that we can reach from AB in k many steps. Note, that even with this number the diameter of the district is not larger than 8. In the denser areas we used 3rd-neighbourhood during initializing the districts, which also results in a diameter not larger than 8. Admittedly, the logic which we used to reduce the number of districts permits the algorithm to create districts with diameters larger than 8, however the way we use these districts to choose the locations of the sensors deals with this problem.

Fig. 13. We can see how using the algorithm explained above results in districts in our real city, Győr. The black edges are the district leaders, the same-colored edges belong to the same district.

Now we have the district leaders and their weights. For the first step of choosing which road should get a sensor, we calculate the shortest routes in time between the district leaders pair-wise, and whichever road makes it to any pairing's shortest route gets into this set of roads. In case of Győr, this can be easily be done, as the time necessary for driving through any road can be calculated from the SUMO file's velocity and length data which is given for every road. With this time-weight applying the Dijkstra algorithm correctly we can easily get the shortest routes. Within this subset of roads we introduce 2 weight system. One of them is a list of the district leader pairs it connects, the other is the sum of the district leaders' weights it connects. If a road connects more than 1 pair of district leaders, then the list is longer, and the traffic volume weight is added up. On figure 14. we have the roads chosen by the Dijkstra algorithm. First we order these roads by these weights such that after the ordering the top road has the longest list. In case of a tie, the deciding factor is the weight, the road with the larger weight should be in front of the one with the smaller. After the ordering we select the first road, this will get a sensor. We update every other roads' list in such a way, that we remove the first road's list's elements from every other road. With these updated lists, we repeat this step until every road's list is empty. This idea was based on relevant literature, we are trying to make sure, that regardless of the veichle's origin and destination, we 'catch' it with one of the sensors. [8], [12], [10], [13]

If we still have sensors to place, we will now move on to select roads based on the weights only. One problem we might run into in this case is that if we have a long, busy road divided into many parts, then we would be choosing all of those parts one-by-one, not leaving any place for completely different roads. In order to avoid this problem, once we choose the road with the highest weight, we eliminate its 1st-step-neighbourhood from the selection. We repeat this step until the set of edges chosen by the Dijkstra algorithm becomes an empty set. If we still have sensors left, we choose from the district leaders randomly.

Our results with Győr was that even if 30% of the roads are covered with sensors, the logic explained above is enough and finds a place for every sensor, therefore we do not need further logic to find the places for the sensors. The figures 14-17. show the process we explained above. The first road we choose will be $(5, 6)$ as it has 2 elements in its list and its weight is higher than the others -we did not implement a third ranking weight, in this case $(5, 6)$ and $(1, 5)$ are in a tie- so it is chosen. After choosing it, we update all edges' list and remove 'AB' and 'AC' from them. This results in only having $(12, 13)$ as an edge that have any element in its list, so we choose this as the next road. Now all of the roads' lists are empty, so we move on to the weights only. The largest weight is on $(1, 5)$ so we choose this. None of the remaining preselected edges are in its 1st-step-neighbourhood, so we do not remove any of them. Among the now remaining edges $(6, 9)$ has the largest weight, so we choose this as the fourth road and remove $(6, 7)$ as it is in the 1st-step-neighbourhood of $(6, 9)$. With only $(7, 12)$ remaining we choose it, and we are finished, as our goal was to find places for 5 sensors.

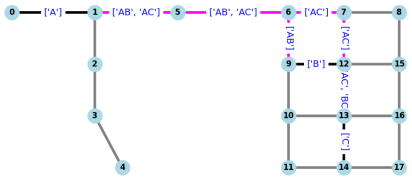

Fig. 14. The magenta roads were chosen by the Dijkstra algorithm. These roads have the list of district leaders they connect as edge labels. The first road we choose is $(5, 6)$ as its list has 2 elements and it has the highest weight.

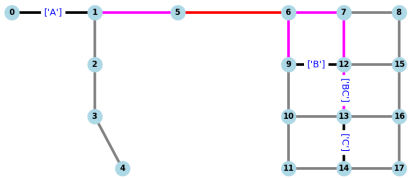

Fig. 15. After updating the lists of the edges, only $(12, 13)$ has a non-empty list, so we choose that next.

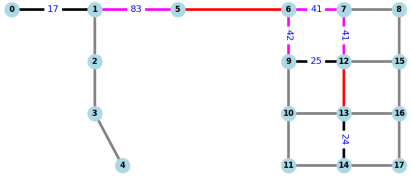

Fig. 16. The third road to choose is the $(1, 5)$ as it has the highest weight. The fourth road to choose is the $(6, 9)$ as it has the highest weight out of the remaining ones. We also remove $(6, 7)$ as it is in the 1st-step-neighbourhood of $(6, 9)$.

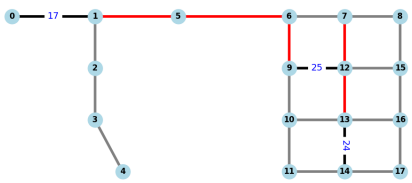

Fig. 17. The only remaining edge from the result of the Dijkstra algorithm is the $(7, 12)$ so we choose this lastly.

In the case of the Győr, we show 2 figures during this step of the algorithm. On figure 18. we show the result of the Dijkstra algorithm, altough out of 504 edges still more than 300 are selected by this step, we still excluded many roads. On figure 19. we show the final result of the selection process with 10% of covered roads. We can see, that near the city-side end of every BEVEZETŐ ÚT there is a sensor, and around the main road of the city there are more.

Fig. 18. The black edges are selected by the Dijkstra algorithm

Fig. 19. The black edges are the chosen edges, which get a sensor. In this example 10 % of the roads are covered with sensors.

V. Training the model and Results

In this section, we provide a summary of the insights gained from our study, which utilized a Graph Neural Network (GNN) to assess road traffic density based on varying levels of sensor data. By testing the model's performance across 10, 20, and 30 percent sensor data availability, we observed that an optimized, zone-based sensor placement consistently outperformed random placements.

A. Training of the model

For the training in order to avoid overfitting for a given datapoint we decided to shuffle the data. We decided that for every epoch, we chose the teaching data point randomly. In this way we avoided overfitting. The loss-epoch graph is on Figure 20. In order to determine our model's performance and to see whether it learns or not, we used a benchmark. Our benchmark was the average loss that we would get, if the model does not work, and produces only zeros as output. This means that our benchmark is essentially the mean squared average of the 1358 data points we are working with. We can see that on average, our model produced 50% decrease compared to this benchmark, however the losses vary a lot, depending on which data point we chose for the given epoch.

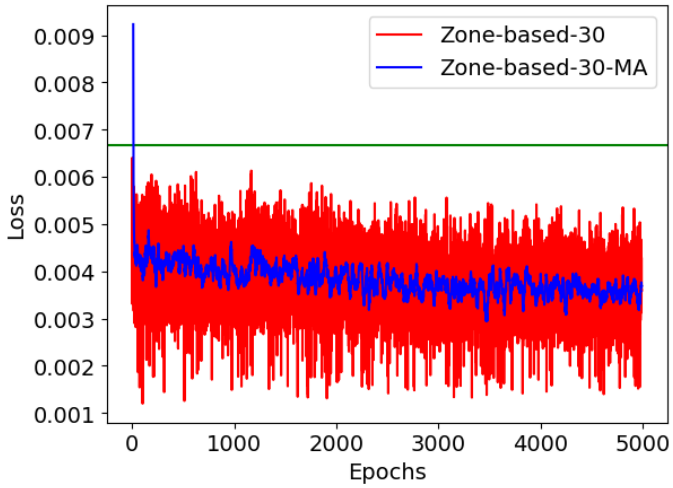

Fig. 20. Loss-epoch graph for the model. On the graph the red line is the loss function's values comparing the test data points with the output of the model. The blue line is the moving average of the loss function. The green line shows what the average loss would be, if the model's output are only zeros. We used this as benchmark to determine our model's performance. The model's performance does not get significantly better as we run more epochs on it, in fact, it reaches its minimum after around 4000 epochs, and then it overfits in a way.

B. Comparing models

In order to capture the average performance better, we calculated after every 20 epoch the average Mean Squared Error for every data point. In order to do this, we created the output of the model for every data point, calculated the mean squared error for them, and calculated an average of that. Comparing the three zone-based models, our findings reveal an interesting progression in performance, particularly regarding sensor coverage at 10%, 20%, and 30%. As we can see on Figure 21 initially, the 20% coverage model surprisingly outperformed the 30% model, likely due to the random initialization of the GNN model.This early performance anomaly persisted for the initial phases of training; however, as the training progressed and the model adjusted over approximately 2000 epochs, the

expected ranking emerged with the 30% model outperforming both the 20% and 10% models. This progression underscores the model's capacity to learn effectively over time, ultimately favoring higher sensor coverage.

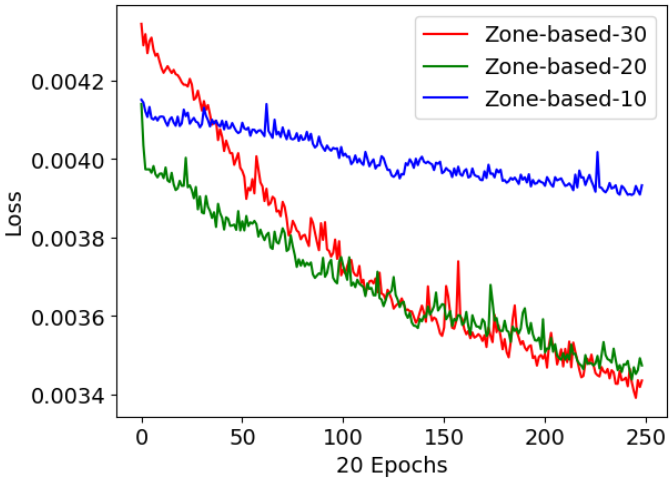

Fig. 21. Loss-epoch graph for the three models. On the graph the red, green ,blue liness are the average loss function value calculated on all the test data graphs at 30%, 20% ,10% known edges.

In the following three Figures22-24, we present a side-by-side comparison of the random and zone-based algorithm deployed sensor models at 10%, 20%, and 30% coverage. The results reveal that all 10%,20% and 30% zone-based models consistently outperformed their random counterparts. Therefore, these findings highlight that while having more data and placing sensors thoughtfully is usually helpful, randomly placed sensors can still work well in some cases, especially when there's a moderate amount of coverage.

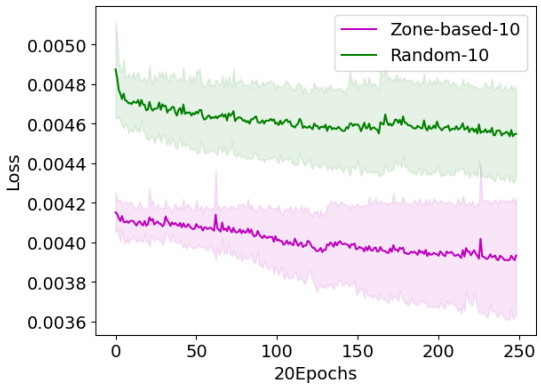

Fig. 22. Loss-epoch graph for two models. On the graph the green and magenta lines are the average loss function value calculated on all the test data graphs at 10% known edges with zone-based and random sensor placements. The green and magenta areas are 2 deviation wide.

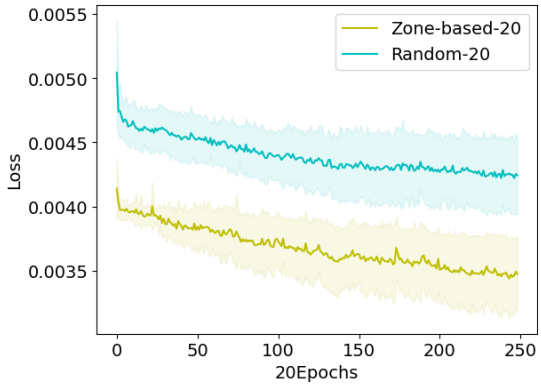

Fig. 23. Loss-epoch graph for two models. On the graph the yellow and cyan lines are the average loss function value calculated on all the test data graphs at 20% known edges with zone-based and random sensor placements.The yellow and cyan areas are 2 deviation wide

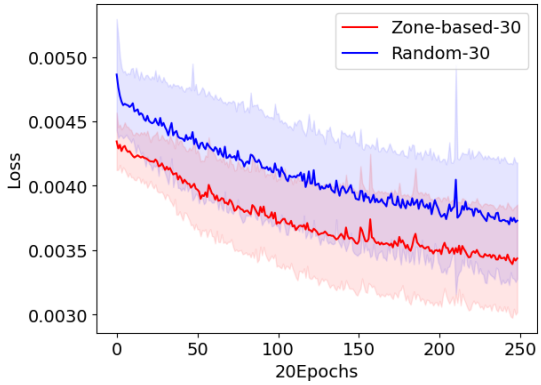

Fig. 24. Loss-epoch graph for two models. On the graph the red and blue liness are the average loss function value calculated on all the test data graphs at 30% known edges with zone-based and random sensor placements.The red and blue areas are 2 deviation wide

Notably, on Figures25 we can see that the 10% zone-based model performed better than the 20% random model, showing that smart sensor placement can be more effective than simply using more sensors. Even with fewer sensors, the 10% zone-based setup provided stronger results, proving that a well-planned layout can make data collection both accurate and cost-effective. Similarly the 20% zone based model outperformed the 30% random model.

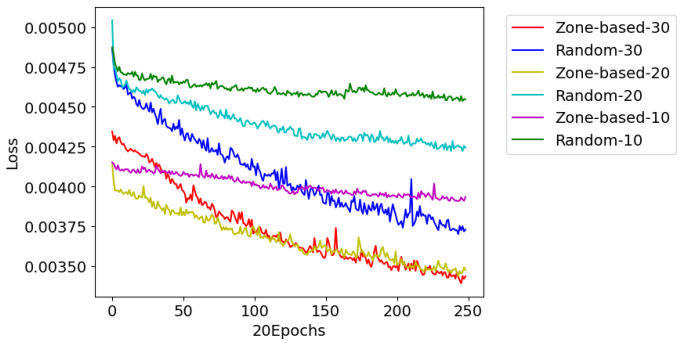

Fig. 25. Loss-epoch graph for two models. On the graph the magenta dots are the average loss function value calculated on all the test data graphs at 10% known edges with zone-based sensor placements and the blue dots are the average loss function value calculated on all the test data graphs at 30% known edges with random sensor placements.

We also wanted to examine what are the spatial nature of the model's output's accuracy. In order to do this, we coloured the graph of Győr based on the absolute value of the differences between the model's output and the validation data. The graphs are on Figures 26-31. We can see that the main roads of the city have larger differences, whereas the model managed to learn the smaller streets fairly well. The 10% random model performs the worst, while the 30% zone-based model performs the best. Errors increase consistently across each road segment as sensor coverage decreases.

Fig. 26. Coloured graph of Győr based on the the absolute value of the differences between the model's output and the validation data with 10% random sensor placements

Fig. 27. Coloured graph of Győr based on the the absolute value of the differences between the model's output and the validation data with 20% random sensor placements

Fig. 28. Coloured graph of Győr based on the the absolute value of the differences between the model's output and the validation data with 30% random sensor placements

Fig. 29. Coloured graph of Győr based on the the absolute value of the differences between the model's output and the validation data with 10% zone-based sensor placements.

Fig. 30. Coloured graph of Győr based on the the absolute value of the differences between the model's output and the validation data with 20% zone-based sensor placements.

Fig. 31. Coloured graph of Győr based on the the absolute value of the differences between the model's output and the validation data with 30% zone-based sensor placements.

We also used another metric to take a look at the spatial accuracy of the model's output. This is GEH statistics which is used in traffic modelling, traffic forecasting and traffic engineering to compare two sets of traffic volumes. The formula of the metric :

$$\text{GEH} = \sqrt{\frac{2(M-C)^2}{M+C}} \tag{2}$$

Where in our equation M is the quarter hourly traffic volume from the model and C is the volume from the simulation. In this statistics the roads with higher traffic volume will be more sensitive the error term will be scaled with the "importance" of the road. The values in general should be somewhere between 3 and 10, ideally under 5.

These weighted errors are shown on graphs on Figure 32.-37. These graphs strengthens the results from the previous figures, that the model managed to capture the smaller roads' traffic volume fairly well, but lacks on the larger ones. It is quite obvious, that for the main roads the GEH value does not get under 5, in some cases it is also larger than 10, however for the side roads usually does get better than that. [3]

Fig. 32. Coloured graph of Győr based on the the GEH values calculated from the model's output and the validation data with 10% random sensor placements

Fig. 33. Coloured graph of Győr based on the the GEH values calculated from the model's output and the validation data with 20% random sensor placements

Fig. 34. Coloured graph of Győr based on the the GEH values calculated from the model's output and the validation data with 30% random sensor placements

Fig. 35. Coloured graph of Győr based on the the GEH values calculated from the model's output and the validation data with 10% zone-based sensor placements

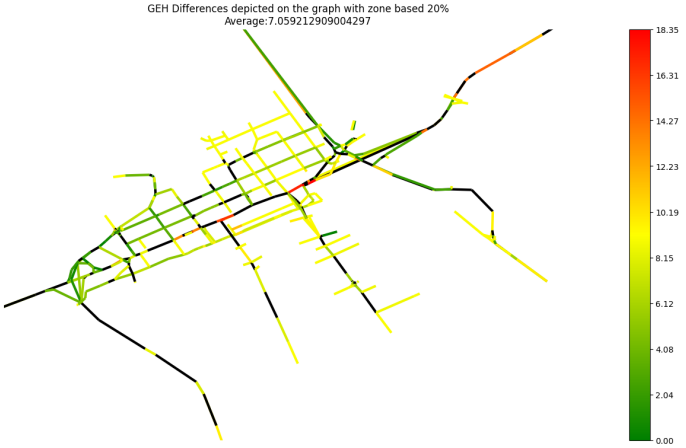

Fig. 36. Coloured graph of Győr based on the the GEH values calculated from the model's output and the validation data with 20% zone-based sensor placements

Fig. 37. Coloured graph of Győr based on the the GEH values calculated from the model's output and the validation data with 30% zone-based sensor placements

In table I. we summarize our results. The best performance belongs to the random 20 % coverage. This is due to the fact, that there is a probability that the random algorithm finds a very good layout for the sensors, however on average it does not perform well. This is supported by figure 25. In the other cases the zone-based algorithm performs better than their random counterparts.

Algorithm	Coverage	Average difference	Average GEH
random	30%	51,5	7,3
zone-based	30%	42,6	6,7
random	20%	42,5	6,5
zone-based	20 %	44,1	7,0
random	10%	53,6	7,5
zone-based	10%	46,1	7,0

TABLE I
SUMMARY OF OUR RESULTS FOR THE 2 ALGORITHMS WITH THEIR GRAPH.

VI. Conclusion

In our thesis we proved that GNNs can be used for solving traffic estimation problems in cities. We also presented a new approach for solving the sensor location problem and proved that -altough choosing the locations randomly can perform well in some cases- in general, our algorithm performs better than the random choice. In the future we are planning to extend our current work to traffic forecasting and also work on more complex model structures.

References

[1] Milad Besharatifard and Fatemeh Vafaee. A review on graph neural networks for predicting synergistic drug combinations. *Artificial Intelligence Review*, 57, 02 2024.

[2] Dami Choi, Christopher J. Shallue, Zachary Nado, Jaehoon Lee, Chris J. Maddison, and George E. Dahl. On empirical comparisons of optimizers for deep learning, 2020.

[3] Olga Feldman. The geh measure and quality of the highway assignment models. 10 2012.

[4] Thomas N. Kipf and Max Welling. Semi-Supervised Classification with Graph Convolutional Networks. *arXiv e-prints*, page arXiv:1609.02907, September 2016.

[5] Daniel Krajzewicz. *Traffic Simulation with SUMO – Simulation of Urban Mobility*, pages 269–293. Springer New York, New York, NY, 2010.

[6] Xiao Li, Li Sun, Mengjie Ling, and Yan Peng. A survey of graph neural network based recommendation in social networks. *Neurocomputing*, 549:126441, 2023.

[7] Highway Capacity Manual. Highway capacity manual. *Washington, DC*, 2(1):1, 2000.

[8] Mahmoud Owais. Traffic sensor location problem: Three decades of research. *Expert Systems with Applications*, 208:118134, 2022.

[9] Amit Roy, Kashob Kumar Roy, Amin Ahsan Ali, M Ashraful Amin, and A K M Mahbubur Rahman. Sst-gnn: Simplified spatio-temporal traffic forecasting model using graph neural network, 2021.

[10] Mostafa Salari, Lina Kattan, William H.K. Lam, Mohammad Ansari Esfeh, and Hao Fu. Modeling the effect of sensor failure on the location of counting sensors for origin-destination (od) estimation. *Transportation Research Part C: Emerging Technologies*, 132:103367, 2021.

[11] Alvaro Sanchez-Gonzalez, Jonathan Godwin, Tobias Pfaff, Rex Ying, Jure Leskovec, and Peter Battaglia. Learning to simulate complex physics with graph networks. In *International conference on machine learning*, pages 8459–8468. PMLR, 2020.

[12] Minhua Shao, Congcong Xie, and Lijun Sun. Optimization of network sensor location for full link flow observability considering sensor measurement error. *Transportation Research Part C: Emerging Technologies*, 133:103460, 2021.

[13] Hai Yang, Chao Yang, and Liping Gan. Models and algorithms for the screen line-based traffic-counting location problems. *Computers Operations Research*, 33(3):836–858, 2006.