CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Automatic parameter selection for multi-speed multi-agent pathfinding solver

Master's thesis in Computer science and engineering

ANDREAS ROSENFELD

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2020

Master's thesis 2020

Automatic parameter selection for multi-speed multi-agent pathfinding solver

ANDREAS ROSENFELD

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2020

Automatic parameter selection for multi-speed multi-agent pathfinding solver
ANDREAS ROSENFELD

Supervisor: Elad Michael Schiller, Department of Computer Science and Engineering
Examiner: Nir Piterman, Department of Computer Science and Engineering

Master's Thesis 2020
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2020

Automatic parameter selection for multi-speed multi-agent pathfinding solver
ANDREAS ROSENFELD
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

Multi-agent pathfinding is a study focused on finding collision-free paths for multiple agents in a shared environment in order for the agents to reach designated destinations. As the problem is well researched, more and more sophisticated algorithms are available that utilize different parameters to fine-tune the search for these solutions. In this paper, we demonstrate the combining of multi-agent pathfinding with machine learning to perform such parameter selection dynamically. This selection is approached as an image classification problem, where unseen problem instances are divided between a collection of available parameter values. For this, typical vehicular road traffic scenarios were implemented for experimentation. The presented results give new knowledge for this previously unstudied approach indicating that there are cases in which visual representation of the problem is sufficient as the dynamic selection can perform better when compared to a fixed setting. Therefore, demonstrating the potential that machine learning can be used to improve the performance of a multi-agent pathfinding algorithm without needing to develop it further.

Keywords: computer science, multi-agent pathfinding, machine learning, transfer learning, AlexNet, parameter selection, urban scenarios.

Acknowledgements

First and foremost, I want to thank my supervisor, professor Elad Michael Schiller, for his knowledge, experience, and support throughout this thesis. I am also very grateful to the examiner, professor Nir Piterman, for taking the time to oversee and evaluate the project. Additionally, I want to mention my fellow students - Johan Gerdin, Jesper Åberg, and Andreas Kuszli - with whom we formed a small but supportive team, as we all worked in a shared field of study. Lastly, I want to thank the Alfred Ots Scholarship Fund for supporting my studies at Chalmers University of Technology.

Andreas Rosenfeld, Gothenburg, August 2020

Contents

List of Figures

List of Tables

1

Introduction

In the multi-agent pathfinding (MAPF) problem, the task is to find paths for multiple agents, each having a specific start and end location. A valid solution is a set of paths that would allow each agent to navigate from its start location to its end location without conflicts, meaning the agents do not collide with each other. Collisions between agents happen when agents occupy or traverse the same location at the same time, and thus, need to be avoided in the planning of the solution. Within the literature, common objectives for a solution are to minimize the sum of the individual path costs of all agents, or to minimize the maximum individual path cost (known as the makespan). Finding such optimal conflict-free solutions is NP-hard [1, 2] as the computational complexity of the problem grows exponentially with the number of agents.

However, many MAPF solvers have been developed, which can model, and thus contribute to, a wide array of industrial and commercial settings. Such problems, that can benefit from numerous teams of agents (e.g. automated robots) cooperating and moving efficiently within their environment, include, for example, GPS navigation [3], traffic control [4], aviation [5], and video games [6]. Additionally, another critical, but natural, application example includes automated warehousing systems [7] such as Amazon order-fulfillment centers (Figure 1.1). In this setting, warehouse robots need to operate autonomously in a grid-like environment between inventory stations and storage locations to move inventory pods. Each robot can carry one pod at a time, either to the inventory station for shipping or back to an empty storage location. Most of the warehouse space is filled with storage locations separated by narrow corridors that allow only one-way traffic. Therefore, path planning is needed to avoid collisions and to enable the agents to work fast and efficiently.

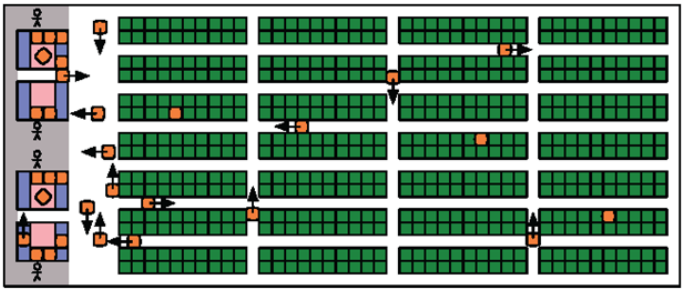

Figure 1.1: Amazon warehouse as a MAPF problem, with inventory stations (left side of figure) and storage locations (green cells) [7].

As such, MAPF problems have been studied intensively since the 1980s [8], and the resulting solvers can be classified in many different ways. Since MAPF is NP-hard, a natural distinction between algorithms comes in terms of optimality. Optimal solvers are practical up to a relatively small number of agents, while sub-optimal solvers aim to trade-off optimality for faster runtime. Examples of optimal algorithms include Conflict Based Search (CBS) [9] and M* [10], while sub-optimal ones cover Hierarchical Cooperative A* (HCA*) [6] and Enhanced-CBS (ECBS) [11], to name a few. Some additional straightforward contrasts between solvers are:

- **Coupled**, i.e., teams of agents are considered as a compound system, on which single-agent pathplanning is applied vs. **decoupled**, i.e., individual paths are found separately for each agent in combination with collision avoidance methods [8].
- **Offline**, i.e., the problem instance is given as is and a solution is calculated before execution vs. **online**, i.e., new agents/obstacles appear during the running time, and paths are recalculated during execution [12].
- **Discrete time**, i.e., agents operate in discrete time, where each action has a uniform duration and is associated with one time step vs. **continuous time**, i.e., agents' actions happen independently within specific time intervals [13].

Currently, no single MAPF algorithm is considered the best one in all situations [14, 15].

In the field of machine learning, computer vision tasks such as image classification have shown increasingly better results in the last decade with the use of Convolutional Neural Networks (CNNs). Even though the preceding idea of CNNs can be regarded to originate from 1980, with the modern framework being established in the 1990s [16]. This recent success can be attributed to the convergence of several trends, namely, the increasing availability of data, significant improvements in specialized computer hardware, and a vast selection of many useful machine learning libraries. The last allows the use of neural networks on a variety of problems. Most significantly, such libraries can provide pretrained CNN models, which already perform well on general visual problems, but can be further utilized and trained in a more specialized setting, as MAPF.

In this thesis, we plan to combine MAPF with machine learning in the context of parameter selection. For this, we will be working with Continuous-Time with Discrete-Speeds (CTDS) [17], an optimal MAPF solver, which finds paths for agents in a decoupled, offline manner and operates in continuous time. Additionally, the model accepts a set of speeds, that dictates the possible speeds that can be used by the agents. However, it is unclear how to perform this parameter selection. Therefore, we present the idea of using a pretrained CNN as an *oracle* that helps us automatically select the optimal speed parameter before solving a specific MAPF problem instance. To analyze the performance of our suggested machine learning approach, we will design various vehicular road traffic scenarios such as highways, intersection, and roundabout, to experiment on.

1.1 Related work

Over the years, various MAPF algorithms have been developed, each having their own strategy to solve MAPF problems and, therefore, performing better on different instances. As such, Sigurdson et al. [15] explored automatic algorithm selection in the context of multi-agent video game maps. They used 20 maps from the game *Baulder's Gate II* and produced MAPF problems of various types - all agents assigned randomly, all agents moving from one side to another, all agents moving from the center of the map to the outside, among others - for a total of seven problem types. For the algorithm selection, three distinct MAPF algorithms were considered, each with its strengths and weaknesses. With this setup, they trained an off-the-shelf deep neural network to automatically select a fitting algorithm for MAPF instances on the chosen maps. For this, problem instances were represented as images, where colored pixels illustrated the following information: white - unblocked node, black - blocked node, green - start location of an agent, and red - goal location of an agent. Their dataset consisted of 2800 samples, as they used 20 maps, where for every map, 20 MAPF instances were created for each of the seven problem types. As their primary evaluation metric, they used the completion rate and demonstrated that their approach performed better than any of the individual algorithms. One of the benefits of such an approach is that it does not require the development of new MAPF algorithms. However, at the same time, it expects the implementation of multiple preexisting algorithms.

In preceding work to the previous, Sigurdson and Bulitko [18] also explored automatic algorithm selection, but in the single-agent domain. Intending to find a pathfinding algorithm with the lowest expected suboptimality, they created a portfolio of algorithms to select from. Similarly, they approach the dynamic algorithm selection as a traditional image classification problem with graphs based on several video game maps. In their dataset, 342 maps were considered from various vide-games as the search graphs, for a total of over 17100 MAPF problems. They demonstrated the performance of the algorithm selection on four different granularity levels: game-type, per-game, per-map, and per-problem. With the lowest expected suboptimality as the evaluation metric, the results showed that there is potential to achieve better results with dynamic algorithm selection rather than always choosing a single algorithm.

This project differs from earlier work as we shift the focus from virtual video-game MAPF problems to more realistic problems based on vehicular road traffic scenarios. Moreover, the automatic algorithm selection is reformulated as the automatic parameter selection allowing to develop a systematic way to work with a single algorithm extensively.

We note the existence of alternative optimization approaches, such as the one by Petig et al. [19]. Moreover, there are algorithms for cooperatively facilitating safe manoeuvres that are needed for the implementation of the studied use cases [20, 21, 22, 23, 24, 25, 26]. We offer the reader to consider the implementation of the studied techniques using simulations approaches, as in [27, 28, 29].

1.2 Our contribution

Our work contributes to the field of MAPF by describing the automatic parameter selection approach and applying it on various vehicular road networks. As the CTDS model [17] comes from recent research, the work has also assisted in making the implementation of the MAPF model more robust. The automatic parameter selection approach permits the use of different variables dynamically to solve new unseen MAPF problem instances. This is beneficial as different instances might require separate parameters depending on the problem setup to achieve a better solution. The results from our experiments showed that this dynamic selection - based on the number of speeds available to the agents, and thus enabling smoother interaction between the agents, if needed - can perform better when compared to a fixed setting. In the best cases, our per-graph model was able to perform 2.04 times better than the best deterministic selection. For the model trained on combined data from multiple graphs, the same measure was 1.88 times better than the best deterministic selection. This was measured by comparing the differences to the optimal cost between the best deterministic selection and our dynamic selection. In conclusion, the work suggests a connection between the visual representation of a MAPF problem and its solution with the potential to improve it further via parameter selection. The novel results obtained from this vision-based learning approach may help to provide a state-of-the-art improvement to the vital problem of planning for autonomous vehicle transportation.

2

Background

In the following chapter, background information of the two main fields - multi-agent pathfinding and machine learning - is given along with the formulation of the automatic parameter selection problem. Fundamentally, the combining of these two fields is done in this project to solve the presented selection problem.

2.1 Multi-agent pathfinding

In this section, the MAPF problem within the CTDS model [17] is explained along with the algorithm to solve such problems. In further chapters, we treat this MAPF solver as a *black box*, which is applied in order to generate data for the automatic parameter selection problem, described in the following Section 2.2.

2.1.1 Continuous-time with discrete speeds model

The CTDS model [17] considers actions in continuous time, meaning that the actions of the agents do not take uniform time at each time step, as is the case with discrete-time models. An important factor is also the attempt to relax the assumption that agents can stop and accelerate instantaneously. For this, the model supports for the agents the ability to use a discrete set of speeds $speeds = \{s_{min}, .., s_{max}\}$, allowing them to accelerate between them and thus, enabling for smoother speed changes and better arrival time.

2.1.1.1 MAPF problem

In this model, the MAPF problem consists of a directed graph $G = (V, E)$, where each node $v \in V$ represents locations and each edge $(v, v') \in E$ the possible connections between the nodes. Additionally, we have a set of agents $A = \{a_1, .., a_k\}$, where each agent $a_i \in A$ is described as a pair of unique start and end nodes $(v^i_{start}, v^i_{goal})$. Agents traverse the graph by transitioning between states $st = (v \in V, t \in \mathbb{R}, s \in speeds)$. The triple is defined by

- the position of the agent (v),
- the time when the agent is in that position (t), and
- the speed of the agent at that time (s).

A solution to the problem is a set of collision free paths $P = \{p_1, .., p_k\}$, where for each agent $a_i \in A$ the path $p_i = (st\{v^i_{start}, 0, s^i_{start}\}, .., st\{v^i_{goal}, t_{goal}, s^i_{goal}\})$ encodes transitions (actions) between states to traverse beginning from the assigned start node to the goal node.

In the next section, we will introduce an optimal solver, which will be used in this project, that finds such solutions.

2.1.1.2 Solver

The solver works in an alternating fashion between finding optimal single-agent paths (low-level search), and then, based on conflicts between the individual paths, constructing a binary constraint tree (CT) by adding new nodes until an optimal multi-agent solution is found (high-level search), assuming one exists. At first, the root node of the CT has no constraints. After the initial low-level search, conflicts, if found, are resolved by imposing constraints on the actions of the agents. Each new constraints adds two child nodes in the CT, as each conflict between two agents can be solved in two ways: allowing an action for one agent and disallowing an action for the other during a specific time interval, or vice versa. This back and forth between the two levels continues until a goal node in the CT is reached, meaning that all constraints are satisfied, and thus all conflicts are resolved. Particularly, the cost of the final solution is the minimum that can be achieved, which is defined for the solver as the sum of the individual path costs (SIC):

$$SIC(P) = \sum_{p_i \in P} cost(p_i) \tag{2.1}$$

where the cost of an single path p_i is the arrival time at the goal node:

$$cost(p_i) = t_{goal} \tag{2.2}$$

To illustrate this, we consider the following example. Figure 2.1 represents a MAPF problem with agents $a_1 = (0, 2)$ and $a_2 = (3, 4)$. Additionally, the discrete set of speeds $speeds = \{0, 1\}$ is given.

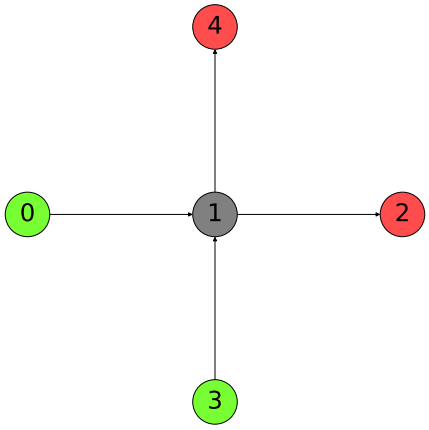

Figure 2.1: MAPF problem instance, with two agents $a_1 = (0, 2)$ and $a_2 = (3, 4)$. Start nodes shown in green, and goal nodes in red.

In order to find an optimal solution, a CT (Figure 2.2) is built starting from the root node, where there are no constraints. The first low-level search gives a corresponding solution P_0, but right away there is a conflict as both agents try to enter node 1 at the same time. This situation is resolved by adding two additional child nodes (N_1 and N_2) to the root node. The first one (N_1) constrains a_1 by forcing it to wait until a_2 has entered node 1 before allowing a_1 to start moving, and vice-versa for the other CT node N_2. The search ultimately stops here as both N_1 and N_2 are goal nodes, because there are no more conflicts and both agents are at their goal nodes. Also, the solutions P_1 and P_2 are equivalent as $SIC(P_1) = SIC(P_2)$.

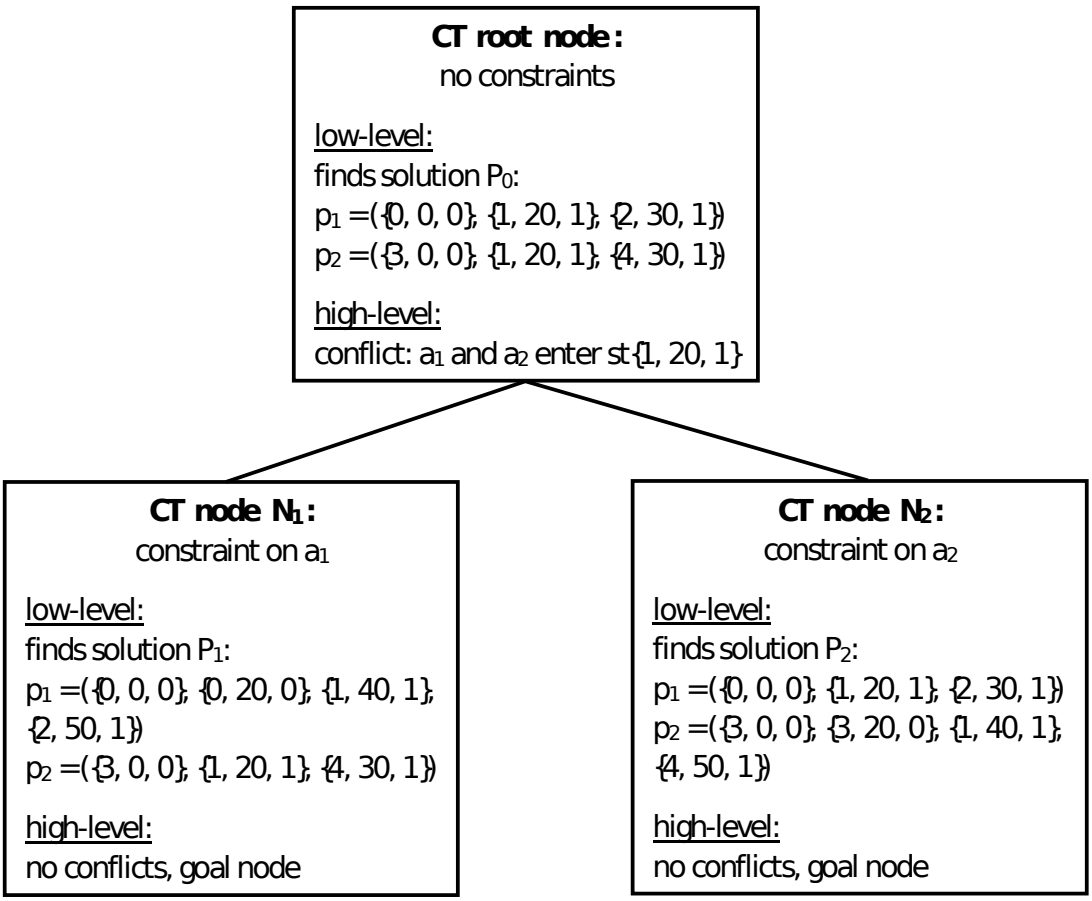

Figure 2.2: Constraint tree demonstrating the interaction between the high-level and low-level search.

2.1.2 Target system

MAPF models and the solutions from these models are typically abstractions of the real world and thus try to imitate it in a more or less simplified manner. As such, the closest model with regards to the real world, can be described with the concept of the target system. In essence, it can be thought of as a subset of the real world, where information is filtered in favor of the most critical and possibly simplified

(e.g., utilizing discrete values) to achieve a satisfactory result. It is necessary, as in the real world, many dynamic forces are constantly involved in the movements of the agents that change continuously. Concerning MAPF, the more interesting ones are the kinematic: the velocity, acceleration, and displacement of the agents, and spacial: the distance and positioning between the agents. This is contrary to other physical forces such as gravity, friction between the tires and the road, or the air resistance force applied to a moving agent, to name a few examples. The presented CTDS model is also a target system, but optimal solutions here might not translate well to more complex target systems that are a step closer to the real world. For this reason, in this project, we also consider a target system that can take a solution from the CTDS model, and apply an optimization and transformation step to translate it to a fully continuous solution that can be simulated on a game engine [30].

2.2 Parameter selection problem

The automatic parameter selection problem is a variation of the automatic algorithm selection problem [31]. For our case, we define the problem by a tuple $(\mathcal{I}, \mathcal{S}, \mathcal{C})$, where:

- $\mathcal{I} = \{i_1, .., i_n\}$ is a set of MAPF problem instances,
- $\mathcal{S} = \{speeds_1, .., speeds_m\}$ is a set of set of speeds that can be used to solve each instance $i \in \mathcal{I}$, and
- $\mathcal{C} : \mathcal{I} \times \mathcal{S} \to \mathbb{R}$ is a function that returns the cost of the solution when solving problem instance i with a speed set $speeds \in \mathcal{S}$.

For this problem, the solution is a mapping $\pi : \mathcal{I} \to \mathcal{S}$ that maps each problem instance $i \in \mathcal{I}$ to a specific set of speeds $speeds \in \mathcal{S}$. The total cost of the solution $\mathcal{T}(\pi)$ is the sum of the individual costs of the solutions for each problem instance:

$$\mathcal{T}(\pi) = \sum_{i \in \mathcal{I}} \mathcal{C}(\pi(i), i) \tag{2.3}$$

In this setting, the optimal solution is defined as:

$$\pi^* = \operatorname*{argmin}_{\pi} \mathcal{T}(\pi) \tag{2.4}$$

Meaning, for each problem instance $i \in \mathcal{I}$ the optimal mapping π^* always chooses the speed set $speeds \in \mathcal{S}$ that results in the smallest individual cost $\mathcal{C}(\pi(i), i)$. Accordingly, it also results in the smallest total cost $\mathcal{T}(\pi)$.

2.3 Machine learning

In order to develop a mapping π, we will look towards machine learning to solve the task of classification. Machine learning is known as the ability to utilize raw data by extracting patterns from it by an artificial intelligence system to acquire their own knowledge [32, Chapter 1]. In classification, a learning algorithm - capable of learning from data - is tasked to identify into which available category some

input belongs [32, Section 5.1.1]. To approach this task, we present an overview of neural networks (NNs) to better understand AlexNet [33], a pretrained Convolutional Neural Network, that we will be working with by utilizing transfer learning. A more detailed view into the subject can be found in the book "Deep Learning" by Goodfellow et al. [32].

2.3.1 Neural networks

Deep feedforward neural networks are typical machine learning models, intending to approximate some function f to map an input x to an output y. The goal with this mapping $y = f(x; \theta)$ is to learn the values of the parameters θ to achieve the best function approximation. The feedforward property comes from the fact that information flows from x, through intermediate computations defining f, to the output y. NNs are called networks as they are commonly composed of many different functions that define these intermediate computations, giving structure to the overall network. These functions are also known as layers, which determine the depth of the model. As an illustration, Figure 2.3 represents an abstract neural network. Lastly, the term *neural* is inspired by neuroscience [32, Chapter 6].

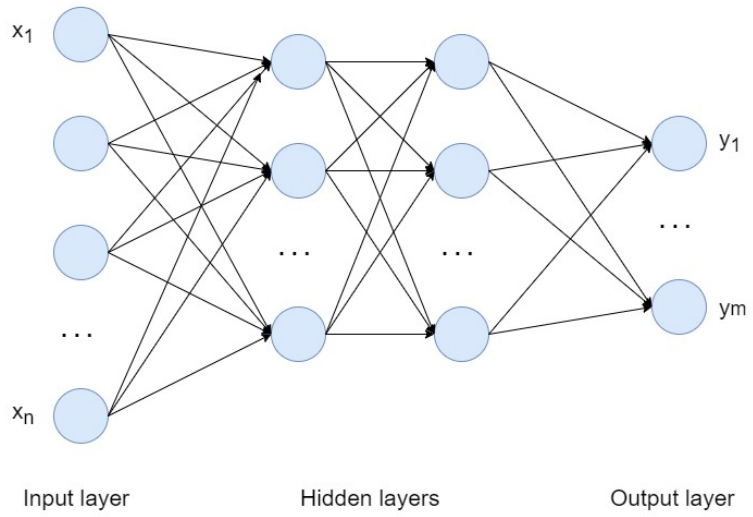

Figure 2.3: An abstract neural network, with four layers. The middle layers are referred to as hidden since the training data does not reveal the desired output for them [32, Chapter 6].

2.3.1.1 Convolutional Neural Networks

Convolutional Neural Networks are deep learning architectures further inspired by visual perception mechanisms found in nature [16, Section 1]. They are specialized for handling data with a known grid-like topology, such as image data - a two-dimensional grid of pixels [32, Chapter 9]. Fundamental CNN components include three types of layers: convolutional, pooling, and fully-connected [16, Section 2].

The convolutional layer consists of multiple convolution kernels to compute corre-

sponding feature maps of feature representations of the inputs. The kernel can be perceived as a small filter that represents some visual feature. It is applied over all the spatial locations of the input and learned during the training of the network. In detail, a neuron in the feature map is connected to adjacent neurons in the previous layer - known as the receptive field of the neuron. The feature map is the results of the convolution of the input with a kernel that is passed through an element-wise nonlinear activation function. Such activation functions are advantageous for multi-layer networks as they allow for the detection of nonlinear features [16, Section 2]. One of the most common and recommended [32, Chapter 6] activation function to use is the rectified linear unit (ReLU) [34] defined by $y = max(0, x)$.

The pooling layer is usually between two convolutional layers and downsizes the resolution of the feature maps to achieve shift-invariance [16, Section 2]. Meaning, it can be seen as a summary statistic of neighboring neurons to help make the input invariant to translations, i.e., more robust in terms of detecting the presence of features rather than the exact pixel-perfect positioning of features. One commonly used pooling operation is max pooling [35], where the maximum output of a rectangular neighborhood is reported [32, Section 9.3].

Fully-connected layers connect all neurons in a previous layer to all neurons in the current layer. With this, the aim is to perform high-level reasoning and generate global semantic information. Additionally, the last layer of the CNN is the output layer, which is commonly a softmax operator for classification tasks. From here, the optimum machine learning parameters θ can be learned by minimizing a suitable loss function [16, Section 2].

2.3.1.2 Splitting of the data

The main challenge in machine learning is generalization [32, Section 5.2], i.e., to perform well on previously unseen data. When training a machine learning model, the data is typically split into training, validation, and test sets. The training set is the dataset that is used for learning the internal parameters θ. Moreover, during training, the model is evaluated with the help of a validation set. This provides feedback on the initial success of the training and helps guide the tuning of hyperparameters, such as the learning rate. After the model is completely trained, the performance is measured on a test set of previously unseen data. How this splitting of the data is done percentage-wise depends on the problem. Datasets with millions of samples might only need a fragment for validation and testing, while on smaller sets, the ratio between the different sets needs to be smaller. An important assumption on the overall training and evaluation process is that all the data samples are independent from each other, and furthermore, that the different datasets are identically distributed [32, Section 5.2].

2.3.2 Transfer learning

Transfer learning refers to the application of knowledge from one setting to better generalize in another setting [32, Section 15.2]. In the context of supervised learning - where each example in the dataset is associated with a target or a label [32, Section

5.1.3] - it can be understood as a situation where the input is the same, but the target is changed between the two settings. We apply this practice with the use of a pretrained CNN model known as AlexNet [33] by one of the authors Alex Krizhevsky. The attribute *pretrained* is used as a description because it can be seen as a first step of the whole training process [Section 15.1][32]. In the case of AlexNet, it has been trained on the ImageNet dataset [36], consisting of over 15 million images labeled roughly into 22 thousand different categories. The advantage of using a pretrained CNN lies in the fact that if there is significantly more data in the first setting, then that can help to generalize from only very few examples in the second setting [32, Section 15.2]. Thus, decreasing the need for a vast dataset in the second setting, which in our case is in the MAPF domain.

AlexNet [33] consists of multiple basic CNN components, described in Section 2.3.1.1, that are applied sequentially. The Listing 2.1 provides an overview of the structure that was used in this project. In general, the input image is processed through the convolutional layers (nn.Conv2d), an activation function (nn.ReLU), and pooling layers (nn.MaxPool2d). These modules are grouped into a sequential container (nn.Sequential). A fully-connected layer (nn.Linear) is used eventually to connect the information to the possible class labels. For example, if the classification is defined between five possible labels, then the last layer would consist of five neurons. Lastly, a softmax operation [32, Section 4.1] (nn.Softmax) can be used to associate the result of the computation to class probabilities in the range $[0, 1]$, and with a total sum of 1.

Code Listing 2.1: Pseudocode of AlexNet based on the PyTorch library [37].

```
self.features = nn.Sequential(
    nn.Conv2d(..),
    nn.ReLU(..),
    nn.MaxPool2d(..),
    nn.Conv2d(..),
    nn.ReLU(..),
    nn.MaxPool2d(..),
    nn.Conv2d(..),
    nn.ReLU(..),
    nn.Conv2d(..),
    nn.ReLU(..),
    nn.Conv2d(..),
    nn.ReLU(..),
    nn.MaxPool2d(..))
self.classifier = nn.Sequential(
    nn.Linear(..),
    nn.Softmax(..))
```

Moreover, as we are dealing with unbalanced datasets, then the weights for the different classes in the machine learning model can be adjusted accordingly. For example, assume a context of two classes, X and Y, with $n_X = 1000$ and $n_Y = 2000$ samples. In this case, we would want the samples in the class that has fewer

datapoints (class X) to have more weight in the learning process. By default, these weights are set to one. In order to balance an uneven distribution of the data, the weights of any class C can be changed to the value of dividing the size of the largest class to the size of class C. Accordingly, in this example, class X would get a weight of $\frac{n_Y}{n_X} = 2$ and class Y would get $\frac{n_Y}{n_Y} = 1$. This technique is known as weight balancing [38].

3

Approach

We approach the parameter selection problem for MAPF as an image classification problem, drawing motivation from Sigurdson et al. [15]. An overview of this process is given in Figure 3.1. The set $\mathcal{I}$ of MAPF problem instances will consist of vehicular road network scenarios, described in Section 4.2. To train the machine learning model and learn the mapping π, we pre-process each MAPF problem instance $i \in \mathcal{I}$ into a form of an image. Additionally, we define a set of possible labels that correspond to speed sets from the set $\mathcal{S}$ via the classification $\pi(i)$, in order to be used by the MAPF solver.

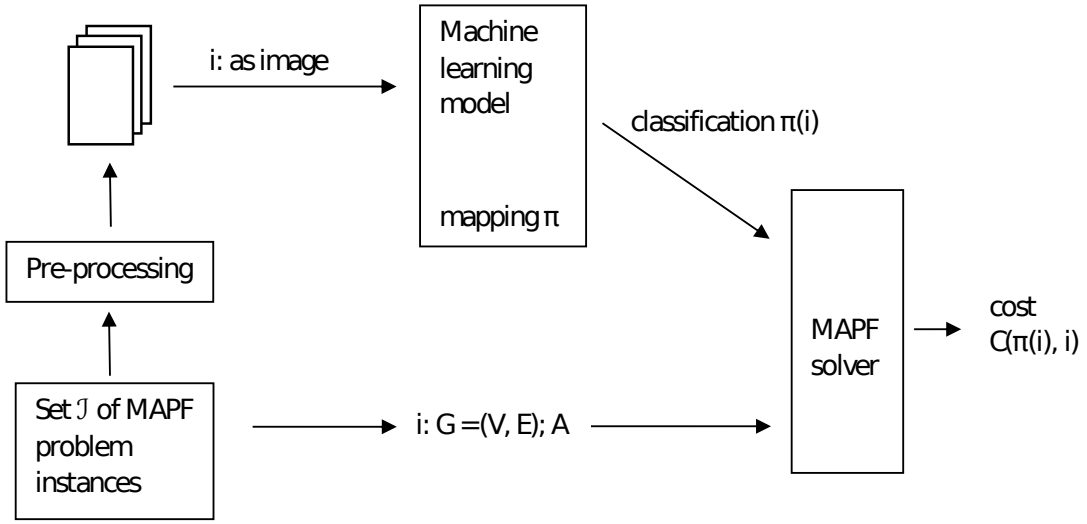

Figure 3.1: Illustration of the workflow between the different components.

Once we have a classification $\pi(i)$ for a given instance $i \in \mathcal{I}$, we can find the cost of a single solution, defined as:

$$\mathcal{C}(\pi(i), i) = w \cdot MS_{avg} + (1 - w) \cdot |\pi(i)| \tag{3.1}$$

where MS_{avg} denotes the average makespan of the MAPF solution P_i:

$$MS_{avg} = \frac{SIC(P_i)}{|A_i|} \tag{3.2}$$

Here, $|A_i|$ is the cardinality of set A_i, i.e. the number of agents within a given problem instance $i \in \mathcal{I}$. Similarly, $|\pi(i)|$ is the cardinality of the assigned speed set $speeds \in \mathcal{S}$ for a given problem instance $i \in \mathcal{I}$.

The additional weight parameter $w = [0, 1]$ allows for balancing the ratio between the two addends in Equation 3.1. The first one focuses on lowering the average makespan, a direct measure of the real cost. While the second one imposes a penalty on the number of speeds, which affects the search space of the solver and, thus, the time it takes to find a solution.

It follows that the total cost of the solution $\mathcal{T}(\pi)$ (Equation 2.3) can be then expanded to:

$$\mathcal{T}(\pi) = \sum_{i \in \mathcal{I}} (w \cdot MS_{avg} + (1 - w) \cdot |\pi(i)|) \tag{3.3}$$

which is the function we will be looking to minimize.

When comparing the two extreme cases, $w = 1$ and $w = 0$, then a value must be found that allows to generate a dataset where one class does not dominate over the others in terms of sample size. In the former case, when $w = 1$, the class with the most speeds would always have an equal or better SIC value, as more speeds allow for more improvement possibilities for the resulting MAPF solution. Thus, because we are only looking at the average makespan, all instances would be classified into the same class - the one with the biggest number of speeds. In the latter case, when $w = 0$, the class with the least speeds would always have a smaller total cost $\mathcal{T}(\pi)$, because all the focus is on the number of speeds that is used. Thus again, all instances would be classified into the same class - the one with the smallest number of speeds.

With this approach, we expect for the machine learning model to be able to identify important features from the input images and to apply them automatically. These features might include, for example, the specific graph setting, and the positioning of the agents.

3.1 Pre-processing

Pre-processing is used to allow for the machine learning model to make use of datasets comprising of MAPF problem instances.

In this preliminary step, an instance, represented by a graph G and a set of agents A, is translated to an image. Additionally, the image is resized to match the input of the machine learning network, 227×227 pixels. By doing so, we enable the machine learning model to use such an image as input.

With this translation, we lose specific individual information of the agents as the

connection between a specific agent's start and goal nodes is anonymized. At the same time, we gain knowledge of the topological positioning between the agents. This is done to better enable the machine learning model to focus on the topological information between the agents as it is highly relevant to the proposed visual approach. Additional features come with the risk of making the visual MAPF problem instance representation excessively detailed, potentially overshadowing the topological features of the input. Additionally, when the instance representation grows more and more detailed, then the demand for a more extensive dataset is also increased as a more complex representation would be harder for the machine learning model to make use of.

In the pre-processed images, a starting position of an agent is illustrated with a green node, a goal location with a red node, and all other nodes are grey. Figure 3.2 displays two such images of problem instances that are produced after pre-processing.

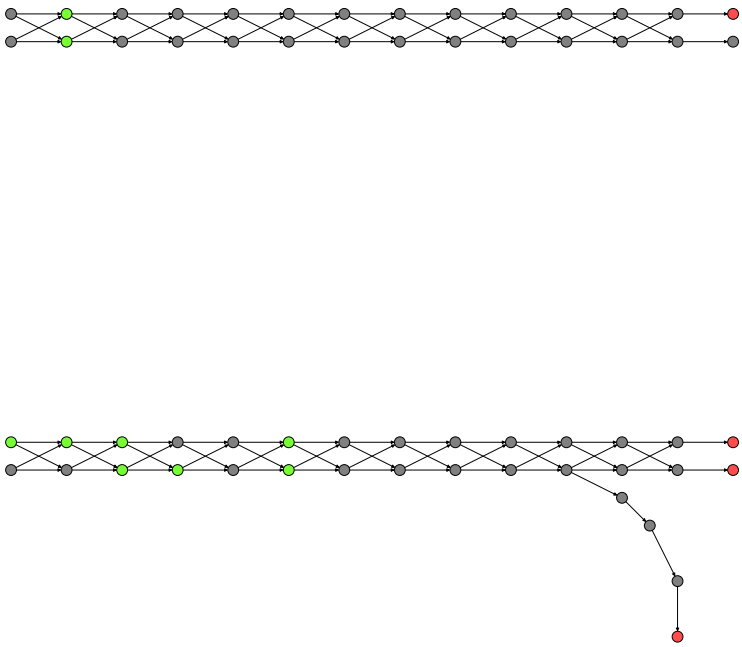

Figure 3.2: Two samples of pre-processed MAPF problem instances. Upper: highway scenario with two agents. Lower: highway exit scenario with seven agents. Starting locations (green nodes) and goal locations (red nodes) of the agents are represented anonymously.

4

Evaluation

This chapter provides the context for the evaluation of our approach by first stating the proposed research questions. Secondly, the different vehicular road traffic scenarios are presented on which the MAPF problem instances are based on along with the relevant parameters of the experiments. Additionally, the evaluation criteria as well as the experiment plan are given to tackle the established research questions. Lastly, the evaluation environment is described.

4.1 Research questions

We evaluate our automatic parameter selection π approach by investigating multiple experiments. First, we look at each of the road scenarios, described in Section 4.2, individually, and train separate models to learn a graph-specific mapping. Then, we form combinations of datasets to train more general models. With this approach, we aim to answer the following research questions:

- Q1. Can a learned mapping perform better than any of the deterministic mappings, i.e., only using a specific set of speeds? If the learned mapping can achieve consistently better results, then this would be a clear indicator that the machine learning approach is working.
- Q2. Would a machine learning model achieve better results when trained on a combined dataset compared to a model only trained on a single graph-specific dataset? With this, we aim to see if the learned mapping π can be made more robust in terms of the MAPF instances to generalize better.
- Q3. Is it worthwhile to learn a mapping between two different speed sets - one with two speeds and the other with four speeds - for the automatic parameter selection? Here, an answer would allow arguing for or against this approach, depending on the application field.
- Q4. How does the cost of a single solution $\mathcal{C}(\pi(i), i)$ (Equation 3.1) of a single MAPF problem instance i compare to the cost in the target system? As $\mathcal{C}(\pi(i), i)$ is a penalty-based function, a correlation between the two would give a hint on how the machine learning model would perform in a more realistic setting. Furthermore, it can help argue if the agents' behavior from using more speeds in the CTDS model [17] can be translated to improved behavior in the target system.

4.2 Test cases

To experiment with the proposed approach, five different road scenarios were designed and implemented: highway, highway exit, highway entrance, roundabout, and intersection. Each road component is defined as a graph $G = (V, E)$, where each node $v \in V$ represents a location on the road. Such nodes are connected via directed edges $(v, v') \in E$ that allow agents to traverse between different locations in order to reach their goal nodes. As a result, these graphs allow for the robust representation of everyday road traffic situations within the CTDS model. Additionally, a standalone 8x8 grid case was also considered.

In the following, we will introduce the graphs in detail. Universally, each road graph incorporates 16 possible start nodes (displayed in blue) and specific goal nodes (displayed in red), enabling us to generate MAPF problems within a fixed range in terms of the number of agents. At the same time, we can ensure that in each generated instance, the start and goal nodes of the agents are connected, and thus, the MAPF solution encodes a logical path in the road scenario. Naturally, these graphs do not represent all scenarios that arise in traffic but rather try to exemplify typical road sections that are part of a bigger road network that can consist of multiple similar fundamental parts.

4.2.1 Highway

The highway graph can be seen in Figure 4.1. It represents a one-way two-lane road, where lane vertices are connected by

1. *forward edges* if both vertices are on the same lane, and
2. *switch edges* if they connect vertices on different lanes.

This setup allows agents to perform overtaking maneuvers to reach their assigned goal nodes. These goal nodes are only connected via forward edges in order to regard the agents who have reached their goal vertex as they would have exited the highway by the MAPF solver.

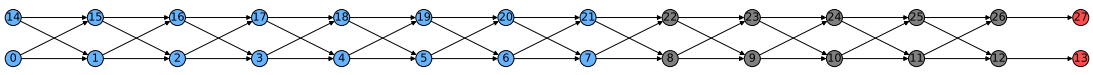

Figure 4.1: Highway graph, with 16 starting nodes (in blue) and 2 possible goal nodes (in red).

4.2.2 Highway exit

Similar to the highway, the highway exit graph can be seen in Figure 4.2. In addition to all the highway aspects, it also incorporates an off-ramp from the main highway allowing agents to possibly exit towards an added goal node.

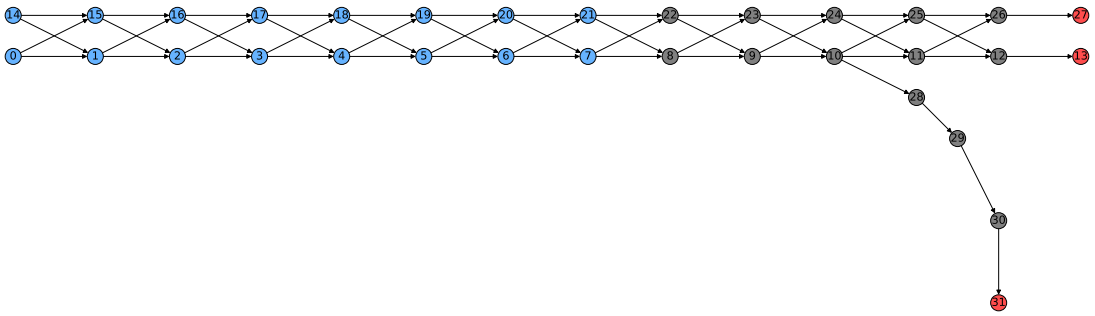

Figure 4.2: Highway exit graph, with 16 starting nodes (in blue) and 3 possible goal nodes (in red).

4.2.3 Highway entrance

Also similar to the highway, the highway entrance graph can be seen in Figure 4.3. Contrary to the highway exit, the highway entrance models an on-ramp to the main highway allowing agents to possibly merge into ongoing traffic.

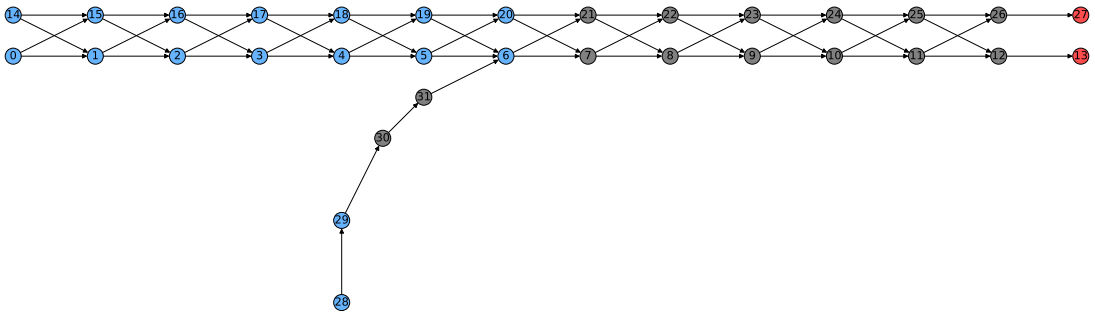

Figure 4.3: Highway entrance graph, with 16 starting nodes (in blue) and 2 possible goal nodes (in red).

4.2.4 Roundabout

The roundabout graph, shown in Figure 4.4, consists of four incoming two-lanes roads, that represent ending segments of the highway graph. Additionally, there are four outgoing one-way exit lanes, that model the possible exits off the roundabout. When assigning agents with possible start and goal nodes, then there are no limitations for this selection, as all goal locations are reachable from all start locations. Figure 4.5 depicts this relation to the reachable nodes for one of the incoming lanes. This is similar for the other incoming lanes. Additionally, extreme maneuvers while on the roundabout are not allowed. Specifically, when an agent is in the inner lane of the roundabout, it can only exit this inner lane when going towards the nearest immediate goal node. An example of this is presented in the caption of Figure 4.4.

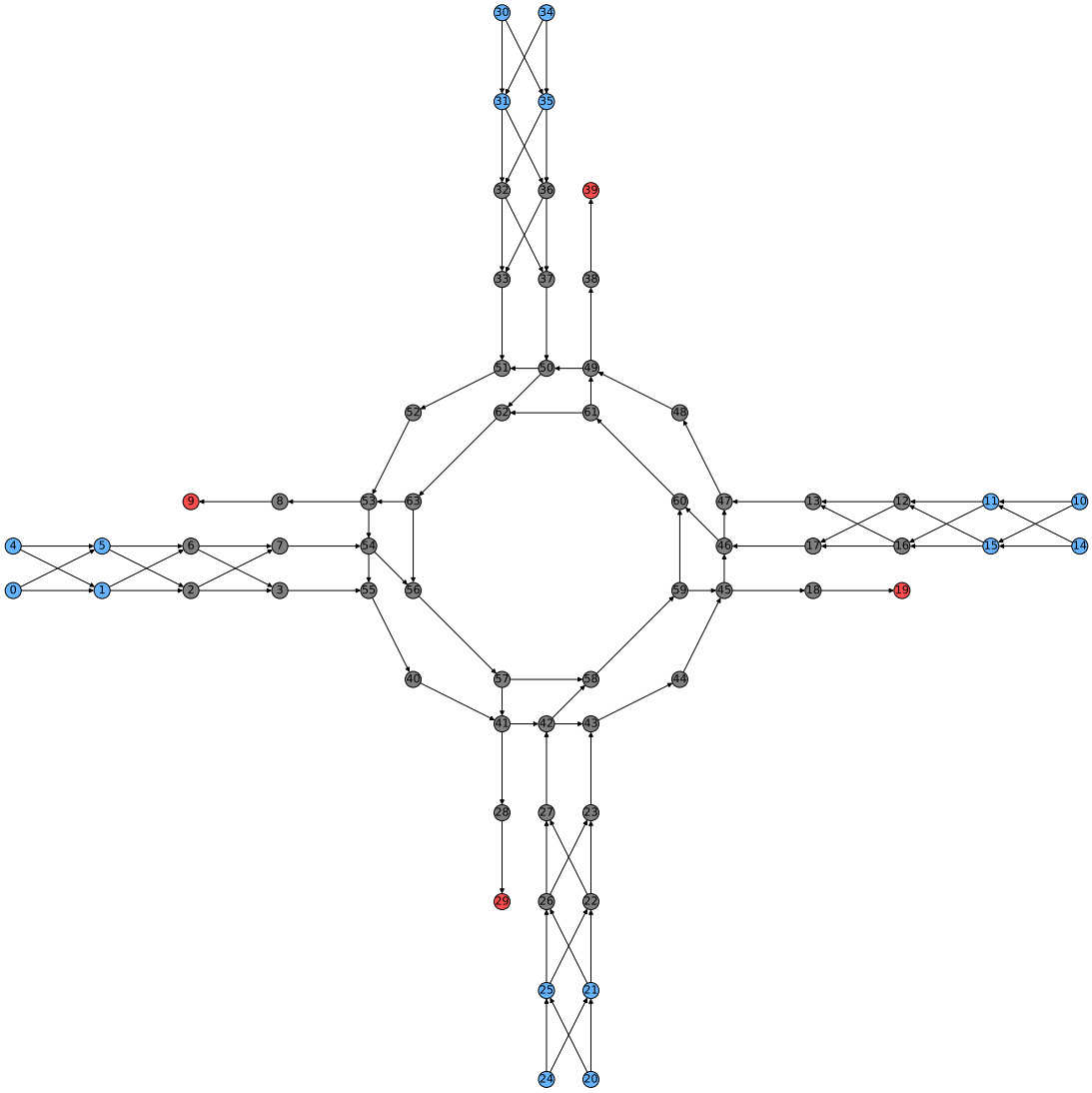

Figure 4.4: Roundabout graph, with 16 starting nodes (in blue) and 4 possible goal nodes (in red). Extreme maneuver example: if an agent starts at node 0 and has a goal node 39, then the agent can not traverse the following path segment: $59 \rightarrow 45 \rightarrow 46$ to to reach the goal node. Exiting the inner lane of the roundabout (represented by nodes 56-63) is only allowed when moving towards the nearest goal node.

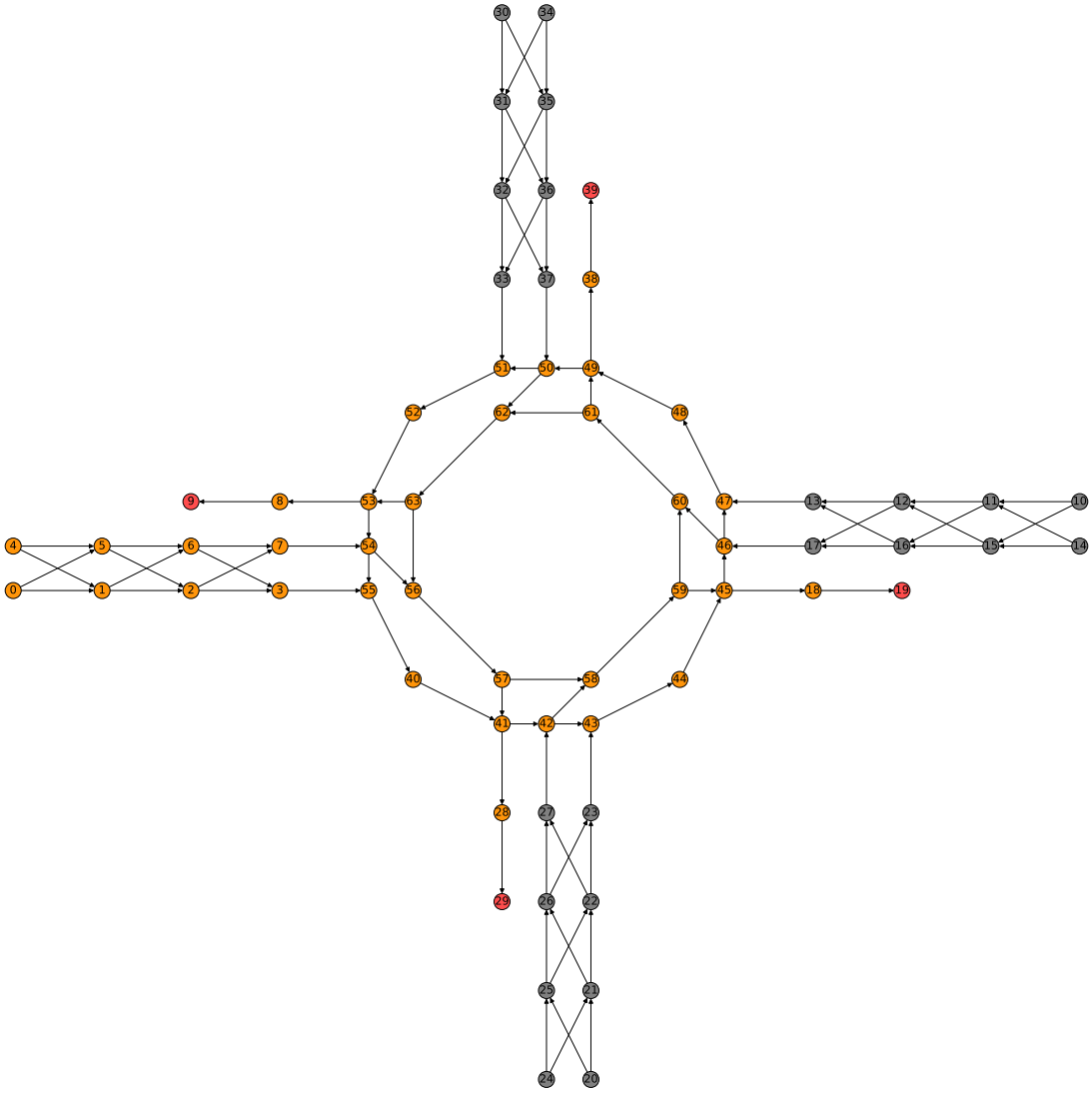

Figure 4.5: Roundabout graph, highlighting (in orange) the reachable nodes for one of the incoming lanes to reach the goal nodes (in red). Other lanes are identical.

4.2.5 Intersection

The intersection graph, shown in Figure 4.6, also consists of four incoming two-lanes roads, that represent ending segments of the highway graph. These lanes intertwine in the middle, and ultimately form connections to the goal nodes by paths that either go straight, left, or right when looking from the perspective of the incoming lane. Here, the possible goal nodes for agents depend on the incoming lane. Figure 4.7 highlights this relation for one of the incoming lanes, but it is similar to all other lanes as well. Additionally, similar to the roundabout, extreme maneuvers on the intersection are also not allowed. Meaning, sudden 90 degree turns when crossing the intersection are disallowed. This is enforced in the CTDS model by checking the angle of the next possible action. In detail, consider the case where an agent arrived from a previous node to the current node, and is considering moving to the next

node. Then, the angle forming between the *previous-current-next* nodes determines the validity of the considered action. An example of this is presented in the caption of Figure 4.6.

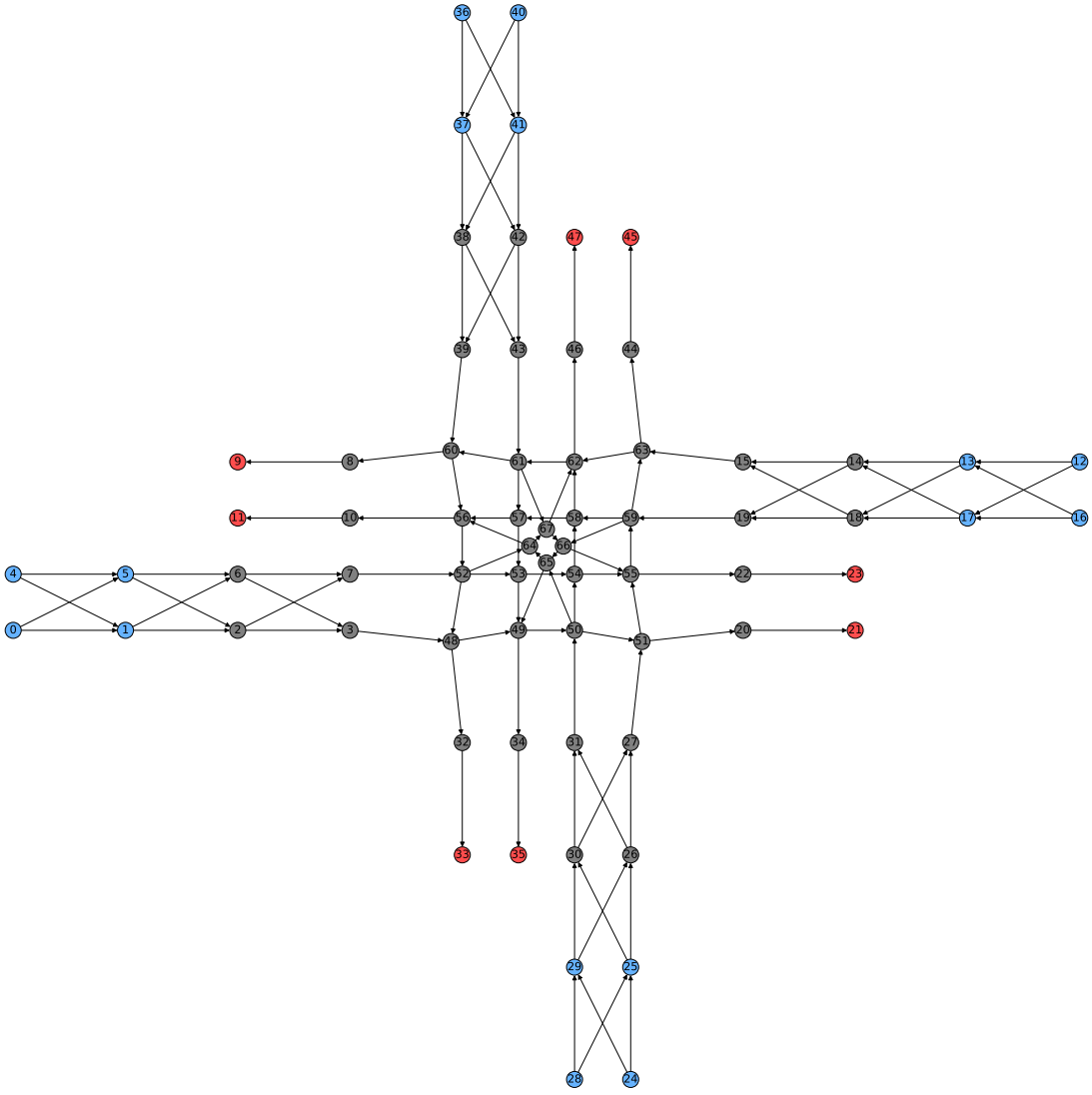

Figure 4.6: Intersection graph, with 16 starting nodes (in blue) and 8 possible goal nodes (in red). Extreme maneuver example: if an agent starts at node 0 and has a goal node 23, then the lane switch can not happen within the following path segment: $49 \rightarrow 50 \rightarrow 54 \rightarrow 55$.

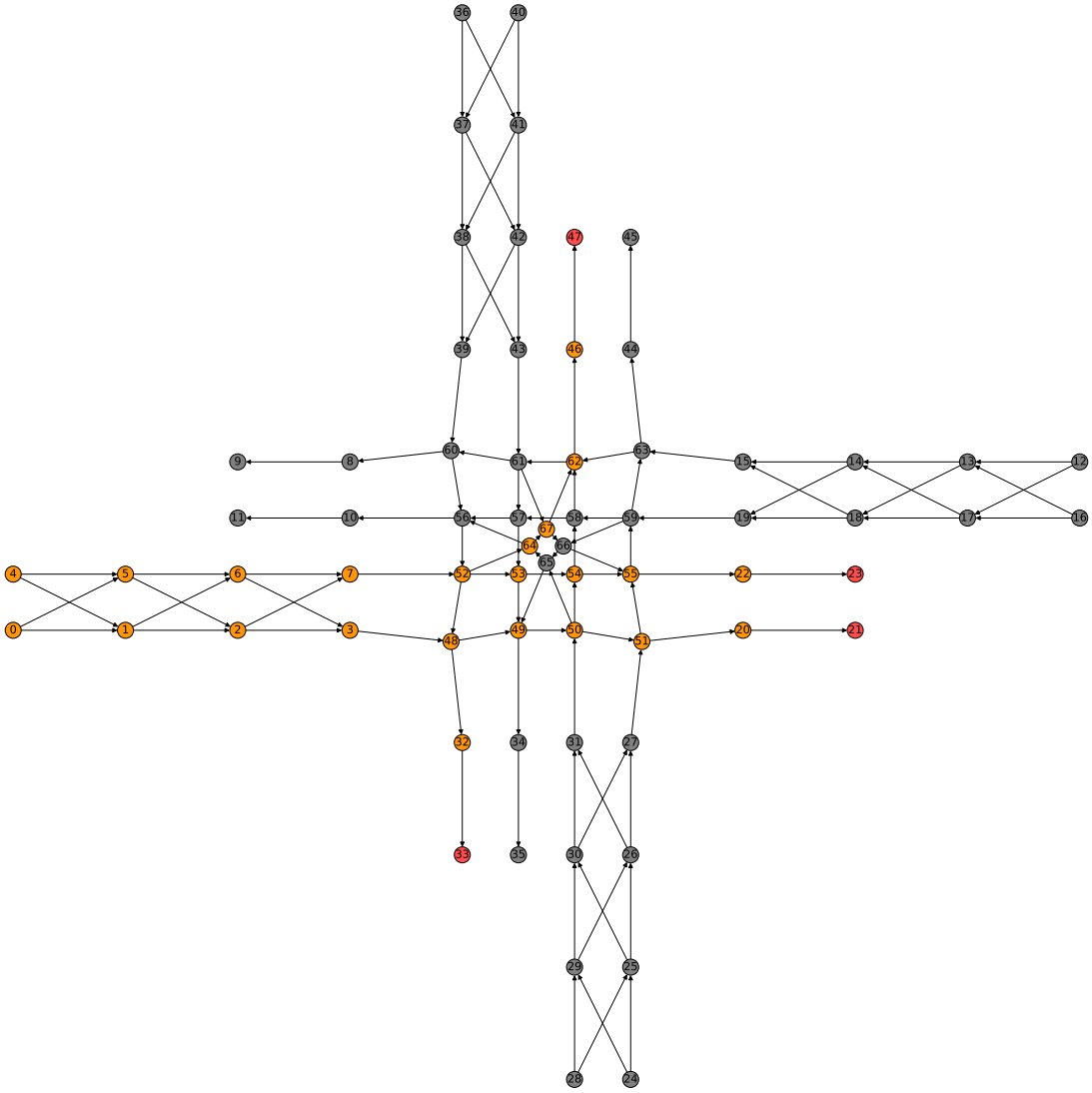

Figure 4.7: Intersection graph, highlighting (in orange) the reachable nodes for one of the incoming lanes to reach the goal nodes (in red). Other lanes are identical.

4.2.6 Grid

The 8x8 grid can be seen in Figure 4.8. In this graph, all nodes are connected to all adjacent neighboring nodes, allowing agents to move between them. Also, a connection between any two nodes v_1 and v_2 represents two directed edges (v_1, v_2) and (v_2, v_1).

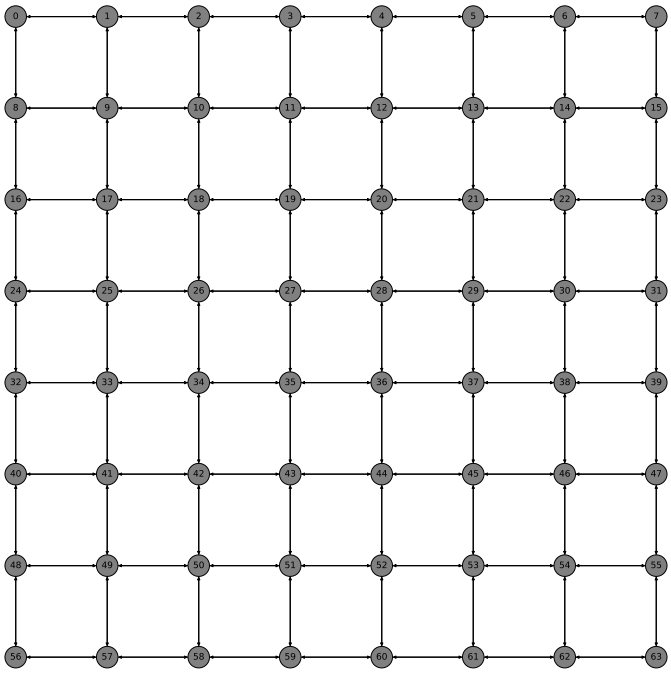

Figure 4.8: Grid graph, with 64 nodes where each node is connected to all four, three, or two neighboring nodes in the case of an inner, edge, or corner node, respectively.

4.3 Parameters

4.3.1 Number of agents

In a similar setting as to our highway graph, it has been shown that there is potential for small performance gains with more than two speeds [17]. For this, the MAPF problem instances should not be too *overcrowded*. By this, we mean that the number of agents should not be too high in order to preserve meaningful topological positioning between the agents. For example, consider the situation when we have 16 agents, the maximum in our setting, and all the starting positions are occupied by these 16 agents. Because of anonymous mapping between a specific agent's start and goal nodes, all such instances would be equivalent to the machine learning model. In fact, the pre-processed input images would appear exactly the same, even if the individual agents differ across the various instances. This would have a significantly negative effect on the learning capability of our proposed models.

Additionally, because we are working with an optimal MAPF solver, the time it takes to find a solution increases exponentially with the number of agents. At some point, the time required to find a solution is too much to produce an adequately sized dataset in a reasonable timeframe, as the number of agents grows too high. Based on previous research by Sigurdson et al. [15], we can reason by analogy that the dataset used for learning by the machine learning model should be in a similar range of 140 samples for each graph and number of agents. Also, we would want to have the same number of samples between the different numbers of agents within a

graph for a uniform assessment. Therefore, a limit had to be decided upon, and for our experiments, we fixed the size of the set of agents $A = \{a_1, .., a_k\}$ in the range $k = [2, 7]$ for the different road scenarios. For the standalone grid test case, the range $k = [5, 13]$ was analyzed.

4.3.2 Set of speeds

The experiments focus on learning a mapping between using two different sets of speeds on the suggested road scenarios. From the parameter selection problem (Section 2.2), we define $\mathcal{S} = \{speed_A, speed_B\}$ as:

- $speed_A = \{s_{min}, s_{max}\}$, and
- $speed_B = \{s_{min}, \frac{1}{3}s_{max}, \frac{2}{3}s_{max}, s_{max}\}$,

where $s_{max} = 10$.

Correspondingly, this defines the possible labels of the classes A and B for classification.

4.3.3 Cost function weight

The weight parameter w from Equation 3.1 determines the balance between the SIC and the assigned penalty. Moreover, it also dictates the distribution of the generated data between the available classes. As such, it has to be chosen carefully to allow for a reasonably balanced dataset to be formed. This allows the machine learning model to learn better, as it can train on more samples from each possible class. Otherwise, the learning process is hindered critically, if one class is too dominant or severely underrepresented. Ultimately, we set the weight parameter $w = 0.95$ as it produced an acceptably balanced final dataset allowing for the representation of different instances that benefit from fewer or more speeds. Thus, providing a setting where dynamic mapping could present an advantage over a fixed parameter in the considered graphs.

4.4 Evaluation criteria

As our evaluation criteria, we will use the total cost function $\mathcal{T}(\pi)$ from Equation 3.3. The learned mapping π, and the corresponding cost $\mathcal{T}(\pi)$ will be compared to:

- deterministic mappings, i.e., always using a specific speed set from $\mathcal{S}$ and thus, labeling all instances as belonging only to class A or B (corresponding to mappings π_A and π_B, respectively),
- π^*, the optimal solution, which always selects the best speed set for every instance, and
- a *worst* mapping, which always selects the worst speed set.

Additionally, we will correlate the costs from the CTDS model to the costs in the target system. To achieve this, a subset of generated samples will be solved for the

target system allowing for a comparison between the relevant costs. In addition to the single cost $\mathcal{C}(\pi(i), i)$ of a sample i we will look at the SIC time (Equation 2.1) of the solution in the CTDS model and the target system.

4.5 Experiment plan

The experiment plan is focused on investigating how a learned mapping π performs when trained on different datasets. The experiments consider datasets comprised of MAPF instances from a single graph and multiple graphs. We refer to the resulting learned mappings trained on these datasets as per-graph and multi-graph mappings, respectively. In detail, we will train and evaluate different mappings with the following experiments.

- **Per-graph mappings.** In these experiments, a dataset is generated (see discussion in Section 4.6) on each of the test cases, described in Section 4.2. The per-graph mappings are trained only on datasets of MAPF problem instances from a single graph and, thus, show the performance of a graph-specific machine learning model. Once a model is trained, a test set is used to compare the total cost values across the different mappings. The optimal mapping π^* (Equation 2.4) and the corresponding total cost $\mathcal{T}(\pi^*)$ (Equation 3.3) represent a perfect model, which always chooses the correct class label for each instance. Additionally, it serves as a lower bound for the machine learning model's performance as the correct label translates to the speed set that achieves the lowest individual cost for each problem instance. We are interested in the learned mapping π and how it compares to the deterministic mappings (π_A and π_B) to answer the research questions Q1 and Q3, established in Section 4.1. The per-graph mappings are based on datasets on the following test cases from Section 4.2, with a reference experiment name given in parenthesis:
  - grid (**Exp. G**),
  - highway (**Exp. H**),
  - highway exit (**Exp. E**),
  - highway entrance (**Exp. N**),
  - roundabout (**Exp. R**), and
  - intersection (**Exp. I**).
- **Multi-graph mappings.** In these experiments, the generated datasets from the per-graph experiments are combined to form more diverse datasets consisting of MAPF problem instances from multiple graphs. The aim of this is to show the performance of a potentially more robust machine learning model than in the previous experiments. Again, once a model is trained, a test set - this time consisting of instances from multiple graphs - is used to compare the total cost values across the different mappings. Similarly, the optimal mapping π^* (Equation 2.4) and the corresponding total cost $\mathcal{T}(\pi^*)$ (Equation 3.3) represent a perfect model, which always chooses the correct

class label for each instance. Additionally, it serves as a lower bound for the machine learning model's performance as the correct label translates to the speed set that achieves the lowest individual cost for each problem instance. We are interested in the learned mapping π and how it compares to the deterministic mappings (π_A and π_B) to answer the research questions Q2 and Q3, established in Section 4.1. The multi-graph mappings are based on datasets on the following combination of test cases from Section 4.2, with a reference experiment name given in parenthesis:

  - highway, highway exit, and highway entrance (**Exp. HEN**),
  - roundabout and intersection (**Exp. RI**), and
  - all five test cases together (**Exp. HENRI**).

  The combinations for Exp. HEN and Exp. RI are motivated by the fact that test cases on which the MAPF problem instances are generated on share a similar structure. The motivation for Exp. HENRI lies in the interest to see how the dynamic selection performs when this is not the case and is trained on the complete dataset.

- **Comparison with target system** In the interest of making a preliminary calibration with the target system (see discussion in Section 2.1.2), a subset of samples will we randomly examined from the best performing per-graph mapping dataset. As the total cost $\mathcal{T}(\pi)$ for a set of instances and the single solution cost $\mathcal{C}(\pi(i), i)$ for an individual instance i are values from a penalty-based function set in the CTDS model, a comparison is needed between the SIC (Equation 2.1) values in the CTDS model and the target system. We are interested in this correlation to answer the research question Q4, established in Section 4.1.

4.6 Data generation

In order to generate datasets composed of MAPF problem instances, agents are assigned random start and goal vertices, depending on the specific test case from Section 4.2. For the presented road scenarios, the selection of possible start and goal vertices depends on the current graph on which the data generation is being done. When considering MAPF problem instances on the 8x8 grid, this selection is not limited and, thus, all nodes are acknowledged. For each agent within an instance, a random independent start node, along with a random goal node, is chosen. In the road scenarios, goal nodes can be the same between different agents because once an agent reaches the assigned goal node, it is assumed to continue moving and, therefore, as exiting the graph. For the 8x8 grid, this is not the case.

Moreover, each single instance $i \in \mathcal{I}$ is fed to the MAPF solver to find the corresponding individual cost $\mathcal{C}(\pi(i), i)$ (Equation 3.1) of the solutions, depending on the defined speed sets in $\mathcal{S}$. Once the solutions are available, a comparison is made to label the problem instance i as belonging under the speed set $speeds \in \mathcal{S}$ that achieved the lowest cost.

4.7 Evaluation environment

The data generation was implemented and handled on a standalone Intel machine (NUC7i7BNH) with an i7-7567U processor. Also, for finding a single MAPF solution, a timeout limit of 30 minutes was set. The utilized CTDS model and MAPF solver were implemented in the programming language C++.

The data pre-processing and the learning of the different mappings were coded in the programming language Python on a mid-range computer. For the machine learning model, we used the PyTorch [37] machine learning library's implementation of AlexNet [33]. Here, the output layer of the model was modified to match the possible number of classification labels. Additionally, the weights of the classes were also adjusted according to the used datasets.

4.8 Problem space

When discussing the number of possible MAPF problem instances for the presented graphs, the automatic parameter selection approach can be more favorable over a manual setting, as there is a vast number of possible problem combinations. Only considering two agents already raises the scale of possible problem combinations to 10^3. For example, on the intersection graph with two agents, the first agent has 16 available starting positions, and the second 15. Furthermore, there are four possible goal positions for both agents. Therefore, the number of different combinations is $16 \times 15 \times 4^2 > 10^3$. With seven agents, the scale is in the order of 10^{10}. For example, on the intersection graph with seven agents, again, the first agent has 16 available starting positions, the second 15, until the seventh has 10 possible starting positions available. Ultimately, the number of different combinations is unfathomable as we have $16 \times 15 \times ... \times 10 \times 4^7 > 10^{10}$ possible problem instances.

4.9 Limitations

Lastly, the presented work includes the following know limitations. The automatic parameter selection approach was tested with the use of a single MAPF algorithm because, currently, only one is known that incorporates the use of discrete speeds. Using this approach on other solvers would be an interesting expansion to gain additional knowledge on the potential of the presented method. Also, the number of configurations were limited, as the number of agents and the size of the available set of speeds both grow the search space exponentially for finding MAPF solutions within the CTDS model [17]. Additionally, the execution time for finding a solution was not taken into account but modeled as a uniform penalty across the different instances.

5

Results

In this chapter, we present the results of our parameter selection π approach, which was applied with datasets of MAPF problem instances based on the introduced test cases from Section 4.2. First, we describe the final dataset, which was formed to train and evaluate the machine learning models. Then, we present the results of the per-graph and multi-graph mappings that show that we were able to constantly achieve better or on-par results when compared to the deterministic selections. Lastly, a correlation between the values from the CTDS model and the target system is presented, indicating parallelism between the two as an argument for the proposed approach. The chapter is concluded with an overall discussion.

5.1 Final dataset

In total, the final dataset consisted of 5040 MAPF problem instances, which were formed into training, validation, and test sets, resulting in a 71.4%, 14.3%, and 14.3% split, respectively (see discussion in Section 2.3.1.2). For the highway test cases, we created 140 MAPF problem instances for each number of agents, for a total of 840 samples for each graph. For the roundabout and intersection test cases, this number was adjusted to 210, for a total of 1260 samples for each graph. The training sets in sum composed of 3600 samples, with ultimately 65.5% belonging to class A and 34.5% to class B (see Section 4.3.2). A more detailed overview of the class ratios across the different graphs is given in Table 5.1. An additional 720 samples formed the test sets, on which the results of the learned mapping π are reported.

graph	number of graphs	class A	class B
highway	1	68.3%	31.7%
highway exit	1	68.5%	31.5%
highway entrance	1	60.3%	39.7%
roundabout	1	77.7%	22.3%
intersection	1	53.0%	47.0%
all	5	65.5%	34.5%

Table 5.1: Results of the data generation showing the data composition of the training sets for the different graphs.

The 8x8 grid graph was not accounted for in the final dataset for reasons that are explained in the next section.

5.2 Per-graph mappings

Exp. G: For the experiment Exp. G from Section 4.5, we started by creating 100 MAPF problem instances for each number of agents for the formation of the training set. Here, the expectation was to have a dataset, where each class is sufficiently represented to allow for the training of a machine learning model. However, the data generation revealed that the machine learning approach could not be applied in this case, as all 900 generated instances were classified into class A. Therefore, there was nothing to be gained with the use of dynamic mapping. This result is unexpected, but in hindsight, reasonable, as the possible distances between randomly assigned start and goal nodes can be extremely minimal - even next to each other. In this case, the use of more speeds does not pose any advantage. Furthermore, when there are conflicts between any two agents, then they can be quite easily solved on the grid without needing to consider the use of more speeds. This is because, in most cases, multiple shortest paths exist between any two start and goal node pairs that share the same Manhattan distance. Thus, stopping or speed changes can be avoided - as a solution with conflicts can be discarded for a conflict-free solution, but with the same path length - and again, using more speeds does not present an advantage.

Exp. H: Figure 5.1 refers to the experiment Exp. H from Section 4.5. Here, we expect the machine learning model and the resulting mapping π to perform better than the two deterministic mappings π_A and π_B for a successful result. This is because we assume the machine learning model to make use of the data it is trained on to learn to recognize useful features from the instances. When looking at the differences with the optimal total cost $\mathcal{T}(\pi^*)$, we can see that our model achieved the smallest difference among the different mappings. The learned dynamic selection (π) was able to perform 1.54 times better than the best deterministic selection (π_A) when comparing the differences to the optimal. Therefore, the expectation for this experiment was met as the training of the model produced a successful result.

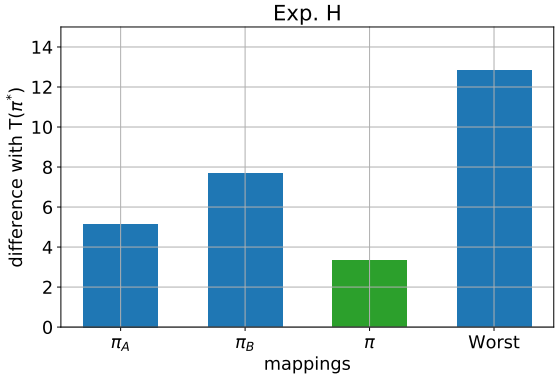

Figure 5.1: Results for Exp. H: difference with the optimal total cost $\mathcal{T}(\pi^*)$ compared to the total costs for the deterministic mappings (π_A and π_B), our model (π in green), and a *worst* mapping.

Exp. E: Figure 5.2 refers to the experiment Exp. E from Section 4.5. Here, we expect the machine learning model and the resulting mapping π to perform better

than the two deterministic mappings π_A and π_B for a successful result. This is because we assume the machine learning model to make use of the data it is trained on to learn to recognize useful features from the instances. When looking at the differences with the optimal total cost $\mathcal{T}(\pi^*)$, we can see that our model achieved the smallest difference among the different mappings. The learned dynamic selection (π) was able to perform 1.76 times better than the best deterministic selection (π_A) when comparing the differences to the optimal. Therefore, the expectation for this experiment was met as the training of the model produced a successful result.

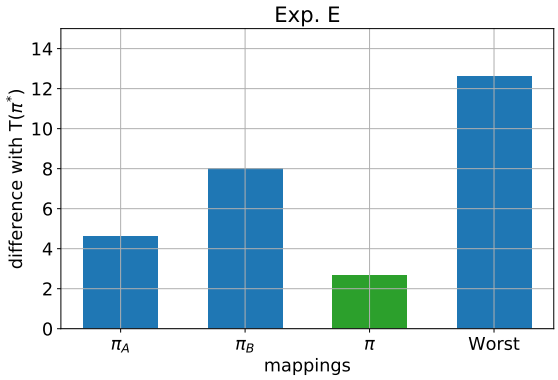

Figure 5.2: Results for Exp. E: difference with the optimal total cost $\mathcal{T}(\pi^*)$ compared to the total costs for the deterministic mappings (π_A and π_B), our model (π in green), and a *worst* mapping.

Exp. N: Figure 5.3 refers to the experiment Exp. N from Section 4.5. Here, we expect the machine learning model and the resulting mapping π to perform better than the two deterministic mappings π_A and π_B for a successful result. This is because we assume the machine learning model to make use of the data it is trained on to learn to recognize useful features from the instances. When looking at the differences with the optimal total cost $\mathcal{T}(\pi^*)$, we can see that our model achieved the smallest difference among the different mappings. The learned dynamic selection (π) was able to perform 2.04 times better than the best deterministic selection (π_B) when comparing the differences to the optimal. Therefore, the expectation for this experiment was met as the training of the model produced a successful result.

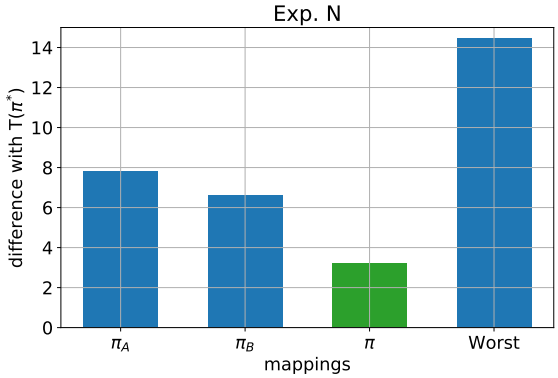

Figure 5.3: Results for Exp. N: difference with the optimal total cost $\mathcal{T}(\pi^*)$ compared to the total costs for the deterministic mappings (π_A and π_B), our model (π in green), and a *worst* mapping.

Exp. R: Figure 5.4 refers to the experiment Exp. R from Section 4.5. Here, we expect the machine learning model and the resulting mapping π to perform better than the two deterministic mappings π_A and π_B for a successful result. This is because we assume the machine learning model to make use of the data it is trained on to learn to recognize useful features from the instances. When looking at the differences with the optimal total cost $\mathcal{T}(\pi^*)$, we can see that our model did not achieve the smallest difference among the different mappings. The learned dynamic selection (π) was 1.05 times worse than the best deterministic selection (π_A) when comparing the differences to the optimal. Therefore, the expectation for this experiment was not met as the training of the model did not produce a successful result. A possible cause for this might be the underrepresentation of class B samples within this dataset (see Section 5.1). Considering a larger dataset might help fix this, but we estimate the reason for this result to lie in the setup of the graph. Meaning, the two circular lanes that permit the anticlockwise flow of agents within the roundabout allow to better avoid conflicts between the agents than in the other test cases. Thus, in this case, using more speeds is not that beneficial as stopping or speed changes can be better avoided.

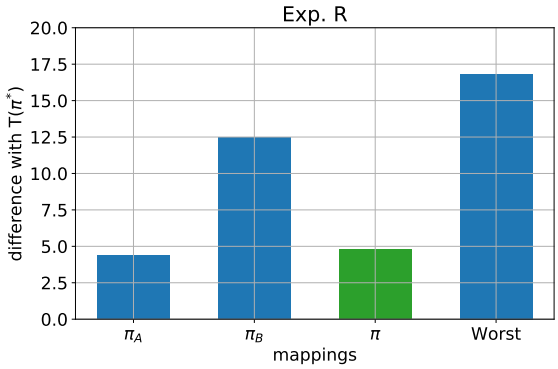

Figure 5.4: Results for Exp. R: difference with the optimal total cost $\mathcal{T}(\pi^*)$ compared to the total costs for the deterministic mappings (π_A and π_B), our model (π in green), and a *worst* mapping.

Exp. I: Figure 5.5 refers to the experiment Exp. I from Section 4.5. Here, we expect the machine learning model and the resulting mapping π to perform better than the two deterministic mappings π_A and π_B for a successful result. This is because we assume the machine learning model to make use of the data it is trained on to learn to recognize useful features from the instances. When looking at the differences with the optimal total cost $\mathcal{T}(\pi^*)$, we can see that our model achieved the smallest difference among the different mappings. The learned dynamic selection (π) was able to perform 1.36 times better than the best deterministic selection (π_B) when comparing the differences to the optimal. Therefore, the expectation for this experiment was met as the training of the model produced a successful result.

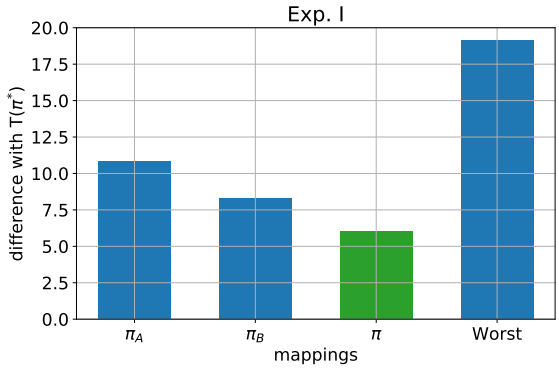

Figure 5.5: Results for Exp. I: difference with the optimal total cost $\mathcal{T}(\pi^*)$ compared to the total costs for the deterministic mappings (π_A and π_B), our model (π in green), and a *worst* mapping.

Table 5.2 summarises the per-graph mapping results and shows how the results compare to the optimal mapping percentage-wise. The per-graph mapping results demonstrate that our dynamic parameter selection performed better than the best deterministic selection in four test cases out of the total five. This matches our expectations for the machine learning model to make use of the input features to learn the mapping π. However, the single worse, but almost on-par performance in the experiment Exp. R was unexpected and may be explained by the underrepresentation of class B samples within the dataset. In other words, for these instances, the performance of the deterministic mapping π_A was already very close to the optimal. As stated, considering a larger dataset might help, but the inherent reason for this result is esimated to come from the graph setup that allows to better avoid conflicts between the agents.

	Exp. H	Exp. E	Exp. N	Exp. R	Exp. I
π_A	+0.43%	+0.37%	+0.61%	+0.20%	+0.78%
π_B	+0.64%	+0.64%	+0.51%	+0.56%	+0.60%
π	+0.28%	+0.21%	+0.25%	+0.21%	+0.44%
Worst	+1.06%	+1.00%	+1.12%	+0.76%	+1.39%

Table 5.2: Per-graph mappings total cost value differences with the optimal mapping π^* in percentages.

5.3 Multi-graph mappings

Exp. HEN: Figure 5.6 refers to the experiment Exp. HEN from Section 4.5. Here, we expect the machine learning model and the resulting mapping π to perform better than the two deterministic mappings π_A and π_B for a successful result. Additionally, we expect π to perform better than the per-graph mappings from experiments Exp. H, Exp. E and Exp. N, which on average performed 1.78 times better than the best deterministic selections when comparing the differences to the optimal. This is because the machine learning model has access to more data, and instances from one graph can share features that apply to instances from another graph, potentially improving the learning. When looking at the difference with the optimal total cost $\mathcal{T}(\pi^*)$, we can see that our model achieved the smallest difference among the different mappings. The learned dynamic selection (π) was able to perform 1.88 times better than the best deterministic selection (π_A) when comparing the differences to the optimal. Therefore, the expectation for this experiment was met as the training of the model produced a successful result.

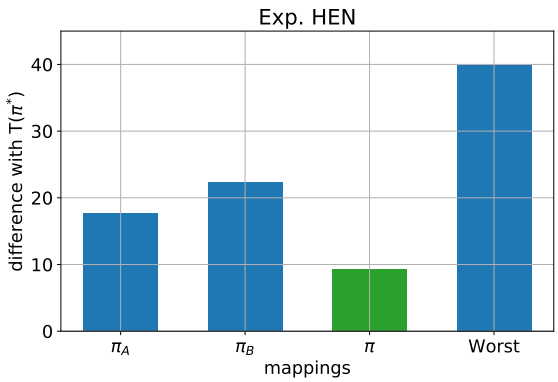

Figure 5.6: Results for Exp. HEN: difference with the optimal total cost $\mathcal{T}(\pi^*)$ compared to the total costs for the deterministic mappings (π_A and π_B), our model (π in green), and a *worst* mapping.

Exp. RI: Figure 5.7 refers to the experiment Exp. RI from Section 4.5. Here, we expect the machine learning model and the resulting mapping π to perform better than the two deterministic mappings π_A and π_B for a successful result. Additionally, we expect π to perform better than the per-graph mappings from experiments Exp. R and Exp. I. This is because the machine learning model has access to more data, and instances from one graph can share features that apply to instances from another graph, potentially improving the learning. When looking at the difference with the optimal total cost $\mathcal{T}(\pi^*)$, we can see that our model achieved the smallest difference among the different mappings. The learned dynamic selection (π) was able to perform 1.56 times better than the best deterministic selection (π_A) when comparing the differences to the optimal. Therefore, the expectation for this experiment was met as the training of the model produced a successful result.

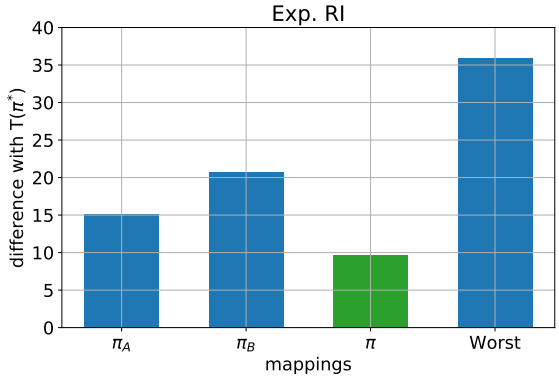

Figure 5.7: Results for Exp. RI: difference with the optimal total cost $\mathcal{T}(\pi^*)$ compared to the total costs for the deterministic mappings (π_A and π_B), our model (π in green), and a *worst* mapping.

Exp. HENRI: Figure 5.8 refers to the experiment Exp. HENRI from Section 4.5. Here, we expect the machine learning model and the resulting mapping π to perform better than the two deterministic mappings π_A and π_B for a successful result. This is because we assume the machine learning model to make use of the data it is trained on to learn to recognize useful features from the instances. When looking at the difference with the optimal total cost $\mathcal{T}(\pi^*)$, we can see that our model achieved the smallest difference among the different mappings. The learned dynamic selection (π) was able to perform 1.36 times better than the best deterministic selection (π_A) when comparing the differences to the optimal. Therefore, the expectation for this experiment was met as the training of the model produced a successful result.

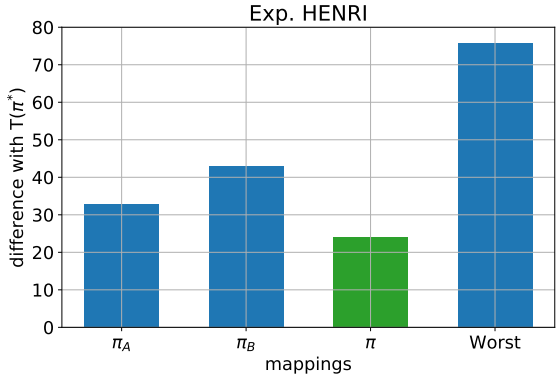

Figure 5.8: Results for Exp. HENRI: difference with the optimal total cost $\mathcal{T}(\pi^*)$ compared to the total costs for the deterministic mappings (π_A and π_B), our model (π in green), and a *worst* mapping.

Table 5.3 summarises the multi-graph mapping results and shows how the results compare to the optimal mapping percentage-wise. The multi-graph mapping results demonstrate that our dynamic parameter selection performed better than the best deterministic selection in all three test cases. This matches our expectations for the machine learning model to make use of the input features to learn the mapping π.

Additionally, it could be said that for Exp. HEN and Exp. RI the performance was better when comparing with the performance of the graph-specific mappings from which the data combinations were used. Lastly, with the Exp. HENRI, it seems that the choice of training a more generalized model on data from all five test cases is possible as it performed better than any of the deterministic mappings. However, this result is the worst between the three multi-graph mappings. The reason for this may be in the missing structure between all of the test cases, that is present in the data combinations of experiments Exp. HEN and Exp. RI.

	Exp. HEN	Exp. RI	Exp. HENRI
π_A	+0.47%	+0.42%	+0.45%
π_B	+0.59%	+0.58%	+0.58%
π	+0.25%	+0.27%	+0.33%
Worst	+1.06%	+1.00%	+1.03%

Table 5.3: Multi-graph mappings total cost value differences with the optimal mapping π^* in percentages.

5.4 Comparison with target system

Figure 5.9 examines the distinct solutions from six randomly selected MAPF instances from the experiment Exp. N dataset as it achieved the best performance between the per-graph mappings. It features the SIC values in the CTDS model and the target system between solutions with different number of agents using the defined set of speeds from $\mathcal{S}$ (see Section 4.3.2) for the different instances, and also the single solution cost from the penalty-based function $\mathcal{C}(\pi(i), i)$ (Equation 3.1). The expectation here is that the SIC values in the CTDS model and the target system behave in a similar manner and that they show a logical connection to the single solution cost $\mathcal{C}(\pi(i), i)$ of a MAPF instance i and, therefore, also to the total cost $\mathcal{T}(\pi)$ (Equation 3.3) comprised of multiple instances $i \in \mathcal{I}$. We expect this because we assume that the agents' behavior from using more speeds in the CTDS model can be translated to improved behavior in the target system if an instance i is labeled to use more speeds ($\pi(i) = B$).

The comparison shows that when an instance i was categorized under class A ($\pi(i) = A$), i.e., to use fewer speeds, then the SIC values are equal both in the CTDS model and the target system when using the different speed sets. This is because using more speeds can only make the final solution better in terms of the SIC time. As the SIC times are equal, then it is better to use fewer speeds as using more speeds does not give an advantage. Also, using fewer speeds allows the solution to be found quicker as the computational complexity grows with the number of available speeds. Thus, the reduced penalty applied to the single cost $\mathcal{C}(\pi(i), i)$ for using fewer speeds allows us to label the instance into class A. This connection is also visible in the equal SIC values for the target system.

Correspondingly, when an instance i was categorized under class B ($\pi(i) = B$), i.e., to use more speeds, then the SIC values are lower both in the CTDS model and

the target system with the use of speed set $speed_B$. Here, the focus lies rather in the fact that the values improve in a similar manner, rather than the exact size of the improvement. This shows that using more speeds for these instances is also beneficial in the target system. Thus, the expectation for this experiment is met, as the behavior with the use of the different speed sets between the SIC values in the CTDS model and the target system is similar. Also, the logical connection to the penalty-based functions is revealed because equal solutions in terms of the SIC time are penalized in favor of the least expensive one in the CTDS model as well as the target system. Therefore, suggesting that the proposed machine learning approach can also be translated to the target system as possible gains present in the CTDS model can also be manifested in the target system.

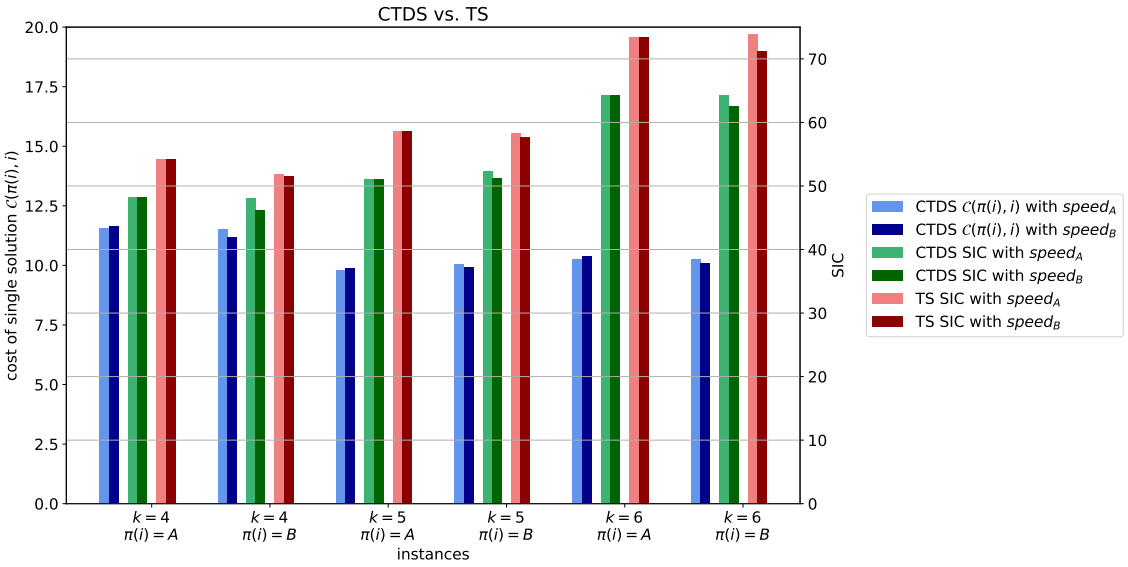

Figure 5.9: Comparison of values in the CTDS model and the target system (TS) using the speedsets $speed_A$ and $speed_B$ for different instances i with k agents.

5.5 Discussion

Coming back to the research questions established in Section 4.1, we have presented the results of the implementation of the automatic parameter selection to learn a mapping π. With this approach, the results showed that the learned per-graph and multi-graph mappings were able to constantly perform better or on-par in comparison to the fixed deterministic selections on the established road scenarios. Thus, answering the first research question (Q1) and suggesting that machine learning can be used to achieve better performance than any of the deterministic mappings as the learned features from a visual instance representation are sufficient. However, as was learned from Exp. R, this comes with the expectation that the problem setup allows for the generation of instances where both cases - using more speeds and using fewer speeds - are better balanced. Additionally, Exp. G showed that the use of more speeds did not present any advantage in this case because conflicts could be easily avoided in the grid environment, and, therefore, a dynamic mapping could

not be applicable as a fixed setting was always preferred.

Moreover, regarding the second research question (Q2), the multi-graph results indicated the potential to train a more robust machine learning model on different problem settings. Like the per-graph mappings, the dynamic mappings were still able to perform better than the deterministic mappings. At the same time, even expressing small gains over the per-graph mappings, but this comes with the expectation that the instances from different graphs share a similar structure. With this, we mean that there is potential for instances from one graph to have features that apply to instances from another graph, In more detail, consider an instance that is on the highway entrance graph. If agents within this graph are not assigned start nodes from the on-ramp to the main highway, then essentially this same instance can also be represented by only using the highway graph. Therefore, features from the highway entrance instance could be applied to an instance on the highway graph during the training of a machine learning model. This connection can also be reasoned to exist between different roundabout and intersection instances, as both share a similar setup of four incoming two-lane roads that model the possible start nodes for the agents. Furthermore, with Exp. HENRI, where a model was trained on MAPF problem instances from all of the five road test cases, the dynamic mapping was still able to outperform the fixed mappings. However, we can see an overall decline in the quality of the result when compared to the other multi-graph mappings. Thus, indicating that *specialiced* machine learning models - that focus on test cases with a similar structure - could be preferred over very robust models that try to achieve towards a *Jack-of-all-trades* styled implementation.

When discussing the third research question (Q3), the proposed problem setup of using two speeds vs. four speeds presented the possible improvement to be slightly under 1%. Although, this is to be expected as the presented road environments are relatively small compared to a bigger road system, consisting of multiple such minor parts, where there is more possibility for improvement. Learning from Exp. G, an important factor to consider is the setup of the applied problem. If most conflicts can be avoided without loss in arrival time by regarding other paths, then this approach might not be worthwhile as it is hard to find instances where more speeds impose a potential improvement.

Lastly, concerning the fourth research question (Q4), we demonstrated that there exists a correlation between the single solution cost $\mathcal{C}(\pi(i), i)$ for a MAPF problem instance i and the SIC values in the CTDS model and the target system. This comparison showed that gains in the CTDS model could be manifested in the target system as the agents' behavior from using more speeds in the CTDS model can be translated to improved behavior in the target system. Therefore, the use of dynamic mapping can also be beneficial in a framework based on the target system.

6

Conclusion

In this thesis, we formulated the automatic parameter selection problem and presented the application of a deep neural network to approach this problem. By doing so, a mapping was learned that is able to connect a MAPF problem instance to a suitable speed parameter. This dynamic parameter selection approach was experimented on with MAPF problem instances from various vehicular road traffic scenarios.

The results showed that a visual representation of MAPF problems is sufficient as there are cases in which the dynamic selection was able to perform better when compared to a fixed setting. One of the benefits of the presented approach is that it allows to work with a single MAPF algorithm extensively without the need to improve it further in order to achieve potentially better performance. Furthermore, by using machine learning to select parameters, developers of MAPF algorithms do not have to implement this selection somehow manually, as the possible number of problem instances may be virtually unlimited.

In conclusion, the presented work gives new knowledge for the previously unstudied approach of parameter selection, indicating a connection between the visual representation of a MAPF problem and its solution with the potential to improve it further via parameter selection. Future work in this area could extend the evaluation to more realistic MAPF problems by extracting information from real-world traffic data. Moreover, once a machine learning model is fully trained, it could be implemented in an online manner, so it could provide parameter selection knowledge to solve MAPF problem instances as they appear in real time. Also, implementing more road scenarios would allow for the training of a possibly more general model. Another possible future direction is considering MAPF problem instance representations other than visual to use as input for machine learning.

Bibliography

[1] Jingjin Yu and Steven M LaValle. Structure and intractability of optimal multi-robot path planning on graphs. In *Twenty-Seventh AAAI Conference on Artificial Intelligence*, 2013.

[2] Pavel Surynek. An optimization variant of multi-robot path planning is intractable. In *Twenty-Fourth AAAI Conference on Artificial Intelligence*, 2010.

[3] Nathan R Sturtevant and Robert Geisberger. A comparison of high-level approaches for speeding up pathfinding. In *Sixth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2010.

[4] Kurt Dresner and Peter Stone. A multiagent approach to autonomous intersection management. *Journal of artificial intelligence research*, 31:591–656, 2008.

[5] Lucia Pallottino, Vincenzo G Scordio, Antonio Bicchi, and Emilio Frazzoli. Decentralized cooperative policy for conflict resolution in multivehicle systems. *IEEE Transactions on Robotics*, 23(6):1170–1183, 2007.

[6] David Silver. Cooperative pathfinding. *AIIDE*, 1:117–122, 2005.

[7] Peter R Wurman, Raffaello D'Andrea, and Mick Mountz. Coordinating hundreds of cooperative, autonomous vehicles in warehouses. *AI magazine*, 29(1):9–9, 2008.

[8] Lynne E Parker. Path planning and motion coordination in multiple mobile robot teams. *Encyclopedia of complexity and system science*, pages 5783–5800, 2009.

[9] Guni Sharon, Roni Stern, Ariel Felner, and Nathan R Sturtevant. Conflict-based search for optimal multi-agent pathfinding. *Artificial Intelligence*, 219:40–66, 2015.

[10] Glenn Wagner and Howie Choset. Subdimensional expansion for multirobot path planning. *Artificial Intelligence*, 219:1–24, 2015.

[11] Max Barer, Guni Sharon, Roni Stern, and Ariel Felner. Suboptimal variants of the conflict-based search algorithm for the multi-agent pathfinding problem. In *Seventh Annual Symposium on Combinatorial Search*, 2014.

[12] Jiří Švancara, Marek Vlk, Roni Stern, Dor Atzmon, and Roman Barták. Online multi-agent pathfinding. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 7732–7739, 2019.

[13] Anton Andreychuk, Konstantin Yakovlev, Dor Atzmon, and Roni Stern. Multi-agent pathfinding with continuous time. *arXiv preprint arXiv:1901.05506*, 2019.

[14] Ariel Felner, Roni Stern, Solomon Eyal Shimony, Eli Boyarski, Meir Goldenberg, Guni Sharon, Nathan Sturtevant, Glenn Wagner, and Pavel Surynek. Search-based optimal solvers for the multi-agent pathfinding problem: Summary and challenges. In *Tenth Annual Symposium on Combinatorial Search*, 2017.

[15] Devon Sigurdson, Vadib Bulitko, Sven Koenig, Carlos Hernandez, and William Yeoh. Automatic algorithm selection in multi-agent pathfinding. *arXiv preprint arXiv:1906.03992*, 2019.

[16] Jiuxiang Gu, Zhenhua Wang, Jason Kuen, Lianyang Ma, Amir Shahroudy, Bing Shuai, Ting Liu, Xingxing Wang, Gang Wang, Jianfei Cai, et al. Recent advances in convolutional neural networks. *Pattern Recognition*, 77:354–377, 2018.

[17] Johan Gerdin. Multi-agent pathfinding with discrete speeds and its application for vehicle road networks. Master's thesis, Computer Science and Engineering, Chalmers University of Technology, Rännvägen 6B, Göteborg, Sweden, S-412 96, 6 2020. To appear.

[18] Devon Sigurdson and Vadim Bulitko. Deep learning for real-time heuristic search algorithm selection. In *Thirteenth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2017.

[19] Thomas Petig, Elad Michael Schiller, and Jukka Suomela. Changing lanes on a highway. In Ralf Borndörfer and Sabine Storandt, editors, *18th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems, ATMOS 2018, August 23-24, 2018, Helsinki, Finland*, volume 65 of *OASICS*, pages 9:1–9:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

[20] António Casimiro, Emelie Ekenstedt, and Elad Michael Schiller. Self-stabilizing manoeuvre negotiation: The case of virtual traffic lights. In *38th Symposium on Reliable Distributed Systems, SRDS 2019, Lyon, France, October 1-4, 2019*, pages 354–356. IEEE, 2019.

[21] Oscar Morales Ponce, Elad Michael Schiller, and Paolo Falcone. How to stop disagreeing and start cooperatingin the presence of asymmetric packet loss. *Sensors*, 18(4):1287, 2018.

[22] Vladimir Savic, Elad Michael Schiller, and Marina Papatriantafilou. Distributed algorithm for collision avoidance at road intersections in the presence of communication failures. In *IEEE Intelligent Vehicles Symposium, IV 2017, Los Angeles, CA, USA, June 11-14, 2017*, pages 1005–1012. IEEE, 2017.

[23] António Casimiro, Oscar Morales Ponce, Thomas Petig, and Elad Michael Schiller. Vehicular coordination via a safety kernel in the gulliver test-bed. In *34th International Conference on Distributed Computing Systems Workshops (ICDCS 2014 Workshops), Madrid, Spain, June 30 - July 3, 2014*, pages 167–

176. IEEE Computer Society, 2014.

[24] Oscar Morales Ponce, Elad Michael Schiller, and Paolo Falcone. Cooperation with disagreement correction in the presence of communication failures. In *17th International IEEE Conference on Intelligent Transportation Systems, ITSC 2014, Qingdao, China, October 8-11, 2014*, pages 1105–1110. IEEE, 2014.

[25] António Casimiro, José Rufino, Ricardo C. Pinto, Eric Vial, Elad Michael Schiller, Oscar Morales Ponce, and Thomas Petig. A kernel-based architecture for safe cooperative vehicular functions. In *Proceedings of the 9th IEEE International Symposium on Industrial Embedded Systems, SIES 2014, Pisa, Italy, June 18-20, 2014*, pages 228–237. IEEE, 2014.

[26] António Casimiro, Jörg Kaiser, Elad Schiller, Pedro Costa, José Parizi, Rolf Johansson, and Renato Librino. The KARYON project: Predictable and safe coordination in cooperative vehicular systems. In *43rd Annual IEEE/IFIP Conference on Dependable Systems and Networks Workshop, DSN Workshops 2013, Budapest, Hungary, June 24-27, 2013*, pages 1–12. IEEE Computer Society, 2013.

[27] António Casimiro, Jörg Kaiser, Johan Karlsson, Elad Michael Schiller, Philippas Tsigas, Pedro Costa, José Parizi, Rolf Johansson, and Renato Librino. Brief announcement: KARYON: towards safety kernels for cooperative vehicular systems. In Andréa W. Richa and Christian Scheideler, editors, *Stabilization, Safety, and Security of Distributed Systems - 14th International Symposium, SSS 2012, Toronto, Canada, October 1-4, 2012. Proceedings*, volume 7596 of *Lecture Notes in Computer Science*, pages 232–235. Springer, 2012.

[28] Mitra Pahlavan, Marina Papatriantafilou, and Elad Michael Schiller. Gulliver: A test-bed for developing, demonstrating and prototyping vehicular systems. In *Proceedings of the 75th IEEE Vehicular Technology Conference, VTC Spring 2012, Yokohama, Japan, May 6-9, 2012*, pages 1–2. IEEE, 2012.

[29] Christian Berger, Erik Dahlgren, Johan Grunden, Daniel Gunnarsson, Nadia Holtryd, Anmar Khazal, Mohamed Mustafa, Marina Papatriantafilou, Elad Michael Schiller, Christoph Steup, Viktor Swantesson, and Philippas Tsigas. Bridging physical and digital traffic system simulations with the gulliver test-bed. In Marion Berbineau, Magnus Jonsson, Jean-Marie Bonnin, Soumaya Cherkaoui, Marina Aguado, Cristina Rico Garcia, Hassan Ghannoum, Rashid Mehmood, and Alexey V. Vinel, editors, *Communication Technologies for Vehicles, 5th International Workshop, Nets4Cars/Nets4Trains 2013, Villeneuve d'Ascq, France, May 14-15, 2013. Proceedings*, volume 7865 of *Lecture Notes in Computer Science*, pages 169–184. Springer, 2013.

[30] Jesper Åberg and Andreas Kuszli. A framework for multi-agent path finding with focus on transport systems. Master's thesis, Computer Science and Engineering, Chalmers University of Technology, Rännvägen 6B, Göteborg, Sweden, S-412 96, 6 2020. To appear.

[31] John R. Rice et al. The algorithm selection problem. *Advances in computers*, 15(65-118):5, 1976.

[32] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[33] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[34] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[35] Y-Lan Boureau, Jean Ponce, and Yann LeCun. A theoretical analysis of feature pooling in visual recognition. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 111–118, 2010.

[36] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[37] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[38] Nikhil Ketkar et al. *Deep Learning with Python*, volume 1. Springer, 2017.