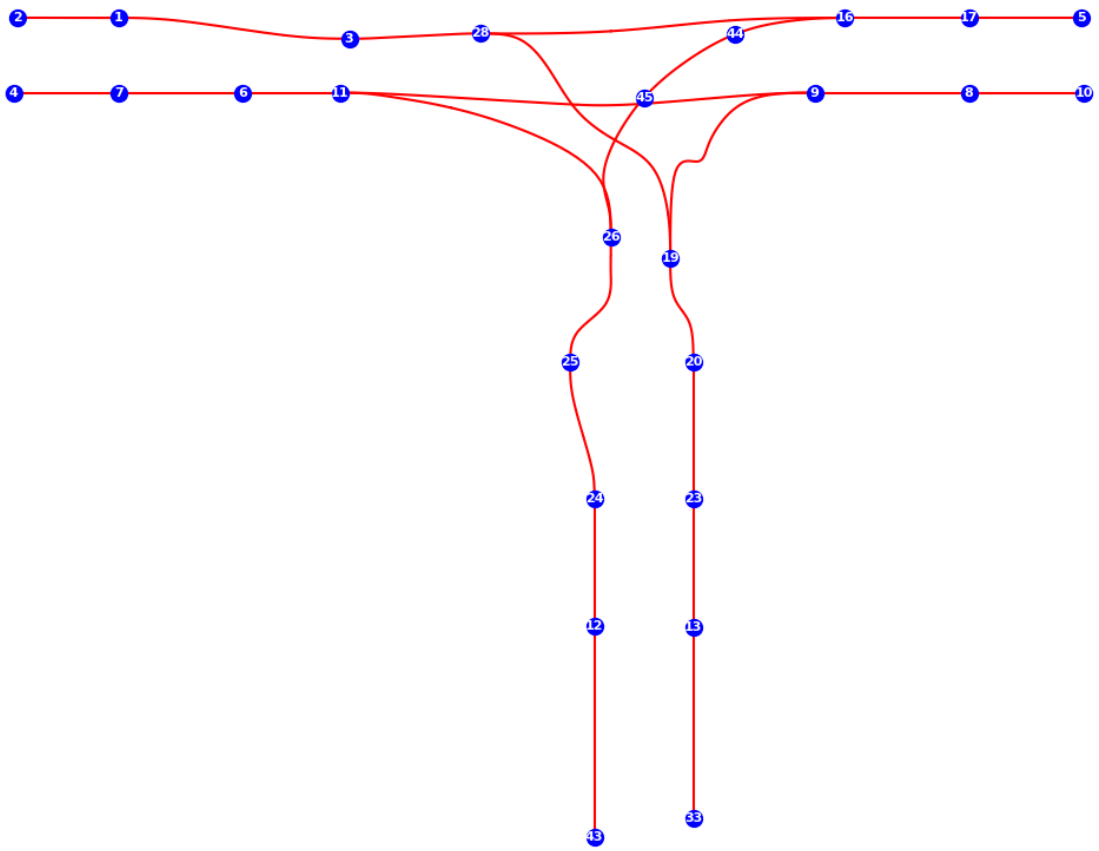
CHALMERS
UNIVERSITY OF TECHNOLOGY



Supervised Learning to Evaluate Road Networks for Automated Guided Vehicles

Using graph neural networks to estimate the efficiency of different road network designs

Master's thesis in Complex Adaptive Systems

JONAS LAURI, MATTIAS WIBERG

DEPARTMENT OF ELECTRICAL ENGINEERING

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2023
www.chalmers.se

Master's thesis 2023

Supervised Learning to Evaluate Road Networks for Automated Guided Vehicles

Using graph neural networks to estimate the efficiency of different road network designs

Jonas Lauri, Mattias Wiberg

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Chalmers University of Technology
Gothenburg, Sweden 2023

Supervised Learning to Evaluate Road Networks for Automated Guided Vehicles
Using graph neural networks to estimate the efficiency of different road network designs
Jonas Lauri, Mattias Wiberg

Supervisor: Dante Landa Vega, Kollmorgen Automation AB
Examiner: Erik Agrell, Electrical Engineering

Master's Thesis 2023
Department of Electrical Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: One of the 50 road network designs from the dataset used.

Typeset in LaTeX
Gothenburg, Sweden 2023

Abstract

To evaluate if a layout for automated guided vehicles is good is a hard task. A simple solution is to simply simulate the proposed layout, but this takes a large amount of time. Using neural networks could be much faster, if they are able to generalize to the task of evaluating layouts. This thesis introduces and examines some different network structures and proposes a way of processing the data. These networks were then trained to be able to generalize a T-intersection. The resulting best network was a novel network, which combines core features of the investigated models in combination with our proposed preprocessing and embedding of the data. However, even this network gave an unsatisfactory result. Different reasons as to why the networks are not able to generalize are then discussed, like the small size of the data set and uncertainty of the simulation software results. If the data is unreliable or lacks any discernible patterns, then the network will not be able to learn, no matter how it is constructed.

Keywords: Graph Attention Network, Graph Neural Network, Graph Convolution Network, Relational Fusion Network, Many2one, Regression, Road Network

Acknowledgements

We would like to thank Dante for his help and friendliness during our time at Kollmorgen as well as Kollmorgen for providing a good office space to work at.

Mattias Wiberg & Jonas Lauri, Gothenburg, June 2023

List of Acronyms

Below is the list of acronyms that have been used throughout this thesis listed in alphabetical order:

AGV	Automated Guided Vehicle
CNN	Convolutional Neural Network
DFS	Depth-First Search
GAT	Graph Attention Network
GCN	Graph Convolutional Network
GF	Global Features
GNN	Graph Neural Network
FF	Feed Forward
LAT	Layout Analyzer Tool
MLP	Multilayer Perceptron (Network)
MSE	Mean Square Error
MTRF	Mean Total Relative Frequency
ORF	Order Relative Frequency
RFN	Relational Fusion Network
RGB	Red Green Blue
TOF	Total Order Frequency
LAT	Layout analyser tool

Contents

List of Figures

List of Tables

1

Introduction

An *automated guided vehicle* (AGV) system is a fleet of mobile robots that automatically transports goods in a layout of fixed virtual roads, designed according to the specification of the warehouse. This layout must have the capacity to handle the intended traffic load without traffic jams, which is verified by running simulations. However, such simulations are time consuming, and since the design process is an iterative one, faster feedback directly translates to decreased costs. In this master thesis project, we will research how machine learning can be used to speed up the design process by predicting the consequences of different design choices.

1.1 Data structure

There are many situations where one wants to use structured data to draw conclusions. One of the most common cases is the classification of images. A common way to use the structure of the data is using *convolutional neural networks* (CNNs). CNNs use the neighbours of each pixel when processing the image. This is since the data is highly ordered, as the number of neighbours is the same for all pixels (not at the edges but there one can use padding). The layers of the CNN could for example be set up to find ordered big changes in pixel values in RGB values which corresponds to a line or color change.

However, our project uses graphs as input data and not images as commonly found in CNNs. Graphs contain a number of nodes and edges connecting the nodes together. Each node and edge can have features such as, node size and edge length. The number of nodes and edges can vary for each graph making the framework of CNNs inapplicable. Therefore one instead uses *graph convolutional networks* (GCNs) which in turn is a type of *graph neural network* (GNN). GCNs use the same ideas as in CNNs, but are suited for the irregular structure of graphs. The fact that the number of nodes and edges are often different for each graph is a problem as the input dimensions to a neural network must be constant. To solve this, one aggregate features from each node and uses those instead. Features can be explained as the contents of the nodes. What the features are depends on the graphs in question but they should always be the same for all the graphs in the data set.

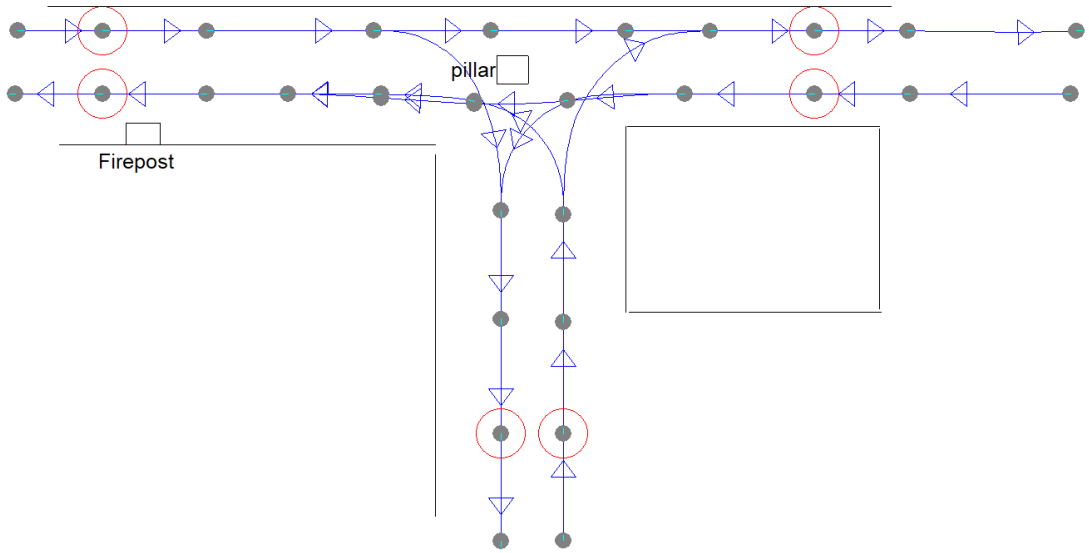


Figure 1.1: Visualization of example layout three. It is quite good but it has a considerable risk for deadlocks in the middle.

1.2 Layout Designer

Two examples of the data can be seen in the figures 1.1 and 1.2. These graphs have some directed roads (edges) and some points (nodes) (the red outlined ones are stations where load is fetched or dropped off). One important thing to note is that how the road is placed spatially will matter for the results. To be able to model the spatial dependency, we will have node positions and the edge lengths as features. Additionally, we will introduce graphs which give even more consideration to the spatial dependency, which we will call *extended graphs* (see section 2.2.2).

1.3 Problem formulation

The problem that we want to tackle in this thesis is whether these *graph neural networks* (GNN) can be used to predict how good a certain layout is. To answer this question, an ensemble of different GNNs will be examined. After this, the processing of the data to get reasonable inputs and outputs. This output should be a representative of how good the layout is. The output will be the so called *throughput* of the layout.

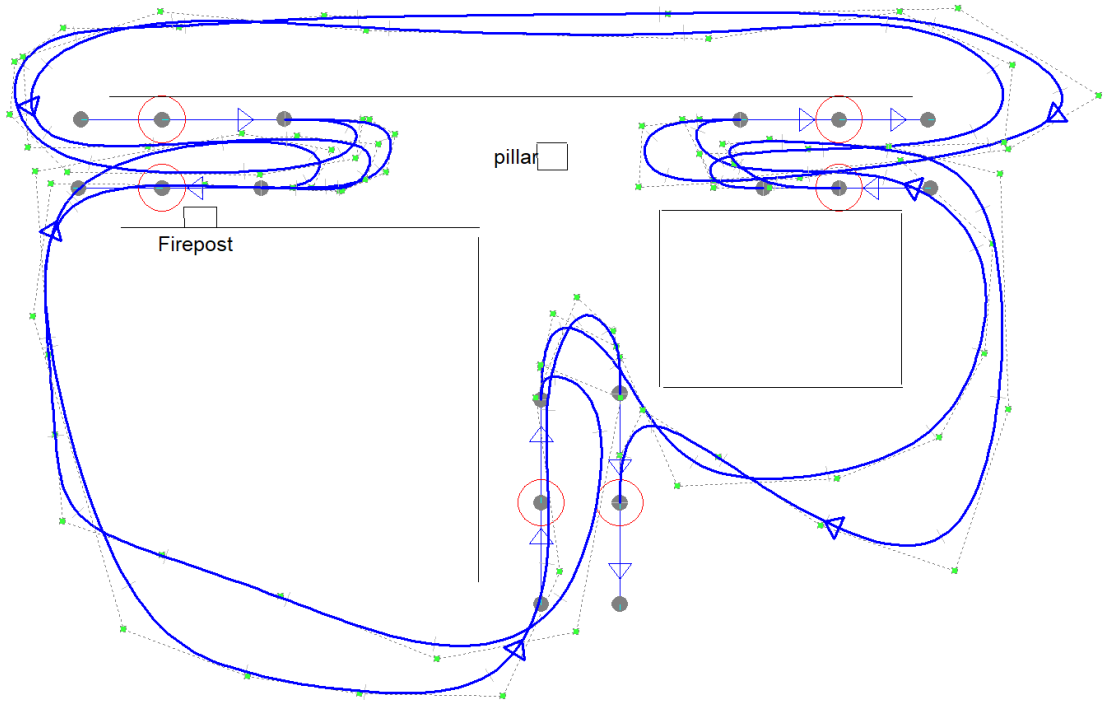


Figure 1.2: Visualization of example layout 40. A bad layout where we have a large risk of deadlocks and the roads are long.

2

Theory

This chapter describes the theory used in this report.

2.1 B-splines

A way to model the roads which the AGV are able to drive on is to use B-splines [1]. B-splines have some advantageous features, like only being dependent on control points a short distance away. In other words, they are only defined by local features, which is good for performance.

A B-spline curve, $\mathbf{p}(u)$, is a vector defined as

$$\mathbf{p}(u) = \sum_{i=0}^{n} \mathbf{p}_i N_{i,k}(u) \tag{2.1}$$

where $\mathbf{p}_i$ are the $n+1$ control points.

The blending functions, $N_{i,k}(u)$ are defined recursively using the following

$$N_{i,1}(u) = \begin{cases} 1 & \text{if } u \in [t_i, t_{i+1}), \\ 0 & \text{else.} \end{cases} \tag{2.2}$$

And, when $k > 1$

$$N_{i,k}(u) = \frac{u - t_i}{t_{i+k-1} - t_i} N_{i,k-1}(u) + \frac{t_{i+k} - u}{t_{i+k} - t_{i+1}} N_{i+1,k-1}(u) \tag{2.3}$$

where $\{t_0, t_1, \ldots, t_{n+k}\}$ are called the *knot values* and $(k-1)$ is the order of the curve.

2.1.1 Uniform B-spline of degree three

To be computationally fast while still providing good accuracy and flexibility one can use cubic splines (degree three). For a uniform B-spline function of order $k = 4$ and degree three, the B-spline is

$$\mathbf{p}(u) = N_{0,4}(u)\mathbf{p}_{i-1} + N_{1,4}(u)\mathbf{p}_i + N_{2,4}(u)\mathbf{p}_{i+1} + N_{3,4}(u)\mathbf{p}_{i+2} \tag{2.4}$$

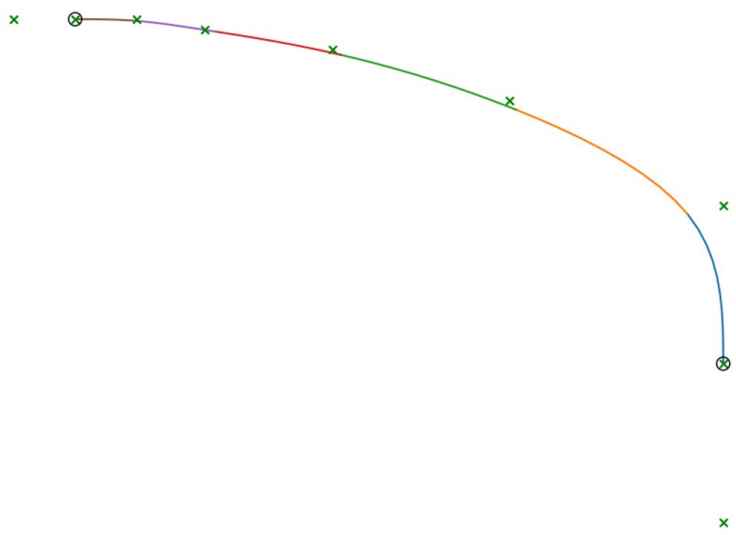


Figure 2.1: An example of a road segment consisting of multiple B-splines. The circles are the start and end nodes, the crosses are the control points $\mathbf{p}_i$, and the different colors represents the different spline parts.

where i is the part number as defined by the four control points. In general, $k = n+1$ control points are enough to uniquely define a spline of order n.

In section A.2 the final matrix for the basis functions is derived. The interested reader is therefore refered to section A.2.

$$\mathbf{p}(u) = \begin{bmatrix} \vdots & \vdots & \vdots & \vdots \\ N_{0,4} & N_{1,4} & N_{2,4} & N_{3,4} \\ \vdots & \vdots & \vdots & \vdots \end{bmatrix} \begin{bmatrix} \mathbf{p}_{i-1} \\ \mathbf{p}_i \\ \mathbf{p}_{i+1} \\ \mathbf{p}_{i+2} \end{bmatrix} \tag{2.5}$$

$$= [u^3\, u^2\, u\, 1] \cdot \frac{1}{6} \cdot \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{p}_{i-1} \\ \mathbf{p}_i \\ \mathbf{p}_{i+1} \\ \mathbf{p}_{i+2} \end{bmatrix} \tag{2.6}$$

We see that with four control points then the spline $\mathbf{p}(u)$ is uniquely defined. An example of a road segment and its composite splines can be seen in figure 2.1.

2.2 Graph representation

A directed graph [2], $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, can be defined as a tuple of a set of nodes $\mathcal{V}$ and a set of edges $\mathcal{E}$. An edge represents a connection between two vertices and is defined as a tuple from one node to another $(v_i, v_j) \in \mathcal{E}$. For an example see the graph in figure 2.2, the nodes in this case would be $\mathcal{V}$ = {A, B, C, D} and the edges $\mathcal{E}$ = {(A, B), (A, C), (B, C), (C, A), (C, D)}. The edges can also have attributes, the most common are weights.

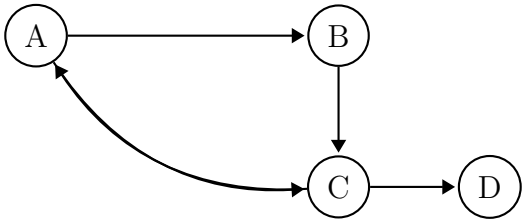


Figure 2.2: An example of a graph with four nodes and five edges.

Although representing the edges of a graph as a list is efficient when it comes to memory, another way that is more intuitive and simpler to implement is expressing the edges in an adjacency matrix A, $A \in \mathbb{B}^{|\mathcal{V}|\times|\mathcal{V}|}$, whose binary elements indicate whether there is a connection between different nodes. It is defined as

$$A_{ij} = \begin{cases} 1, & \text{if } (v_i, v_j) \in \mathcal{E}, \\ 0, & \text{otherwise.} \end{cases} \tag{2.7}$$

As an example, figure 2.2 would have an A equal to

$$A = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}. \tag{2.8}$$

2.2.1 Dual graphs

There is also a dual graph representation of $\mathcal{G}^P$ defined in [3], which is different from the usual definition of a dual graph. It is defined as $\mathcal{G}^D = (\mathcal{E}, \mathcal{B})$, where $\mathcal{E}$ are the edges and $\mathcal{B}$ are the *between-edge* connections. The *between-edge* connections are in turn defined as $\mathcal{B} = \{((u, v), (v, w)) \,|(u, v), (v, w) \in \mathcal{E}\}$. In words, the *between-edge* connections are the connections of the edges. In figure 2.3 the dual graph of figure 2.2 can be seen. The reasoning behind the introduction of dual graphs is that the focus of the dual graph is the edges, which could be more important than the nodes.

2.2.2 Extended graphs

To incorporate the curvature of the spline (also called segment see section 3.1.3) one needs to include the control points that define it. The amount of control points are dependent on the amount of parts the spline consists of, which is then in turn defined by the user when creating the spline. Due to the edge attribute dimensions having to be constant, it is not possible to incorporate these control points into edge attributes for the networks. Therefore an extended graph is proposed, where each part of the spline is converted to a new edge defined by four control points and an extra node see figure 2.4. Note that in figure 2.4 the edge connecting the two nodes in the top image has been replaced in the bottom image with three new edges although using the same control points and two new nodes have been added

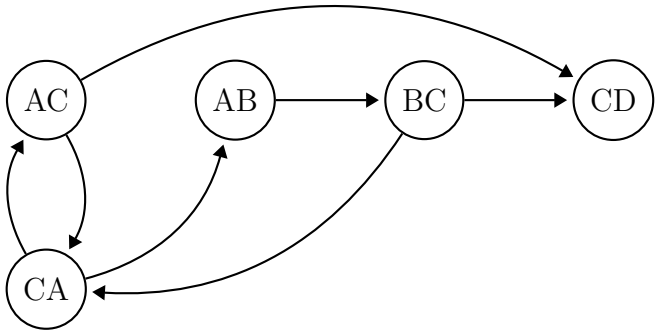


Figure 2.3: The dual graph representation of figure 2.2. The edges are the *between-edge* connections (e.g. from A to B and B to C there are edges, therefore we have an edge between AB and BC).

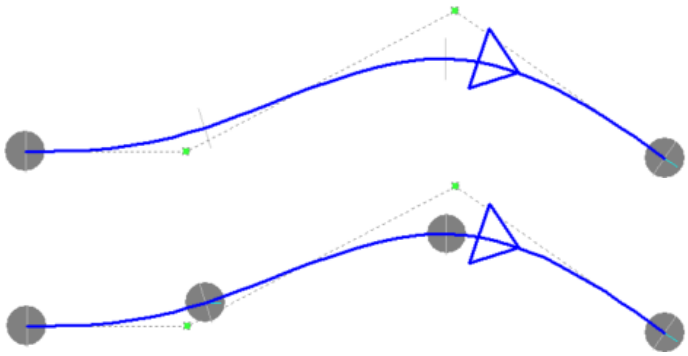


Figure 2.4: In the top image is an example of a spline before it has been extended, and in the image below the extended version of the same spline. Where the parts of the spline start and end are shown with dashed gray lines and the green squares are the control points.

between the three parts. This kind of extension is especially important when the edge has a lot of turns and therefore a straight line would be a poor representation.

2.3 Graph neural networks

Graph neural networks have become more and more popular in recent years. Some example applications are in recommendation systems, social networks and bioinformatics. In [4] these applications (and more) of GNNs are discussed. Since our road networks can be described in the form of a graph, these neural network structures will be the ones we use.

A GNN takes node and edge features as input and attempts to predict values or classes. The classic example is the Cora, Citeseer and Pubmed data sets introduced in [5]. The nodes of these data sets are articles. The words of each article are the features of the node (article). The edges are the articles which reference each other. To input into a GNN, take the features from each node using its edges and aggregate

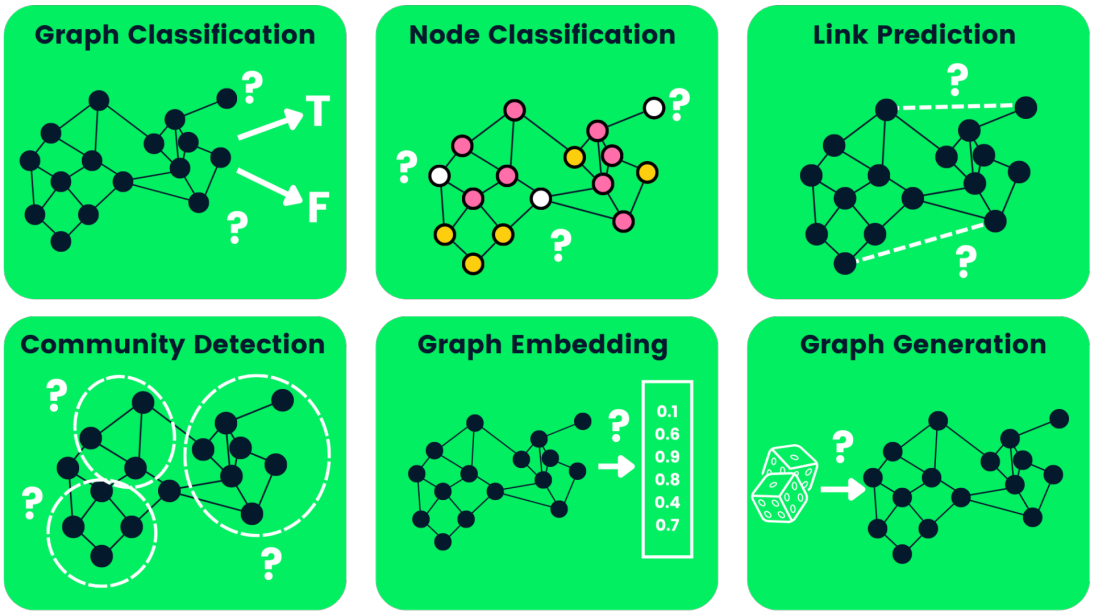


Figure 2.5: Examples of different types of GNN tasks. In our case we have a graph embedding task where we only predict the total performance of the graph as a single number. Made by [6].

this information in some way. One simple way is to average the feature value over the total number of nodes. Given this input, the objective is to predict the class each paper belongs to, given references and the occurrences of each word within each of the papers. This is a type of node classification task.

What the features, edges and nodes are depends on the problem. As stated before, the reason one introduces these features is that if we input the features of each node, we allow for freedom in the number of nodes. Compare with a picture for example, each pixel must be connected to eight others and the dimensions must be the same for each picture in the data set. The nodes in a graph do not have these restrictions. This is the reason to apply graph neural networks, to increase the freedom of the input dimensions.

The output to the above example is equal to the number of nodes being predicted. However, in our case, we have a many to one task. This means that we will only predict one single value, given a graph. Specifically, we will predict the throughput, which will be defined as a continuous number between $[0, 1]$. In figure 2.5 [6], we have examples of some different types of GNN tasks. Our problem is of the graph embedding type.

Finally, feature aggregation, introduced in [7], will be used in the networks. As shown in [7], many graphs cannot be differentiated by just inputting the mean of the features, for example. An example would be two graphs with two features each, $[1, 3]$ and $[2, 2]$. The mean would be the same (two) but the max (one and three) would be different. Therefore, one should use different kinds of aggregators to be

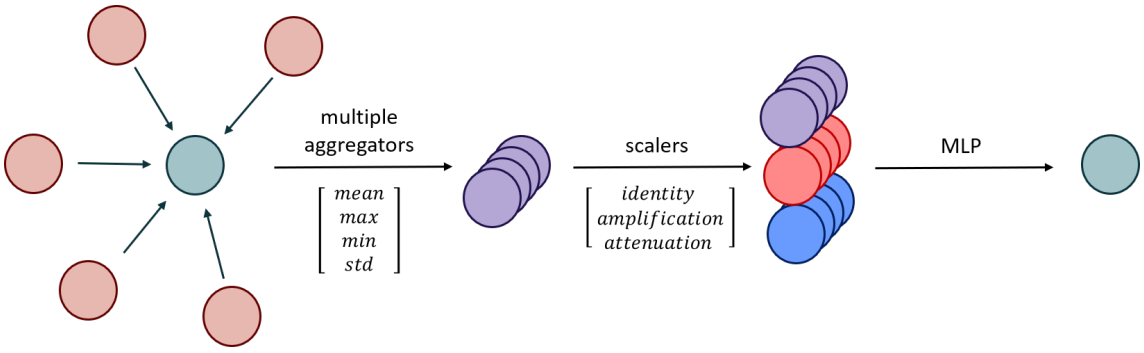


Figure 2.6: Introduction of aggregators. In this thesis, no scalers (only identity) will be used. Above figure is made by [7].

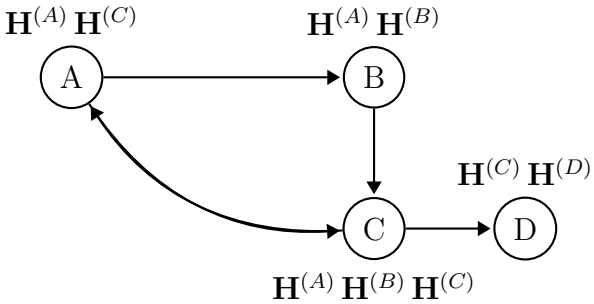


Figure 2.7: A schematic of how a GCN works. The features are labeled with $\mathbf{H}$ and we can see the spread of each feature next to each node. Here $\mathbf{H}^{(A)}$ is used to label the features of the A node, while $\mathbf{H}^{(l)}$ in 2.9 refers to the all features used as input to a GCN-layer.

able to discriminate between different networks.

Finally, graphs can have *global features* (GF). This concept was used in the *General Powerful Scalable Graph Transformers* (GPS) paper by [8]. An example of a global feature could be the total number of nodes in a graph.

2.3.1 Graph convolutional networks

The analogue to CCNs, but for graphs are the *graph convolutional networks*, or GCNs [9]. These layers aggregate the information from the neighbours of each node in a similar way to ordinary convolutional networks. A schematic of the information flow in a GCN can be seen in figure 2.7. However, as the structure is irregular we instead have to rely on the adjacency matrix to get the neighbours. We also get a varying amount of information for each node so we will need to normalize to confine the gradients to the range $[-1, 1]$. These gradients are then used in the gradient decent. (Gradient decent is the way in which neural networks train their weights.) These aspects give rise to the following definitions.

The GCN uses the following propagation rule (which is standard in libraries such

as PyTorch Geometric) [9]:

$$\mathbf{H}^{(l+1)} = \sigma\left(\overbrace{\tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}}^{N\times N}\overbrace{\mathbf{H}^{(l)}}^{N\times F}\overbrace{\mathbf{W}^{(l)}}^{F\times H}\right) \tag{2.9}$$

Here $\tilde{A} = A + I$, where A is the adjacency matrix and I an appropriate identity matrix. Then we have $\tilde{D}$ which is the degree matrix, which is defined as, $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$. In words, the elements in $\tilde{D}$ are the number of connections of each node (given that each self connection is only worth one). $\mathbf{W}^{(l)}$ is a layer-specific trainable weight matrix and σ is an activation function. Finally, $\mathbf{H}^{(l)}$ is the matrix of activations (or features) in the l-layer which means that $\mathbf{H}^{(0)} = \mathbf{X}$, the input layer. For the first layer, the overbraces are the dimensions at each step of the matrix multiplications. N is the number of nodes, F, the number of features per node and H the length of the output. For an eventual second layer, the dimensions of $\mathbf{H}^{(l)}$ are $N \times H$, so an appropriate weight matrix has dimensions $H \times H_2$, where H_2 is the length of the output from the second layer.

In [9], using work done by [10] and [11], the propagation rule 2.9 is motivated with a first-order approximation of localized spectral filters on graphs. problem of exploding or vanishing gradients as shown in [9] . They made an approximation using the Chebyshev polynomials. Using this approximation, equation 2.9 has complexity of $O(K|\mathcal{E}|)$, which is linear in the number of edges (K is a constant).

To train the weights of the network one needs an error function to minimize. The article which introduced the GCN solves a classification problem and therefore chooses the cross-entropy error [9]. We have a regression problem as our targets will be continuous numbers and we will therefore use the L^1 or L^2 (MSE) errors.

If one were to do the approximations with Chebyshev polynomials with A, D, this would yield eigenvalues in the range $[0, 2]$ [9]. This range is inappropriate as we do not want our gradients to scale like this. Therefore, GCNs use $\tilde{A}, \tilde{D}$ to help with this problem [9].

Equation 2.9 is formulated with two degree matrices ($\tilde{D}$). The reason for this is that we want not only the current nodes degree (number of connections per node) to affect the input to the neural network, we also want the degree of the connecting node to matter. Let us take a look at an example to see the differences between $\tilde{D}^{-1}\tilde{A}$ and $\tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}$. We have

$$\tilde{A} = A + I = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.10}$$

and therefore

$$\tilde{D}_{ii} = \sum_j \tilde{A}_{ij} \rightarrow \tilde{D} = \begin{bmatrix} 3 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{2.11}$$

Then

$$\tilde{D}^{-1}\tilde{A} = \begin{bmatrix} \frac{1}{3} & 0 & 0 & 0 \\ 0 & \frac{1}{2} & 0 & 0 \\ 0 & 0 & \frac{1}{3} & 0 \\ 0 & 0 & 0 & \frac{1}{1} \end{bmatrix} \times \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \frac{1}{3} & \frac{1}{3} & \frac{1}{3} & 0 \\ 0 & \frac{1}{2} & \frac{1}{2} & 0 \\ \frac{1}{3} & 0 & \frac{1}{3} & \frac{1}{3} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.12}$$

and

$$\tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2} = \cdots = \begin{bmatrix} \frac{1}{3} & \frac{1}{\sqrt{6}} & \frac{1}{3} & 0 \\ 0 & \frac{1}{2} & \frac{1}{\sqrt{6}} & 0 \\ \frac{1}{3} & 0 & \frac{1}{3} & \frac{1}{\sqrt{3}} \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{2.13}$$

Normalization such as in Equation 2.13 makes sure each feature gets weighted based on not only the current node, but also the nodes which the edges of the node are connected to.

2.3.2 Graph attention networks

A recent development of GNNs are the introductions of attention layers, which are used in *graph attention networks*, GATs [12]. (GAN was already taken so it became Graph ATtention networks.) They are based on the idea that some nodes are more important than others and that we want a way to model this. First we need the input which is a set of node features, $\mathbf{H} = \{\mathbf{H}_1, \mathbf{H}_2, \ldots, \mathbf{H}_N\}, \mathbf{H}_i \in \mathbb{R}^F$, where again, N is the number of nodes and F is the number of features in each node. Then apply a weight matrix $\mathbf{W} \in \mathbb{R}^{F' \times F}$ on the node features. It is trained within the layers. Here, a shared attention mechanism is used $a : \mathbb{R}^{F'} \times \mathbb{R}^{F'} \to \mathbb{R}$. Using a we calculate the *attention coefficients*, e_{ij}. The a is responsible for the attention and weights the connections between nodes. This a is trained using an MLP.

$$e_{ij} = a(\mathbf{W}\,\mathbf{H}_i, \mathbf{W}\,\mathbf{H}_j) \tag{2.14}$$

Some kind of normalization is then needed (to compare between nodes). The GAT paper [12] uses softmax to calculate the *normalized attention coefficients*, α_{ij}.

$$\alpha_{ij} = \text{softmax}_j(e_{ij}) = \frac{\exp(e_{ij})}{\sum_{k \in \mathcal{N}_i} \exp(e_{ik})} \tag{2.15}$$

Using this definition we get the final expression for the normalized attention coefficients.

$$\alpha_{ij} = \frac{\exp\left(\text{LeakyReLU}\left(\mathbf{a}^T \left[\mathbf{W}\,\mathbf{H}_i \| \mathbf{W}\,\mathbf{H}_j\right]\right)\right)}{\sum_{k \in \mathcal{N}_i} \exp\left(\text{LeakyReLU}\left(\mathbf{a}^T \left[\mathbf{W}\,\mathbf{H}_i \| \mathbf{W}\,\mathbf{H}_k\right]\right)\right)} \tag{2.16}$$

where $\mathbf{a}$ is the weight vector from the attention mechanism a, $\|$ stands for concatenation and $\mathcal{N}_i$ is the neighbourhood of node i. Finally, concatenate the features, resulting in the output features $\mathbf{H}_j^{(l+1)}$.

$$\mathbf{H}_j^{(l+1)} = \sigma\left(\sum_{j \in \mathcal{N}_i} \alpha_{ij}\,\mathbf{W}\,\mathbf{H}_j^{(l)}\right) \tag{2.17}$$

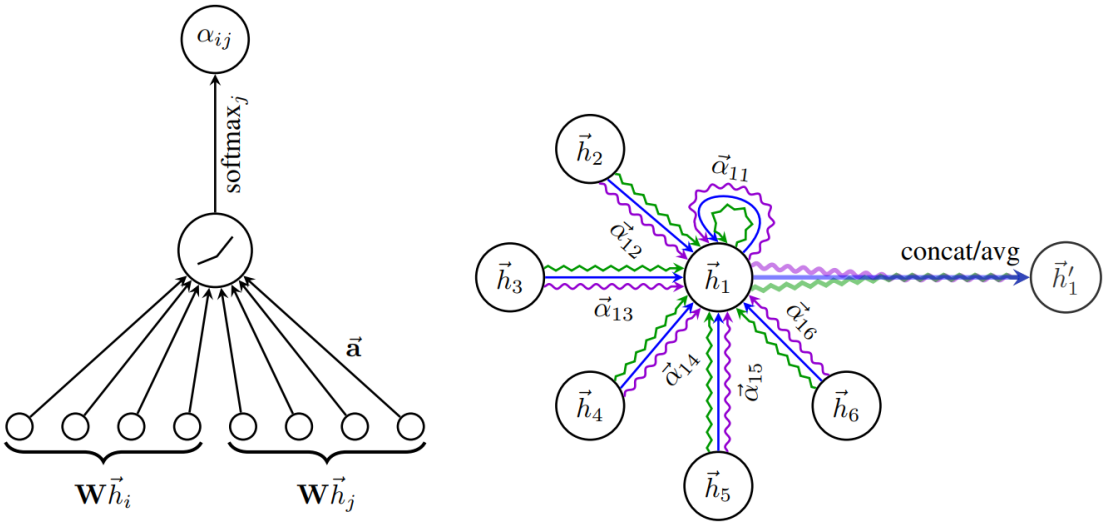


Figure 2.8: To the left we see the attention mechanism **a** and a LeakyReLU activation function to get the normalized attention coefficient. To the right we have an example of multi-head attention with three heads. The figure is taken from the GAT paper [12].

The GAT architecture is inherently less stable because of the extra attention vector, **a**. It is therefore often good to use so called *multi-head attention*, where one repeats the above steps multiple times and afterwards concatenate the results. If multi-head attention is used then we instead have the following equation

$$\mathbf{H}_j^{(l+1)} = \Big\Vert_{k=1}^{K} \sigma\left(\sum_{j\in\mathcal{N}_i} \alpha_{ij}^k \, \mathbf{W}^k \, \mathbf{H}_j^{(l)}\right) \tag{2.18}$$

where K is the number of heads used. In figure 2.8 we see an example GAT network with three heads [12]. They use a different notation for the features: $\vec{h}$ instead of **H**.

2.3.2.1 GATv2

A better version of GAT has been found [13] which changes the order of the calculations. In [13] it was proven that it potentially performs better with no downside and they called their version GATv2. It is defined as below.

$$e_{ij} = \mathbf{a}^T \,\text{LeakyReLU}\left(\mathbf{W} \cdot [\mathbf{H}_i \| \mathbf{H}_j]\right) \tag{2.19}$$

GATv2 calculates dynamic attention and for more details please read [13]. As GATv2 is strictly better than GAT, when we say GAT in later parts of the report we are referring to GATv2.

2.3.3 Relational fusion networks

One problem with GCNs and GATs is that they are not optimized for road networks. Road networks often have smaller node degrees and instead rely on infor-

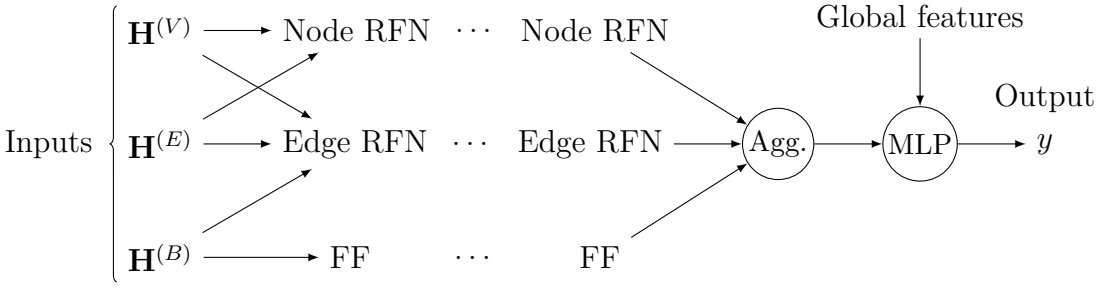


Figure 2.9: A schematic of the RFN network. The information propagates in the same way each layer. After the RFN layers there is an aggregation layer whose output will be fed into an MLP together with the global features of the graph.

mation stored within the edges. To make sure this information is used correctly to obtain better performance we should therefore use *relational fusion networks*, RFNs, according to [3].

An RFN layer consists of three parts, a node relational fusion, an edge relational fusion and a *feed-forward* layer (FF). It uses the node features, $\mathbf{H}^{(V)}$, the edge features, $\mathbf{H}^{(E)}$ and the between-edge features, $\mathbf{H}^{(B)}$. The node fusion takes $\mathbf{H}^{(V)}$ and $\mathbf{H}^{(E)}$ as input. The edge fusion has $\mathbf{H}^{(V)}$, $\mathbf{H}^{(E)}$ and $\mathbf{H}^{(B)}$ as input. Finally, there is a feed-forward network which inputs the between-edge features, $\mathbf{H}^{(B)}$. In figure 2.9 we see the RFN structure and how information propagates through the network as a whole.

In our implementation, only the extended primary graph and extended dual graphs are used as inputs, as all information from the non-extended graphs are embedded in the extended ones. The output of the final RFN layer will be fed into aggregation functions which will then be the input to a *multi layer perceptron* (MLP) layer. The global features of the graphs are added to the MLP as well.

2.3.3.1 Relational fusion

One part of the RFN which is still unclear, is how the relational fusion should be performed. In the RFN article by [3], two examples of possible fusion mechanisms are given. The most simple one is called *AdditiveFuse* and inputs the concatenation of start node features, end node features and their edge features, to what we will call $\mathbf{H}^{(R)}_{(v,n)}$. Then we have weights, bias and an activation function

$$AdditiveFuse = \sigma\left(\mathbf{H}^{(R)}_{(v,n)}\mathbf{W}^R + \mathbf{b}\right) \tag{2.20}$$

where $\mathbf{H}^{(R)}_{(v,n)} \in \mathbb{R}^{1\times d_R}$, $\mathbf{W}^R \in \mathbb{R}^{d_R\times d_0}$ and $\mathbf{b} \in \mathbb{R}^{1\times d_0}$. This means that equation 2.20 has output dimensions d_0. The size of the hidden features, d_0, is free to choose. When concatenating features into the edge RFN, each between-edge has two edges which in turn has two nodes each. This is why the edge RFN takes $\mathbf{H}^{(V)}, \mathbf{H}^{(E)}$ and $\mathbf{H}^{(B)}$ as input while the node RFN only takes $\mathbf{H}^{(V)}$ and $\mathbf{H}^{(E)}$. The node RFN has no between-edges.

The other example of fusion is what they call *InteractionalFuse* and instead of just concatenating all features, it models the interactions by using an extra weight matrix. Using the same input, the concatenation of all features, $\mathbf{H}_{(v,n)}^{(R)}$ we have

$$InteractionalFuse = \sigma\left(\left(\mathbf{H}_{(v,n)}^{(R)}\mathbf{W}^{I} \odot \mathbf{H}_{(v,n)}^{(R)}\right)\mathbf{W}^{R}\right) + \mathbf{b} \tag{2.21}$$

where $\mathbf{W}^{I} \in \mathbb{R}^{d_R \times d_R}$ is a weight matrix modelling the interactions and $\odot$ stands for element-wise multiplication. The introduction of $\mathbf{W}^{I}$ gives us better possibilities to model the interactions (hence the name) but the problem with *InteractionalFuse* is the increase from a linear number of weights to a quadratic number which could drastically increase the training time.

The input to each new layer is the output of the previous layer. Therefore, the output dimension does not have to be the same as the input dimension.

As in the case of the GCN, each node only gets information about its neighbours. Therefore, an RFN-network with L layers is able to get information about its neighbours up to L edges away.

2.3.4 Novel network

We present a novel network which was designed to combine the benefits from the three different papers, the dual graph framework [3], global and local features [8] and aggregators [7].

Earlier research [8], found that being able to expand the data given from the graph using local and global features proved to be relevant for the learning of the model. The local features being the extended graph as well as the order flow embedding and the global features being the number of nodes and edges for each graph as the ratio between them.

Research [7] also shows that having multiple different aggregators instead of one can also help to distinguish between different graph structures. This was incorporated in parallel with the different graph representations to allow for the first MLP layer to interpret what aggregations proved to be of greater importance. This also leads to more information from the graph to be passed further forward into the network.

With this in mind, a new network was created, the structure of which can be seen in figure 2.10. We call this network *Random Network Generator* (RNG).

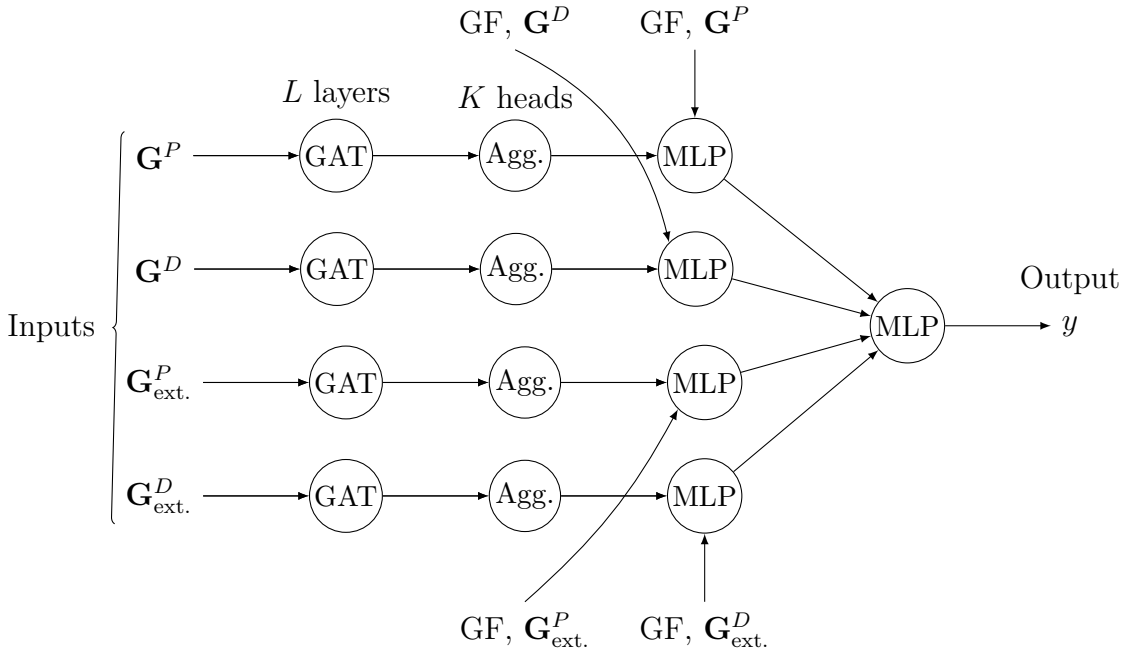


Figure 2.10: The novel network proposed taking inspiration from RFN, GPS and PNA using the aggregators mean, min, max and sum. It also utilizes of all the previously proposed graph representations of a layout in the form of a primal and extended version and their dual graphs respectively.

3

Methods

In section 3.1 the core concepts which are needed in order to understand how the data was augmented and embedded in a graph representation are explained. Section 3.2 contains the creation of the data sets which are used to train the models.

3.1 Introduction

This section introduces the core concepts of the project to fully understand the scope and applications of the work.

3.1.1 Automated guided vehicle

An AGV is a vehicle that can follow a given path from one point to another. In our case the AGVs are vehicles in warehouses or in manufacturing transporting supplies or raw materials for the machines. An AGV cannot deviate from its path nor reverse and never stops on an edge except in the case of an obstacle.

3.1.2 Layout designer

Layout designer is Kollmorgen's software created for partners to create and design layouts for specific use cases.

3.1.3 Layout

A layout defines the paths an AGV can traverse and is represented as an extended version of a graph (see section 2.2) with nodes and edges. In a layout, some nodes are defined as *stations*. These stations are either a source or a target node for an order. The edges are represented as splines (see section 2.1) and are also called segments. An example of a layout can be seen in figure 1.1 where the station nodes are circled in red.

3.1.4 Layout analyzer tool

The *Layout analyzer tool* (LAT) is a tool that uses the layout and the order flow to calculate certain metrics. This tool is developed by Kollmorgen and how these

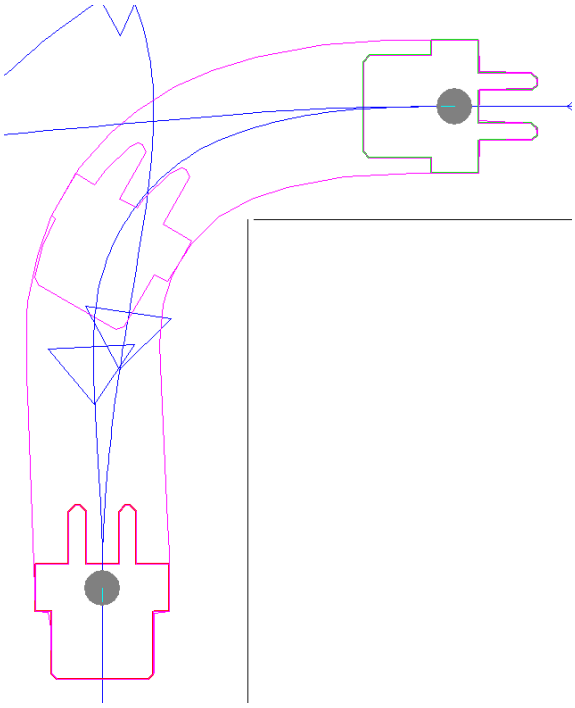


Figure 3.1: An example of the sweep (shown in purple) created around a segment in layout designer.

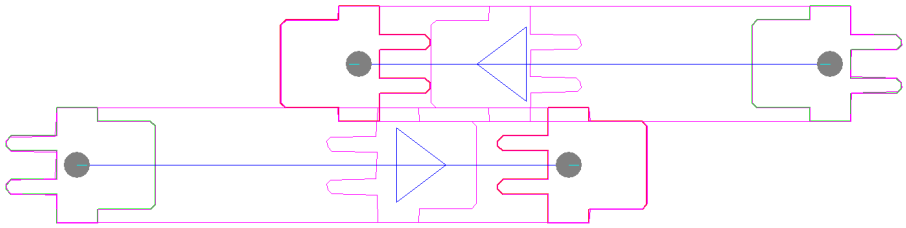


Figure 3.2: An example of a *head-on* deadlock in layout designer.

metrics are calculated are not disclosed.

3.1.5 Sweep

A sweep can be created given an AGV and its path. This shows the area which the AGV will cover while traversing its path.

3.1.6 Deadlock

In the case where an AGV detects another AGV as an obstacle a deadlock can occur. This implies that both AGVs are trying to traverse a segment so that their sweeps collides leading to both of the vehicles stopping and freezing up. Since an AGV cannot reverse, this leads to both of the vehicles getting stuck causing a deadlock.

3.1.7 Order flow

The order flow describes what should happen in a layout and sends commands to the AGVs on where to go. The order flow $\mathcal{F} = \{f\}$ is defined as a set of tuples $f = (t, p, \mathcal{O})$ where t is the time the order was placed, p is the order priority and $\mathcal{O}$ a tuple (v_i, v_j) $i \neq j$ and $v_i, v_j \in \mathcal{S} \subseteq \mathcal{V}$ where v are vertices that are specified as stations $\mathcal{S}$ (source or target nodes) in a graph $\mathcal{G}$.

Although in our case all orders have the same priority and we define the order flow $\mathcal{O}$ as a set of orders $\mathcal{O}_i = (f_i, v_j, v_k), f_i \neq 0$ where f_i is the order frequency in orders/hour and (v_j, v_k) are vertices specifically stations $\mathcal{S}$ (source or target nodes) so that $j \neq k$ meaning $v_j, v_k \in \mathcal{S} \subseteq \mathcal{V}$ in a graph $\mathcal{G}$. From this point forward the order flow denoted by $\mathcal{O}$ refers to this definition and the word "order" indicates $\mathcal{O}_i$.

3.1.8 Throughput

The throughput $\mathcal{T} = \{t\}$ of a layout is a measure of how many orders it can fulfill given an order flow $\mathcal{O}$. It is represented as a set of tuples $t_i = (O_i, \hat{f}_i)$ where $\hat{f}_i$ is the simulated frequency of order i.

3.2 Data

This section describes how the data given from the layouts and the order flow was combined to create the data set used to train the models. The order flow and the priority of the orders are identical for all layouts.

3.2.1 Layouts

The data set consists of 50 different layouts for the same T-intersection ranging from 25 to 120 nodes with the ratio between the amount of nodes and edges being around one. Although there are different amounts of changes some graphs are more similar than others which was visually assessed. The results of this assesment can be seen in table 3.1. Examples of the changes deemed to be minor enough for the layout to be similar to another one can be the change of the position of a node or a split of an edge, meaning that one edge is split into n edges with $n - 1$ new nodes equally distant on the previous edge.

3.2.2 Initial data set

This is the data set that was initially presented to us when we started the thesis. This subsection describes what the inputs and outputs are for one graph in this data set.

3.2.2.1 Input

The input $X = \mathbb{R}^{|\mathcal{V}| \times 4}$ consist solely of the node features without any edge attributes nor weights. These features are calculated using the layout and the order flow:

Table 3.1: An overview of the layouts in the data set visually assessed.

layout	similar to	layout	similar to	layout	similar to	layout	similar to
1	-	14	-	27	14	40	-
2	-	15	14	28	14	41	40
3	2	16	14	29	14	42	40
4	2	17	14	30	14	43	40
5	1	18	14	31	-	44	40
6	-	19	14	32	-	45	40
7	6	20	14	33	-	46	40
8	-	21	14	34	33	47	-
9	8	22	14	35	33	48	47
10	8	23	14	36	33	49	47
11	8	24	14	37	33	50	47
12	8	25	14	38	33		
13	8	26	14	39	33		

[*avg_outgoing_seg_weight*, *avg_incoming_seg_weight*, *incoming_traffic_load*, *wait_probability*].

The *avg_outgoing_seg_weight* is the average travel time for the edges going out from the node. It is calculated as

$$X_{i,0} = \begin{cases} \frac{1}{|\mathcal{N}_i|} \sum_{k \in \mathcal{N}_i} t(e_{ik}), & |\mathcal{N}_i| > 0 \\ 0, & \text{otherwise} \end{cases} \tag{3.1}$$

where $\mathcal{N}_i$ is node i's neighbour nodes and $t(e)$ gives the travel time of edge e.

The *avg_incoming_seg_weight* is the average travel time for the edges going in to the node. It is calculated as

$$X_{i,1} = \begin{cases} \frac{1}{|\mathcal{K}_i|} \sum_{k \in \mathcal{K}_i} t(e_{ki}), & |\mathcal{K}_i| > 0 \\ 0, & \text{otherwise} \end{cases} \tag{3.2}$$

where $\mathcal{K}_i$ is node v_i's neighbour nodes with edges directed into node v_i and $t(e)$ gives the travel time of edge e.

The *incoming_traffic_load* is the sum of the active usage of the incoming segments, meaning how much time of the total time that those segments are occupied by an AGV. This is a metric that is calculated using the layout analyzer tool.

$$X_{i,2} = \begin{cases} \frac{1}{|\mathcal{K}_i|} \sum_{k \in \mathcal{K}_i} \frac{1}{\sum_{t \in T} t} \sum_{t \in T} b(e_{ki}, t), & |\mathcal{K}_i| > 0 \\ 0, & \text{otherwise} \end{cases} \tag{3.3}$$

where $\mathcal{K}_i$ is node v_i's neighbour nodes with edges directed into node v_i, T is the set of the time differences in the non uniform time samplings of the simulation and b is

the piecewise function defined as:

$$b(x,t) := \begin{cases} t, & \text{AGV present at } x \\ 0, & \text{otherwise} \end{cases} \tag{3.4}$$

where x is a node.

The *wait_probability* is the probability that an AGV would have to stop and wait for another AGV at a node. This is a metric that is calculated using the layout analyzer tool.

$$X_{i,3} = \frac{1}{\sum_{t \in T} t} \sum_{t \in T} b(e_{ki}, t) \tag{3.5}$$

T is the total simulation time, t is each time step and $b(x,t)$ is the piecewise function defined in equation 3.4.

3.2.2.2 Target

The target $Y = \mathbb{R}^{1\times 2}$ consists of two values: [*average_flow_fulfillment*, *ok_order_flows_percentage*].

The *average_flow_fulfillment* is the average ratio between the simulated frequency and the order frequency respectively. Calculated as

$$Y_0 = \begin{cases} \dfrac{1}{|\hat{\mathcal{O}}|} \displaystyle\sum_{i=1}^{|\hat{\mathcal{O}}|} \frac{\hat{f}_i}{f_i}, & |\hat{\mathcal{O}}| > 0 \\ 0, & \text{otherwise} \end{cases} \tag{3.6}$$

where f_i is the order frequency, $\hat{\mathcal{O}}$ is all orders $\mathcal{O}_i$ where $\hat{f}_i \neq 0$ and $\hat{f}_i$ is the frequency of fulfillment of an order after a simulation. Since $\hat{f}_i \leq f_i$ the throughput is limited to be between $[0, 1]$.

The *ok_order_flows_percentage* is the percentage of orders with throughput above 95% of orders with a nonzero simulated frequency. Calculated as

$$Y_1 = \begin{cases} \dfrac{1}{|\hat{\mathcal{O}}|} \displaystyle\sum_{i=1}^{|\hat{\mathcal{O}}|} H\left(\frac{\hat{f}_i}{f_i}\right), & |\hat{\mathcal{O}}| > 0 \\ 0, & \text{otherwise} \end{cases} \tag{3.7}$$

with the same notations as in Equation 3.6 and where $H(x)$ is the piecewise function

$$H(x) := \begin{cases} 1, & x > 0.95 \\ 0, & x \leq 0.95. \end{cases} \tag{3.8}$$

3.2.2.3 Identified problems

A problem with the target values is that using $\hat{\mathcal{O}}$ in Y_0 calls for an unfair comparison between graphs. For example $\hat{f}_{i,1}$ in table 3.2 would have an *average_flow_fulfillment*

Table 3.2: Example order flow and simulated throughput of 3 different example layouts and the two different throughput metrics ORF($\mathcal{O}$) and TOF($\mathcal{O}$) and their mean MTRF($\mathcal{O}$) as well as the two initial targets Y_0 (Equation 3.6) and Y_1 (Equation 3.7) for each layout.

i	1	2	3	4	5	ORF	TOF	MTRF	Y_0	Y_1
f_i	10	20	5	5	10	-	-	-	-	-
$\hat{f}_{i,1}$	0	0	0	5	0	0.2	0.1	0.15	1	1
$\hat{f}_{i,2}$	10	0	5	5	0	0.6	0.4	0.5	1	1
$\hat{f}_{i,3}$	5	20	5	5	5	0.8	0.8	0.8	0.8	0.6

of $\frac{1}{1}\left(\frac{5}{5}\right) = 1$ whilst $\hat{f}_{i,3}$ would have $\frac{1}{5}\left(\frac{5}{10} + \frac{20}{20} + \frac{5}{5} + \frac{5}{5} + \frac{5}{10}\right) = \frac{4}{5} = 0.8$ although having a higher total quantity frequency. Since all of the layouts have the same order flow during simulations it is an unjust comparison to average over the nonzero orders.

The measure of Y_1 can be a good idea to get an overview of how well the layout can keep up with the orders at an acceptable rate (over 95%). However, this is an unnecessary nonlinearity (see equation 3.8) for the model to predict and would therefore be good to exclude.

3.2.3 Refined data set

To address the issues and concerns described in Section 3.2.2.3 a new data set was introduced with the following input and target.

3.2.3.1 Input

The input consists of two different types of graphs for each layout, one primal and one dual graph. The primal graph is similar to the initial input data set explained in Section 3.2.2.1 although the node features are now [x, y, $^*\mathcal{O}$] where x is the Cartesian x coordinate, y is the Cartesian y coordinate and $^*\mathcal{O}$ is the expanded form of the order flow frequencies associated with the node. In our case the station nodes are either drop off or fetch stations and cannot be both.

The primal graph also contains edge features [*edge_length*, *travel_time*] where *edge_length* is the length in meters of the edge and *travel_time* is the time it takes in seconds for the AGV to traverse the edge.

The dual graph is represented as described in section 2.2.1, where the node features are the same as the edge features in the primal graph, [*edge_length*, *travel_time*]. The edge features (the *between_edge* connections) are $[\theta, l(\theta), r(\theta), s(\theta)]$ where θ is the angle in degrees in the range [−180°, 180°] and

$$l(\theta) = \begin{cases} 1, & \theta < -\delta \\ 0, & \text{otherwise} \end{cases} \tag{3.9}$$

$$r(\theta) = \begin{cases} 1, & \theta > \delta \\ 0, & \text{otherwise} \end{cases} \tag{3.10}$$

$$s(\theta) = \begin{cases} 1, & -\delta \leq \theta \leq \delta \\ 0, & \text{otherwise} \end{cases} \tag{3.11}$$

where δ is the *straight deadband parameter*, which is how many degrees the road can turn while considering it straight. $(l(\theta), r(\theta), s(\theta))$ is essentially the one-hot encoding of 3 different direction types *left*, *right* and *straight*. The parameter δ would have to be larger for layouts with more nodes since θ would be low because of the edges being short and the opposite for layouts with less nodes. We used $\delta = 2$ for our data set.

3.2.3.2 Target

The goal of the model is to predict the throughput which can be represented as a point value. This point value has to be calculated in such a way that it represents how well the layout is performing compared to other layouts with respect to the order flow (although in our case the order flow is the same for all of the layouts).

We propose two different ways to calculate this target throughput value. The *order relative frequency* $\text{ORF}(\mathcal{O})$ calculated as:

$$\text{ORF}(\mathcal{O}) = \frac{1}{|\mathcal{O}|} \sum_{i=1}^{|\mathcal{O}|} \frac{\hat{f}_i}{f_i} \tag{3.12}$$

This metric weighs the fulfillment of each order in the order flow in contrast to the *total order frequency* $\text{TOF}(\mathcal{O})$ calculated as:

$$\text{TOF}(\mathcal{O}) = \frac{\sum_{i=1}^{|\mathcal{O}|} \hat{f}_i}{\sum_{i=1}^{|\mathcal{O}|} f_i} \tag{3.13}$$

which weighs the overall throughput higher. Which one of these two to use is a designer's choice, depending on what is considered the most important for the layout; A higher fulfillment where orders with a lower frequency play a bigger role although they might be easier to keep up with because of their lower frequency (ORF) or a higher overall throughput where fulfilling the orders with a higher frequency play a bigger role (TOF). To consider both metrics equally one could consider the mean of ORF and TOF, we call this the *mean total relative frequency* $\text{MTRF}(\mathcal{O})$ and it is calculated as :

$$\text{MTRF}(\mathcal{O}) = \frac{\text{TOF}(\mathcal{O}) + \text{ORF}(\mathcal{O})}{2} \tag{3.14}$$

3.2.4 Final data set

Although section 3.2.3 gives a better representation of the data associated with the layouts than the inital data set described in section 3.2.2, it is still missing the representation of the curvature of the edges and how the orders will flow within the graph. Therefore a final data set was created to contain all information needed to completely recreate the layout.

3.2.4.1 Global features

Taking inspiration from GPS [8] global features were added as an attribute on a graph level. This feature vector consists of the following:
$[number\ of\ nodes, number\ of\ edges, \frac{number\ of\ nodes}{number\ of\ edges}]$.

3.2.4.2 Order flow DFS embedding

Using the order flow and the graph we use the *Depth-First Search* (DFS) algorithm to find all possible paths from the fetch station and the drop off station and embed the order's frequency in the nodes in these paths. These frequencies are then aggregated using the mean function although an arbitrary aggregation function could be used. The full algorithm of this embedding can be seen in algorithm 2.

Algorithm 1 The find all paths algorithm using Depth-First Search to find all possible paths from node A to B.

```
1:  function FindAllPaths(src: Node, dst: Node)
2:      allPaths ← [ ]
3:      path ← [src]
4:      edgePath ← [ ]
5:
6:      function DFS(src, dst, path: list[Node], edgePath: list[Edge])
7:          if src = dst then
8:              append tuple (path, edgePath) to allPaths
9:          else
10:             for all edge from src do
11:                 append edge to edgePath
12:                 append head of edge to path
13:                 DFS(src, dst, path, edgePath)
14:                 remove last from path
15:                 remove last from edgePath
16:             end for
17:         end if
18:     end function
19:
20:     DFS(src, dst, path, edgePath)
21:     return allPaths
22: end function
```

Algorithm 2 Algorithm to embed the order flow using the paths given from Algorithm 1 and the order flow

Require: *orderFlow* is unidirectional

$flows \leftarrow [\,]$
for every *order* **do**
    $src \leftarrow$ order source node
    $dst \leftarrow$ order destination node
    $allPaths \leftarrow$FindAllPaths(src, dst)

    for *path* **in** $allPaths$ **do**
        $nodes \leftarrow$ nodes in *path*
        $edges \leftarrow$ edges in *path*
        for *node* **in** *nodes* **do**
            $flow \leftarrow [0, \ldots, 0]$ ▷ shape $1 \times |stations|$
            $flow[src] \leftarrow$ frequency of *order*
            append $flow$ to $flows[order][node]$
        end for
    end for

    $flows[order] \leftarrow$OrderAggregator($flows[order]$)
end for

$flows \leftarrow$TotalAggregator($flows$)

Table 3.3: Order flow given by fetch and place stations and a frequency of how often the order should be placed in hours. This is the order flow used when simulating the throughput for the layouts.

Order ID	Fetch Station	Place Station	Frequency
1	A	F	200
2	C	F	380
3	B	E	320
4	B	D	170
5	A	E	215

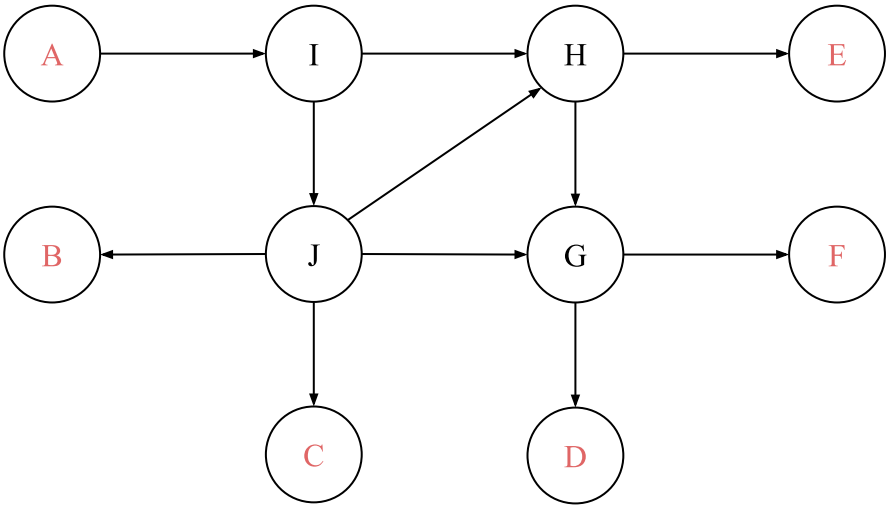


Figure 3.3: An example directed graph where the letters A, B, C, D, E and F are in red indicating those nodes being a station.

An example of the order flow DFS embedding using algorithm 2 can be seen in figure 3.3 using the graph seen in figure 3.4 and the order flow seen in table 3.3.

3.2.4.3 Graph extension

To fully include the B-spline curvature of the edges a graph can be extended as described in section 2.2.2. The four control points are then embedded into the new edges from the parts along with the length of that part as well as the travel time. As it was not possible to get the exact travel time, a linear mapping from the segment travel time was made according to:

$$\hat{t} = \frac{\hat{l}}{l} \cdot t \tag{3.15}$$

where $\hat{t}$ is the travel time for the part, $\hat{l}$ is the length of the part, l is the length of the segment and t is the travel time for the segment. This leads to the edges having $2 + 4 \cdot 2 = 10$ features $[\mathit{edge\_length}, \mathit{travel\_time}, c_{0,x}, c_{0,y}, c_{1,x}, c_{1,y}, c_{2,x}, c_{2,y}, c_{3,x}, c_{3,y}]$ where the position of each control point is embedded as c.

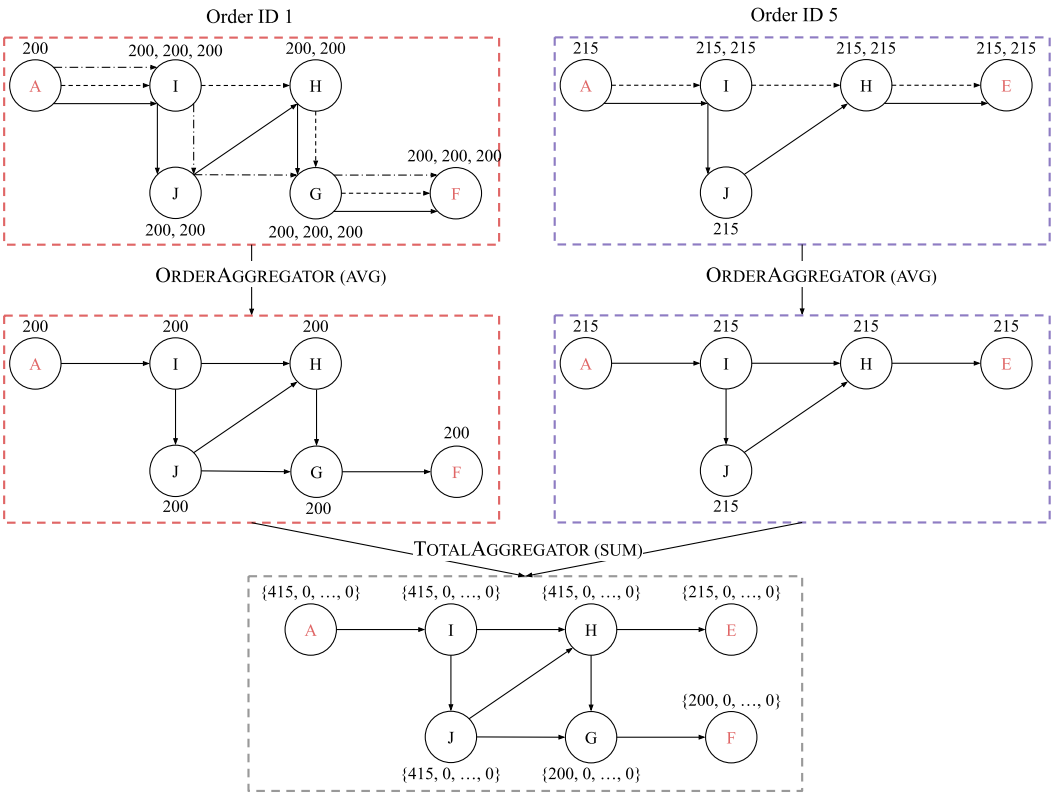


Figure 3.4: An example of the DFS order embedding considering all the order with node A as their fetch station according to algorithm 2 (indicating the first position in the frequency vector F_0).

3.2.5 Preprocessing

The node features [x, y, $*\mathcal{O}$], the edge for the primal graph/node features for the dual graph [*edge_length*, *travel_time*] and the angle θ in the edge features for the dual graph are normalized using a max min normalisation. This is done since it helps with the numerical stability of the model according to [14].

To allow for the layouts to not be fully connected, there are three nodes in each layout where the AGV is manually moved instantaneously when it has delivered an order. These nodes as well as the edges connecting them are removed in all layouts. The reason behind these nodes being removed is that they give no relevant information to the model and their purpose is solely to be used in the simulation software.

3.2.5.1 Data split

Both the initial and final data set are shuffled after they are placed into three sets, the training, test and validation set with a split of 80%, 10% and 10% respectively. The initial and the final data set both contains the same layouts although the embedding is different.

The data set contains some simulations that have zero in throughput, this is due to deadlocks in the layout causing the AGVs to just stand still in one position for the entire simulation. We use stratified sampling on the functional and deadlock sets as shown in figure 3.5 [15]. The reason being that it is important that we keep a similar amount of low throughput layouts and high throughput layouts in the train, test and validation sets. (By looking at the mean throughput we can see how equal the split is between the different sets.) The reasoning being that it would be hard for the model to generalize to the poorly performing layouts if it has never seen one before. To consider a layout to be of low throughput, 0.01 was used as a cutoff value. The throughput of the different graphs can be seen in figure 3.5.

3.3 Loss function

Since this is a regression task and not a classification task it is not viable to use loss functions such as cross entropy. 24% of all the target values are zero which could lead to a bias in the model towards estimating zero. We use the mean square error (MSE) as our loss function to help combat this issue.

3.4 Networks

The following network structures were compared using the final data set as described in section 3.2.4. Although the MLP, GCN and GAT networks only use the primal graph representation the RFN and RNG networks use several different graph representations of the layout as described in section 2.3.3 and 2.3.4 respectively.

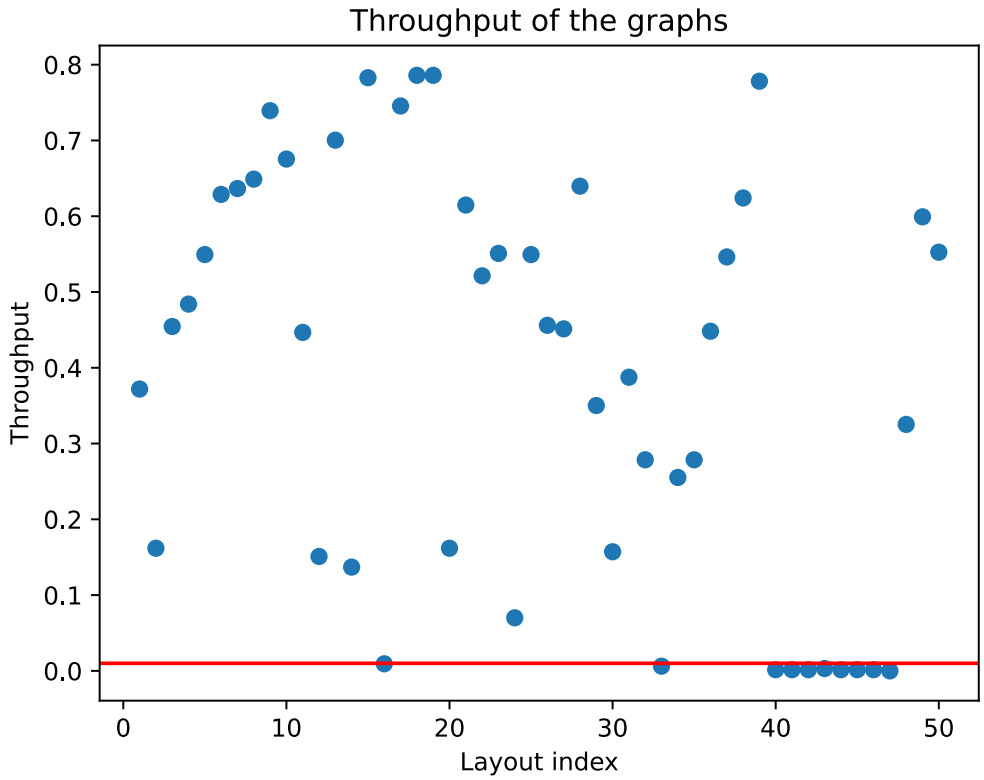


Figure 3.5: The throughput of all the layouts in the data set. The red line indicates the cutoff value where the data is split into a functional data set and a deadlock one.

- **MLP** is applied to the node features after they are aggregated using the max, min, mean and sum aggregators and is then passed into another MLP.
- **GCN** is applied on the node features of the graph and are then aggregated using the max, min, mean and sum. This is then passed into an MLP.
- **GAT** is applied to the graph, including the use of edge features. After applying the GAT, the node features are aggregated using the max, min, mean and sum, and then passed into an MLP. This aggregation is done for each head and concatenated.
- **RFN** see section 2.3.3
- **RNG** see section 2.3.4

The results of these networks can be seen in chapter 4.

3.4.1 Dropout

To allow for the models to better generalize and prevent overfitting a dropout of 50% is used on all message passing layers for all networks stated above. A 50% dropout is also used for all the layers in the MLP except for the output layer (last layer).

3.4.2 Training

When training all of the networks a constant learning rate of 0.01 was used using the ADAM optimizer [16]. An early stopper was used with a patience of 40 epochs without a minimum delta. The early stopper ensures that the training stops if the validation loss has stagnated.

4

Results

In this chapter the results from the chosen networks are presented.

4.1 Hyperparameter search

A hyperparameter search was preformed to find the optimal combination of layers and hidden channels, for example, in the different networks. Which hyperparameters were varied can be seen in table 4.1. For each configuration the networks were trained ten different times to conclude the overall convergence of the model. The list below describes how these hyperparameters were varied.

- *hidden layers*: 1, 2 and 3
- *hidden channels*: 8, 16, 24, 32, 40, 48, 56 and 64
- *hidden factor*: 1, 2, 3, 4, 5, 6, 7 and 8
- *heads*: 1, 2, 3 and 4
- *interactional fuse*: yes and no

Table 4.1: This table showcases the different hyperparameters that were varied for the different models. Note that hidden channels is similar to the hidden factor that solely RFN uses in the network structure. The reason that the RFN hidden factor is introduced, is so that the hidden channels in the RFN can scale with the input size of the fusions.

parameter/network	MLP	GCN	GAT	RFN	RNG
hidden layers	YES	YES	YES	YES	YES
hidden channels	YES	YES	YES	NO	YES
heads	NO	NO	YES	NO	YES
hidden factor	NO	NO	NO	YES	NO
interactional fuse	NO	NO	NO	YES	NO

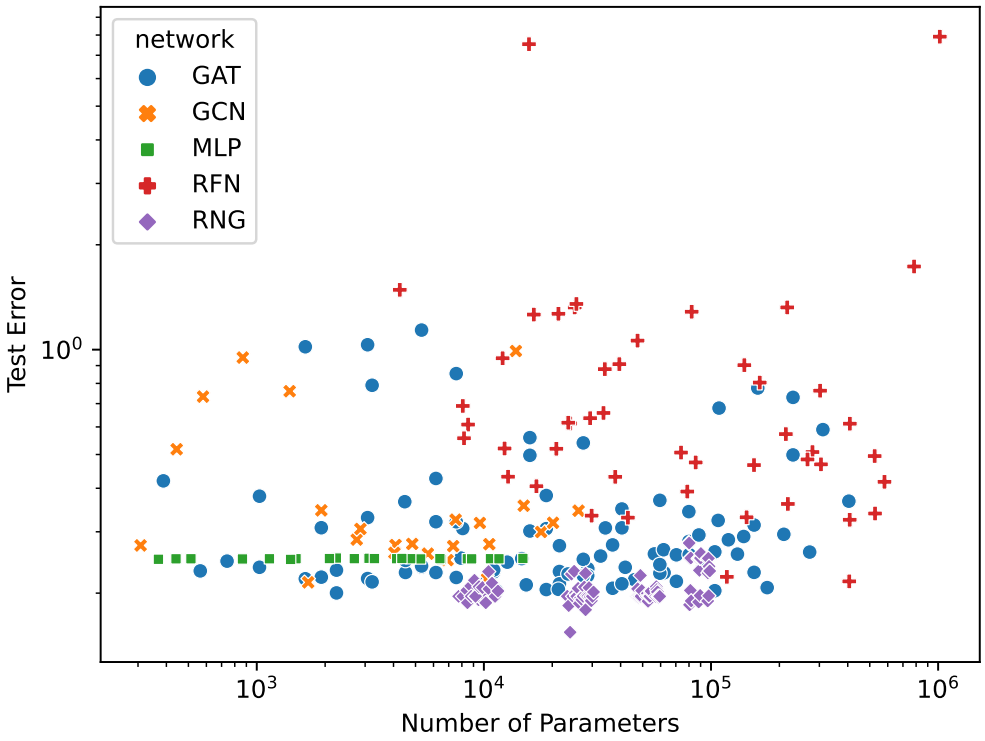


Figure 4.1: The five different network types scatter plotted with the different configurations averaged over the 10 training iterations. All networks in this figure were trained on the final data set.

4.2 Network comparison

For a fair comparison to be made the amount of learnable parameters has to be taken into account. These learnable parameters are the weights and biases for the different layers in the models. In figure 4.1 a comparison between the test error and different amounts of learnable parameters can be seen. In this figure it is quite clear that our novel network, the RNG, outperforms all the other networks with most configurations. The four different clusters that can be seen from the RNG are created when varying the amount of heads. The RNG network structure also seems to be robust to changes in the hyperparameters. The models' increase in learnable parameters does not seem to to affect the test error apart from random fluctuations. This is surprising since the expected behaviour when adding more parameters is that this would decrease the test error. Possibly, the main reason behind this is that the data set is too small, leading to the larger models to converge to a similar test error as the smaller ones. This issue is further discussed in section 5.3.1.

Further analysis into the RNG network can be made by creating a heatmap of one of the heads. Such a heatmap can be seen in figure 4.2 for an RNG network with one head. From the looks of this heatmap, one layer and more than 24 hidden channels

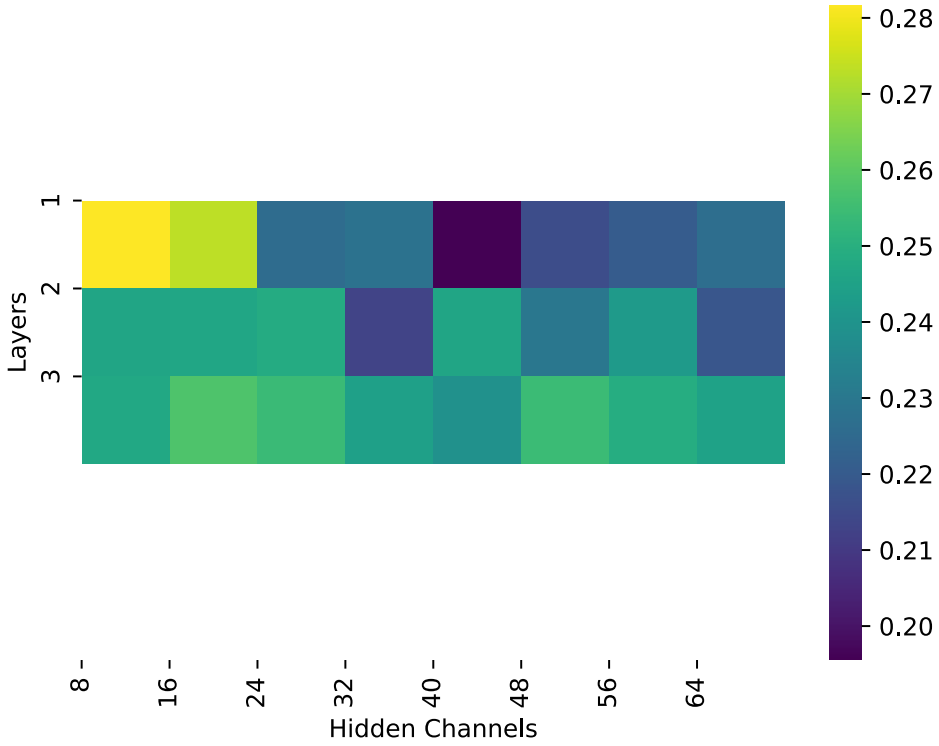


Figure 4.2: A heatmap showing the test error for RNG for one head.

seems to be the best combination.

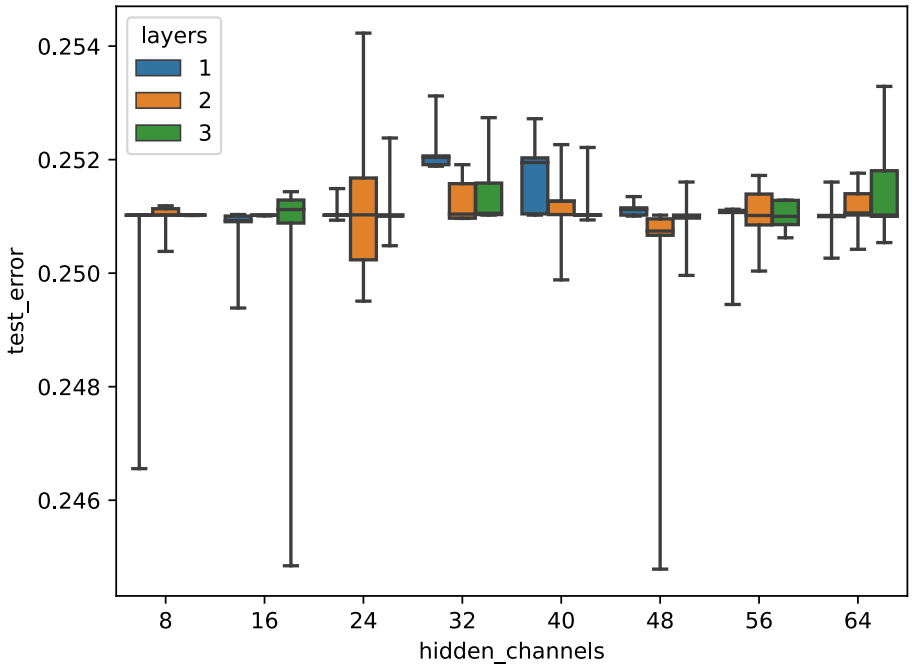
Even with the same hyperparameters, there is a considerable amount of variance for some networks. The GCN (see figure 4.3b), GAT (figure 4.3c) and RFN (figure 4.3e and 4.3f (especially with interactional fuse), show a high amount of variance in the training. However the MLP and RNG networks are quite stable in their convergence.

4.2.1 Average versus minimum error comparison

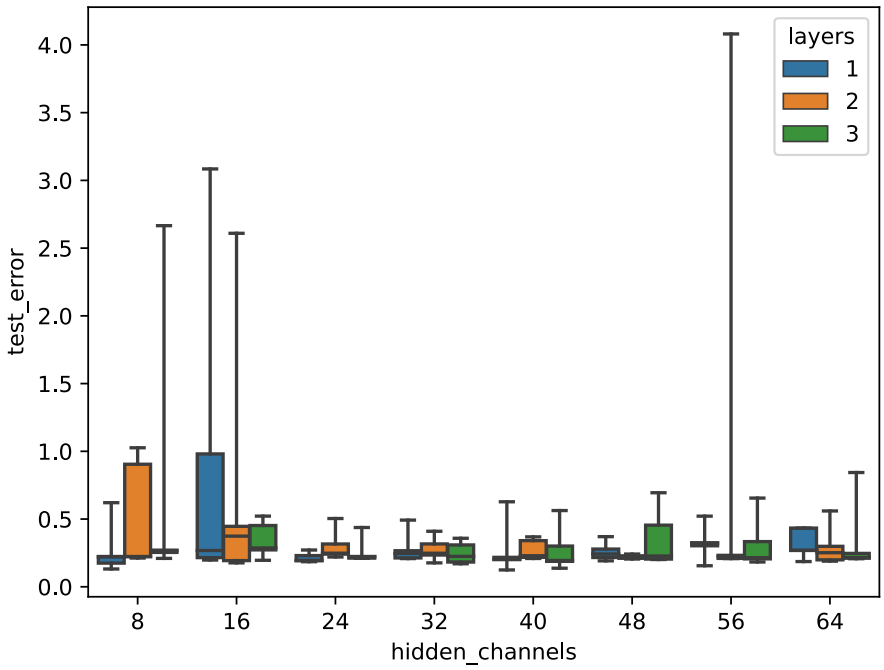
In figure 4.4 the mean and minimum values for the test and train errors for the networks are plotted. One interesting thing to note is that while the RNG network often is the best, the RFN can outperform RNG in testing. This could mean that the RFN has learnt to generalize better than the RNG network in these cases. However, what is more likely is that this difference is caused by the small size of the test set and not because of some property of the RFN. It basically just gets lucky.

4.2.2 Comparison of the initial and final data set

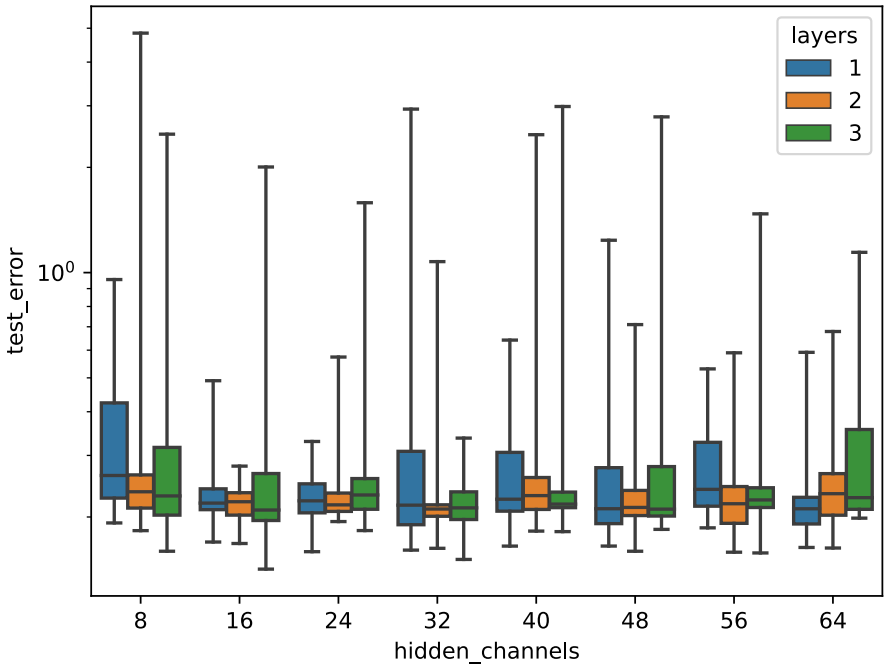
A comparison of the initial and final data set is also in order, as to motivate the changes made. The comparison will only be made between the networks: MLP,



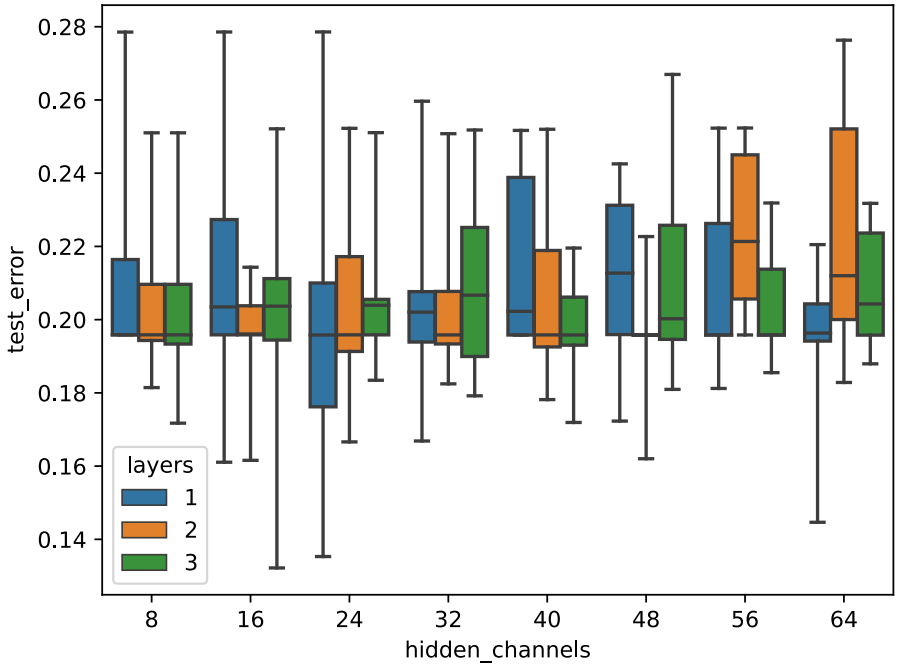
(a) The results for the MLP. Linear scale.



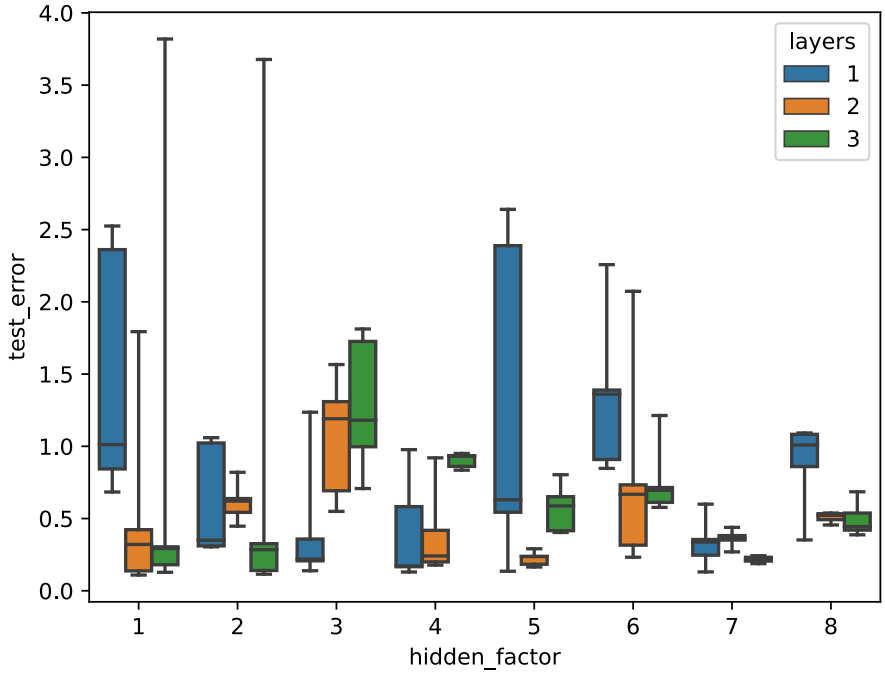
(b) The results for the GCN. Linear scale.



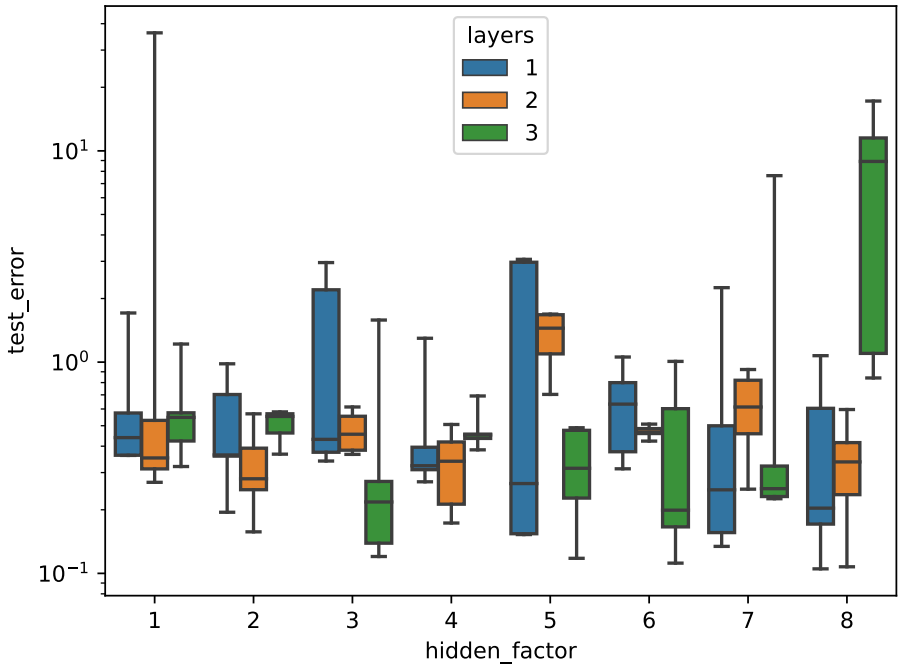
(c) The results for the GAT. Log scale. All heads included.



(d) The results for the RNG. Linear scale. All heads included.

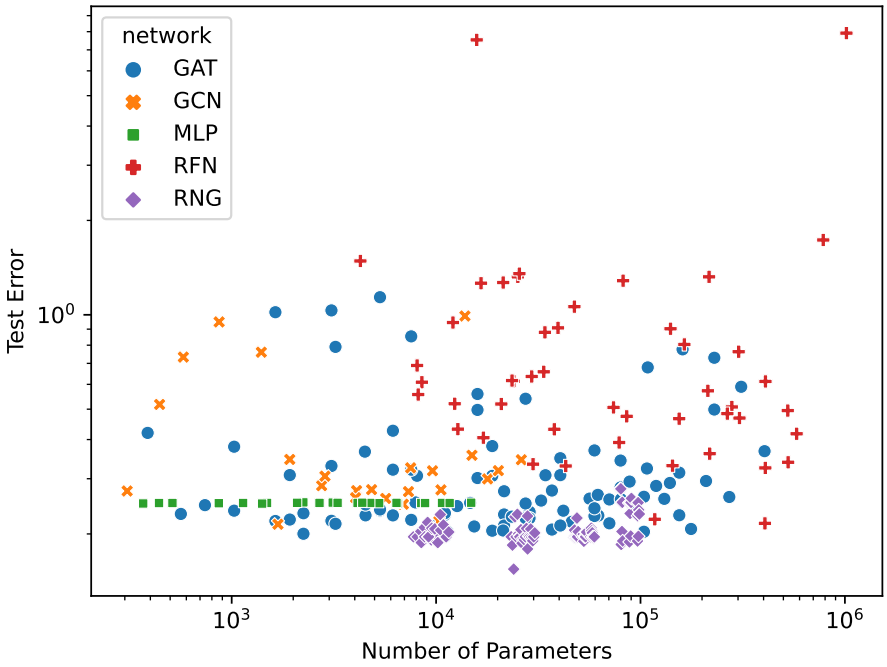


(e) The results for the RFN with additive fuse. Linear scale.

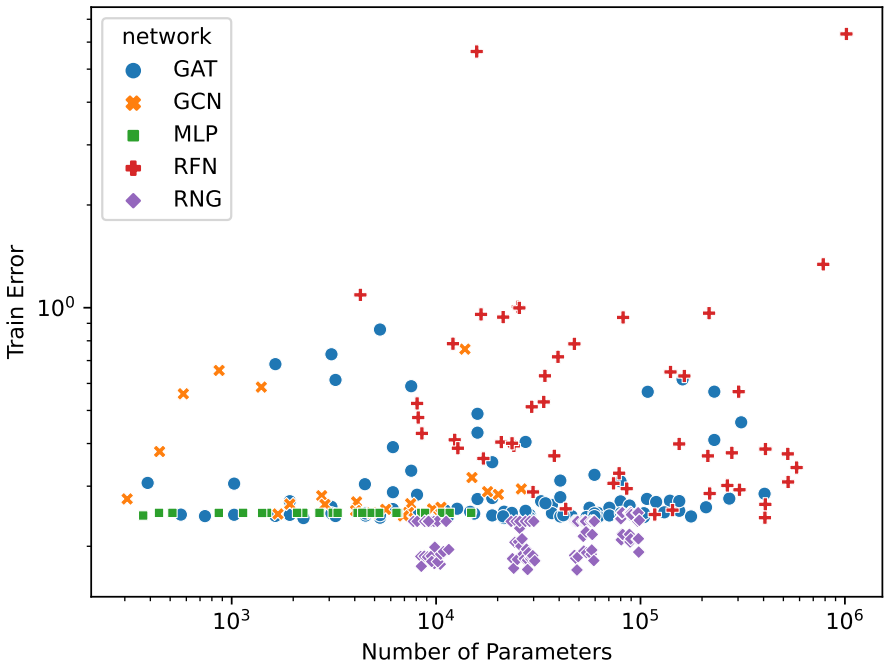


(f) The results for the RFN with interactional fuse. Log scale.

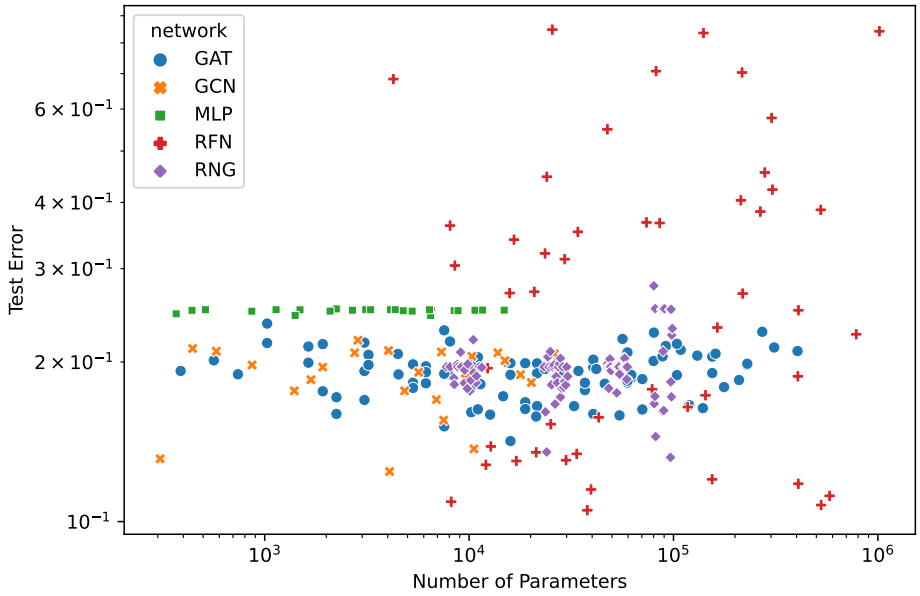
Figure 4.3: Box plots over the ten different runs with the same network hyperparameters. The edges of the box are the quartiles and the whiskers are the max/min values of the data.



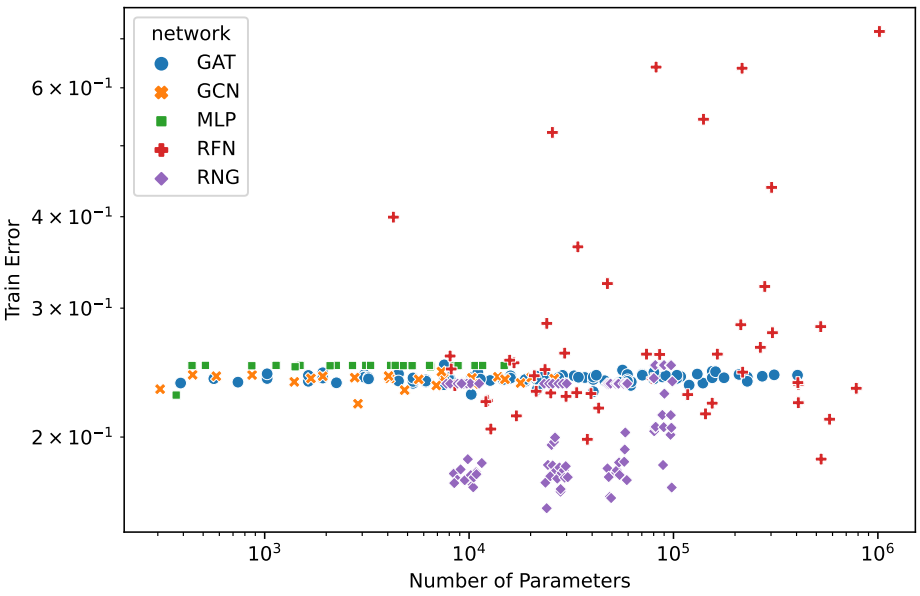
(a) Plot of the test error averaged over the ten runs.



(b) Plot of the train error averaged over the ten runs.



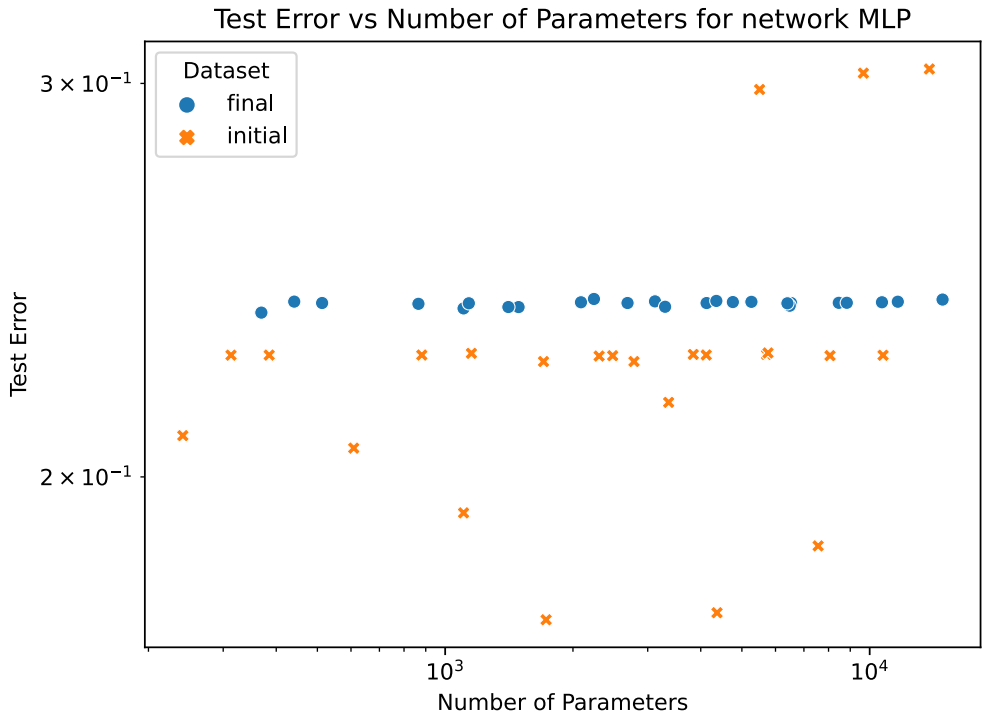
(c) Plot of the minimum of the test error.



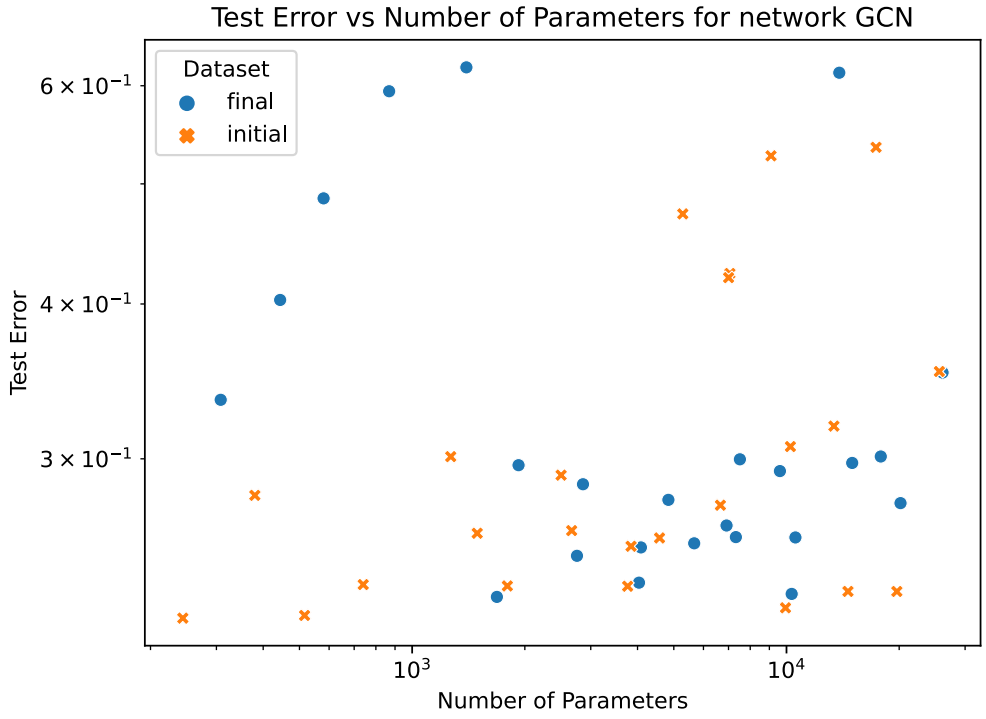
(d) Plot of the minimum of the train error.

Figure 4.4: Scatter plots showing the test/train error for different networks and hyperparameters. Sometimes the RFN performs well on the test set, which is interesting. This is probably just a fluke though.

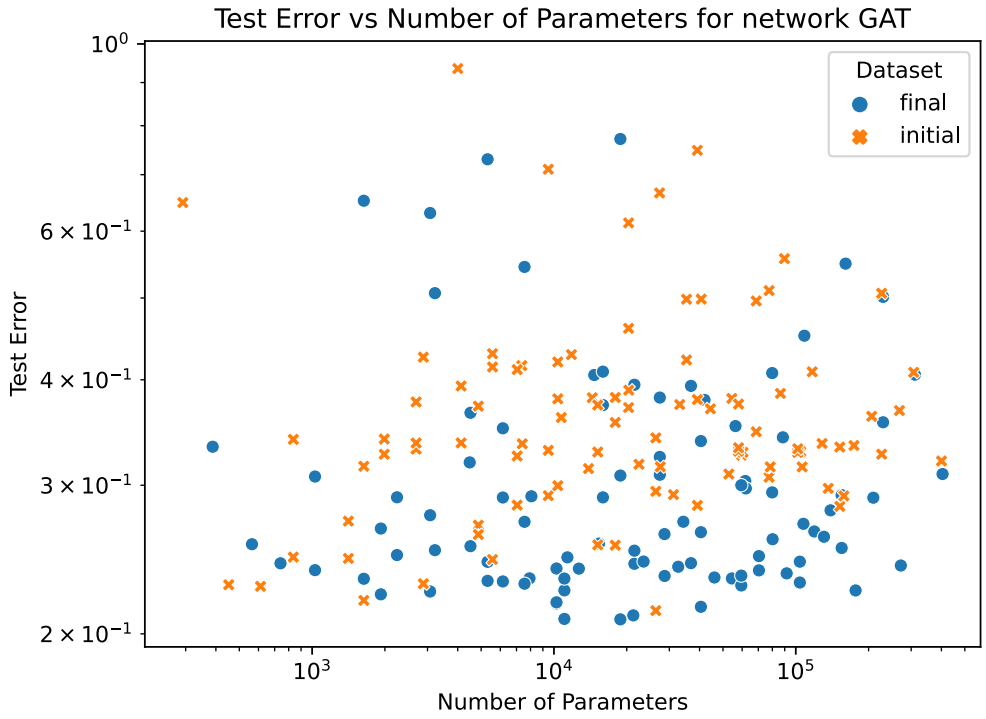
GCN and GAT as the RFN and RNG networks need dual graphs and extended graphs in their current implementation. The MLP, GCN and GAT are also more rigorously studied networks and are not specially made to fit the final data set in any way. In figure 4.5 we can see the comparisons. In general it seems that the GCN and GAT perform better on the final data set.



(a) Comparison of the test errors for the initial and final data set using the MLP network.



(b) Comparison of the test errors for the initial and final data set using the GCN network.



(c) Comparison of the test errors for the initial and final data set using the GAT network.

Figure 4.5: Different comparisons of the test errors for some networks. In general, they perform better on the final data set, especially when the number of parameters is large. There is some randomness however, and it is hard to say for sure.

5

Conclusion

This chapter explains the key findings of the results in the report as well as the implications of the data set and the limitations of the models investigated.

5.1 Key findings

Graph neural networks have proven to be a good network structure in our case, although, a concern is how much of the data is lost. Since an aggregation is required between all of the nodes in the graph to be able to allow for an MLP to do the regression task. Therefore inspiration from [7] was taken, using multiple aggregators as well as using global features of the graph as used in [8]. This proved to be of great importance and as shown in figure 4.1 led to our novel network being the best of the tested networks.

5.2 Environmental implications

There were several environmental implications of the data set (and the thesis as a whole) that was given to us at the start of the project. The simulations take a lot of computing power and therefore energy. If the number of simulations could be reduced or removed entirely then a lot of energy could be saved. Of course, having more efficient layouts is also good in itself. If the productivity can be achieved with less energy (e.g. using a good layout) then energy could be saved there as well. One problem is that training the networks is costly. How the balance of the advantages and disadvantages is in the real world is hard to say for sure but if the networks were to work properly, it could be environmentally friendly.

5.3 Limitations

In this section we identify the limitations in our research and explain how these impact the validity of our conclusions.

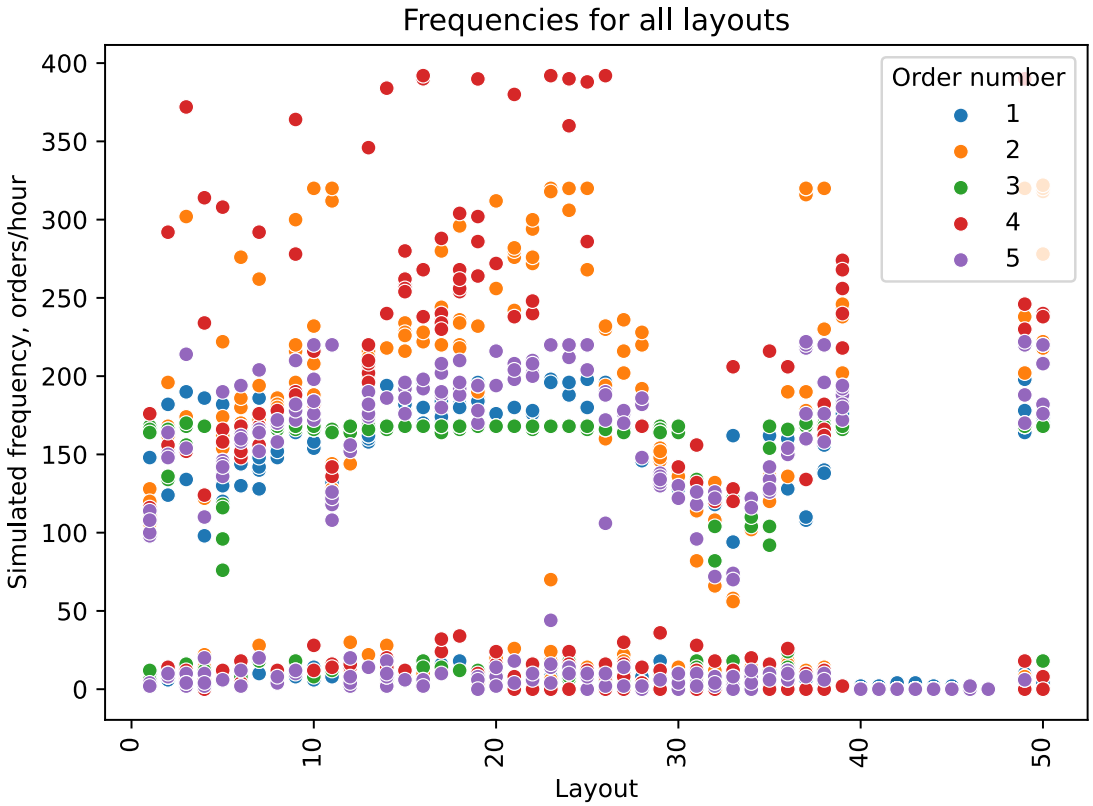


Figure 5.1: Plot of the throughputs of each order per layout over six different simulations. We see a large spread of the frequencies.

5.3.1 Data set

An obvious limitation to the models is the amount of data that is being used to train on. Containing only 49 unique layouts leads to the test set containing only five graphs when using a data split of 80/10/10 for the training, validation and test set. These proportions are commonly used but as the data set is so small this might lead to problems. For example, the networks not being able to generalize could be because the graphs in the validation and test sets simply do not appear in the training set. Of course, the stratified sampling should help with this problem but there is still a lot of uncertainty.

Another reason why it is hard for the models to generalize to the data, is because of the uncertainty in the simulation software. When simulating the same layout several times, the throughput is not consistent, and the distribution of the throughput seems to have a lot of randomness. This makes it difficult to predict and find a correlation to the input data which is what the training of the model is trying to accomplish. In figure 5.1 we can see the variance of the throughputs from these simulations.

5.3.2 Edge travel time

When extending the graph, the travel times for the new edges had to be recalculated. Currently, the new travel time is calculated with a linear scaling using the ratio of the length of the new edge and length of the old edge, as can be seen in equation 3.15. In reality, the new travel time should not use a linear scaling since the travel time of the new edge depends on its curvature. This leads to an incorrect scaling for all the edges in the extended graph.

5.3.3 LAT statistics

When extending a graph the LAT would have to be rerun to generate the new LAT metrics such as the *incoming_traffic_load* and the *wait_probability* for the new edges. The current implementation assumes that these metrics are the same even when the edge is being extended. Other edge attributes such as the *travel_time* and *length* are changed accordingly, although the *travel_time* is not exact as mentioned in the previous section.

There are some edges that do not have any metrics set from LAT. These edges have their metrics set to zero.

5.4 Future directions

This section discusses the potential future directions for research or analysis in this area. It includes new research questions, methods, or approaches that build on our findings.

5.4.1 Further limiting the problem

A way to advance this thesis would be to limit the problem space even further since it seems, judging from the simulation results, that very minor changes in even a single node's position can cause a big change in the throughput. This is likely due to the nature of deadlocks appearing due to a change in an edge's curvature caused by a change in a node's position.

Therefore, it would be beneficial to have a data set containing very similar layouts, possibly even the same amount of nodes, edges and connections, changing solely the positions of the nodes and the curvature of the edges. This would limit the problem space even further and function as a first stepping stone to understand the problem better and develop a novel network to better predict the throughput.

It would also be of great interest to only change the shapes of the edges (B-splines) while keeping the nodes positions static, limiting the problem even further.

5.4.2 Bigger data set

A data set described in section 5.4.1 could be generated using limited random node positions. For example each node has a given region where they would be randomly positioned as the data set is generated. This would allow for a bigger data set to be generated quickly and the possibility to also change the edges' control points in a similar way.

5.4.3 Many to some to one

It would be interesting to conduct further research into how to incorporate more of the node and edge features of the graph as input for the final MLP responsible for the many to one task and limiting the use of aggregators. One way of doing this would be the use a graph network to rank the importance of each node then concatenating the n most important nodes and use that as input for the final MLP. This would however introduce a constraint of a minimum size of a layout but would pass more unaggregated information about the nodes and edges of the graph to the final MLP.

5.4.4 Different models

Another direction would be to test different networks not used in this thesis. Because of the scope of the project, these networks were not included in the simulations but it could be interesting to examine whether other networks could perform better than the ones presented. Examples of different networks could be the *Transformer Convolutional Network* introduced in [17] or the already mentioned GPS [8] or PNA [7] networks.

5.4.5 Different targets

Aside from changing the network structure one could also change the target values. One example could be trying to predict the throughput of each order flow instead of an order flow aggregation. However, as can be seen in figure 5.1, there is a lot of uncertainty in the distribution of the orders. This is the reason why this thesis has focused on a single value calculated from the different throughputs.

Also changing the way throughput is calculated could make the networks better. Maybe the ORF, TOF and the MTRF are all bad at classifying if the layout is good. This is being mentioned as there has not been any testing of these different targets.

Bibliography

[1] C. de Boor, *A Practical Guide to Splines*. New York: Springer-Verlag, 1978, pp. 325–326, ISBN: 978-0-387-95366-3. [Online]. Available: https://link.springer.com/book/9780387953663.

[2] E. A. Bender and S. G. Willamson, *Lists, Decisions and Graphs*. S. Gill Williamson, 2010, p. 161. [Online]. Available: https://books.google.se/books?id=vaXv_yhefG8C.

[3] T. S. Jepsen, C. S. Jensen, and T. D. Nielsen, "Relational fusion networks: Graph convolutional networks for road networks," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 1, pp. 418–429, 2022. DOI: 10.1109/TITS.2020.3011799.

[4] J. Zhou, G. Cui, S. Hu, *et al.*, "Graph neural networks: A review of methods and applications," *AI Open*, vol. 1, pp. 57–81, 2020, ISSN: 2666-6510. DOI: https://doi.org/10.1016/j.aiopen.2021.01.001. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2666651021000012.

[5] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Galligher, and T. Eliassi-Rad, "Collective classification in network data," *AI Magazine*, vol. 29, no. 3, pp. 93–106, Sep. 2008. DOI: 10.1609/aimag.v29i3.2157. [Online]. Available: https://ojs.aaai.org/aimagazine/index.php/aimagazine/article/view/2157.

[6] A. A. Awan, *A comprehensive introduction to graph neural networks (gnns)*, accessed 02-May-2023, 2022. [Online]. Available: https://www.datacamp.com/tutorial/comprehensive-introduction-graph-neural-networks-gnns-tutorial.

[7] G. Corso, L. Cavalleri, D. Beaini, P. Liò, and P. Veličković, "Principal neighbourhood aggregation for graph nets," in *Proceedings of the 34th International Conference on Advances in Neural Information Processing Systems*, vol. 16, Red Hook, NY, USA: Curran Associates Inc., 2020, pp. 13 260–13 271. [Online]. Available: https://proceedings.neurips.cc/paper/2020/file/99cad265a1768cc2dd013f0e740300ae-Paper.pdf.

[8] L. Rampášek, M. Galkin, V. P. Dwivedi, A. T. Luu, G. Wolf, and D. Beaini, "Recipe for a general, powerful, scalable graph transformer," in *Proceedings of the 36th International Conference on Advances in Neural Information Processing Systems*, vol. 35, Red Hook, NY, USA: Curran Associates Inc., 2022, pp. 14 501–14 515. [Online]. Available: https://proceedings.neurips.cc/

paper_files/paper/2022/file/5d4834a159f1547b267a05a4e2b7cf5e-Paper-Conference.pdf.

[9] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *Proceedings of the 5th International Conference on Learning Representations*, Palais des Congrès Neptune, Toulon, France, Apr. 2017. DOI: 10.48550/ARXIV.1609.02907. [Online]. Available: https://arxiv.org/abs/1609.02907.

[10] D. K. Hammond, P. Vandergheynst, and R. Gribonval, "Wavelets on graphs via spectral graph theory," *Applied and Computational Harmonic Analysis*, vol. 30, no. 2, pp. 129–150, 2011, ISSN: 1063-5203. DOI: https://doi.org/10.1016/j.acha.2010.04.005. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1063520310000552.

[11] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *Proceedings of the 30th International Conference on Neural Information Processing Systems*, Barcelona, Spain: Curran Associates Inc., 2016, pp. 3844–3852, ISBN: 9781510838819.

[12] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," in *Proceedings of the 6th International Conference on Learning Representations*, Vancouver, BC, Canada, Feb. 2018. DOI: 10.48550/ARXIV.1710.10903. [Online]. Available: https://arxiv.org/abs/1710.10903.

[13] S. Brody, U. Alon, and E. Yahav, "How attentive are graph attention networks?" In *Proceedings of the The Tenth International Conference on Learning Representations*, Virtual Event: OpenReview.net, Apr. 2022. [Online]. Available: https://openreview.net/forum?id=F72ximsx7C1.

[14] G. Aksu, C. O. Güzeller, and M. T. Eser, "The effect of the normalization method used in different sample sizes on the success of artificial neural network model," *International Journal of Assessment Tools in Education*, vol. 6, no. 2, pp. 170–192, 2019. DOI: 10.21449/ijate.479404.

[15] S. Menon, *Stratified sampling in machine learning*, Analytics Vidhya, accessed 09-Jun-2023, Medium, Dec. 2020. [Online]. Available: https://medium.com/analytics-vidhya/stratified-sampling-in-machine-learning-f5112b5b9cfe.

[16] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proceedings of the 3rd International Conference on Learning Representations*, San Diego, CA, USA, May 2015. [Online]. Available: http://arxiv.org/abs/1412.6980.

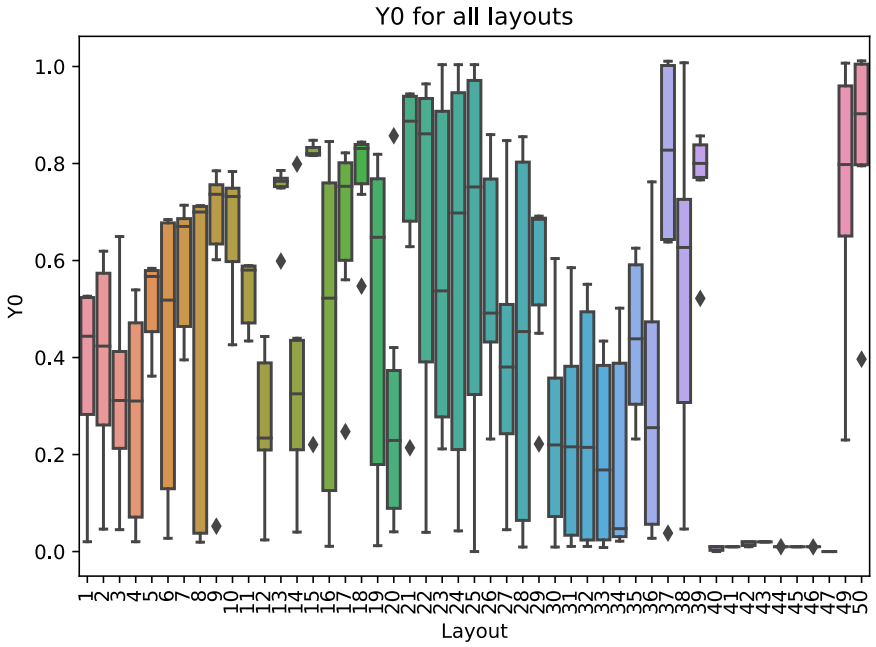
[17] Y. Shi, Z. Huang, S. Feng, H. Zhong, W. Wang, and Y. Sun, "Masked label prediction: Unified message passing model for semi-supervised classification," in *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence*, Montreal, Québec, Canada: International Joint Conferences on Artificial Intelligence Organization, Aug. 2021, pp. 1548–1554. DOI: 10.24963/ijcai.2021/214. [Online]. Available: https://proceedings.neurips.cc/paper/2020/file/99cad265a1768cc2dd013f0e740300ae-Paper.pdf.

A

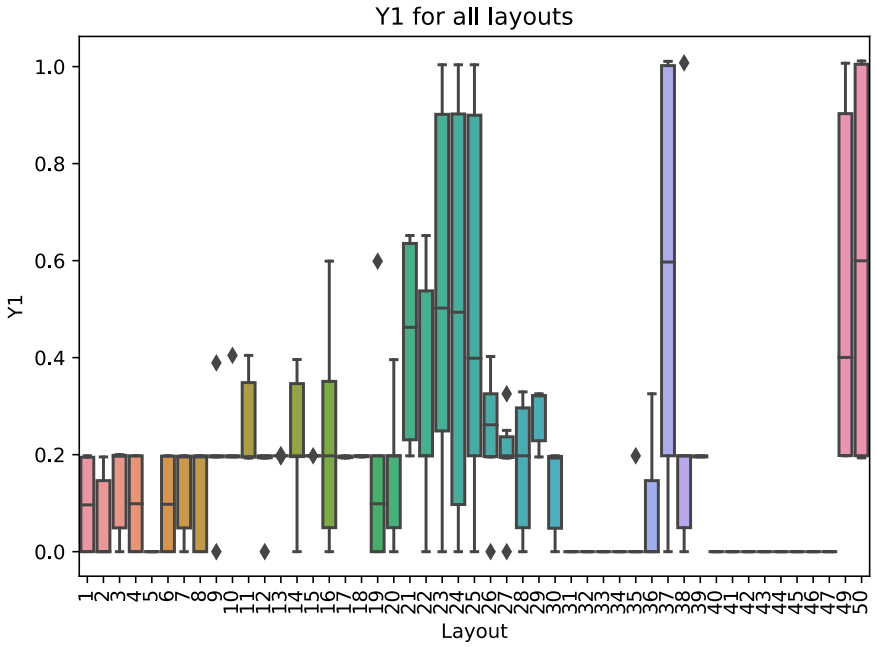
Appendix

A.1 Additional plots of the throughput variation

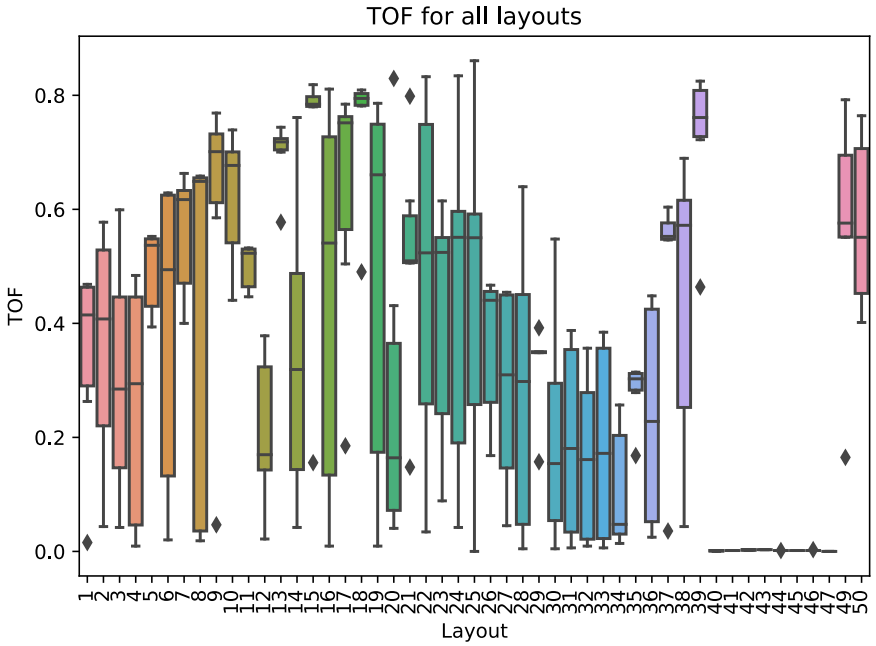
Additional plots of the throughput variation are presented in figure A.1.



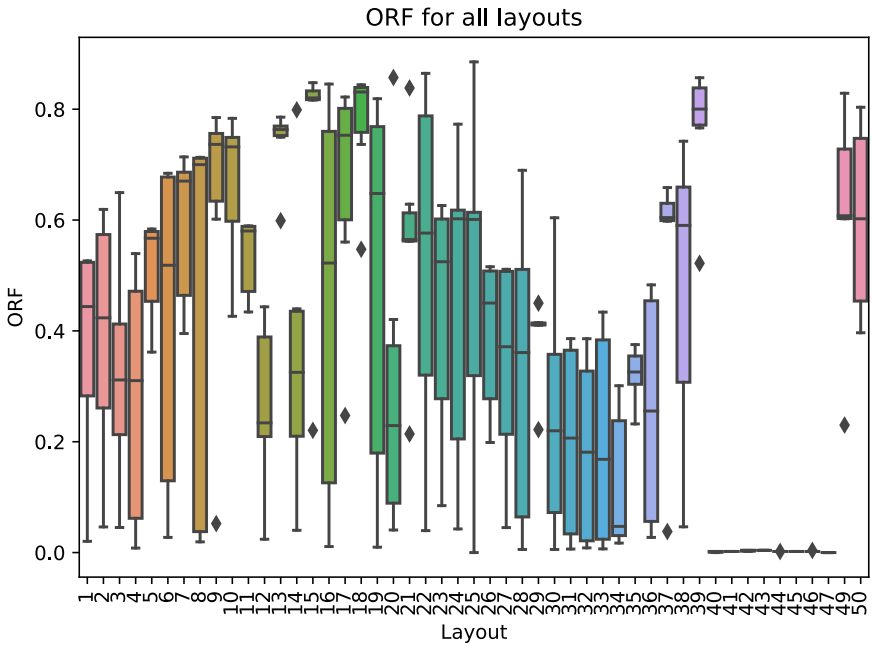
(a) The Y_0 target value as discussed in section 3.2.2.



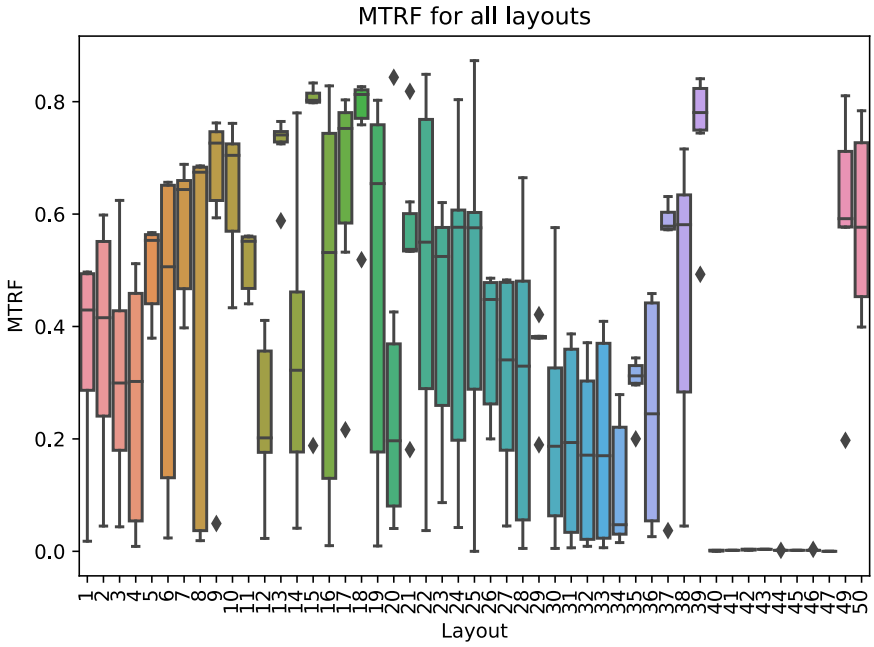
(b) The Y_1 target value as discussed in section 3.2.2.



(c) The TOF target value as discussed in section 3.2.3.2.



(d) The ORF target value as discussed in section 3.2.3.2.



(e) The MTRF target value as discussed in section 3.2.3.2.

Figure A.1: Displaying the variance in all of the layouts in the data set for the different target metrics. The whiskers has a maximum length of 1.5 times the interquartile range.

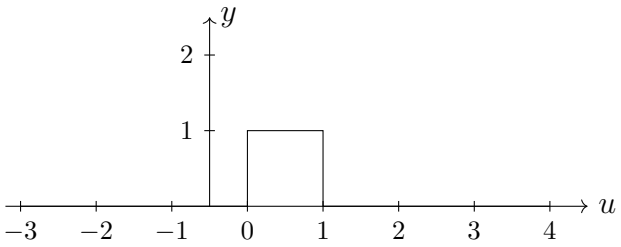


Figure A.2: The blending function $N_{3,1}$ over the interval $[-3, 4]$.

A.2 Derivation of the basis function for the uniform B-spline of degree three

To do the derivation of the basis function, we first define a knot vector from the knot values. Remember that the knot values are $\{t_0, t_1, \ldots, t_{n+k}\}$. Because $n = 3$ and $k = 4 \rightarrow n + k = 7$, we have a knot vector with eight knot values. Choose $T = [-3, -2, -1, 0, 1, 2, 3, 4]$. In the uniform case, we only want the spline to be defined in the interval $u \in [0, 1]$. Using this together with equation 2.2, we get

$$N_{i,1}(u) = \begin{cases} 1, & \text{if } i = 3, \\ 0, & \text{otherwise.} \end{cases} \tag{A.1}$$

In figure A.2 see the blending function, $N_{3,1}$. We chose this function because outside of the interval $[0, 1]$ the blending functions are equal to zero, which is a normalization which we want to get a uniform B-spline.

For $k = 2$, let us do the calculations. (Here we did derivation ourselves, but they have also done the derivations at Kollmorgen and in [1].) Start with $N_{0,2}$. Use equation 2.3.

$$N_{0,2} = \frac{u - t_0}{t_1 - t_0} N_{0,1}(u) + \frac{t_2 - u}{t_2 - t_1} N_{1,1}(u) \tag{A.2}$$

$$= \frac{u - (-3)}{(-2) - (-3)} \cdot 0 + \frac{(-1) - u}{(-1) - (-2)} \cdot 0 \tag{A.3}$$

$$= 0. \tag{A.4}$$

$N_{1,2}$, $N_{4,2}$ and $N_{5,2}$ is zero in the same way. However,

$$N_{2,2} = \frac{u - t_2}{t_3 - t_2} N_{2,1}(u) + \frac{t_4 - u}{t_4 - t_3} N_{3,1}(u) \tag{A.5}$$

$$= \frac{u - (-1)}{0 - (-1)} \cdot 0 + \frac{1 - u}{1 - 0} \cdot 1 \tag{A.6}$$

$$= 1 - u \tag{A.7}$$

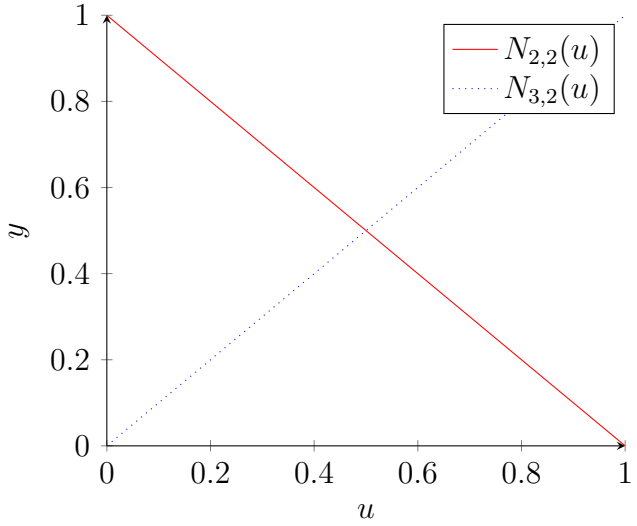


Figure A.3: The hat functions over the interval $[0, 1]$.

and

$$N_{3,2} = \frac{u - t_3}{t_4 - t_3} N_{3,1}(u) + \frac{t_5 - u}{t_5 - t_4} N_{4,1}(u) \tag{A.8}$$

$$= \frac{u - 0}{1 - 0} \cdot 1 + \frac{2 - u}{2 - 1} \cdot 0 \tag{A.9}$$

$$= u. \tag{A.10}$$

Our results are thereby

$$N_{i,2}(u) = \begin{cases} 1 - u & \text{if } i = 2, \\ u & \text{if } i = 3, \\ 0 & \text{else} \end{cases} \tag{A.11}$$

These are the well known *hat functions*. In figure A.3 see the blending functions over the interval $[0, 1]$. Over the rest of the interval, all blending functions are zero.

For $k = 3$ we quickly realize that $N_{0,3} = 0$ and $N_{4,3} = 0$. Then

$$N_{1,3} = \frac{u - t_1}{t_3 - t_1} N_{1,2}(u) + \frac{t_4 - u}{t_4 - t_2} N_{2,2}(u) \tag{A.12}$$

$$= \frac{u - (-2)}{0 - (-2)} \cdot 0 + \frac{1 - u}{1 - (-1)} \cdot (1 - u) \tag{A.13}$$

$$= \frac{u^2 - 2u + 1}{2}. \tag{A.14}$$

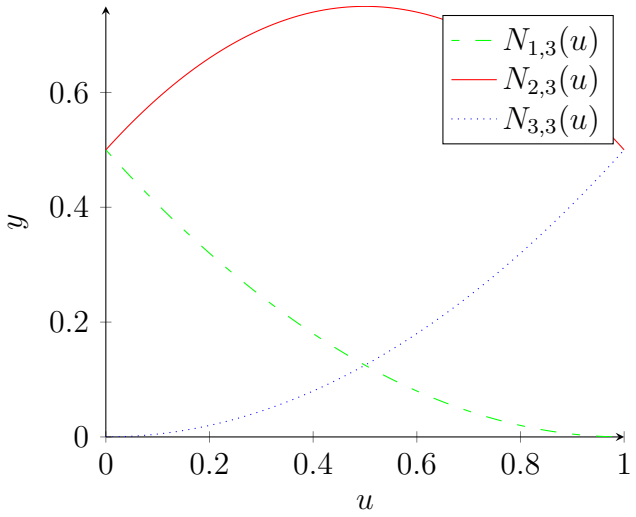


Figure A.4: The blending functions for $k = 3$ over the interval $[0, 1]$.

$$N_{2,3} = \frac{u - t_2}{t_4 - t_2} N_{2,2}(u) + \frac{t_5 - u}{t_5 - t_3} N_{3,2}(u) \tag{A.15}$$
$$= \frac{u - (-1)}{1 - (-1)} \cdot (1 - u) + \frac{2 - u}{2 - 0} \cdot u \tag{A.16}$$
$$= \frac{1 - u^2}{2} + \frac{2u - u^2}{2} \tag{A.17}$$
$$= \frac{-2u^2 + 2u + 1}{2}. \tag{A.18}$$

$$N_{3,3} = \frac{u - t_3}{t_5 - t_3} N_{3,2}(u) + \frac{t_6 - u}{t_6 - t_4} N_{4,2}(u) \tag{A.19}$$
$$= \frac{u - 0}{2 - 0} \cdot u + \frac{3 - u}{3 - 1} \cdot 0 \tag{A.20}$$
$$= \frac{u^2}{2}. \tag{A.21}$$

In figure A.4 see the blending functions over the interval $[0, 1]$.

Finally, $k = 4$.

$$N_{0,4} = \frac{u - t_0}{t_3 - t_0} N_{0,3}(u) + \frac{t_4 - u}{t_4 - t_1} N_{1,3}(u) \tag{A.22}$$
$$= \frac{u - (-3)}{0 - (-3)} \cdot 0 + \frac{1 - u}{1 - (-2)} \cdot \frac{u^2 - 2u + 1}{2} \tag{A.23}$$
$$= \frac{u^2 - 2u + 1 - u^3 + 2u^2 - u}{3 \cdot 2} \tag{A.24}$$
$$= \frac{-u^3 + 3u^2 - 3u + 1}{6}. \tag{A.25}$$

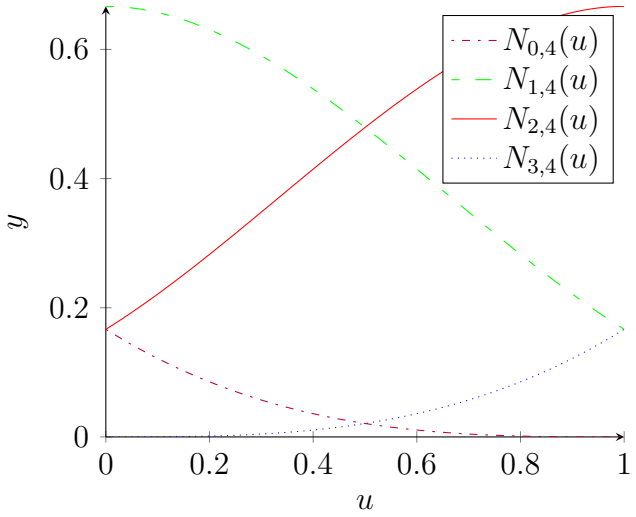


Figure A.5: The blending functions for $k = 4$ over the interval $[0, 1]$.

$$N_{1,4} = \frac{u - t_1}{t_4 - t_1} N_{1,3}(u) + \frac{t_5 - u}{t_5 - t_2} N_{2,3}(u) \tag{A.26}$$

$$= \frac{u - (-2)}{1 - (-2)} \cdot \frac{u^2 - 2u + 1}{2} + \frac{2 - u}{2 - (-1)} \cdot \frac{-2u^2 + 2u + 1}{2} \tag{A.27}$$

$$= \frac{u^3 - 2u^2 + u + 2u^2 - 4u + 2}{6} + \frac{-4u^2 + 4u + 2 + 2u^3 - 2u^2 - u}{6} \tag{A.28}$$

$$= \frac{3u^3 - 6u^2 + 4}{6}. \tag{A.29}$$

$$N_{2,4} = \frac{u - t_2}{t_5 - t_2} N_{2,3}(u) + \frac{t_6 - u}{t_6 - t_3} N_{3,3}(u) \tag{A.30}$$

$$= \frac{u - (-1)}{2 - (-1)} \cdot \frac{-2u^2 + 2u + 1}{2} + \frac{3 - u}{3 - 0} \cdot \frac{u^2}{2} \tag{A.31}$$

$$= \frac{-2u^3 + 2u^2 + u - 2u^2 + 2u + 1}{6} + \frac{3u^2 - u^3}{6} \tag{A.32}$$

$$= \frac{-3u^3 + 3u^2 + 3u + 1}{6}. \tag{A.33}$$

$$N_{3,4} = \frac{u - t_3}{t_6 - t_3} N_{3,3}(u) + \frac{t_7 - u}{t_7 - t_4} N_{4,3}(u) \tag{A.34}$$

$$= \frac{u - 0}{3 - 0} \cdot \frac{u^2}{2} + \frac{4 - u}{4 - 1} \cdot 0 \tag{A.35}$$

$$= \frac{u^3}{6}. \tag{A.36}$$

In figure A.5 see the blending functions over the interval $[0, 1]$.

With these derivations done we can finally write down the final matrix.

$$\mathbf{p}(u) = \begin{bmatrix} \vdots & \vdots & \vdots & \vdots \\ N_{0,4} & N_{1,4} & N_{2,4} & N_{3,4} \\ \vdots & \vdots & \vdots & \vdots \end{bmatrix} \begin{bmatrix} \mathbf{p}_{i-1} \\ \mathbf{p}_i \\ \mathbf{p}_{i+1} \\ \mathbf{p}_{i+2} \end{bmatrix} \tag{A.37}$$

$$= [u^3\, u^2\, u\, 1] \cdot \frac{1}{6} \cdot \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{p}_{i-1} \\ \mathbf{p}_i \\ \mathbf{p}_{i+1} \\ \mathbf{p}_{i+2} \end{bmatrix} \tag{A.38}$$

We see that with four control points then the spline $\mathbf{p}(u)$ is uniquely defined.

DEPARTMENT OF ELECTRICAL ENGINEERING
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY