CHALMERS
UNIVERSITY OF TECHNOLOGY

Optimization of resource allocation in air defence systems using graph neural networks

An NP-complete optimization problem

Master's thesis in Complex Adaptive Systems

SOFIA KÄLLHAMMER

DEPARTMENT OF SPACE, EARTH AND ENVIRONMENT

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2025
www.chalmers.se

MASTER'S THESIS 2025

Optimization of Resource Allocation in Air Defence Systems using Graph Neural Networks

An NP-complete optimization problem

SOFIA KÄLLHAMMER

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Space, Earth and Environment

CHALMERS UNIVERSITY OF TECHNOLOGY

Gothenburg, Sweden 2025

Optimization of Resource Allocation in Air Defence Systems using Graph Neural Networks
An NP-complete optimization problem
SOFIA KÄLLHAMMER

Supervisor: Håkan Warston, Saab Surveillance & Claes Andersson, Chalmers
Examiner: Claes Andersson, Department of Space, Earth and Environment

Master's Thesis 2025
Department of Space, Earth and Environment
Chalmers University of Technology
SE-412 96 Gothenburg
Sweden
Telephone + 46 31-772 1000

Gothenburg, Sweden 2025

Optimization of Resource Allocation in Air Defence Systems using Graph Neural Networks
An NP-complete optimization problem

SOFIA KÄLLHAMMER
Department of Space, Earth and Environment
Chalmers University of Technology

Abstract

Graph neural networks (GNN) represent a specialized class of artificial neural networks designed for operating on graph-structured data. This neural network model was first introduced by Gori et al. [1] in 2005, but has seen increasing popularity in recent years. In this master's thesis, it was investigated how the resource allocation in an air defence system could be optimized using GNNs. It is an NP-complete combinatorial optimization problem, where the number of combinations surpasses an exponential scaling with the number of objects. This project examines how the model performs, but also how the model performance and training time scales to the amount of data.

A GNN model with a graph convolutional operator is proposed, which is trained to predict the impact of each allocation and find the combination of allocations that maximizes the impact, without violating any capacity constraints for the defence resources. The findings suggests that the proposed model yields satisfactory results for problems with low computational complexity. Nevertheless, for practical application in real-world scenarios, the model necessitates further development for solving problems of increased complexity with high confidence.

Keywords: Graph neural network, machine learning, deep learning, NP, combinatorial optimization.

Acknowledgements

I would like to thank everyone who has helped me during this thesis project. Especially, I would like to thank my supervisor Håkan Warston at Saab Surveillance in Gothenburg for his guidance and feedback throughout this project.

I would also like to thank my examiner Claes Andersson at Chalmers for feedback and interesting ideas regarding the methodology.

Sofia Källhammer, Gothenburg, June 2025

List of Acronyms

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

ANN	Artificial Neural Network
BCE	Binary Cross-Entropy
CCE	Categorical Cross-Entropy
CNN	Convolutional Neural Network
GCN	Graph Convolutional Network
GNN	Graph Neural Network
MOE	Margin Of Error
NP	Nondeterministic Polynomial time
ReLU	Rectified Linear Unit
SGD	Stochastic Gradient Descent

Table of Content

List of figures

List of tables

1 Introduction

The development of defence systems has been of high importance for thousands of years, because of wars and conflicts around the world. However, the current global situation is increasing the interest in well-functioning defence systems. Over the past few years, the usage of drones in the military has increased rapidly [2], leading to higher complexity in conflicts, since the drones are both hard to detect and can be used on a much larger scale, compared to larger manned aircrafts. As attacking military units evolves to become faster, large-scale, and less predictable, defence systems must evolve to be able to meet more advanced adversaries.

Surveillance and defence systems could be used to protect and control any type of vital installation or land border. How the defence system is structured may vary and depends on situational factors. Previously, the systems were entirely manual, but now the trend is towards more automated systems [3].

Generally, in semi-automatic systems, sensor systems gather information about traffic and other activities in a limited air space around the vital installation or land border, to create a situational awareness. The situational awareness is used as a basis for the decision-making of how to distribute resources in the defence system in the most optimal way. The decision should be made in a short time frame, usually only a few fractions of a second. However, the number of alternatives could be very large. The number of combinations surpasses an exponential scaling with the number of targets. Hence, this problem is NP-complete and traditional searching algorithms, such as linear search and binary search [4] are not applicable within the time frame.

Typically, for NP-complete problems is that a solution to the problem can be validated quickly, but there is no known method for solving the problems efficiently [5]. A possible approach for solving NP-complete problems in a reasonable time is to use a pre-trained model. Then, most of the execution time is done before the model is used in a critical situation, enabling the model to solve problems fast and make decisions within the time frame. Artificial Neural Networks (ANN) [6] are a type of machine learning algorithm based on such approach and can possibly be used to solve complex optimization problems. One type of ANN that has become more

popular in recent years is Graph Neural Networks (GNN) [7], which uses graphs as input data. The graph data represents any data structure comprising of entities and their relationships.

Previous studies [8, 9, 10, 11, 12] have shown a promising progress in the usage of GNNs for solving a large variety of NP-complete optimization problems, such as the Travelling Salesman Problem (TSP), Maximum Independent Set (MIS) and Minimum Vertex Cover (MVC). However, the GNNs are not able to solve all these problems faultlessly yet, although some newer models can solve some problems with a very high accuracy. For instance, Li et al. [10] has developed a model that can solve 1,000 instances of the MIS problem. Indications suggest that GNNs have the potential to yield better results than existing suboptimal solutions. Hence, this study will investigate how GNNs can be used to optimize the distribution of resources in defence systems.

1.1 Problem description

A defence system with an arbitrary number of resources is used to protect a vital installation. Each of the resources has an operating distance and a maximum capacity. The maximum capacity specifies how many targets each resource can operate on simultaneously. An arbitrary number of threats are moving towards the vital installation. The resources should be distributed in the most optimal way to maximize the defence against threats. Prioritizing is based on threat level and the probability that a resource can hit the target. Only one resource should be assigned to each target. Redistribution of resources should be done dynamically when positions are changing. An illustration of a possible scenario is shown in Figure 1.1.

1.2 Aim

This study aims to investigate how GNNs can be used for combinatorial optimizations problems, where resources in a defence system should be distributed among the targets to protect a vital installation. The aim is to find a fast and accurate method, which in the long term can replace existing suboptimal methods, where resources are not used with maximum efficiency. The optimization of resource allocation has the potential to enable more efficient utilization of the resources within the defence system, resulting in positive outcomes in both an economic and environmental perspective.

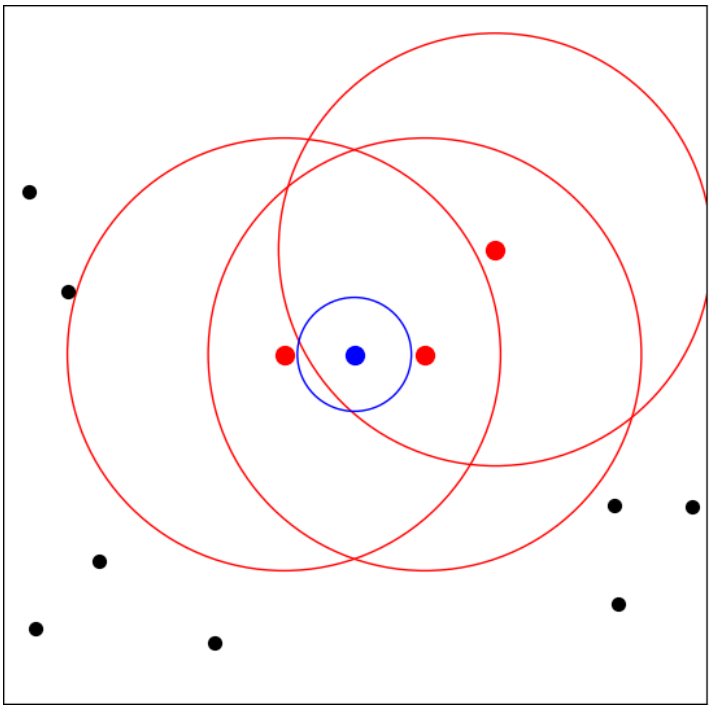

Figure 1.1: Illustration of a possible scenario with a vital installation (blue dot), three resources (red dots) and eight targets (black dots). When the targets are within the resources operating distances (red circles), the resources should be allocated to the targets. The goal is to stop the targets from reaching the area within the blue circle.

1.3 Research questions

The research questions to answer in this study are the following:

1. How does the model perform in scenarios with different complexities?
2. How does the model's performance and training time scale to the amount of data?
3. How does the parameter settings impact the model's performance?

1.4 Scope and delimitations

This thesis will focus on how the distribution of defence resources can be optimized using graph neural networks. The main part of this project is to design a GNN aimed for this combinatorial optimization problem. Decisions should be made based on the current situation in every timestep and the model will neither take the situation in the previous timestep into account, nor future possible scenarios.

Optimal positioning of the defence units is not a part of this project. Instead, the defence units have predefined and fixed positions. The problem is also limited into a two-dimensional space, where the vertical positions are disregarded.

2 Theory

In this chapter, the necessary theory regarding graph neural networks and NP-complete problems is introduced, including definitions of NP-completeness and NP-hardness, graph neural networks and training methods for neural networks. Some findings from previous studies are also mentioned.

2.1 Definition of NP-completeness and NP-hardness

NP stands for nondeterministic polynomial time and is a class of problems where a potential solution for a problem could be verified in polynomial time, but there is no efficient way to find such a solution [5]. If all problems in NP could be reduced to another problem in polynomial time, that other problem is NP-hard, implying it is at least as hard to solve than any problem in NP.

NP-complete problems belong to both NP and NP-hard and can be described as the hardest problems in NP. If an efficient algorithm were discovered for solving one single NP-complete problem, it could be leveraged to solve all other NP-complete problems due to their polynomial-time reducibility. All NP-complete problems are at least as hard as any other NP-complete problem, and they have approximately exponential growth.

2.1.1 Previous studies on NP-hard problems

Finding methods for effectively solving NP-hard problems has been a field of high interest for many years. Various methods, such as approximation algorithms [13] and genetic algorithms [14] has been used, trying to solve these problems. While approximation algorithms can’t give the exact solution, genetic algorithms may yield more satisfying results, although the algorithms are time-consuming.

In recent years, methods based on Graph Neural Networks (GNN) has increased in popularity [8, 10, 15]. Several studies have led to pleasing results for different famous NP-hard problems. Li et al. [10] have proposed a model that solves 1,000 instances of the Maximum Independent Set problem and Liu et al. [15] have designed a GNN model that can solve the Max Cut problem

with a relative error below 1%. Further, Liu et al. [8] have presented a new and innovative model, called AutoGNP, which uses search algorithms to find the best GNN architecture for a given NP-hard problem.

2.2 Computational complexity

Given a set of objects of two different types, the number of combinations for how the objects could be arranged in pairs, where each pair consists of one object of each type, is defined as the number of permutations [16]:

$$\mathrm{P} = \begin{cases} \dfrac{N!}{(N-M)!} & ,N \geq M \\ \dfrac{M!}{(M-N)!} & ,N < M \end{cases} \tag{2.1}$$

In equation (2.1), N is the number of objects of type 1 and M is the number of objects of type 2. The formula in equation (2.1) describes the lower bound of the number of combinations of pairs of different types. If the objects of any type could be paired with more than one object of the other type, the number of combinations is increasing.

2.3 Graph Neural Networks

Graph neural networks (GNN) has been developed in over 15 years [7, 17]. Unique for GNNs is that they operate on data presented as graphs. All data consisting of a set of objects and relationships between them can be described as graphs, where the objects correspond to the graph's nodes and their relationships are described by the edges. The node and edge attributes are stored as features for each node and edge. The graphs could be either undirected or directed, where the former one means that information flows in any direction, while the latter one has edges with a source node, v_{src}, and a destination node, v_{dst}. Then, information can only flow from v_{src} to v_{dst}. Graphs could also be classified as either homogenous or heterogenous. Homogenous graphs have nodes and edges of same type, while heterogenous graphs have nodes and edges of different types. The types of the nodes and edges have a significant impact of how the GNN should be implemented.

Usually, graph prediction tasks for GNNs are either graph-level tasks, node-level tasks or edge-level tasks. A typical graph-level task is to classify an entire graph. Common node-level tasks are node classification and node clustering. A potential edge-level task could be predicting the relationships between nodes.

2.3.1 GNN architecture

A basic GNN architecture consist of several layers, which uses convolutional and recurrent operators to propagate the information forward in the network [11, 17]. The convolution operator is the most common propagation operator in GNN models, and it is used to extract features from the graph and convert convolutions in a non-graph domain to graph domain [18]. Convolutions on graphs are described in more detail in section 2.3.2.1. The recurrent operator operates similarly to the convolution operator. The difference is that the recurrent operators use the same weights in all layers, while the convolution operators have different weights in each layer. Figure 2.1 illustrates a simple structure of a GNN with an input of size three, two hidden layers with eight and four neurons, and an output layer of size two.

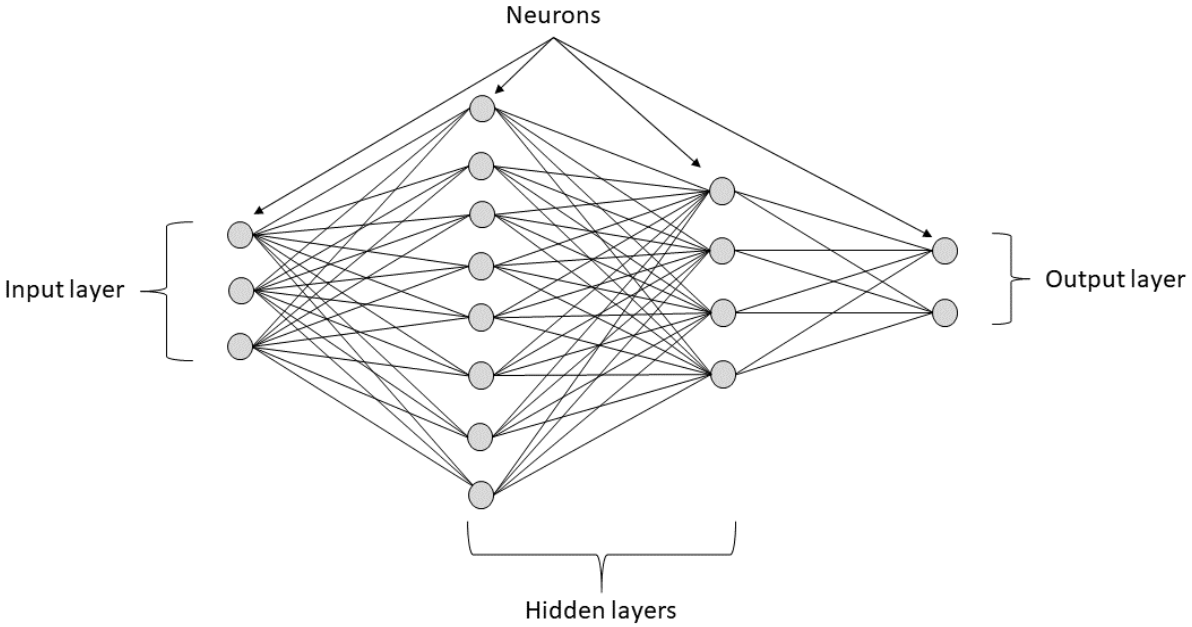

Figure 2.1: An example of a GNN architecture. In this example an input feature vector of size three is fed through a GNN with two hidden layers with eight and four neurons in respective layer. The output is a feature vector of size two, in other words, the size of the output layer.

In the forward propagation, each of the input nodes are fed through the first layer in the network. To learn the model's complex non-linear patterns, a non-linear activation function could be added to the output before feeding it through the next layer. Activation functions are described more in section 2.3.1.1. The procedure is repeated until reaching the output layer, where the final output is calculated. The number of neurons in the output layer should be the same as the number of classes. The forward propagation is illustrated in Figure 2.2.

A pooling layer could be added at the end of the network, which uses an aggregation function to gather and generalize information over several nodes. It is essential in the reduction of dimensions, extraction of important substructures and embedding information into graph representations. In general, there are two types of pooling on graphs; global pooling and hierarchical pooling [19]. Global pooling reduces the graph to a single vector, while hierarchical pooling reduces the input graph into a smaller graph. Common aggregation functions in pooling layers are max, mean and sum [17].

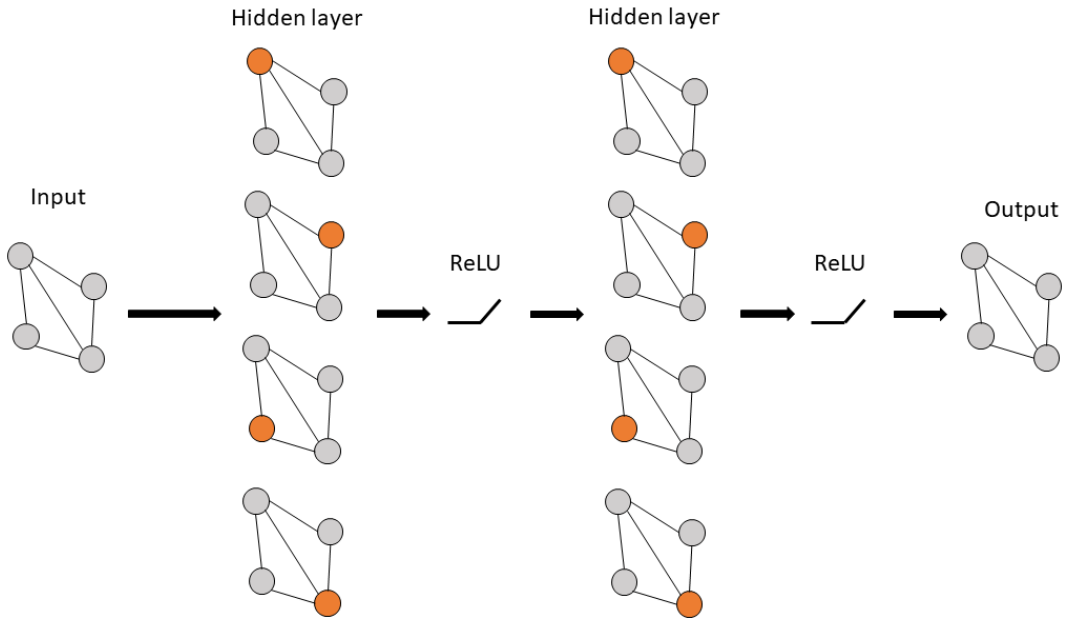

Figure 2.2: Forward propagation in a GNN. The model takes the features of each node as input, and processes it to create the output. Here, ReLU activation is added to the output after each hidden layer.

2.3.1.1 Activation functions

Both hidden layers and output layers uses activation functions, such as Rectified Linear Unit (ReLU), sigmoid and SoftMax, to decide whether a node should be activated or not [20]. Non-linear activation functions are necessary for learning the network complex non-linear patterns in the data. ReLU is a non-linear activation function which should only be used in the hidden layers. When using ReLU, a neuron will only be activated if the output is strictly positive. The ReLU activation function is defined in equation (2.2):

$$\sigma(x) = max(x, 0) \tag{2.2}$$

Another non-linear activation function is sigmoid, which transforms the output values to values between 0 and 1. It is commonly used in binary classification. The function takes a smooth S-shaped form and is continuously differentiable. The sigmoid function is defined in equation (2.3):

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.3}$$

SoftMax activation function can be described as a combination of several sigmoid functions. The value from each sigmoid function can be treated as a probability for a certain class. Hence, SoftMax is usually used for multi-classification tasks, where the number of sigmoid functions used in the SoftMax function is equal to the number of classes. The SoftMax function is defined in equation (2.4):

$$\sigma(\boldsymbol{z})_j = \frac{e^{z_j}}{\sum_{n=1}^{N} e^{z_n}}, j = 1, \dots, N \tag{2.4}$$

Here, $\boldsymbol{z}$ is the output vector from the neural network and N is the number of classes.

2.3.2 Graph Convolutional Networks

Graph convolutional networks (GCN) is a type of GNN. It works similarly to a Convolutional Neural Network (CNN), but can be used on graph data [21]. GCN uses both the node features

and the nodes locality in the graph, making it suitable for graph-structured data where the connectivity is important.

In a GCN, the graph information is propagated forward using the propagation rule described in equation (2.5), where the matrix of activations in the $(l+1)^{th}$ layer, $H^{(l+1)}$, is calculated, based on the matrix of activation functions in the previous layer, $H^{(l)} \in \mathbf{R}^{N \times D}$, together with different weight matrices.

$$H^{(l+1)} = \sigma\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}\right) \tag{2.5}$$

Here, $\tilde{A} = A + I_N$ is the adjacency matrix describing the graph's connectivity in an undirected graph and I_N is the identity matrix. $D_{ii} = \sum_j \tilde{A}_{ij}$ and $W^{(l)}$ are trainable weight matrices which are specific for each layer and σ is the activation function used between the layers.

2.3.2.1 Convolution on graphs

Convolution on images or other grid-like structures has proved successful for extracting features. Generally, convolutions aggregate information about the neighbouring nodes features and combine them with the node's own features, by applying a convolutional kernel on each pixel [18]. The grid-like structure makes the number of nodes or pixels used in each convolution constant.

An image can easily be converted to a graph, where each pixel in the image can be treated as a node in the graph. However, most real-world graphs do not have the same grid-structure as for images. Although images and graphs have some similarities, graph's irregular structure and higher complexity makes it difficult to apply the same convolutional operator on graphs as for images [22]. Further, the pixels in an image have an inherent ordering, where each pixel is determined by its position in the grid. Nodes in a graph does not have such ordering. Instead of applying the convolution operator on a sub-grid of a fixed size, the number of neighbours used in each convolution on a graph may vary. The neighbourhood used for convolution in a standard CNN and a GCN is shown in Figure 2.3.

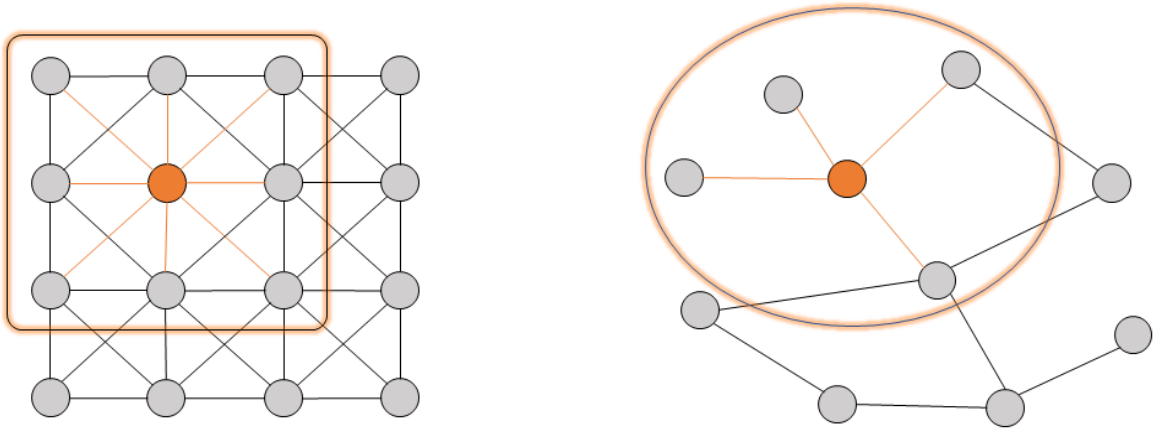

Figure 2.3: Neighbourhood used for convolution in CNN (left) and GCN (right). The neighbourhood of a node in a CNN is always grid-structured and has a size of eight nodes, while the neighbourhood of a node in a graph can vary both in size and structure.

2.3.3 Message passing in graph neural networks

To share information between entities in the graph, message passing through layers is necessary, allowing the model to learn about the graph connectivity and improving the predictions [17]. Traditionally, message passing occurs between nodes along the edges, but it could also occur between edges. The message passing is a three-steps procedure:

1. Each node in the graph gathers all the node embeddings or messages from its neighbouring nodes and uses it to compute a new message.
2. The messages are aggregated via an aggregation function, e.g., max, mean or sum.
3. All nodes update their attributes. The current values and the aggregated messages are fed through an update function, i.e., a trained neural network.

In each message passing GNN layer, a node can share information with nodes one step away from itself. After N layers, the node can share information with nodes N steps away. By using a sufficient number of layers, information could be shared over the entire graph.

2.4 Training methods for graph neural networks

Generally, GNNs are trained using message passing through the network, loss computation, backpropagation and computation of gradients. The training continues during a predefined number of epochs, or until the loss has converged. The latter method, called early stopping, is

preferred, since it prevents overfitting by stopping the training when the model starts to memorize the training data. In Algorithm 2.1, a simple GNN training algorithm with backpropagation is shown.

In addition to the general steps, different training approaches can be used. The choice of training method depends on the learning goal and the content in the training data. In this section, some common approaches for training GNNs are presented.

```
begin
    while convergence not reached do
        for epoch = 1, 2, …, number of epochs do
            for data in training data loader do
                Clear gradients
                Propagate data forward
                Calculate loss
                Propagate loss backward
                Update model parameters
            end for
            Check if convergence is reached
        end for
    end while
end
```

Algorithm 2.1: A simple GNN training algorithm with backpropagation.

2.4.1 Backpropagation

Backpropagation is essential in the use of supervised learning [23]. The method is used for calculating how changes in the network parameters affects the model's accuracy. After feeding the input forward through the whole network, the loss is calculated. Loss functions are described more in section 2.4.5. Afterwards, the gradients are calculated by propagating the loss backward in the network. The gradients are used to determine how much and in which direction each variable should be changed. The variables with the worst impact on the loss are modified most. In equation (2.6), the backpropagation rule is described [6].

$$\delta_j^{(l-1)} = \sum_i \delta_i^{(l)} w_{ij}^{(l)} \, \sigma'\left(b_j^{(l-1)}\right) \tag{2.6}$$

Here, $\delta_j^{(l-1)}$ is the error propagated to the j^{th} neuron in the $(l-1)^{th}$ layer, $\delta_i^{(l)}$ is the error from the i^{th} neuron in the l^{th} layer and $w_{ij}^{(l)}$ is the weight between the two neurons. $\sigma'\left(b_j^{(l-1)}\right)$ is the derivative of the activation function with respect to the local field of the j^{th} neuron in the $(l-1)^{th}$ layer.

2.4.2 Supervised learning and Unsupervised learning

The main difference between supervised and unsupervised learning is that supervised learning uses labelled data, while unsupervised learning uses unlabelled data [24]. The labelled data used in supervised learning is useful for learning the model classification or regression tasks. Further, the labelled data helps the model to measure its accuracy. In supervised learning, the model makes predictions continuously during training and adjusting its parameters based on the outcome.

In the absence of ground-truth labels, unsupervised learning is not suitable for classification or regression. However, unsupervised learning can be used to discover hidden patterns in large datasets, such as clusters, relationships, which can be performed without supervision.

Previous studies related to supervised and unsupervised learning in GNNs are found in [8, 9, 15].

2.4.3 Reinforcement learning

Reinforcement learning is a training method used on unlabelled data, independent from external instructions [6]. Unlike supervised and unsupervised learning, the model learns by trial and error, instead of extracting information.

The reinforcement learning process can be described as a relationship between the environment, an agent, and a goal. This is called the Markov decision process [25]. In this process, the current state is given from the environment, which the agent uses to determine an action. If the action gives an output that relates to the goal, the environment assigns a reward

to the output. Otherwise, a penalty is assigned. The model learns from the rewards and penalties which actions should be taken in which state. Further, the action yields a new state, which is given to the agent. This process is repeated until the training terminates.

Reinforcement learning is not used in this project, but could potentially be used as an alternative method. This method has also proved to be useful in other graph-related problems. Previous studies related to reinforcement learning for GNNs are found in [26, 27, 28].

2.4.4 Stochastic gradient descent

Stochastic gradient descent (SGD) is an algorithm used for optimizing in machine learning models [6]. The SGD algorithm adjusts the weights, w and thresholds, ϑ in the hidden neurons, according to equation (2.7) and (2.8).

$$\delta w_{mn} = \eta \delta_m^{(\mu)} x_n^{(\mu)} \tag{2.7}$$

$$\vartheta_m = -\eta \delta_m^{(\mu)} \tag{2.8}$$

In equation (2.7) and (2.8), η is the is the learning rate and $x_n^{(\mu)}$ is the input pattern. The errors, $\delta_m^{(\mu)}$, determining how the weights and thresholds should be changed, are achieved from the forward propagation, defined in equation (2.9), and backward propagation, described in section 2.4.1.

$$V_j^{(l)} = \sigma\left(\sum_k w_{jk}^{(l)} V_k^{(l-1)} - \vartheta_j^{(l)}\right) \tag{2.9}$$

Here, $V_j^{(l)}$ is the state variable for the j^{th} neuron in the l^{th} layer, $w_{jk}^{(l)}$ is the weights connecting to the neurons on the left side, $\vartheta_j^{(l)}$ is the threshold for the current neuron and σ is the activation function.

The forward propagation and backpropagation are combined with a stochastic update of weights and thresholds, where only one random data point or a random set of data points is used for computing the gradients. The stochastic behaviour reduces the risk of getting stuck in

a local minimum. During training, the average weight increment is reduced, but the direction of the weight update in each update step fluctuates. The complete SGD algorithm is described in Algorithm 2.2.

begin

Initialise random weights $w_{mn}^{(l)}$, set threshold $\vartheta_m^{(l)} = 0$

for $epoch = 1, \dots, e_{max}$ **do**

Choose a random value μ

Apply input $\boldsymbol{x}^{(\mu)}$ to the input layer $\boldsymbol{V}^{(0)}$

for $l = 1, \dots, L$ **do**

forward propagation: $V_j^{(l)} = \sigma(\sum_k w_{jk}^{(l)} V_k^{(l-1)} - \vartheta_j^{(l)})$

end for

compute errors $\delta_i^{(L)} = \sigma'(b_i^{(L)})(t_i - V_i^{(L)})$

for $l = L, \dots, 2$ **do**

backpropagation $\delta_j^{(l-1)} = \sum_i \delta_i^{(l)} w_{ij}^{(l)} \sigma'(b_j^{(l-1)})$

end for

for $l = 1, \dots, L$ **do**

update weights $w_{mn}^{(l)} \mathrel{+}= \eta \delta_m^{(l)} V_n^{(l-1)}$

update thresholds $\vartheta_m^{(l)} \mathrel{-}= \eta \delta_m^{(l)}$

end for

end for

end

Algorithm 2.2: Stochastic gradient descent [6]. In this algorithm, μ is a random value determining the datapoint used for computing the gradients, $\boldsymbol{V}^{(l)}$ is the l^{th} layer, σ is the activation function, t_i is the i^{th} element in target vector $\boldsymbol{t}$ and $\boldsymbol{\delta}$ is the error.

2.4.5 Loss function

After feeding the input data through the network, the loss is computed by using a loss function. This function is designed based on the type of prediction task [11]. The design of the loss

function is crucial for learning the model the right things. The loss is used to update the network parameters during training.

2.4.5.1 Categorical Cross-Entropy loss

Categorical Cross-Entropy (CCE) is a differentiable loss function used in multi-class classification problems [29]. It measures how well the predicted probability for each class complies with the actual class labels. CCE works similarly to Binary Cross-Entropy (BCE), but operates on classification problems with more than two classes, instead of only class 0 and 1. Since CCE is differentiable, it is particularly useful in optimization algorithms with gradient descent. For a set of *N* predictions with *C* different classes, CCE is defined in equation (2.10):

$$L = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{C} y_{i,j}\log\left(p_{i,j}\right) \tag{2.10}$$

Here, $y_{i,j}$ is the ground-truth labels and $p_{i,j}$ is the predicted probability of being in class *j* for the i^{th} prediction.

2.4.6 Batching and sampling on graphs

A common approach when training neural networks is using mini-batches with a constant batch size of the training data. However, this requires that the datapoints have the same dimensions, causing problems when training on graph data with varying number of nodes and edges [17]. A possible solution is to sample a fixed number of nodes or edges from the graphs to use in training [11]. This is a common operation on large graphs. The nodes and edges could either be selected randomly or belong to a connected subset of the graph. Depending on the importance of having the complete graph structure in training, this operation might either be effective or changing the graph structure too much. Thus, the choice of sampling method is important.

3 Methodology

In this chapter, the different steps in the methodology are explained. Important assumptions and simplifications are also mentioned. Finally, it is described how the model's performance is tested.

3.1 Threat assessment

A ranking of the targets based on threat level is done, where the most threatening one is given rank 1 and the least threatening one is given rank N_{target}, where N_{target} is the number of targets. How the ranking is done depends on the specific task of the defence system. The prioritization of targets is done based on equation (3.1).

$$\frac{1}{rank + 1} \tag{3.1}$$

The probability that a specific resource hits the target is specified by the variable P_{kill}. The expected impact given when a resource is assigned to a target is given by equation (3.2).

$$P_{kill} \cdot \frac{1}{rank + 1} \tag{3.2}$$

3.2 Problem definition

The problem is treated as a multiclass classification problem. For each target, the goal is to assign a class label in the range $[0, n_{resources}]$, specifying which resource should be allocated to each target. Targets with class 0 has no resource allocated to them.

3.2.1 Datasets

Several datasets were used, representing two different systems. The first system consists of three resources and eight targets, while the second system consists of five resources and 20 targets. Six different datasets on the system with dimension ($n_{resources} = 3, n_{targets} = 8$) were used, containing different amounts of data. The datasets are fully independent from each

other. Further, an aggregation of all datasets was also used, allowing testing of the model on an even larger dataset. For the ($n_{resources} = 5, n_{targets} = 20$) system, four different datasets were used.

In all datasets, each datapoint consists of a biadjacency matrix of size ($n_{resources}, n_{targets}$). The matrices are filled with values from equation (3.2), corresponding to the edge values between the target nodes and the resource nodes. The ground truth labels are represented as a list, where each element in the list corresponds to a target node and has a label representing the class.

All datasets are based on randomly simulated data and the dataset generation was not within the scope of this project.

3.2.1.1 Pre-processing of data

A large fraction of the datapoints consists of matrices with only zeros, meaning no target is in any resource's operating distance. To reduce the training time without reducing the training quality, all fully zero-filled matrices, except from one in each dataset, were removed. After reduction, 55.3% of the data were left in the first six datasets of the smaller system. On the rest of the datasets, the reduction was much smaller. After reduction, the dataset is shuffled and divided into training, validation, and test sets with the split ratio 80:10:10. The size of each dataset and the amount of data in the training sets are presented in Table 3.1.

Dataset	$n_{resources}$	$n_{targets}$	Size after reduction	Training set size
Small system 1	3	8	18 212	14 569
Small system 2	3	8	36 203	28 962
Small system 3	3	8	54 465	43 572
Small system 4	3	8	72 447	57 957
Small system 5	3	8	90 496	72 396
Small system 6	3	8	108 846	87 076
Small system, total	3	8	380 664	304 531
Large system 1	5	20	45 602	36 482
Large system 2	5	20	91 205	72 964
Large system 3	5	20	273 705	218 964
Large system 4	5	20	661 387	529 109

Table 3.1: Size of all datasets after reduction of the number of datapoints. All removed datapoints are completely zero-filled and does not affect the model's performance.

3.2.1.2 Data distribution

In Figure 3.1, the distribution of datapoints with different number of targets existing within any operating distance. The distribution is based on datasets corresponding to the smaller system with three resources and eight targets. In Figure 3.2, the corresponding distribution for the datasets based on the larger system is shown. The distribution in the smaller system has a small variation, where datapoints with seven targets are marginally more represented, while datapoints with eight targets are marginally less represented. However, the distribution in the larger system is more uneven, with a variation between 1.2% and 9%.

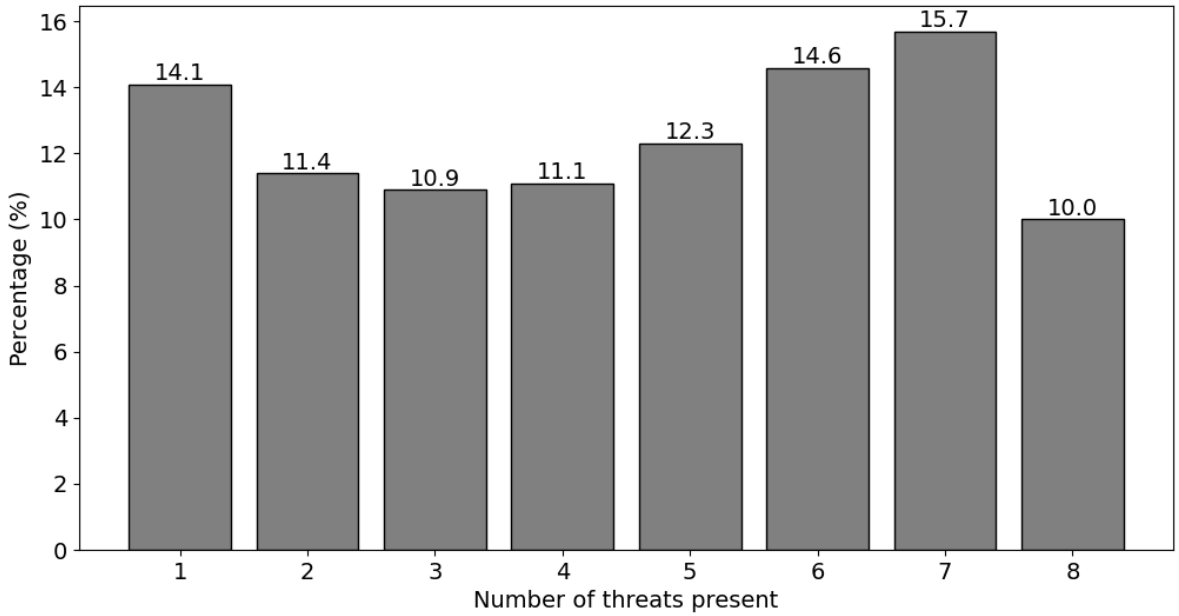

Figure 3.1: Average distribution of the fraction of datapoints with different numbers of targets in the smaller system. The distribution shown here is after the data reduction. Also note that there is only one datapoint with no targets in each dataset.

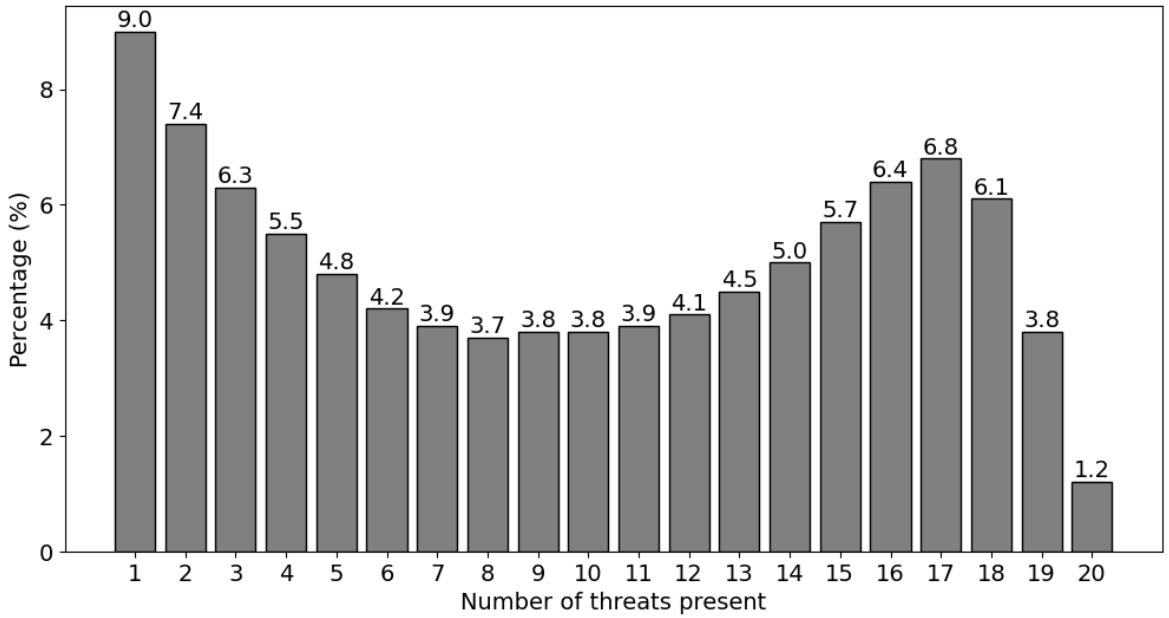

Figure 3.2: Distribution of the fraction of datapoints with different numbers of targets in the larger system. The distribution shown here is after the data reduction. Also note that there is only one datapoint with no targets in each dataset. The distribution is equal for all datasets on the larger system.

3.2.2 Constraints

The maximum degree of each resource node is determined by its maximum capacity. For both systems, the maximum capacity for each resource is set to 4. Thus, a resource node is not allowed to be allocated to more than four targets. The maximum degree of each target node is 1, meaning only one resource should be allocated to each target.

3.3 Graph data representation

The relationships between the targets and resources are represented as a graph with a set of nodes and edges. Each defence resource and target corresponds to a node with several feature values. For target nodes, the values correspond to the edge weights between the target node and all resource nodes. For resource nodes, the feature values are set to the mean of all edges connected to the node. The number of feature values for each node is equal to the number of resources.

The edges are also represented in two list, edge index and edge attribute. The former one corresponds to the indices of the nodes connected by each edge. The latter one corresponds to the edge weights. The edge weights are determined by equation (3.2).

All information about the graph is stored in a data object. The data object holds a number of attributes, presented in Table 3.2.

Attribute	Description	Size
x	Node feature matrix	$n_{targets} + n_{resources}) \times n_{resources}$
edge index	List of the nodes each edge is connected to	$2 \times (n_{targets} \cdot n_{resources})$
edge attribute	Edge weights	$(n_{targets} \cdot n_{resources})$
y	Ground truth class labels showing which resource should be allocated to which target, or if no resource should be allocated	$n_{targets}$

Table 3.2: Data attributes stored in the data objects, including a description of the attribute and its size.

3.4 Model architecture

The model consists of two layers with the torch-geometric graph convolutional operator GCNConv [21] and one fully connected linear layer. ReLU activation function and dropout is applied between the layers. The number of input neurons and output neurons is the same as the number of resources. To prevent overfitting, a small dropout fraction is used, causing a randomly selected set of neurons and their connections to be dropped during training [30]. This method adds some random noise to the data and prevents the model to adapt to the training data too much. In Figure 3.3, the used model architecture is shown. The parameters used in the model are presented in Table 3.3.

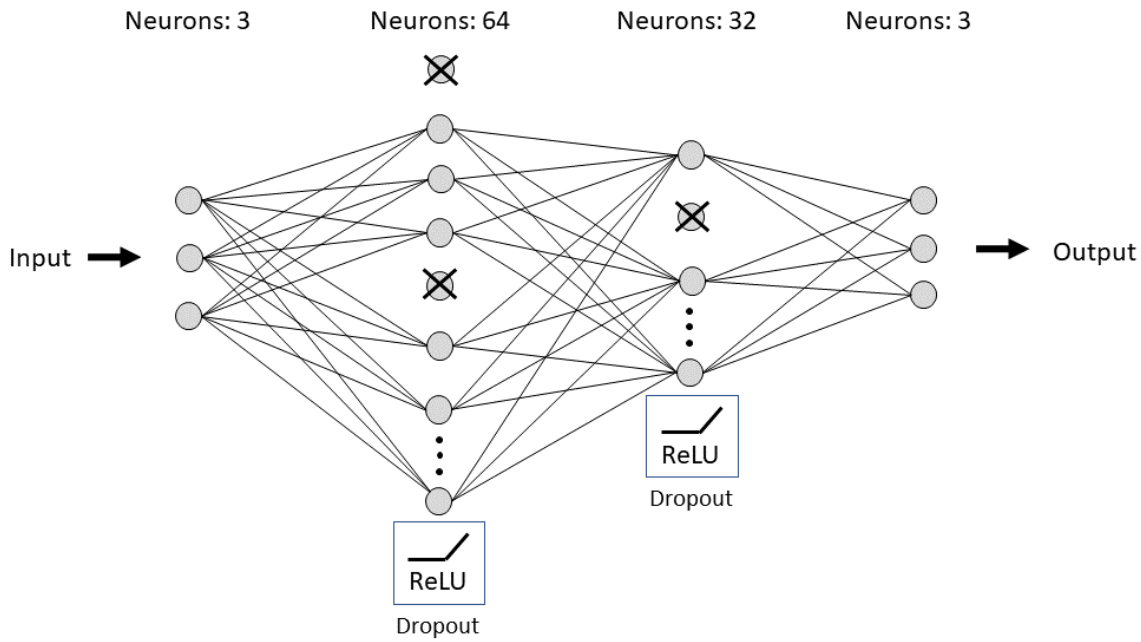

Figure 3.3: Model architecture with three input neurons, 64 and 32 hidden neurons and three output neurons. ReLU activation function and dropout is applied to the output after each hidden layer. Dropped neurons are crossed out. When training on the dataset with the large system, five input and output neurons are used instead.

Parameter	Value
Number of graph convolution layers	2
Number of linear layers	1
Input neurons	$n_{resources}$
Hidden neurons	64
Output neurons	$n_{resources}$
Dropout	0.15

Table 3.3: Model parameters describing the GNN architecture.

3.4.1 Post-processing of the model's output

The output from the model is a matrix of size $(n_{resources} + n_{targets}) \times n_{resources}$, where the elements in the matrix corresponds to the importance of each edge. The output matrix consists of both positive and negative values. Negative values mean the edge is not optimal and edges corresponding to large positive values are more likely to be a part of the optimal solution.

Only edges between a resource and a target are of interest. Hence, the first $n_{resources}$ rows are filtered out from the output matrix. Since the number of classes are $n_{resources} + 1$, including the 0 class, a column filled with zeros is added first in the output matrix.

The predicted best solution is determined by, for each row in the matrix, choosing the index of the column with the maximum value. If some class index is assigned to more targets than allowed by the maximum capacity, the targets giving the lowest values are instead assigned class 0.

3.5 Training and evaluation

The training of the network is based on supervised learning, since it is suitable for classification problems and useful when labelled data is available. For each epoch, the training data is shuffled and fed into the network. The training loss is calculated for each batch and is propagated backward in the network to calculate the gradients. The gradients determine how each variable in the model should be changed. The learning rate controls how much the model should change. The learning rate is gradually reduced with a predefined scheduler factor when the validation loss has stopped improving during the training to adapt finetuning later in the

training process. The number of epochs the model should wait before reducing the learning rate is determined by the scheduler patience.

The gradients are reset before feeding the next batch of data. The loss for each batch in the training data is summed up to calculate the total training loss for the epoch.

The performance is validated after each training epoch, by calculating the loss on the validation set. The validation loss for each epoch is compared by the best achieved validation loss. If a new minimum in the loss is reached, the current model is saved. The patience determines how many epochs the training should continue if no improvements has occurred in the validation loss. If no improvement in the validation loss has occured during a certain number of epochs, the training stops and the overfitted model is replaced with the best model. The training and evaluation algorithm used are shown in Algorithm 3.1.

A batch size of 1 is used in both the training and the validation, meaning the whole graph in each data point is used and each graph is processed independently. By feeding each datapoint separately, it is easier to keep track of the capacity constraints. All training parameters are presented in Table 3.4.

Parameter	Value/type
Optimizer	SGD
Learning rate	0.005
Scheduler patience	5
Scheduler factor	0.1
Weight decay	$5 \cdot e^{-4}$
Maximum number of epochs	1000
Stop training patience	10
Tolerance	e^{-5}
Penalty factor	0.2

Table 3.4: Training parameters.

```
begin
    for epoch = 1, 2, …, number of epochs do
        Set the model in training mode
        Shuffle the training data
        for training data in training data loader do
            Clear gradients
            Propagate training data forward
            Calculate training loss
            Propagate loss backward
            Update model parameters
        end for
        Set the model in training mode
        Disable gradient computation
        for validation data in validation data loader do
            Propagate validation data forward
            Calculate validation loss
        end for
        Adjust the learning rate if reaching a plateau
        Check is convergence in validation loss is reached
        if convergence is reached do
            Stop training
            break
        end if
    end for
    Save model with lowest loss
end
```

Algorithm 3.1: Training and evaluation algorithm.

3.5.1 Definition of loss function

A customized loss function is used. Since the class 0 is assigned to some targets when the model tries to exceed the maximum capacity, the model possibly learns to assign class 0 too often. To avoid this, the loss function adds an extra penalty to predictions of class 0 where the actual class is non-zero. First, CCE-loss is calculated using the PyTorch loss function CrossEntropyLoss with mean reduction [31], which combines a logarithmic SoftMax layer with negative log likelihood loss.

The number of violations in the prediction is calculated by counting the number of false negative predictions. Predictions with violations are penalized with a higher loss, depending on the number of violations and a penalty factor.

The customized loss function used in training is described in equation (3.3):

$$loss = CCE + \lambda \cdot {n_{violations}}^2 \tag{3.3}$$

Here, λ is the penalty factor.

3.6 Evaluation metrics

The model's performance is tested using different measurements. The first measurement, presented in equation (3.4), measures in how many predictions are identical with the ground-truth vector. The second one measures the fraction of individual targets in all datapoints which are assigned the correct class, according to equation (3.5).

$$\text{Accuracy} = \frac{\text{number of fully correct predictions}}{\text{number of predictions}} \tag{3.4}$$

$$\text{Elementwise accuracy} = \frac{\text{number of correctly predicted items}}{\text{number of predictions} \cdot \text{size of predictions}} \tag{3.5}$$

Additionally, precision, recall and F1-score is calculated for each of the classes using the metrics in equation (3.6 - 3.8). Precision refers to the fraction of true positive predictions in a set of all positive predictions of a given class. Recall refers to the proportion of true positives

in the set of both true positive and false negative predictions. F1-score is a balanced combination of both precision and recall.

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{3.6}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{3.7}$$

$$\text{F1 - score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{3.8}$$

In equation (3.6), TP represent the number of true positive predictions and FP represents the number of false positive predictions. In equation (3.7), FN is the number of false negative predictions.

In addition, the number of predictions where the sum of the edge values differs by at most by some given threshold from the ground-truth sum is measured. How the fraction of predictions within a given threshold from the actual solution is measured is described in Algorithm 3.2.

```
begin
    counter = 0
    for data in test set do
        determine the prediction using the model
        calculate sum of edges used in the prediction
        calculate sum of edges used in optimal solution
        if (optimal sum - prediction sum) < (threshold × optimal sum) do
            counter = counter + 1
        end if
    end for
return counter / (number of predictions)
end
```

Algorithm 3.2: Algorithm used for determining the fractions of predictions within a given threshold from the actual solution

3.6.1 Simulation

For a clear visual comprehension of the model performance, simulations of various possible scenarios are visualized, illustrating how the model allocates resources to moving targets. In the simulation, the ranking of targets is based on the Euclidean distance between the targets and the vital installation. The value of P_{kill} is determined by the Euclidean distance between the resource and the target, the velocity of the target and a random probability of hitting the target, specified for each resource.

3.6.2 Testing performance at varying complexities

Additionally, the performance is tested when different number of targets is present, for validating how many targets can exist in the operating distance, without reducing the model's performance. Equation (3.9) is used for testing this.

$$\text{Accuracy}_x = \frac{\text{number of fully correct predictions on data with x targets}}{\text{number of datapoints with x targets}} \tag{3.9}$$

In equation (3.9), x is the number of targets present within some of the resources operating distances.

4 Results

This chapter presents the results for each dataset, including training time and performance metrics. Furthermore, the correlation between the dataset size and both training time and performance is visualized. Additionally, the model's performance relative to the number of targets is presented. All results were obtained with the same computer, with a 24-core laptop CPU (Intel Core i9-13950HX 2.20 GHz), 64 GB RAM and Windows 10 64-bit operating system.

On the smaller system, three independent models were trained on each dataset. In this chapter, the result with the highest accuracy for each dataset is presented. Results from all runs are presented in Appendix A. For the larger system, only one model was trained on each dataset, due to the long training time and the limited time for this project. All models were trained until no improvement in the loss were observed for ten epochs.

4.1 Network training and evaluation on the smaller system

Dataset	Size	Best loss	Epoch with best loss	Training time (hours)
1	18 212	0.089869	61	1
2	36 203	0.081129	58	1.9
3	54 465	0.077983	102	5.2
4	72 447	0.079339	108	7.3
5	90 496	0.080322	106	7.8
6	108 846	0.080245	116	9.8
Total	380 664	0.082422	23	9.7

Table 4.1: Training time and number of epochs for all datasets on the smaller system with three resources and eight targets.

As seen in Table 4.1, the training took one hour on dataset 1 with 18,212 datapoints, and almost 10 hours on both dataset 6 with 108,846 datapoints and the total dataset with 380,664 datapoints. The lowest loss of 0.077983 was achieved when training on dataset 3. In Figure

4.1, a fast growth in training time is shown between the first six datasets. Thereafter, the training time stabilizes.

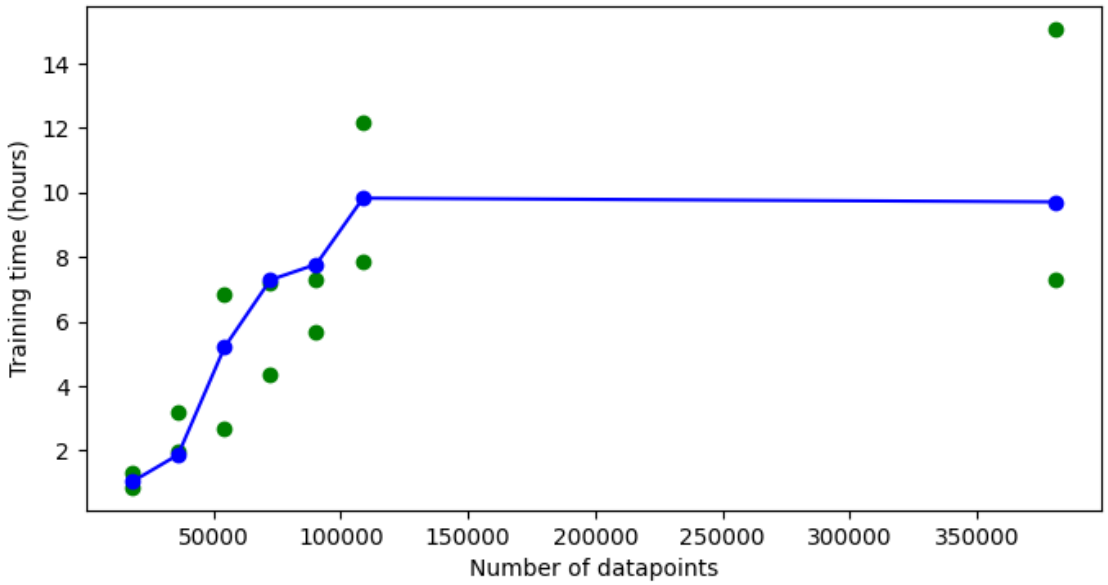

Figure 4.1: Training time as a function of the size of the training set, when training on the smaller system with three resources and eight targets. The blue dots and line represent the training time of the runs where the best accuracy was achieved on each dataset. The green dots represent the remaining runs.

4.2 Model performance on the smaller system

Dataset	Size	Accuracy (%)	Accuracy, MOE <1% (%)
1	18 212	87.6	92.48
2	36 203	90.56	93.29
3	54 465	89.72	93.72
4	72 447	90.08	94.02
5	90 496	90.19	93.62
6	108 846	89.55	93.50
Total	380 664	89.07	93.11

Table 4.2: Accuracy when training on all datasets on the smaller system with three resources and eight targets.

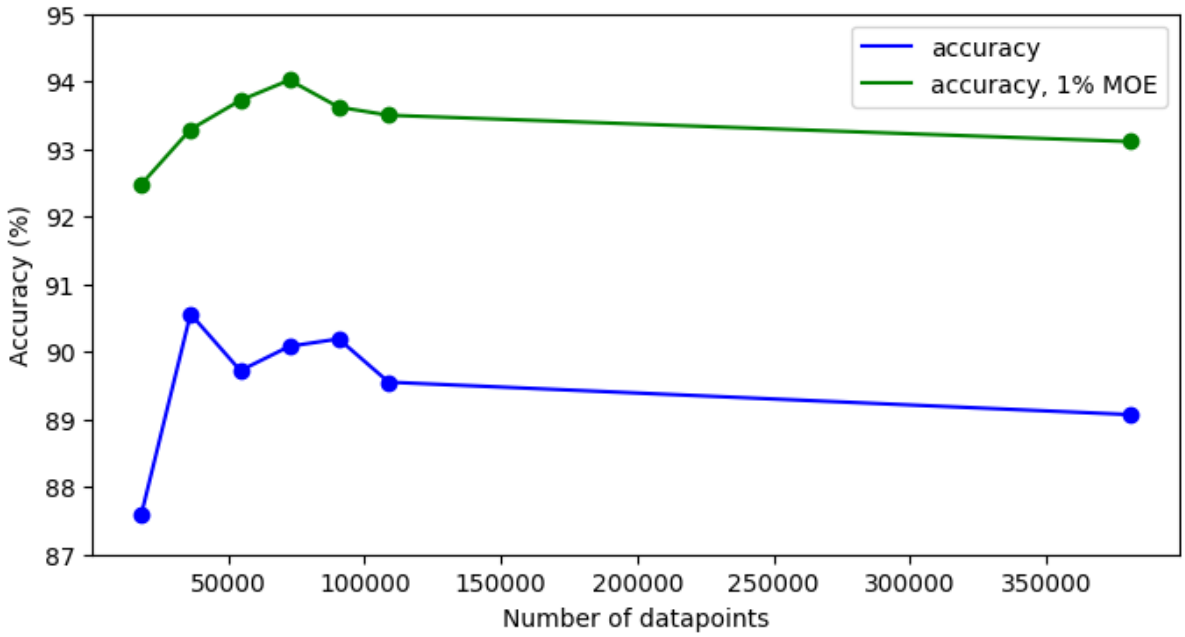

Figure 4.2: Accuracy as a function of the number of datapoints. The training is done on the smaller system with three resources and eight targets.

Table 4.2 and Figure 4.2 illustrate that models trained on datasets with a size between 36,203 and 90,496 datapoints outperform those trained on both smaller and larger datasets. When training on the smallest dataset, an accuracy of 87.6% was achieved, while an accuracy of 89.07% was achieved when training on the largest dataset. The best accuracy of 90.56% was achieved when the model was trained on dataset 2.

Further, on the smallest dataset, 92.48% of the predictions fell within 1% margin from the optimal solution. On the largest dataset, this fraction is instead 93.11%. The highest accuracy with 1% margin of error was 94.02% and was achieved from training on dataset 4.

4.2.1 Elementwise performance when training on the smaller system

Dataset	Size	Elementwise accuracy (%)
1	18 212	97.30
2	36 203	97.96
3	54 465	97.84
4	72 447	97.89
5	90 496	97.90
6	108 846	97.81
Total	380 664	97.66

Table 4.3: Accuracy when classifying each target independently. The training is done on all datasets on the smaller system with three resources and eight targets.

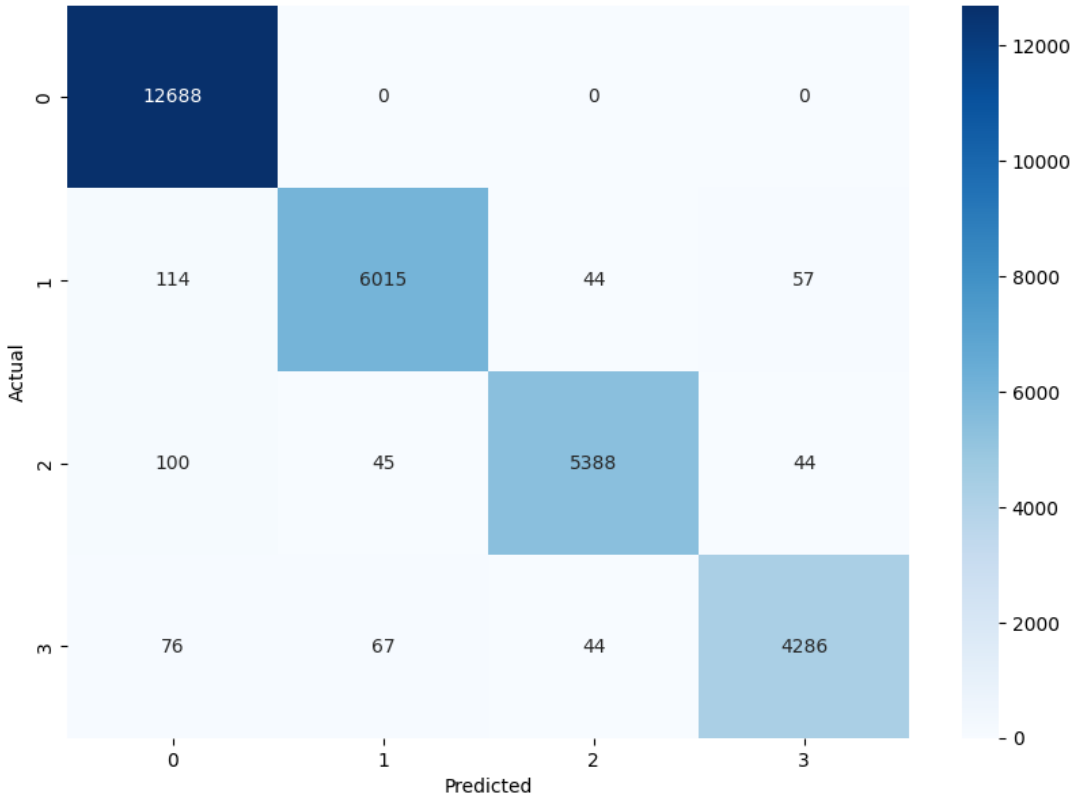

Figure 4.3: Confusion matrix of the elementwise predictions for the best model, i.e., the model trained on dataset 2. The predicted classes are on the x-axis and the actual classes are on the y-axis.

Class	Precision (%)	Recall (%)	F1-score (%)
0	97.77	100	98.87
1	98.17	96.55	97.35
2	98.39	96.61	97.49
3	97.70	95.82	96.75

Table 4.4: Precision, Recall and F1-score for the best model trained on the smaller system, i.e., the model trained on dataset 2.

In Table 4.3, the elementwise accuracy when classifying each target independently is shown. The elementwise accuracy varies between 97.3% and 97.96%. In Figure 4.3, a confusion matrix showing the elementwise predictions for the model trained on the second dataset on the smaller system is shown, where a clear majority of the elements were assigned the correct classes. The first row in the matrix has zeros in all incorrect boxes, indicating the model never allocates resources to targets outside the operating distances.

Table 4.4 presents the precision, recall and F1-score of the different classes existing in the smaller system. Class 0 has 100% recall, meaning there are no false negatives. For the other classes, the precision is marginally higher than the recall.

4.2.2 Performance at varying complexities in the smaller system

Dataset	Number of targets							
	1	2	3	4	5	6	7	8
1	99.23	98.91	97.86	98.54	93.81	88.42	71.90	47.56
2	99.81	98.51	97.66	98.56	94.85	91.18	78.52	62.35
3	99.47	99.50	99.16	97.59	96.13	88.67	80.24	57.78
4	99.51	98.79	98.51	97.55	93.47	89.54	78.17	64.53
5	99.68	98.96	97.91	97.07	95.09	87.22	75.40	59.51
6	99.10	98.97	98.44	98.45	95.39	87.45	78.03	59.40
Total	99.19	98.26	97.93	97.39	95.16	88.43	76.46	59.66

Table 4.5: Accuracy (%) for each dataset when different number of targets are present within some of the resources operating distances. The accuracies are measured on all models trained on a dataset based on the smaller system with three resources and eight targets. Accuracies over 90% are underlined.

Dataset	Number of targets							
	1	2	3	4	5	6	7	8
1	99.23	100	100	100	99.05	96.14	81.05	59.15
2	100	99.50	100	100	98.36	94.85	83.33	67.17
3	100	100	100	100	98.51	95.05	88.38	67.38
4	99.90	100	100	100	97.89	95.21	84.16	74.64
5	100	99.91	99.90	100	98.73	94.10	84.13	70.91
6	100	100	100	100	99.05	93.45	85.17	68.76
Total	99.92	99.77	99.81	99.86	98.53	93.91	84.22	67.94

Table 4.6: Accuracy (%) with MOE < 1% for each dataset when different number of targets are present within some of the resources operating distances. The accuracies are measured on all models trained on a dataset based on the smaller system with three resources and eight targets. Accuracies over 90% are underlined.

In Table 4.5, the accuracies when different number of targets are presented is shown. All models achieved an accuracy of 90% or more when up to five targets are present. Further, an

accuracy of 91.18% was achieved on scenarios with six targets for the model trained on dataset 2. In Table 4.6, the accuracies with 1% margin of error are shown for cases with different number of targets available. Here, all models achieved an accuracy of at least 90% when six or less targets existed.

4.3 Network training and evaluation on the larger system

Dataset	Size	Best loss	Epoch with best loss	Training time (hours)
1	45 602	0.222464	217	9.5
2	91 205	0.216663	229	18.6
3	273 705	0.212956	208	50.2
4	661 387	0.216169	204	125.4

Table 4.7: Training time and number of epochs for all datasets on the larger system with five resources and 20 targets.

The training times for each dataset is shown in Table 4.7. On the smallest dataset with 45,602 datapoints, the training took 9.5 hours, while the training took 125.4 hours on the largest dataset with 661,387 datapoints. The lowest loss, 0.212956, was achieved when training on dataset 3. The number of epochs needed for convergence varied between 204 and 229.

4.4 Model performance on the larger system

Dataset	Size	Accuracy (%)	Accuracy, MOE <1% (%)	Elementwise accuracy (%)
1	45 602	42.92	58.62	83.17
2	91 205	54.72	62.73	91.35
3	273 705	54.56	62.12	91.18
4	661 387	53.07	62.52	91.24

Table 4.8: Accuracy when training on all datasets on the larger system with five resources and 20 targets.

Table 4.8 shows the accuracies of all models trained on the larger system. The accuracy varies between 42.92% and 54.72%. The lowest accuracy was achieved from training on the smallest dataset with 45,602 datapoints. If using the largest dataset, containing 661,387 datapoints, an

accuracy of 53.07% was reached. The best accuracy of 54.72% was achieved when the model was trained on dataset 2 with 91,205 datapoints. This model outperformed models trained on both larger and smaller datasets, although the difference between the three largest dataset was much smaller than the difference between the smallest and the second smallest dataset. However, the highest accuracy on the larger system is much lower than for all models on the smaller system.

Further, on the smallest dataset, 58.62% of the predictions fell within 1% margin from the optimal solution. On the largest dataset, this fraction is instead 62.52%. The highest accuracy with 1% margin of error was 62.73% and was achieved from training on dataset 2.

4.4.1 Elementwise performance when training on the larger system

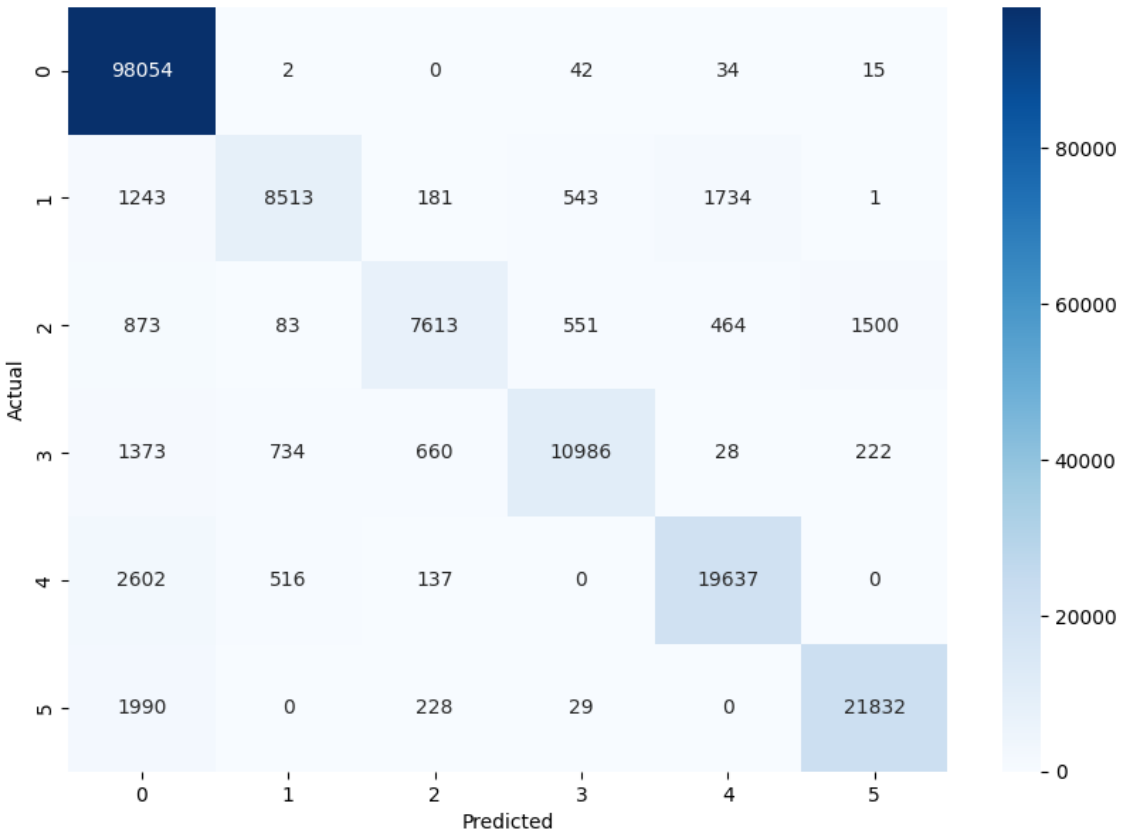

Figure 4.4: Confusion matrix illustrating the elementwise predictions for the best model trained on the larger system. The predicted classes are on the x-axis and the actual classes are on the y-axis.

Class	Precision (%)	Recall (%)	F1-score (%)
0	92.39	99.91	96.00
1	86.44	69.69	77.17
2	86.32	68.68	76.50
3	90.41	78.45	84.01
4	89.68	85.78	87.69
5	92.63	90.67	91.64

Table 4.9: Precision, Recall and F1-score for the best model trained on the larger system, i.e., the model trained on dataset 2.

In Figure 4.4, a confusion matrix showing the elementwise predictions for the model trained on the second smallest dataset on the larger system is presented. The confusion matrix shows a discernible diagonal, albeit less distinct than the diagonal in Figure 4.3. The first row in the matrix has strictly positive values in almost all boxes, both the correct box and the incorrect boxes, suggesting the model either allocates a resource to a target where capacity constraints prevent this inclusion in the optimal solution, or assigns resources to targets located outside their operating distances.

Table 4.9 presents the precision, recall and F1-score of the different classes existing in the larger system. Class 0 has the highest F1-score, where the recall is significantly higher than the precision. For all other classes, the precision is higher than the recall. Further, the scores are lower on class 1 and 2, compared to the other classes.

4.4.2 Performance at varying complexities in the larger system

Dataset	Number of targets									
	1	2	3	4	5	6	7	8	9	10
1	100	99.38	99.22	99.58	94.44	90.11	85.05	79.13	70.99	52.87
2	99.88	100	99.66	98.91	94.01	94.58	87.78	81.49	72.89	56.19
3	99.92	99.80	99.60	97.40	96.01	93.48	87.96	79.96	74.68	61.68
4	99.98	99.75	99.58	98.48	97.20	93.30	89.58	80.70	74.51	58.51

	11	12	13	14	15	16	17	18	19	20
1	43.98	32.04	14.98	9.54	8.42	4.78	3.59	1.23	0	0
2	49.10	34 29	21.88	12.59	9.40	4.55	3.02	1.68	0.30	0
3	46.81	34.46	24.53	16.39	10.14	6.18	3.04	2.43	0.65	0.59
4	46.83	36.05	24.87	16.35	11.50	6.36	3.52	1.90	0.82	0.57

Table 4.10: Accuracy (%) for each dataset when different number of targets are present within some of the resources operating distances. The accuracies are measured on all models trained on a dataset based on the larger system with five resources and 20 targets. Accuracies over 90% are underlined.

In Table 4.10, the accuracies on scenarios with different number of targets are presented. All models achieved an accuracy of 90% or more on scenarios with up to six targets. Thereafter, the accuracy drastically reduces when adding more targets.

4.5 Simulation results

In this section, different steps from two resource allocation simulations are shown, one simulation illustrating each of the systems. In both simulations, the best model for each respective system was used. The targets were randomly distributed and had a velocity vector that directed them towards the vital installation.

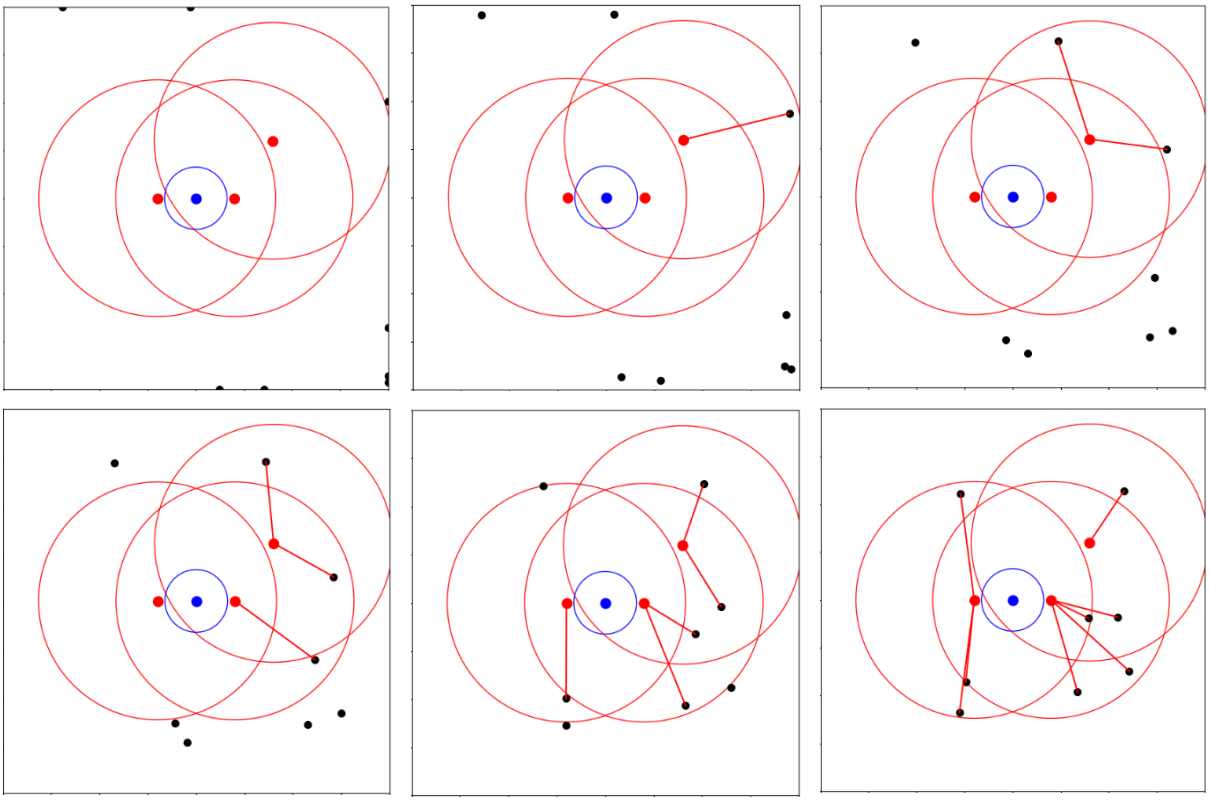

Figure 4.5: Step-by-step showing how the best model on the smaller system assigns resources to the targets in different timesteps. In this simulation, tree resources and eight targets were used.

In Figure 4.5, different steps from the resource allocation simulation in the smaller system are shown. Here, the model was trained on the second dataset with 36,203 datapoints. The corresponding simulation steps for the larger system are shown in Figure 4.6. Here, the used model was trained on dataset 2 with 91,205 datapoints. The first model manages to allocate resources to all targets in the less complex system, shown in Figure 4.5. However, the latter model fails to allocate resources to some reachable targets. Additionally, this model sometimes allocates resources to targets outside the resource's operating distance, shown in the fifth step in Figure 4.6.

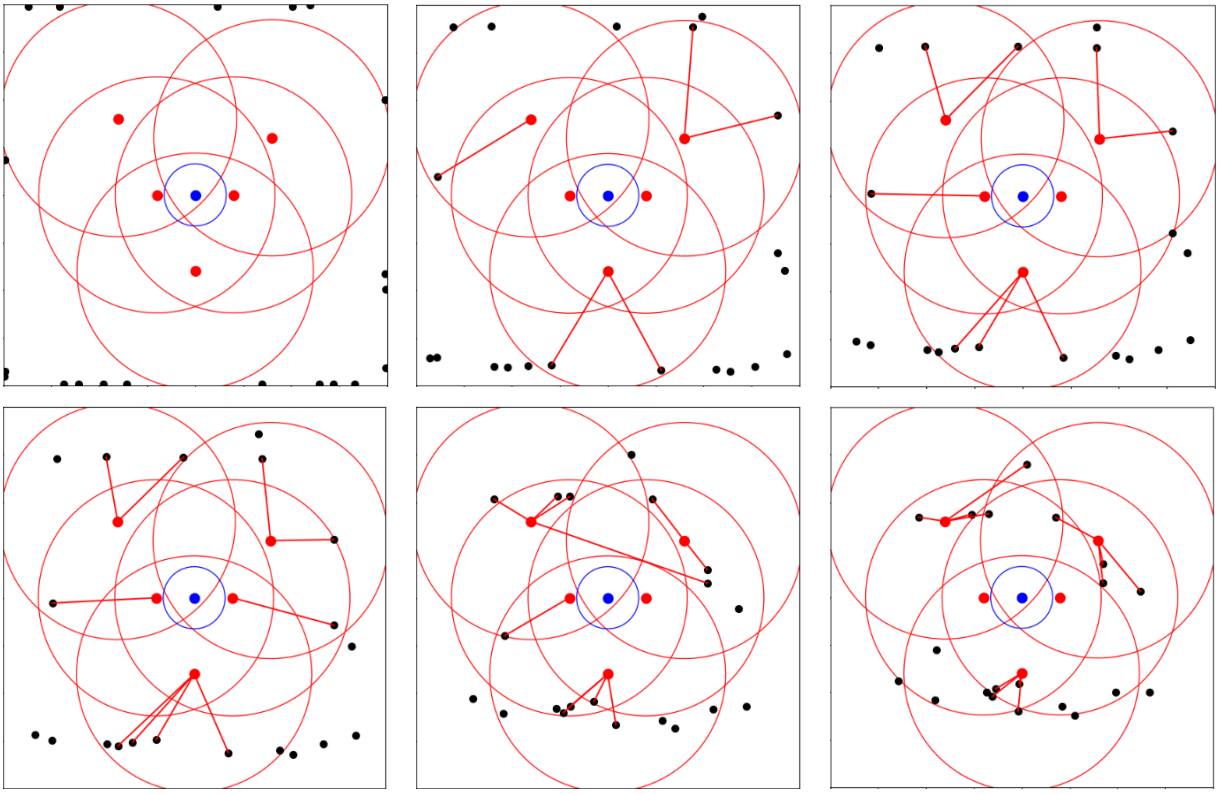

Figure 4.6: Step-by-step showing how the best model on the larger system assigns resources to the targets in different timesteps. In this simulation, five resources and 20 targets were used. Note that the resource at the top left is allocated to a target outside the operating distance in step 5.

5 Discussion

In this chapter, the results from chapter 4 are discussed, along with the implementation and some of the parameter settings. The chapter ends with a brief discussion of possible future work related to this project.

5.1 Model performance and training time

Based on the results in section 4.2, we can conclude that the model achieves the best performance when the dataset size is neither too small nor too large. For the smaller system, the accuracy is peaking when training on a dataset with a size between approximately 36,000 and 90,000 datapoints. Further, the highest accuracy of 90.56% was achieved when training on dataset 2 with 36,203 datapoints, and not the largest dataset, confirming that adding data will not necessarily yield better results. However, the training time still increases when adding datapoints, as shown in section 4.1, even if the accuracy is not improving. Training a neural network model on a small dataset increases the risk of overfitting, which negatively affects the model performance. On the other hand, if the dataset is large and contain too small variations, overfitting could occur on common, and usually easy scenarios, while performing worse on less common and more complex scenarios. In, Figure 4.1, it can be observed that the training time increases rapidly between the first six datasets. However, the largest dataset requires approximately the same training time as the sixth dataset. The lower training time related to the number of datapoints, along with the lower accuracy, indicates overfitting for the model trained on the largest dataset. Then, the new data is similar to the old data and does not add any value. However, the accuracy for the smallest dataset is significantly lower than the accuracy for the largest dataset, indicating the consequence of overfitting due to lack of data is here larger than the consequence of overfitting due to too small variations.

All models trained on the smaller system manages to achieve an accuracy of 90% or more on scenarios with up to five targets, as can be seen in Table 4.5. Further, the model trained on dataset 2 also achieved an accuracy over 90% at scenarios with six targets. The accuracy drastically reduces for scenarios with more targets. This clearly shows that the model has some problems with more complex cases.

However, when six targets existed, between 93.45% and 96.14% of the predictions are within 1% from the optimal solution, meaning that even if the model has problems with finding the optimal solution, it is still able to find decent solutions. Since the complexity has exponential growth with respect to the number of elements in the system, it is reasonable the accuracy has a fast decline.

Unfortunately, the models trained on the larger system did not perform as well as the models trained on the smaller and less complicated system. As presented in Table 4.8, the accuracy varied between 42.92% and 54.72%. Additionally, the models trained on the large system occasionally allocated resources to targets outside the resources operating distances, shown in the fifth step in Figure 4.6. This did never happen for any model trained on the smaller system. This happened due to the models on the larger system converting some of the edges with edge value zero to positive values in the output matrix, while the models on the smaller system convert all these edges to negative values. All models on the larger system, succeed to reach an accuracy of 90% or more on scenarios with up to six targets, as shown in Table 4.10. This suggests that the model begins to produce incorrect predictions when the problem's complexity makes it impossible to allocate the best resource to every target.

In addition, the training time was much longer on the larger system. Comparing the training time on dataset 5 on the smaller system and dataset 2 on the larger system, which contain approximately the same amount of data, the training took more than twice as long time on the larger system. Nevertheless, the accuracy was much lower. Adding large amounts of data does not improve the accuracy after some point for the larger system. This suggests that the amount of data is not the bottleneck, but the implementation and the complexity of the model. The same implementation was used for both systems and the parameter tuning were done for optimizing the model on the smaller system. Since the complexity of the other system was much higher, this problem probably requires a model with higher complexity to be solved.

5.1.1 Class-wise performance

In section 4.2.1 and section 4.4.1, the elementwise predictions on the smaller system and the larger system are shown. In the confusion matrix for the best model trained on the smaller system, it is shown that the model has a high probability of giving the correct class to an individual target. In Table 4.3, it is shown that the best model achieves an accuracy of 97.96%,

when classifying targets independently. However, for the best model trained on the larger system, this probability is instead 91.35%, shown in Table 4.8. For both models, the precision is higher than the recall for all classes, except from class 0. The reason is that the models tend to incorrectly assign class 0 more often than incorrectly assigning any other class, causing a lower precision of class 0, while benefitting the precision of the other classes. Additionally, class 0 has 100% recall on the smaller system, shown in Table 4.4, meaning there is no false negatives in this class.

5.1.2 Stochastic behaviour in model training

Given that the neural network training is a stochastic learning process, both performance, training time and loss can vary across different training runs. In this project, the models trained on the smaller system were trained three times on each dataset and the best results were saved. This minimizes the risk of obtaining poor results for any dataset, as the risk of all three runs being bad is lower than the risk of a single run being bad. In the results obtained, there was some variation in the results across different runs, especially with respect to training time, shown in Appendix A. However, even with multiple runs per dataset, individual runs on any dataset could still outperform or underperform runs on other datasets, explaining why both the model performance on the smaller system, shown in Figure 4.2, and the training time, shown in Figure 4.1, have moderately uneven curves.

5.2 Implementation and hyperparameter tuning

The choice of layers had a significant impact on the model performance. Across all layers tested, graph convolutional layers were more robust and resulted in the best model. Adding a linear layer after the graph convolutional layers yielded better results than a similar model without a linear layer. Further, models with 64 and 32 neurons in the hidden layers were in general better than models with other dimensions.

Including a small penalty factor in the loss function for false negative predictions led to marginally better results compared to using no penalty factor. However, the model training was highly sensitive to the size of the penalty factor and the model accuracy was sharply reduced when the penalty factor increased.

5.3 Ethical considerations

The integration of machine learning into defence systems brings new challenges regarding ethical aspects. While machine learning offers unique and revolutionary possibilities, it needs to be used carefully. It is important to be aware that even a computer-based model can make mistakes, and the models should never bear the full responsibility for its actions. Further, the model has absence of both intent and ethical thinking. Therefore, a human decision-maker should always make the final decision on whether to execute an action or not. This is particularly important in the case of usage in defence systems, where weapons are involved and the consequences of an incorrect action can be devastating.

Although a machine learning model should never make the final decision, it is still important that the model produces results of high quality, to appear reasonably credible. An unreliable model drastically reduces the trust in the system, which usually makes the model unusable.

5.4 Future work

The model achieved during this project does not yet meet the requirements for use in a sharp, real-world system. However, this thesis project resulted in better understanding of how such model could be implemented and how different hyperparameter settings affect the model.

Possibly, different hyperparameter settings can yield better performance, including learning rate, weight decay, penalty function and dropout between the layers. Other parameters in the model architecture, such as the size and number of layers, type of layers and type of activation functions could also be adjusted to improve the model. While this thesis examines various parameter settings, it is crucial to acknowledge that the tested combinations only present a small subset of all potential hyperparameter combinations.

Further, it is also possible that a different training method would be more suitable for this problem. Instead of treating this problem as a classification of nodes, the problem could be treated as a different type of problem, for example binary classification of the edges. Also, introducing sampling could possibly benefit the training. During this thesis, only graph neural networks were used, but possibly, other types of neural networks could also be used.

As shown in Figure 3.1 and Figure 3.2, the datasets were imbalanced. Using a more balanced dataset can affect the results. Both histograms are showing that datapoints with one target and datapoints with slightly less than the maximum number of targets are more common than datapoints with the maximum or around half of the maximum number of targets. However, in Figure 3.1 it is shown than six to seven targets are more common than two to four targets in the smaller system, but the accuracies for two to four targets are higher than the accuracies for six and seven targets. Thus, using a more balanced dataset will not necessarily yield better results. However, in Figure 3.2 showing the distribution for datasets based on the larger system, it is shown that the least common number of targets are between six to eleven, which is when the model performance decreases drastically. Therefore, it is possible that a more balanced dataset would yield better performance on the larger and more complex system.

6 Conclusion

This master's thesis project has shown that graph neural networks possibly could be used for resource allocation in air defence system. The best model achieved from training on a smaller system with three resources and eight targets reached an accuracy of 90.56% after less than two hours of training. Further, it was concluded that going from 18,000 to 36,000 datapoints increased the accuracy drastically, but afterwards, only the training time increased without improving the model's performance.

When increasing the complexity of the system, by using five resources and 20 targets, the model did not reach the same performance as on the smaller system. This model only reached an accuracy of 54.72% after 50 hours of training on a dataset with approximately 91,000 datapoints. Thus, the model has problems with more complex scenarios. Further, both increment in complexity and amount of data requires more training time. However, the accuracy will not necessarily increase, especially not if adding complexity to the problem.

In summary, the model performed well on problems with low complexity, but the accuracy was drastically reduced when the complexity increased. Since the problem is NP-complete, it is reasonable the accuracy has a fast decrement when adding elements in the system. Although the current model is not yet suitable for a real-world application, graph neural networks hold significant potential as a component of a future solution to this combinatorial optimization problem, enabling more efficient usage of resources. This pending further enhancements in model architecture, training method and hyperparameter settings.

Bibliography

[1] M. Gori, G. Monfardini and F. Scarselli, “A new model for learning in graph domains,” in *IEEE International Joint Conference on Neural Networks*, Montreal, 2005.

[2] R. Stephens, "How drones have shaped the nature of conflict," Vision of Humanity, [Online]. Available: https://www.visionofhumanity.org/how-drones-have-shaped-the-nature-of-conflict/. [Accessed 18 February 2025].

[3] J. Svensson, “Swedish Defence University,” 12 12 2023. [Online]. Available: https://www.fhs.se/en/swedish-defence-university/stories/2023-12-05-innovative-ai-research-about-military-decision-support.html. [Accessed 28 February 2025].

[4] V. P. Parmar and C. Kumbharana, "Comparing Linear Search and Binary Search Algorithms to Search an Element from a Linear List Implemented through Static Array, Dynamic Array and Linked List," *International Journal of Computer Applications*, vol. 121, no. 3, p. 15, 2015.

[5] M. Sisper, Introduction to the Theory of Computation, Second Edition, Boston: Thomson Course Technology, 2006, pp. 264-282.

[6] B. Mehling, Machine learning with neural networks, Gothenburg: Cambridge University Press, 2022.

[7] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner and G. Monfardini, "The Graph Neural Network Model," *IEEE Transactions on Neural Networks*, vol. 20, no. 1, pp. 61-80, 2009.

[8] Y. Liu, P. Zhang, Y. Gao, C. Zhou, Z. Li and H. Chen, "Combinatorial Optimization with Automated Graph Neural Networks," arXiv, 2024.

[9] N. Yadati, P. Yadav, A. Louis, M. Nimishakavi, V. Nitin and P. Talukdar, "HyperGCN: A New Method of Training Graph Convolutional Networks on Hypergraphs," Indian Insitute of Science, Bangalore, 2019.

[10] Z. Li, Q. Chen and V. Koltun, "Combinatorial Optimization with Graph Convolutional Networks and Guided Tree Search," 32nd Conference on Neural Information Processing Systems (NIPS 2018), Montréal, 2018.

[11] J. Zhou, G. Cul and et al., "Graph neural networks: A review of methods and applications," *AI Open,* vol. 1, pp. 57-81, 2020.

[12] R. Sato, M. Yamada and H. Kashima, "Approximation Ratios of Graph Neural Networks for Combinatorial Problems," 33rd Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, 2019.

[13] D. S. Hochba, "Approximation Algorithms for NP-Hard Problems," *ACM SIGACT News,* vol. 28, no. 2, pp. 40-52, 1997.

[14] G. Panchal and D. Panchal, "Solving NP hard Problems using Genetic Algorithm," *International Journal of Computer Science and Information Technologies,* vol. 6, no. 2, pp. 1824-1827, 2015.

[15] Y. Liu, C. Zhou, P. Zhang, S. Pan, Z. Li and H. Chen, "Decision-focused Graph Neural Networks for Combinatorial Optimization," arXiv, 2024.

[16] A. Nicolaides, Combinations, Permutations, Probabilities, vol. 10, Bodmin, Cornwall: Pass publications, 1994.

[17] B. Sanchez-Lengeling, E. Rief, A. Pearce and A. B. Wiltschko, "A Gentle Introduction to Graph Neural Networks," *Distill,* 2021.

[18] A. Daigavana, G. Ravindran and G. Aggarwal, "Understanding Convolutions on Graphs," *Distill,* 2021.

[19] Y. Wang, W. Hou, N. Sheng, Z. Zhao, J. Liu, L. Huang and J. Wang, "Graph pooling in graph neural networks: methods and their applications in omics studies," *Artificial Intelligence Review,* vol. 57, no. (294), 2024.

[20] S. Sharma and S. Sharma, “Activation functions in neural networks,” *International Journal of Engineering Applied Sciences and Technology,* vol. 4, no. 12, pp. 310-316, 2020.

[21] T. N. Kipf and M. Welling, "Semi-Supervised Classification with Graph Convolutional Networks," in *International Conference on Learning Representations*, Toulon, 2017.

[22] S. Zhang, H. Tong, J. Xu and R. Maciejewski, “Graph convolutional networks: a comprehensive review,” *Computational Social Networks,* vol. 6, no. 11, 2019.

[23] D. Bergmann and C. Stryker, "What is backpropagation?," IBM, 2 July 2024. [Online]. Available: https://www.ibm.com/think/topics/backpropagation. [Accessed 29 January 2025].

[24] J. Delua, "Supervised versus unsupervised learning: What's the difference?," IBM, 21 March 2021. [Online]. Available: https://www.ibm.com/think/topics/supervised-vs-unsupervised-learning. [Accessed 29 January 2025].

[25] J. Murel and E. Kavlakoglu, "What is reinforcement learning?," IBM, 25 March 2024. [Online]. Available: https://www.ibm.com/think/topics/reinforcement-learning. [Accessed 29 January 2025].

[26] K.-H. Lai, D. Zha and X. Hu, "Policy-GNN: Aggregation Optimization for Graph Neural Networks," Texas A&M University, 2020.

[27] Y. Song, Y. Li, S. Fan, D. He and J. Liao, "A New Graph Neural Network-based Join Optimization Algorithm," in *International Conference on Algorithms, Data Mining, and Information Technology (ADMIT)*, Xi'an, 2022.

[28] Y. Gao, H. Yang, P. Zhang, C. Zhou and Y. Hu, "GraphNAS: Graph Neural Architecture Search with Reinforcement Learning," arXiv, 2019.

[29] J. R.Terven, D. M. Cordova-Esparza, A. Ramiez-Pedraza, E. A. Chavez-Urbiola and J. A. Romero-Gonzalez, "Loss Functions and Metrics in Deep Learning," arXiv, 2024.

[30] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever and R. Salakhutdinov, “Dropout: a simple way to prevent neural networks from overfitting,” *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929-1958, 2014.

[31] "CrossEntropyLoss," PyTorch, 2024. [Online]. Available: https://pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html. [Accessed 3 March 2025].

Appendix A

In this chapter, accuracy, loss, number of epochs and training time from all runs on the smaller system is presented.

Dataset	Accuracy (%)	Best loss	Epoch with best loss	Training time (hours)
1	87.6	0.089869	61	1
1	87.54	0.088581	55	0.9
1	87.54	0.08677	88	1.3
2	90.22	0.082683	61	2
2	90.08	0.08148	102	3.2
2	90.56	0.081129	58	1.9
3	89.39	0.075459	154	6.9
3	89.48	0.077237	56	2.7
3	89.72	0.077983	102	5.2
4	89.77	0.080908	67	4.4
4	90.08	0.079339	108	6.8
4	89.59	0,079191	116	7.2
5	90.19	0.080322	106	7.8
5	89.55	0.080188	99	7.3
5	89.87	0.080878	72	5.7
6	89.32	0.079411	137	12.2
6	89.21	0.081034	84	7.9
6	89.55	0.080245	116	9.8
Total	88.32	0.079859	43	15.1
Total	88.51	0.082935	16	7.3
Total	89.07	0.082422	23	9.7

Table A.1: Accuracy, loss, number of epochs and training time from all runs on the smaller system. The highest accuracy on each dataset is underlined.

DEPARTMENT OF SPACE, EARTH AND ENVIRONMENT
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2025
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY