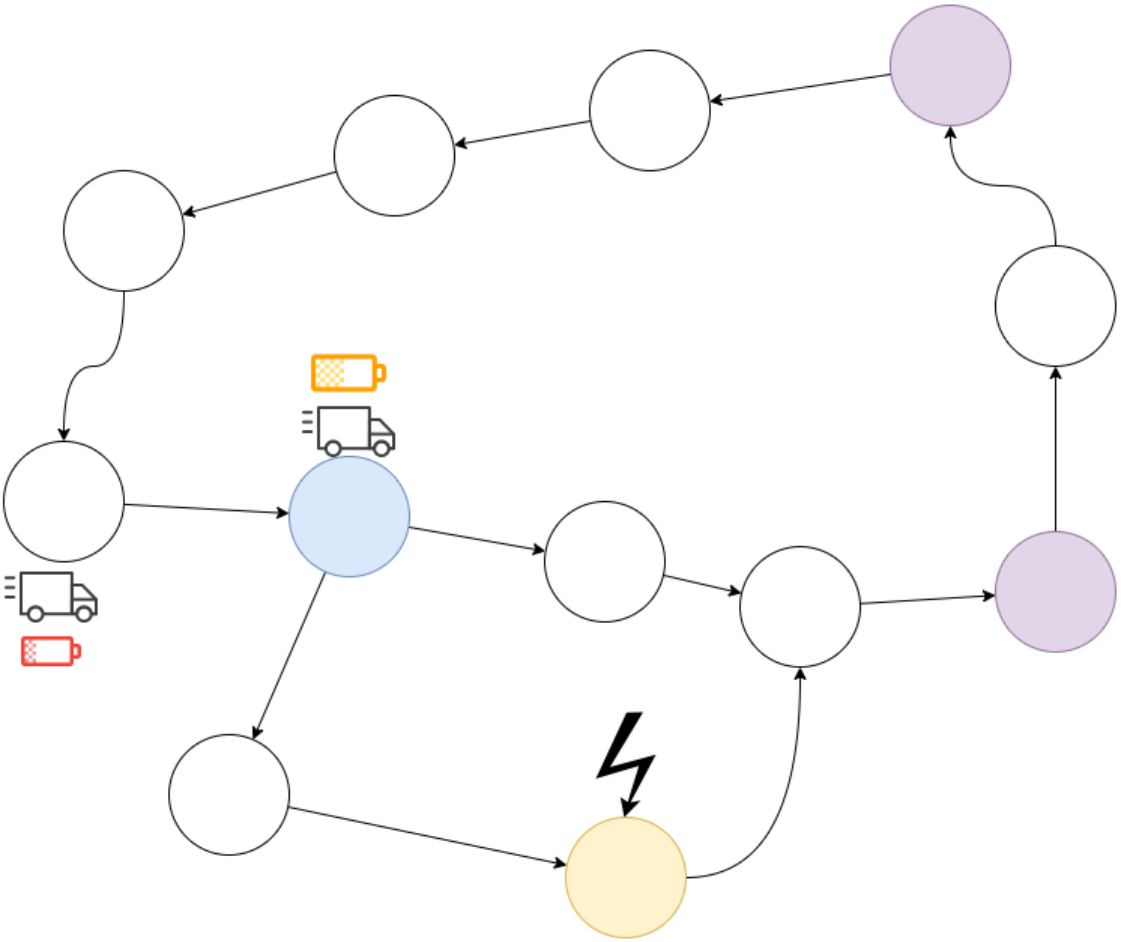


Machine Learning Based Charging Decision Policy for a Fleet of Electric Vehicles

Master's thesis in Engineering Mathematics and Computational Science

GUSTAV JOHANNESSON

ALEXANDER LINDHARDT

DEPARTMENT OF MATHEMATICAL SCIENCES

CHALMERS UNIVERSITY OF TECHNOLOGY

Gothenburg, Sweden 2022

www.chalmers.se

Master's thesis 2022

Machine Learning Based Charging Decision Policy for a Fleet of Electric Vehicles

GUSTAV JOHANNESSON
ALEXANDER LINDHARDT

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Mathematical Sciences
Division of Applied Mathematics and Statistics
Chalmers University of Technology
Gothenburg, Sweden 2022

Machine Learning Based Charging Decision Policy for a Fleet of Electric Vehicles

GUSTAV JOHANNESSON
ALEXANDER LINDHARDT

Supervisor: Jonas Hellgren, Volvo Autonomous Solutions
Supervisor: Axel Ringh, Department of Mathematical Sciences, Chalmers University of Technology and University of Gothenburg
Examiner: Rebecka Jörnsten, Department of Mathematical Sciences, Chalmers University of Technology and University of Gothenburg

Master's Thesis 2022
Department of Mathematical Sciences
Division of Applied Mathematics and Statistics
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: A conceptual site with two vehicles. Displaying a scenario where the vehicle at the front should choose to not charge.

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2022

Machine Learning Based Charging Decision Policy for a Fleet of Electric Vehicles
GUSTAV JOHANNESSON
ALEXANDER LINDHARDT
Department of Mathematical Sciences
Chalmers University of Technology

Abstract

In a fleet of autonomous electric vehicles, each vehicle must frequently charge its battery. The charging of the vehicles should ideally be as efficient as possible without causing roadblocks or battery depletion. The aim of this thesis, which is performed in collaboration with Volvo Autonomous Solutions, is to develop such charging policies using reinforcement learning methods. More precisely, the problem is formulated as a Markov decision process (MDP), and tabular Q-learning and deep Q-learning are used to learn (optimal) charging policies. The learned policies are evaluated and compared against each other, as well as against a rule-based policy used as a benchmark. The robustness and generalizability of the learned policies are tested by adjusting a number of site-specific parameters, such as charging time and number of vehicles at the site. The results indicate that machine learning-based charging policies can increase the efficiency for a fleet of vehicles compared to a simple rule-based charging policy. Moreover, the importance of such policies is also shown to vary depending on the site configuration and its complexity.

Keywords: charging policy, machine learning, reinforcement learning, Q-learning, DQN, deep Q-learning, Markov decision process

Acknowledgements

First of all we would like to thank our supervisor Jonas Hellgren at Volvo Autonomous Solutions, for arranging the project and providing us with guidance throughout the entire process. We would also like to extend our gratitude to VAS for giving us the tools and logistics that made it possible to complete this thesis work.

We are also incredibly thankful for our supervisor Axel Ringh at Chalmers, for interesting discussions and great advice. Without your continuous help this thesis would not be the same. Finally we would like to thank our examiner Rebecka Jörnsten at Chalmers.

Gustav Johannesson & Alexander Lindhardt, Gothenburg, May 2022

Contents

1
Introduction

Autonomous vehicles are not widely used in traffic yet, however they have already been implemented in confined spaces such as mines and harbors where they perform specific tasks, for example transporting cargo [1]. The vehicles working in such confined spaces are often part of a team and the decision each vehicle takes might affect the total benefit of the team. It is therefore crucial that the decision making by each individual vehicle is carefully determined and beneficial for the task at hand.

If a fleet of vehicles is powered by electricity, each vehicle occasionally needs to charge its battery. In order for the fleet of vehicles to operate efficiently, it needs a stable and effective charging strategy such that each vehicle can perform their task in a profitable manner while never running out of battery. Such a charging strategy is therefore highly important when a fleet of autonomous vehicles is deployed in order to complete a given task.

One company that has already deployed fleets of autonomous electric vehicles at a few sites is Volvo Autonomous Solutions (VAS) [2], [3]. Their aim is to increase the efficiency for these fleets and they are therefore interested in finding optimized charging strategies.

1.1 Background and related work

A charging strategy in its simplest form is based on some basic rules that are predefined. One such policy could for example be to charge a vehicle if its battery level is below a certain threshold. A more complex charging policy should instead be able to make decisions based on a long-term benefit analysis. To make the fleet of vehicles as efficient as possible the system has to weigh the benefit against the time and energy it takes to charge, while also considering the risk of energy depletion. In addition to this, a charging policy should consider the risk of queues arising at charging stations. When a vehicle is approaching a charging station these aspects need to be accounted for and a decision which benefits the system as a whole has to be made.

With the current growth of electric vehicles and plug-in hybrid vehicles, the interest in finding optimal charging policies has increased. A charging policy may however be considered optimal for different reasons, i.e., in different contexts the cost function which is optimized is different. For example, there have been a large amount of research on finding charging policies that decrease the load of the power

grid [4], [5], [6]. Another example is the development of charging policies that try to optimize the lifetime of batteries [7]. The common theme in the research is that learned charging strategies with different aims can be successfully employed.

Vehicle route planning is another active area of research. Today, with the advancement of electric vehicles, the charging of vehicles is now typically included in the planning [8], [9]. However, solving this by finding a charging policy using machine learning independent of the route planning has (to the best of our knowledge) not yet been investigated thoroughly.

There are multiple different approaches that can be adopted in order to find an optimal charging policy. One of these approaches, the one studied in this thesis, is reinforcement learning. Reinforcement learning has proven to be powerful in solving sequential decision problems before. Maybe the most famous example of this is when the program *AlphaGo Zero* was developed by a team at DeepMind in 2017. The program used reinforcement learning to learn how to play the game of Go and surpassed the level of any human player [10].

Reinforcement learning has grown quickly in the last decade as it has been able to solve problems which many thought were impossible for a long time. With the success of deep learning, this helped scale reinforcement learning to be able to solve problems with high dimensional state and action space [11]. Some areas where it has been proven useful are robotics, computer vision, finance, and games, just to name a few [12].

These things together lay the foundation for what we want to explore in this thesis.

1.2 Objective and limitations

The aim of this master's thesis is to find ways of optimizing the charging policy of a fleet of autonomous electric vehicles that follow an independent route planning algorithm. More specifically, reinforcement learning methods will be investigated, implemented, and evaluated. One of the key components of reinforcement learning is to mathematically model the problem at hand as a Markov Decision Process.

The vehicle route planning algorithm together with the modelling of sites and vehicles were developed and distributed by VAS. It was upon this that the charging policy would be implemented, trained, and evaluated. Due to time constraints, the amount of training time of the different reinforcement learning methods had to be limited. The number of possible configurations of sites, vehicles, and other parameters is infinite. However, most of the work of this thesis will be implemented on one simple model of a site with one setup of vehicles and only one charging station. Nevertheless, the methodology used in this thesis can still be applied on different setups with small adjustments.

2
Theory

This chapter gives a short introduction to the main theoretical concepts of this thesis. The sections about Markov decision processes and reinforcement learning is based on the book by Sutton and Barto [13] which gives a more extensive description about these subjects.

2.1 Markov decision process

The charging decision problem can be turned into a discrete decision making process where there is a finite amount of configurations of inputs. A Markov decision process (MDP) can be used to mathematically formulate such a problem, where agents are making decisions in an uncertain and stochastic environment [13, Ch. 3.1]. An MDP is defined as the tuple ($\mathcal{S}$, $\mathcal{A}$, T, R) where:

- The state space $\mathcal{S}$ is the set of all possible states in the environment.
- The action space $\mathcal{A}$ is the set of all possible actions the agent can make.
- The transition function T defines the dynamic of the MDP. The probability of transitioning to state $s' \in \mathcal{S}$ when taking action $a \in \mathcal{A}$ from state $s \in \mathcal{S}$ is given by $T(s'|s, a)$.
- The reward function R returns a numerical value for a state-action pair. The reward $r \in \mathbb{R}$ of taking action $a \in \mathcal{A}$ from state $s \in \mathcal{S}$ is given by $R(r|s, a)$.

If the set of states, actions and rewards are finite, it is called a finite MDP.

The interactions between the agent and its environment are conceptually illustrated in Figure 2.1 and take place in the following way [13, Ch. 3.1]: For each time step t, the agent starts in a state within the environment $s_t \in \mathcal{S}$. It then takes an action $a_t \in \mathcal{A}$ based on s_t and ends up in a new state $s_{t+1} \in \mathcal{S}$. The environment then returns a reward signal $r_t \in \mathbb{R}$ to the agent for this action taken. The agent is given an immediate reward for each action, however every action has an impact on the future evolution of states and therefore also the total future reward. The agent will therefore need to consider and learn how an action at a given state will influence the future.

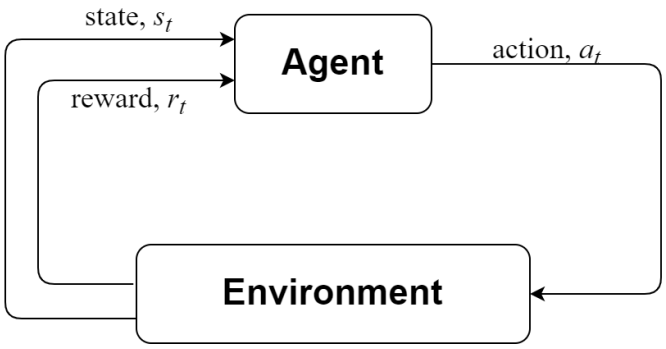


Figure 2.1: Agent-environment interaction. An agent in state $s_t \in \mathcal{S}$ takes an action $a_t \in \mathcal{A}$ based on s_t and ends up in a new state $s_{t+1} \in \mathcal{S}$. The environment returns a reward signal $r_t \in \mathbb{R}$ to the agent for this action taken. This process is then repeated.

2.1.1 Policy and value functions

A policy $\pi : \mathcal{S} \to \mathcal{A}$, is a function that maps a state to an action and is used to describe the agent's behavior. The policy $\pi(s)$ specifies which action the agent will choose when it observes the state s. This function could be either deterministic or probabilistic. A common goal of the agent is to find the optimal policy which maximizes the expected discounted return. The discounted return is defined as

$$G_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + ... = \sum_{k=0}^{\infty} \gamma^k r_{t+k}. \tag{2.1}$$

Here, $0 \leq \gamma \leq 1$, is the discount factor that determines how important the immediate rewards are compared to future rewards. To maximize the expectation of G_t, a value function is used to define the expected discounted return for an agent when it is in a given state. The value function is defined with respect to an agent's policy π. The value function $V^\pi(s)$ is the expected discounted return of an agent starting in state s, following the policy π. For MDPs, the value function is formally defined as [13, Ch. 3.5]

$$V^\pi(s) = \mathbb{E}_\pi\Big[G_t \mid s_t = s\Big] = \mathbb{E}_\pi\Big[\sum_{k=0}^{\infty} \gamma^k r_{t+k} \mid s_t = s\Big]. \tag{2.2}$$

The value function $V^\pi(s)$ is called the state-value function for policy π.

In a similar fashion, a value function $Q^\pi(s, a)$, called the action-value function for policy π, is defined. This function is defined as the expected discounted return of an agent being at state s, taking the action a, and thereafter following policy π [13, Ch. 3.5]. Mathematically, this means that

$$Q^\pi(s, a) = \mathbb{E}_\pi\Big[G_t \mid s_t = s, a_t = a\Big] = \mathbb{E}_\pi\Big[\sum_{k=0}^{\infty} \gamma^k r_{t+k} \mid s_t = s, a_t = a\Big]. \tag{2.3}$$

2.1.2 Optimal policy

For any finite MDP, meaning that there is a finite number of states, actions and rewards, there exists at least one optimal policy π^* that satisfies [13, Ch. 3.6]

$$V^*(s) := V^{\pi^*}(s) = \max_{\pi} V^{\pi}(s), \ \ \forall s \in \mathcal{S}$$

and

$$Q^*(s,a) := Q^{\pi^*}(s,a) = \max_{\pi} Q^{\pi}(s,a), \ \ \forall s \in \mathcal{S}, \ \ \forall a \in \mathcal{A}.$$

The functions $V^*(s)$ and $Q^*(s,a)$ are called the optimal state-value function and the optimal action-value function respectively. Moreover, $Q^*(s,a)$ can be written in terms of $V^*(s)$ as

$$Q^*(s,a) = \mathbb{E}\Big[r_t + \gamma V^*(s_{t+1}) | s_t = s, a_t = a\Big]. \tag{2.4}$$

Now since $V^*(s) = \max_a Q^*(s,a)$, the Bellman equation for the action-value function $Q^*(s,a)$ can be written as [13, Ch. 3.6]

$$Q^*(s,a) = \mathbb{E}\Big[r_t + \gamma \max_{a'} Q^*(s_{t+1}, a')) | s_t = s, a_t = a\Big]. \tag{2.5}$$

If $Q^*(s,a)$ is known, an optimal policy can be directly retrieved from the function as

$$\pi^*(s) = \arg\max_a Q^*(s,a). \tag{2.6}$$

Obtaining an exact value of $Q^*(s,a)$ using recursion is difficult and often computationally impossible. Instead, learning algorithms can be used to approximate $Q^*(s,a)$ [13, Ch. 3.7].

2.2 Reinforcement learning

Reinforcement learning is a field within machine learning that uses a decision maker (agent) that makes sequential decisions in a system (environment). The agent is given a reward after every such decision by the environment, and the goal in each step is to maximize the expected cumulative future reward.

2.2.1 Q-learning

Q-learning is a model-free reinforcement learning method that aims to find an optimal policy by approximating the optimal action-value function, $Q^*(s,a)$ [13, Ch. 6.5]. The learning is carried out by simulating discrete steps in an MDP. For a state s_t, an action a_t is chosen and executed. The next state s_{t+1} is then observed and a reward r_t is given depending on this transition. Using this information, a function approximator $Q(s,a)$ is updated for each observed state-action pair.

The action a_t at state s_t can be chosen completely random, to explore the dynamics, or it can be chosen partially random and partially by using the current best action according to $Q(s, a)$ at state s_t. The latter is called ϵ-greedy, and for a given $\epsilon \in [0, 1]$ the best action is then chosen with probability $1 - \epsilon$. This ensures that all states get sampled continually [13, Ch. 2.2].

2.2.2 Tabular memory

The most simple, and also the original, implementation of Q-learning uses a table as function approximator $Q(s, a)$. This table has an estimate of Q^* stored for all possible state-action pairs. An example matrix representation of a tabular Q for a system with three states and two actions can be seen in Table 2.1. Such a tabulation is reasonable for a finite small set of possible state-action pairs, but is computationally impractical for large systems where the combinatorial nature of all possible combinations of states and actions makes the number of entries explode.

State/Action	a_1	a_2
s_1	$Q(s_1, a_1)$	$Q(s_1, a_2)$
s_2	$Q(s_2, a_1)$	$Q(s_2, a_2)$
s_3	$Q(s_3, a_1)$	$Q(s_3, a_2)$

Table 2.1: Tabular representation of $Q(s, a)$ for a system with three states and two actions.

Updating the table
Before training, the entries of table Q are initialized arbitrarily, often as random numbers. Experiences are then gathered by simulating the system and observing the state transitions and rewards as the tuple (s_t, a_t, r_t, s_{t+1}). For each simulated step the table is updated according to the rule

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha_t \left[r_t + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right], \tag{2.7}$$

where $\alpha_t \in (0, 1]$ is the learning rate, r_t the reward for performing action a_t in state s_t, and $\gamma \in [0, 1]$ the discount factor. From Equation 2.1 it is clear that $G_t = r_t + \gamma G_{t+1}$, which Q-learning utilizes to update its estimates.

Learning rate and episodes
A small learning rate, α, makes the updates of $Q(s, a)$ small and thus making the learning slow. However, with a larger α, the updates are larger and the learning can become unstable. Therefore α is often updated during learning, starting with a large value and then decreasing towards zero. If all state-action pairs are continually updated, and α_t fulfills the conditions

$$\sum_{t=0}^{\infty} \alpha_t = \infty, \tag{2.8a}$$

$$\sum_{t=0}^{\infty} \alpha_t^2 < \infty, \tag{2.8b}$$

throughout the learning process, Q-learning finds an optimal policy for any finite MDP, given enough time and space [14].

The learning is executed in episodes, where the initial state s_0 of an episode is usually observed from a random initialization of the environment. An episode is ended when a terminal state is reached or after a fixed amount of steps, or a combination of these criteria. The learning process can be summarized as in Algorithm 1 [13, Ch. 6.5].

Algorithm 1 Q-Learning algorithm

Initialize $Q(s, a) = 0$, $\forall s \in \mathcal{S}$ and $\forall a \in \mathcal{A}$
for episode **do**
    Initialize environment and observe s_0
    for each step t **do**
        Choose and execute action a_t according to ϵ-greedy
        Observe r_t and s_{t+1}
        $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha_t \left[r_t + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right]$
    end for
end for

2.2.3 Deep Q-Learning

Using a tabular memory can be troublesome for several reasons. If there is a large state-action space the memory capacity might not be enough, or the computations needed for convergence might be too demanding. Moreover, since a tabular function approximator needs discrete input, continuous variables in the system have to be discretized. To combat these problems deep Q-Learning (DQL) was introduced [15]. DQL is similar to Q-learning, but the main difference is that the table is replaced by a neural network. Instead of tabulating the estimated value of Q^* for every pair (s, a), neural networks are used as a function approximator to reduce dimensions. Using some sophisticated techniques and tricks to keep the learning of the neural network stable, this method has shown good results on high-dimensional problems. One of the first examples of a successful use of DQL is the computer program AlphaGo. Using DQL, AlphaGo was able to produce a policy for playing the board game Go, beating top ranked players easily. This feat was something previously thought to be far in the future if conventional techniques were used [15].

Feedforward neural networks

Before introducing deep Q-learning in more detail, we first need the concept of feedforward neural networks. A feedforward neural network is built of layers of nodes which are connected by weights as shown in Figure 2.2 and is used to approximate functions. The first layer, called the input layer, receives the input values. To

calculate the value x_j^l of node j in layer l, the values x_i^{l-1} of the nodes in the previous layer are used together with the weights $\boldsymbol{w^l}$ according to

$$x_j^l = f(b_j^l + \sum_i w_{ij}^l \cdot x_i^{l-1}) \tag{2.9}$$

where f is the activation function and $\mathbf{b}$ the bias term [16]. This process is then repeated for each node and between each layer until the output layer is reached and a final output is produced.

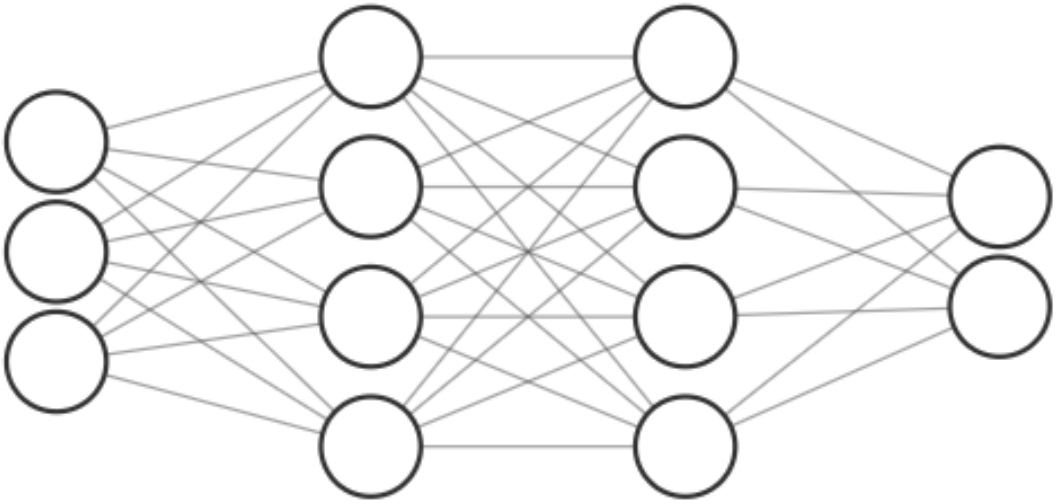


Figure 2.2: An example architecture of a neural network with three input neurons, two hidden layers with four neurons each, and an output layer with two neurons.

Feedforward neural networks with at least one hidden layer can approximate any continuous function [17]. They have shown great results in recent projects across many fields, ranging from image classification to stock market prediction [18].

Updating weights

The function that a neural network approximates depend on the interactions between nodes, and these interactions are controlled by the weights. To configure the weights such that the neural network does a good approximation the weights are updated iteratively, using pairs of input x and target output y. Such pairs are called labeled data. The input is fed through the network and the output $\hat{y}$ is retrieved and compared to the target output y, and the weights are thereafter updated to better resemble the true function.

The goal of updating weights is to minimize a loss function, which is a measure of how wrong the network predicted for a given input. One commonly used loss function is Mean Squared Error (MSE),

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2, \tag{2.10}$$

where n is the number of data points in the labeled data set. To minimize the loss, the weights are updated according to some optimizer. The optimizer uses backpropagation, which is a technique to calculate the gradients of the loss, to train the network and minimize the loss. One of the simplest optimizers, and also often part of more refined optimizers, is stochastic gradient descent. It updates the weights in the opposite direction of the gradient. The simple stochastic gradient descent method is known to find and stay at local extreme points. In an attempt to improve upon this, other optimizers have been developed, one such example is Adam [19].

Deep Q-network

The learning process for deep Q-learning is similar to tabular Q-learning. However, instead of storing the information directly in a table, it is implicitly stored in the weights of a neural network F. The network F has an input node for each state feature in s_t, and an output node for the value $Q(s_t, a)$ for each action a. For a system of three state features and two actions, the network F could look like in Figure 2.2. Note that the function approximator $Q(s_t, a_t)$ then essentially means that we feed s_t to F, and then choose the output value representing the action a_t.

By simulating steps of the model, state transitions are observed and rewards received. These experiences are used to create the labeled data when the network is updated. In the most simple form of DQL the target y is set to

$$y \leftarrow \begin{cases} r_t & \text{if } s_{t+1} \text{ is terminal} \\ r_t + \gamma \max_a Q(s_{t+1}, a) & \text{otherwise,} \end{cases} \tag{2.11}$$

where Q is the neural network, approximating Q^*. In previous work on DQL it has been discovered that using one network to produce y, which is then used to update that same network, can cause problems. Specifically, in certain cases this can lead to overestimation [20]. Instead a setup using two neural networks can be used to counteract this. The online network, Q, is the standard network estimating Q^* and updated every time new experience is gained. The weights for the target network, $\tilde{Q}$, is copied from Q in intervals. $\tilde{Q}$ is used to produce the target y as

$$y \leftarrow \begin{cases} r_t & \text{if } s_{t+1} \text{ is terminal} \\ r_t + \gamma \max_a \tilde{Q}(s_{t+1}, a) & \text{otherwise.} \end{cases} \tag{2.12}$$

The setup with double neural networks has been shown to not only reduce the overestimation, but also produce better performance overall [20].

Experience replay

Another way to increase stability in the training phase of the neural networks is experience replay. By storing experiences, (s_t, a_t, r_t, s_{t+1}), and drawing random samples of these in batches when training, the training becomes more stable and an experience can be used more times. This mitigates the risk of only learning from a subset of states if the agent happens to only perform these in a sequence [21]. The algorithm for DQL can then be summarized as in Algorithm 2.

Algorithm 2 Deep Q-Learning algorithm

Initialize weights $\boldsymbol{w}$ and $\boldsymbol{v}$ for networks Q and $\tilde{Q}$, respectively
Initialize replay memory $\mathcal{D}$
for episode **do**
    Initialize environment and observe s_0
    for each step t **do**
        Choose and execute action a_t according to ϵ-greedy
        Observe r_t and s_{t+1}
        Store (s_t, a_t, r_t, s_{t+1}) in $\mathcal{D}$
        Sample batch D_t from $\mathcal{D}$
        for each experience (s_d, a_d, r_d, s_{d+1}) in D_t **do**
            Produce target y as
            if s_{d+1} is terminal **then**
                $y \leftarrow r_d$
            else
                $y \leftarrow r_d + \gamma \max_a \tilde{Q}(s_{d+1}, a)$
            end if
            Retrieve estimate $\hat{y} \leftarrow Q(s_d, a_d)$
        end for
        Update $\boldsymbol{w}$, using the produced targets and estimates from D_t
        Every $\mathcal{T}$ steps, set $\boldsymbol{v} \leftarrow \boldsymbol{w}$
    end for
end for

3
Problem formulation

The aim of this thesis is to generate charging policies for a fleet of vehicles driving in a confined space. In the following sections, a model of such a confined space will be formulated. Based on this model an MDP is defined to solve the problem of finding an optimal charging policy.

3.1 Model

The site of a real-world charging decision problem is modeled as a graph where vehicles are moving between nodes in discrete steps. This model was developed by VAS but some small modifications were made. The physical distance between nodes can vary, but the transition time between nodes is constant; τ will be used as the time index for the discrete time step in this graphical model. All vehicles in the site are modeled as electric vehicles with the same properties. Each vehicle is equipped with a battery with a set capacity, and its state of charge (SoC) fluctuates based on the vehicle's actions. The SoC decreases continuously with time τ, and at a higher rate when a vehicle is moving. A site contains a charging node and when a vehicle arrives at this node, it stays stationary for a fixed amount of time steps and charges its battery during this time. The node before a charging node, called the decision node, leads to two different nodes, where one of these is the charging node. When a vehicle arrives at the decision node, it must decide to either charge or not. After the charging node, the two paths merge again. Between two decisions, the vehicles are moving around the site according to an underlying route planning algorithm. A visualization of the model is shown in Figure 3.1, where three vehicles are moving in a site.

The decision to charge is made according to some charging policy. A policy determines the choice of action based on information available about the site. This information could for example be the vehicles' positional data and their SoCs. If a vehicle is forced to charge while the charging node is occupied, a queue arises. If the SoC of any vehicle in the site falls below the lower limit, SoC_{low}, the system fails. This limit is the lowest acceptable SoC for a vehicle. If a vehicle falls below this limit, it can be thought of as battery depletion.

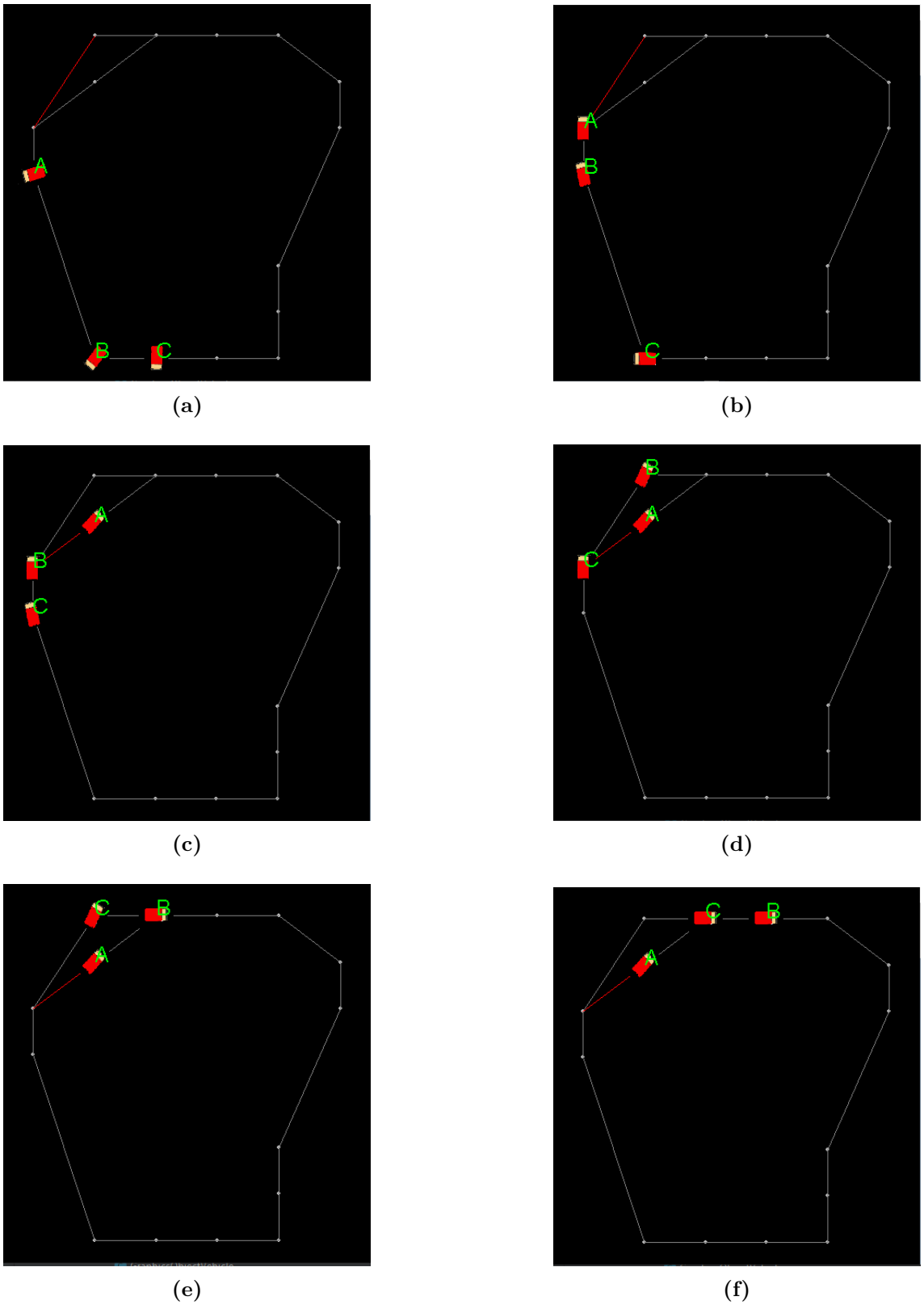


Figure 3.1: The dynamics of the graphical model of a site, illustrated with three cars and for six sequential time steps. In (b), vehicle A arrives at the decision node and chooses to charge. In (c), vehicle A arrives at the charging node, vehicle B arrives at the decision node and chooses not to charge. In (d), vehicle C arrives at the decision node and chooses not to charge.

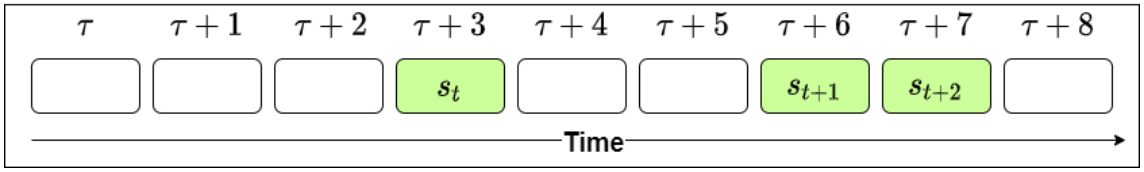


Figure 3.2: Illustration of the relationship between a time step τ and a state s_t. Each rectangle symbolize an update in the environment. For each time step τ where there is a vehicle at the decision node, a state s_t is observed. This is represented with a filled green rectangle.

3.2 MDP formulation

The decision to charge or not is modeled as a finite MDP where an agent is making decisions based on the current state of the site and a reward is given based on the consequences following that decision. This is done in the following way: each time a vehicle arrives at the decision node, a state s_t is observed. Based on this state, an action a_t is chosen to either force the vehicle at the decision node to go to the charging node, or force it to pass. The next state s_{t+1} is observed the next time a vehicle arrives at the decision node and a reward r_t is calculated. Between two states, a number of time steps occur. The difference between states s_t and time steps τ is clarified in Figure 3.2. In particular, note that the number of discrete time steps between state observations in general vary. This repeats until either a terminal state is reached or a certain amount of time has elapsed. A terminal state happens when any of the vehicles' SoC falls below SoC_{low}. The MDP is now completely described by specifying the following components:

- A state $s_t \in \mathcal{S}$ describing the observed state of the environment as a charging decision is made. The features of a state s_t can be either partially or completely describing the information about all vehicles in the site.

- The action space $\mathcal{A} = \{A_0, A_1\}$ where
  - A_0 forces the vehicle at the decision node not to charge.
  - A_1 forces the vehicle at the decision node to go to the charging node or wait in queue until the charging node is available.

- The transition function T defines the density function of the next state s_{t+1} conditioned on the pair (s_t, a_t). Irrespectively of choice of state spaces, this is given by the simulator that acts on a state space with full information of where all the vehicles are and what their respective SoC is.

- The reward for each pair (s_t, a_t), $s_t \in \mathcal{S}$ and $a_t \in \mathcal{A}$, is defined by a reward function R.

4
Solution methods

The task of finding a charging policy was solved using a reinforcement learning approach. The two methods, tabular Q-learning and deep Q-learning, are presented in the following chapter. In addition to these, a rule-based benchmark policy used to evaluate their performance is introduced.

4.1 Rule-based charging policy

A charging policy based only on the level of SoC of the vehicle at the decision node was developed as a benchmark policy. A state s_t was defined as the SoC of the vehicle at the decision node, $s_t = SoC_t^{decision}$. The action A_1 (to charge) is then chosen if s_t is below a certain threshold, $SoC_{threshold}$. The policy π_b can be expressed as

$$\pi_b(s_t) = \begin{cases} A_0 & \text{if } s_t > SoC_{threshold} \\ A_1 & \text{otherwise.} \end{cases} \tag{4.1}$$

This policy is simple, and in many cases not sophisticated enough to avoid queues. This policy is called a rule-based policy and will be used as a benchmark measurement throughout this report.

4.2 Q-learning-based charging policy

Charging policies were generated by applying Q-learning to approximate $Q^*(s, a)$, and then obtaining a policy from that function as in Equation 2.6. This was done by simulating the model with a number of N_V vehicles. Each episode was initialized by randomly placing each vehicle in the site with a random SoC in the range $[SoC_{low}+\xi$, 100]. Here, ξ is set to a constant value such that the initial SoC is enough to complete a full lap without reaching a fail state. Each time a vehicle reached the decision node, the agent observed a state s_t and chose the action a_t using an ϵ-greedy strategy. Next time a vehicle arrived at the decision node, a reward r_t was retrieved and the value $Q(s_t, a_t)$ was updated. This continued for N_E episodes where each episode lasted τ_E time steps or until a terminal state was reached. In order to apply Q-learning successfully, a state and a reward function must first be defined. Additionally, a method of storing and updating the action-value function $Q(s, a)$ must be decided. In this thesis, two methods were used: tabular Q-learning and deep Q-learning.

4.2.1 Feature selection

As mentioned earlier, a state s_t is a subset of the complete information about the environment. The complete information in this case is the position and SoC of each vehicle together with how much time left until the charging node will be available.

4.2.1.1 State s_t^{all}

The state definition with the complete information is defined as s_t^{all} and contains the following features

- The position of vehicle i, x_t^i
- The SoC of vehicle i, SoC_t^i
- Time left until the charging node is available, TC_t

where $i = 1, ..., N_V$.

4.2.1.2 State $s_t^{partial}$

Another state configuration, $s_t^{partial}$, was defined using only three features. Using less features simplifies the model and decreases the demand of memory space and computational power. The three features defining $s_t^{partial}$ were

- The SoC of the vehicle that is currently on the decision node, $SoC_t^{decision}$
- The lowest SoC of any vehicle in the site, not including the vehicles at the decision node or the charging node, SoC_t^{lowest}
- Time left until the charging node is available, TC_t.

The feature SoC_t^{lowest} is selected in order for the agent to know if there is a vehicle in the site that more urgently needs charging. Having TC_t as a feature gives the agent the opportunity to avoid queues if possible.

4.2.2 Reward function

There are many possible reward functions that can be used to solve any MDP. A reward function should be defined such that it reflects what the objective of the agent is. Therefore, a reward function used for this problem can be defined such that it rewards actions that advances the fleet of vehicles in the long-term. One way of doing this could be to penalize actions that negatively affects the progression of the fleet. A reward function that was found to be suitable and used for this problem is described below in detail.

The number of time steps between two states s_t and s_{t+1} depend on the underlying route planner and can vary. For each time step, a cost c_τ is obtained from the

environment according to

$$c_\tau = \begin{cases} -p_f & \text{if fail state} \\ -p_q & \text{if queue} \\ 0 & \text{otherwise.} \end{cases}$$

Here, $p_f > 0$ is a penalty for when any vehicle's SoC is below SoC_{low}. A smaller penalty $0 < p_q \ll p_f$ is given if there is a queue behind the charging node.

The reward r_t for taking action a_t at state s_t is then computed when the next state s_{t+1} is observed according to

$$r_t = \frac{\sum_{\tau}^{N_\mathrm{T}} c_\tau}{N_\mathrm{T}}, \tag{4.2}$$

where N_T is the number of time steps between states s_t and s_{t+1}. Thus the reward r_t is just the average cost between the two states s_t and s_{t+1}

4.2.3 Tabular Q-learning

In order to implement tabular Q-learning, all state features must take discrete values. Additionally, since the number of states can not be too large in order for $Q(s, a)$ to converge (at least in practice), state configuration $s_t^{partial}$ with discretized values was used. Algorithm 1 was used to generate a policy based on such a discretized state. Parameters used while training were chosen using trial and error.

4.2.4 Deep Q-learning

Using a neural network in order to approximate $Q^*(s, a)$, the state features can take continuous values. A network for each of the state configurations $s_t^{partial}$ and s_t^{all}, defined in Sections 4.2.1.2 and 4.2.1.1, was constructed and used for training an agent to generate policies. In order to decide the architecture of the networks used in training, hyperparameter optimization was done using a grid search approach. The parameter configurations that gave the best result during the grid search was then chosen and used during training. The training was performed following Algorithm 2.

5
Implementation and results

In this chapter, the constructed site and vehicles from VAS are presented. The aforementioned solution methods are then implemented on the modeled MDP and charging policies are generated. These generated policies are visualized and compared to a benchmark policy. Finally, the policies are applied on the site with a few small adjustments in order to test their generalizability and robustness.

5.1 Configuration of site

Design choices such as number of nodes in the site, number of vehicles, battery capacity, etc. were made such that the problem was interesting. For example, a vehicle should not be able to drive too many laps before its battery is depleted. At the same time, a vehicle's battery should not deplete too fast since that would only generate a non-complex policy that charges each time a vehicle arrives at the decision node.

The same site configuration was used for each solution method during training in order to make fair comparisons and performance evaluations. The site consisted of $N_N = 15$ nodes with one charging node. The charging time was set to $T_C = 30$ time steps. This meant that charging took approximately the same amount of time as two full laps around the site. One of the nodes in the site had a set probability $p_c = 0.8$ of being closed in each time step. This was added to increase the stochasticity in the model and to better reflect a real traffic situation with sudden stops, such as stopping for pedestrians or traffic lights. The number of vehicles in the site was set to $N_V = 3$, and their properties were the same throughout. The limit SoC_{low} was set to 30, meaning that the vehicles' SoC must be kept above this in order to avoid a fail state. The site-specific parameters are summarized in Table 5.1.

Number of nodes, N_N	15
Number of vehicles, N_V	3
Probability of closed node, p_c	0.8
Charging time, T_C	30
Lowest SoC allowed, SoC_{low}	30

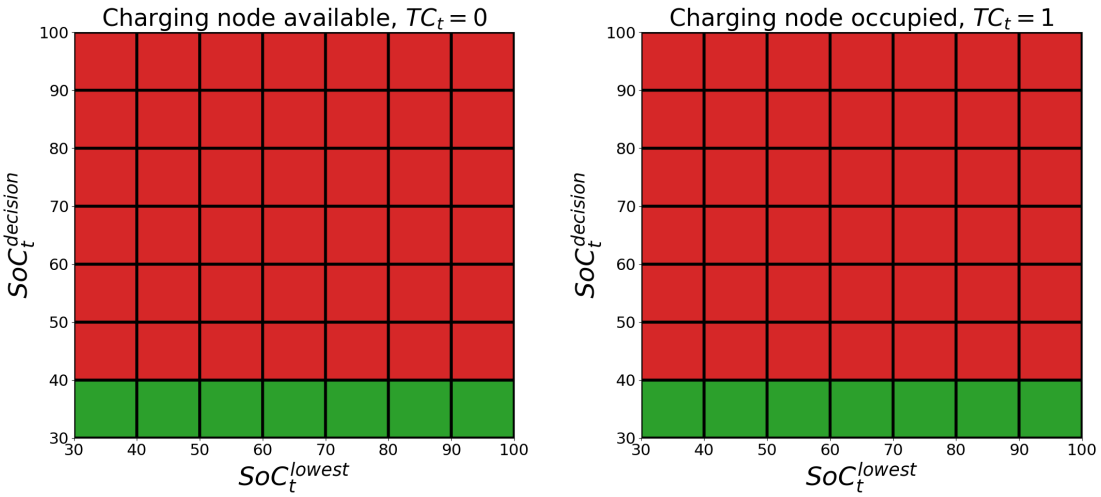
Table 5.1: Site-specific parameters and their values used during training. A charging policy was generated from each solution method with these parameters.

5.2 Rule-based charging policy configuration

A rule-based charging policy π_b of the form given in Equation 4.1 was constructed using a threshold set to $SoC_{threshold} = 40$. This was large enough for the site configuration in Section 5.1 to make sure that no vehicle ever fell below the SoC_{low} limit.

5.2.1 Resulting policy

A visualization of the policy π_b is shown in Figure 5.1 using the features of $s_t^{partial}$ to show its behavior. An agent following this policy simply takes the action to charge only if the SoC of the vehicle at the decision node is below $SoC_{threshold}$. It does not take into consideration whether the charging node is occupied or not, as well as ignores other vehicles' SoCs.



(a) **(b)**

Figure 5.1: A visualization of a rule-based charging policy with $SoC_{threshold} = 40$. With $SoC_t^{decision}$ on the y-axis and SoC_t^{lowest} on the x-axis. The plot on the left (a), shows the policy when the charging node is available, and the right plot (b), shows the policy in the case where the charging node is occupied. The red color represents action A_0, i.e., to not charge, and the green color represents action A_1, i.e., to charge.

5.3 Q-learning-based charging policy configuration

Both tabular Q-learning and deep Q-learning were implemented and trained on the site model explained in Section 5.1, with discount factor $\gamma = 0.9$. They both used the reward function described in Equation 4.2 with $p_f = 50$ and $p_q = 1$. The number of time steps per episode was set to $\tau_E = 3000$. During the training phase, the

performance of the current policy was recorded. If the performance did not improve for a certain amount of episodes, the training stopped and the best performing policy was obtained and saved. The maximum number of episodes for training was set to $N_E = 10000$.

5.3.1 Tabular Q-learning

Tabular Q-learning was implemented using a learning rate α_t that gradually changed using the formula $\alpha_t = \frac{k}{t+k}$, where $k = 100$ was used. This fulfills the criteria in Equation 2.8. An ϵ-greedy policy was used during training with a starting value of $\epsilon_0 = 1$ which exponentially decayed every episode until it reached the lower limit $\epsilon_{min} = 0.3$ at which it stopped decaying. The state configuration $s_t^{partial}$ was used and all features were discretized according to

- $SoC_t^{decision} \in \{0, 10, 20, 30, 40, 50, 60, 70, 80, 90\}$
- $SoC_t^{lowest} \in \{0, 10, 20, 30, 40, 50, 60, 70, 80, 90\}$
- $TC_t = \begin{cases} 0 & \text{if charging node is available} \\ 1 & \text{if charging node is occupied.} \end{cases}$

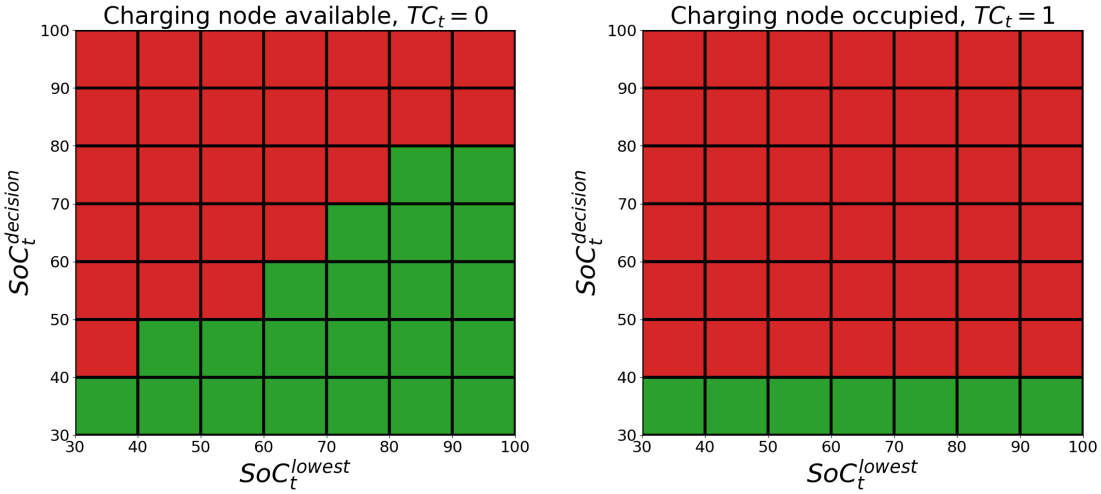
The total number of states was therefore $|\mathcal{S}| = 10 \times 10 \times 2 = 200$.

5.3.1.1 Resulting policy

The policy obtained after training is shown in Figure 5.2. Here we can see that the agent learns to charge at more occasions compared to the rule-based policy shown in Figure 5.1. It learns a behavior that takes the opportunity to charge while the charging node is available and consequently avoids unnecessary queues later on. The vehicle's SoC will be kept high and therefore avoid situations where a vehicle arrives at the decision node with low SoC when the charging node is occupied.

5.3.2 Deep Q-learning

Using deep Q-learning with a neural network as a function approximator for $Q(s, a)$, the state features are the input to the network. The features are taken as their true value, meaning there is no need to discretize the features. During training, an ϵ-greedy strategy was used with $\epsilon = 0.9$ being constant throughout. Since DQL can handle a much larger state space than tabular Q-learning, the state configuration s_t^{all} was used in addition to $s_t^{partial}$. For each state configuration, a neural network was constructed using hyperparameter optimization with grid search to determine their respective architecture. The final architectures are summarized in Table 5.2. As mentioned, two networks were constructed, one for each state configuration. Both were trained according to Algorithm 2 and their resulting policies are presented in Figure 5.3.



(a) **(b)**

Figure 5.2: A visualization of the charging policy generated with tabular Q-learning. With $SoC_t^{decision}$ on the y-axis and SoC_t^{lowest} on the x-axis. The plot on the left (a), shows the policy when the charging node is available, and the right plot (b), shows the policy in the case where the charging node is occupied. The red color represents action A_0, i.e., to not charge, and the green color represents action A_1, i.e., to charge.

State configuration	$s_t^{partial}$	s_t^{all}
Number of input neurons	$\lvert s_t^{partial} \rvert$	$\lvert s_t^{all} \rvert$
Number of hidden layers	3	3
Number of hidden neurons per layer	32	64
Number of output neurons	2	2
Optimizer	Adam	Adam
Initial learning rate for the optimizer	0.00001	0.00001
Activation function hidden layers	ReLU	ReLU
L2-regularization	0.01	0.001
Activation function output layer	Identity	Identity
Loss function	MSE	MSE
Experience replay buffer size	10000	10000
Batch size	64	128
Target network update frequency	500	500

Table 5.2: The deep Q-network architecture used for each state configuration during training.

5.3.2.1 Resulting policy with state $s_t^{partial}$

Using state $s_t^{partial}$, the resulting policy can be visualized in a similar fashion as in the tabular case. However, since all features are continuous, it is difficult to show

the policy without some simplifications. In Figure 5.3 the behavior of the policy can be seen for a subset of the state space. Each plot shows the behavior for different values of TC_t and in discrete steps of $SoC_t^{decision}$ and SoC_t^{lowest}. The policy shows a similar behavior as the tabular case when $TC_t = 0$, meaning when the charging node is available the policy prefers to charge more often. For the other values of TC_t it seems to behave similarly to a rule-based policy.

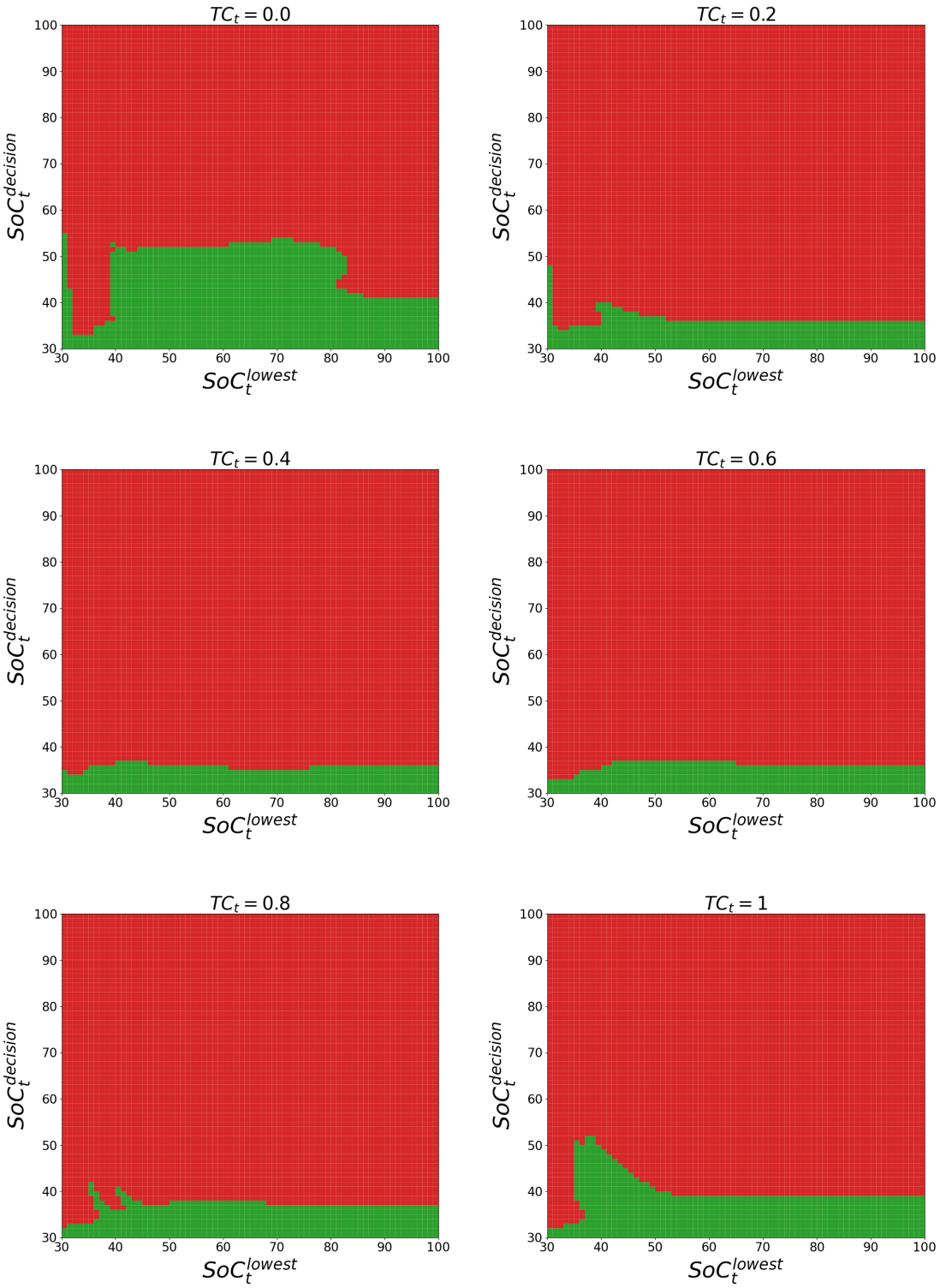


Figure 5.3: A visualization of the charging policy generated with deep Q-learning with state configuration $s_t^{partial}$. With $SoC_t^{decision}$ on the y-axis and SoC_t^{lowest} on the x-axis, both discretized. Each plot shows the policy's behavior for different values of TC_t. The red color represents action A_0, i.e., to not charge, and the green color represents action A_1, i.e., to charge.

5.3.2.2 Resulting policy with state s_t^{all}

The state configuration s_t^{all} was implemented as a vector $\boldsymbol{\phi}$ where

$$\phi_j = \begin{cases} SoC_t^i & \text{if vehicle } i \text{ is at node } j \\ 0 & \text{otherwise} \end{cases} \qquad \forall j \in \{1, 2, ..., N_N\}. \tag{5.1}$$

and $\phi_{N_N+1} = TC_t$ where N_N is the total number of nodes in the site. Hence the number of features of s_t^{all} and inputs to the network was equal to $|s_t^{all}| = 16$ for the site described in Section 5.1. As the state-action space is high dimensional, visualizing the policy is not practical in this case.

5.4 Performance evaluation

The performance of each generated policy was evaluated by simulating the site model multiple times where the charging decision making was determined by that policy. Each simulation was initialized randomly and each policy saw the exact same initializations in order to make a fair comparison. A simulation was executed for a specific number of time steps or until a fail state was reached. The metric used to compare the different policies was the number of completed trips during a simulation. A completed trip was defined as one lap around the site, see Figure 3.1. The box plots in Figure 5.4 illustrate the performance of each generated policy after 1000 simulations with a maximum of 10000 time steps each (or until a fail state was reached). The rule-based charging policy was used as a benchmark measurement and the other policies were evaluated in terms of how many more trips they completed than the rule-based case, measured in percentages. From Figure 5.4, it is clear that each of the generated policies perform better than the benchmark. However, there seem to be no definitive difference of the performance between the generated policies. Furthermore, the Table 5.3 shows additional statistics about the policies' performance. The average number of completed trips in actual numbers, together with the standard deviation of the completed trips between simulations. The same information is given about the amount of time spent in queue for the different policies.

Policy	Completed trips average	Completed trips standard deviation	Time steps in queue average	Time steps in queue standard deviation
Rule-based	1175.80	14.86	469.75	95.33
Tabular Q-learning	1203.61	13.09	1.53	6.28
DQL-$s_t^{partial}$	1203.05	12.75	3.03	7.12
DQL-s_t^{all}	1203.51	13.51	16.40	24.87

Table 5.3: Performance statistics for each policy after 1000 simulations of the site model described in 5.1 with 10000 time steps each.

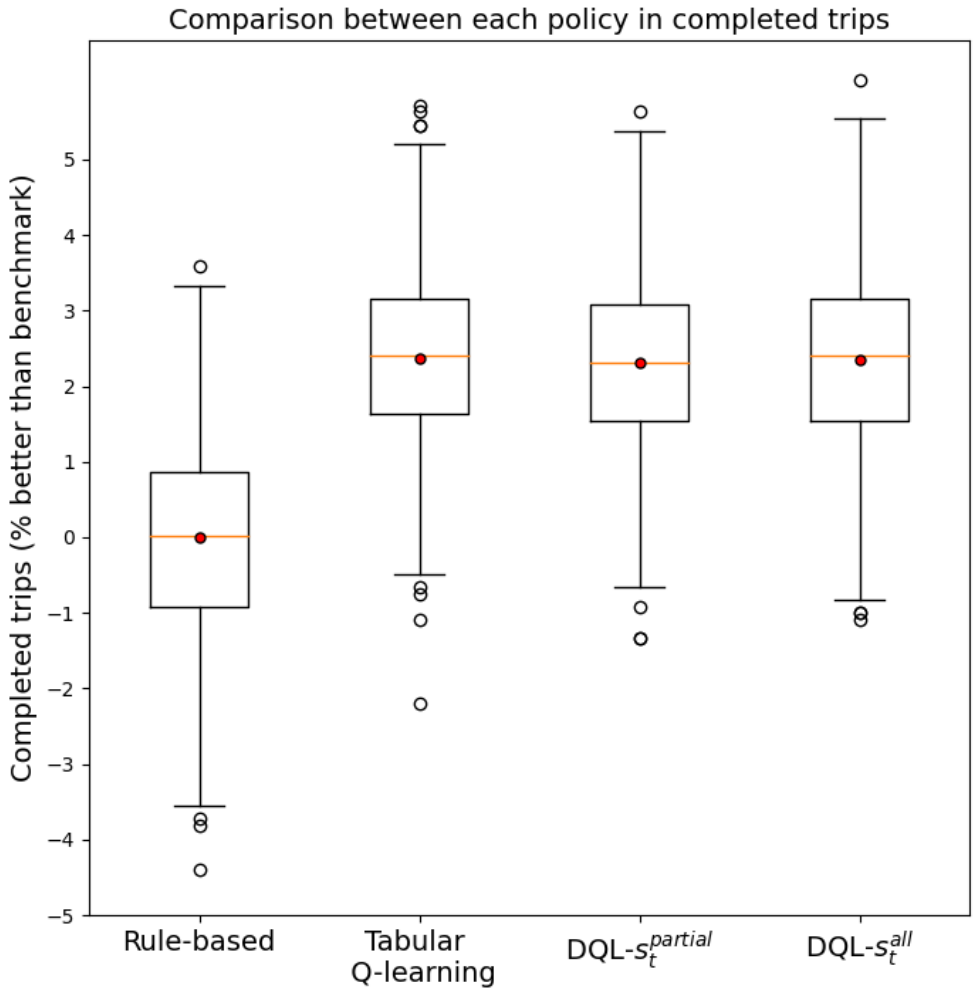


Figure 5.4: Boxplots showing the number of completed trips by different policies compared to the mean of the benchmark rule-based policy, in percentages. Each policy was used for the same 1000 randomly initialized simulations with 10000 time steps each.

5.4.1 Adjusting parameter p_c

As earlier mentioned, a certain node in the site was set to have a probability of being closed in each time step. This was implemented to represent sudden stops in real traffic situations. The probability of how often the node was closed was controlled by the parameter p_c. If $p_c = 0$, this meant that the node was always open and if $p_c = 1$, then the node was always closed. Therefore having $0 < p_c < 1$ meant that the node was closed with probability p_c in each time step. In order to see how well the different policies handled different levels of stochasticity, simulations for different values of p_c were carried out. The plot in Figure 5.5 shows the average amount of completed trips for each policy compared to the rule-based policy, shown in percentages for

different values of p_c. For each policy and value of p_c, 500 simulations with 10000 time steps were made. Note that each policy was trained with $p_c = 0.8$.

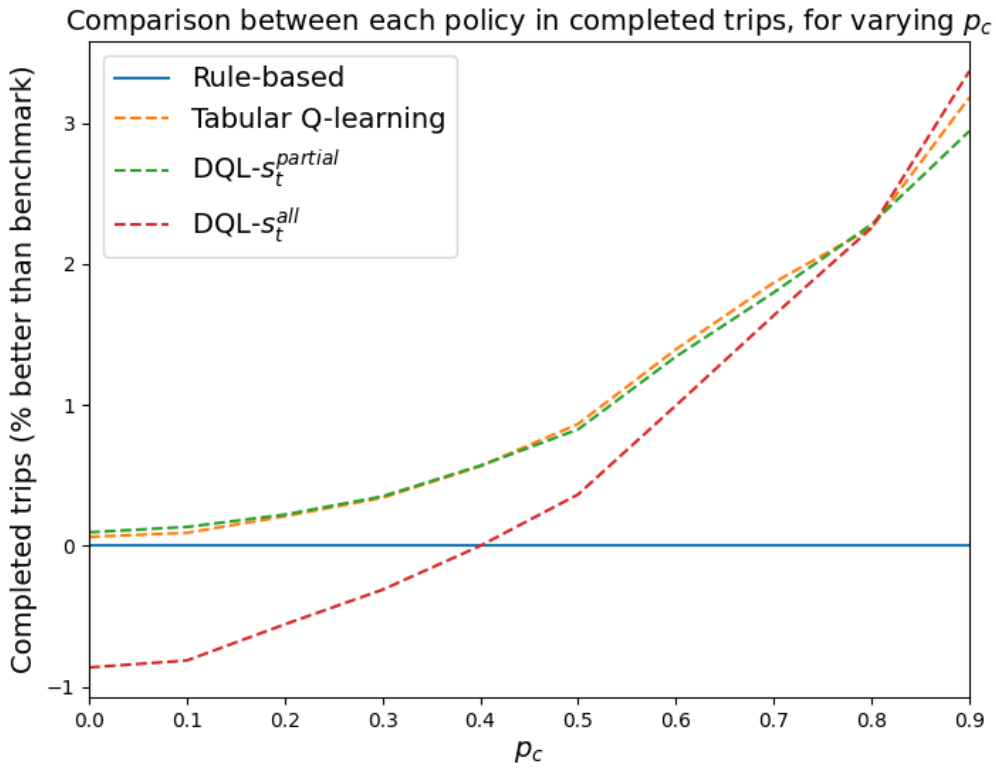


Figure 5.5: A performance comparison between different policies and the benchmark rule-based policy for different values of p_c. A total of 500 simulations with 10000 time steps each were executed for each policy. The graph shows the difference in average completed trips for each policy and value of p_c. Note that each policy was trained with $p_c = 0.8$.

5.4.2 Adjusting parameter T_C

As can be seen in Table 5.3, the time spent in queues with the generated policies is close to zero. This means that the policies have learned to avoid almost any situation where a queue happens. This likely depends on how long the charging time is. In order to see how much this affects the performance of the different policies, we adjusted the charging time and measured the performance of each policy compared to the benchmark. To maintain the same dynamic in the model, the total amount of power into a vehicle during charging remained constant. So, if the charging time increased, the power per time unit decreased and vice versa. The result of this is shown in Figure 5.6. Note that each policy was trained with $T_C = 30$.

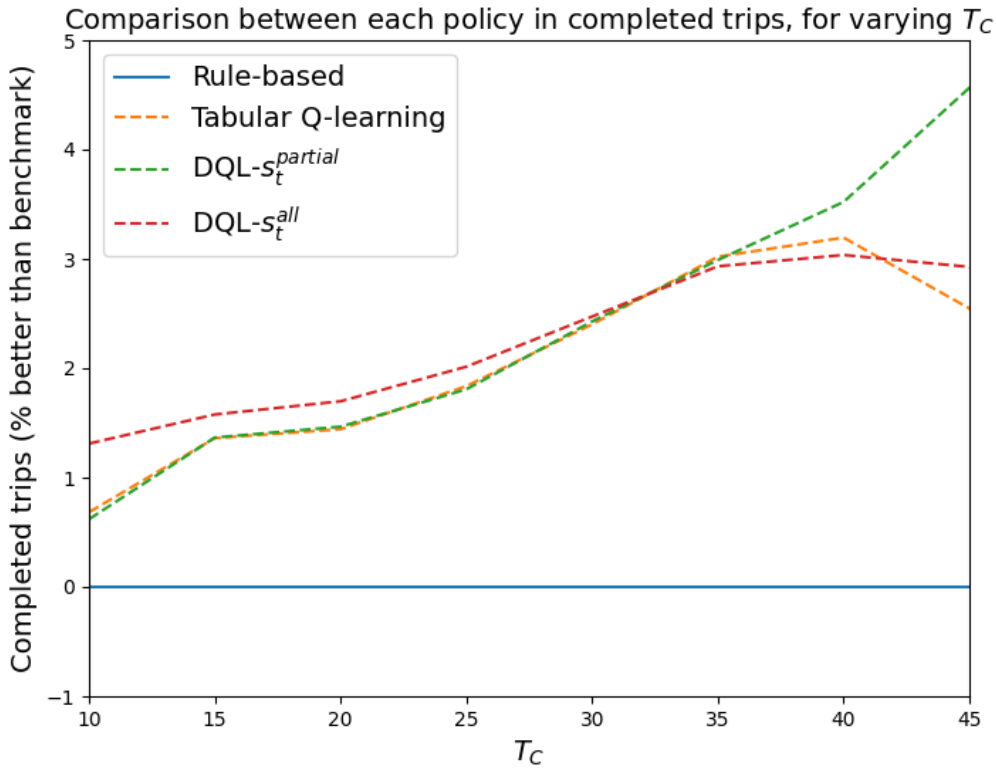


Figure 5.6: A performance comparison between different policies and the benchmark rule-based policy for different charging times. A total of 500 simulations with 10000 time steps each were executed for each policy. The graph shows the difference in average completed trips for each policy and charging time. Note that each policy was trained with $T_C = 30$.

5.4.3 Adjusting parameter N_V

To see how well the generated policies performed with a different number of vehicles than what they were trained with, a number of simulations were executed with each policy for varying N_V. This was done to further evaluate the generalizability of the policies and to investigate if more vehicles could be added successfully to the site without the need of re-training. The number of vehicles used during training was set to $N_V = 3$. The results for the simulations are shown in Figure 5.7.

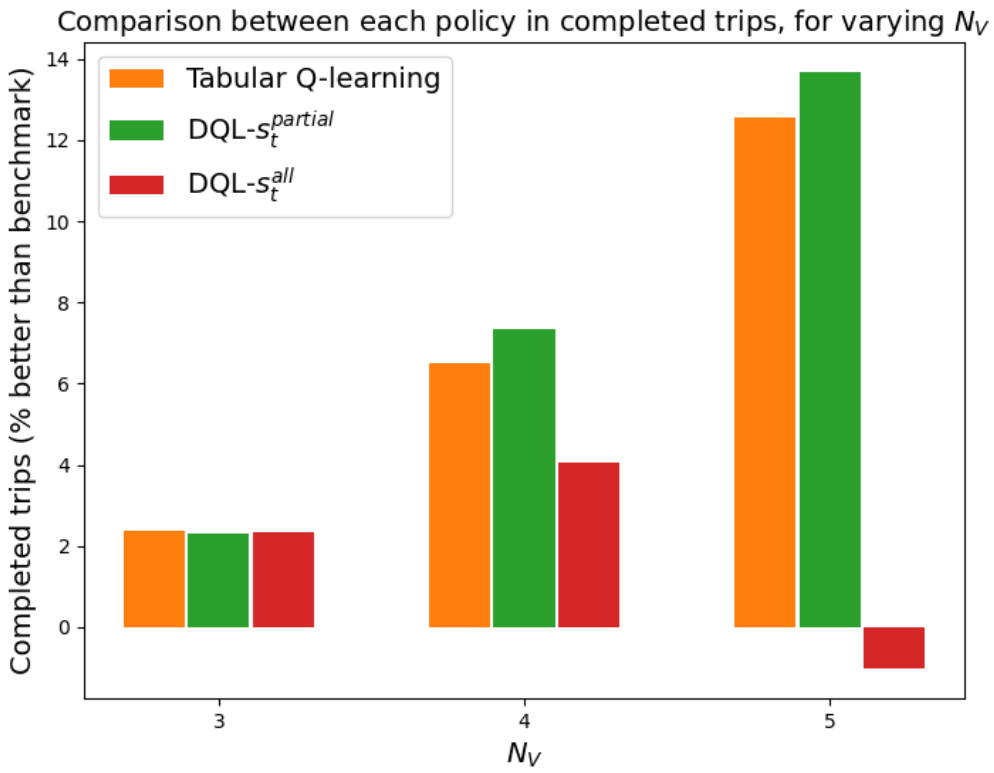


Figure 5.7: A performance comparison between different policies and the benchmark rule-based policy for different number of vehicles in the site. A total of 500 simulations with 10000 time steps each were executed for each policy. The graph shows the difference in average completed trips for each policy and number of vehicles. Note that each policy was trained with $N_V = 3$.

6
Discussion

This chapter begins with discussing the main results of the generated policies. Thereafter design choices, limitations, and future work are discussed.

6.1 Interpretation of results

The results from Chapter 5 show that policies generated from reinforcement learning are capable of outperforming a benchmark rule-based policy. In Figure 5.4, the policies all have a similar performance where they complete around 2.5% more trips than the benchmark on average. Furthermore, the Table 5.3 shows that the queue time for each generated policy is very low, and they all avoid queues almost completely.

By visually inspecting the policies generated from tabular Q-learning and deep Q-learning with state $s_t^{partial}$ shown in Figure 5.2 and Figure 5.3 respectively, a few properties can be identified. One property that both policies possess, is the fact that the action A_0 (to charge) is taken at more states when the charging node is available, $TC_t = 0$. This happens as long as the lowest SoC in the site is not in danger of reaching below the limit $SoC_{low} = 30$. Then with larger TC_t, the policies essentially follow a rule-based policy where the action to charge is only taken when the vehicle at the decision node otherwise would risk depleting its battery. This attribute is expected from a learned policy as it otherwise would cause queues to occur more often. Thus the policies have learned to charge more frequently when they can do it without causing a queue, and then wait until the charging node is available again before charging. Seeking an optimal policy for the charging decision problem has two key parts. Primarily the policy should account for the whole fleet, and thus try to be unselfish when considering to charge. Secondly, and perhaps even more important, is to account for long-term risks. If the SoC of the vehicles never comes close to the threshold the hard choices are never encountered.

Looking closer at the policy generated from tabular Q-learning in Figure 5.2, a pattern resembling a staircase function can be observed when $TC_t = 0$. This produces a behavior that approximately can be summarized as: the vehicle at the decision node takes the action to charge if the charging node is available and it has the lowest SoC in the site. The same staircase pattern is not quite revealed for the deep Q-learning policy in Figure 5.3. However, it is worth noting that the figure only shows the policy for a discretized subset of the entire continuous state-space. Exactly how the

policies' behavior differ in practice is difficult to say by only inspecting a visualization of them. As they clearly perform equivalently when evalutated, see Table 5.3, both policies are likely to operate in a similar way when deployed. One possible advantage of using deep Q-learning is that continuous values are used for the state features. This can produce a policy that more precisely can decide when charging or not charging is more beneficial.

An additional remark about the deep Q-learning policy shown in Figure 5.3 is the spikes appearing in the plots with $TC_t = 0$ and $TC_t = 0.2$ when SoC_t^{lowest} is close to $SoC_{low} = 30$. The reason for this might be because when these states are reached, a fail state will be reached regardless of the action taken. This complicates the learning for these particular states.

6.2 Altering site-configuration

The site created and used for simulation and training is greatly simplified compared to a real-life work site. It is used as a proof of concept and benchmark setting for reinforcement learning techniques on the charging decision problem. It was constructed such that a rule-based policy would be able to operate the fleet without failure, but still (most likely, at least) not do it optimally. Since it is a quite simple site there could be many policies which perform near optimally without almost any queue time when using three vehicles. This makes it hard to compare the good policies against each other, which can be seen in Figure 5.4 where the generated policies achieve almost identical results. The results still show the strength of reinforcement learning for this problem since there are several ways to find a good policy.

In order to examine how well the generated policies performed on other site-setups, they were tested with a few site-specific adjustments. One parameter is adjusted at a time, while everything else stays constant. This was done in order to investigate the generalizability and robustness of the policies when applied in different but similar environments. This could prove to be useful in real-world situations as it could be time consuming to re-train a policy for each small adjustment of a site.

6.2.1 Random stopping in the site

In Figure 5.5 we can see a clear indication that the performance of the policies compared to the benchmark policy improves as the value of p_c increase. When having $p_c = 0$, we could see that the rule-based policy ended up in a rhythm where the dynamic between states became fully deterministic. The vehicles in the site deplete their battery exactly the same amount during a trip. The vehicles also gain the same amount power during charging. This means that the vehicles would eventually end up in a perfect flow where they alternate the charging node between each other and consequently have little to no queues. As p_c increase, the flow is disrupted more often, leading to more queue times and therefore a decrease in performance. As the other policies have more information than just the SoC of the vehicle at the decision node to base their action on, the value of p_c is not affecting their performance as

much.

An interesting observation is that the policy generated from DQL using state s_t^{all} has a worse performance than the benchmark when $p_c \leq 0.4$. The reason for this might be that using all features to represent a state has a negative effect on the generalizability of that policy. The other policies use a state with only three features, but the amount of information contained in these features seem to be enough to make beneficial decisions.

6.2.2 Charging time

The benefit of using the generated policies over the rule-based policy for different charging times can be seen to vary in Figure 5.6. Note that even when the charging time changes, the total amount of power into the vehicle for one charging action remains constant. Shortening the charging time will clearly lead to less queues and we see in Figure 5.6 that the difference between the policies' performance decrease. If we instead increase the charging time and therefore also the amount of queues, we see that the generated policies can handle this better than the rule-based policy. It is also noteworthy that DQL with sate configuration $s_t^{partial}$ can deal with an increase of charging time the best out of all policies. Since it uses a continuous value for the feature TC_t (time left until charging node is available), it can make smarter decisions when the charging node is occupied. The decrease in performance of DQL using state s_t^{all} compared to the other DQL policy indicates that it is worse at generalizing.

6.2.3 Number of vehicles

Having more vehicles in the site leads to having more occasions where a queue can arise as there are more vehicles that needs to use the charging station. Additionally, when a queue happens, there are more vehicles that can potentially be affected by the roadblock. What we could see earlier in Figure 5.4 is that the efficiency of the fleet of vehicles can improve if it follows a smart policy compared to a rule-based policy. Furthermore, in the earlier sections when adjusting parameters such that queues are harder to avoid, the difference in performance between the generated policies and the benchmark increased even more. This pattern is even more pronounced when adjusting the amount of vehicles in the site, see Figure 5.7. Here we can see that the best performing policy completes almost 14% more trips than the benchmark when having five vehicles. The same number was around 2.5% in the case with three vehicles.

In Figure 5.7, we can see that especially tabular Q-learning and DQL with state $s_t^{partial}$ clearly outperforms the rule-based policy and that the difference increases when more vehicles are added. As for DQL with state s_t^{all} we can see that with three vehicles, it performs similar to the other generated policies. However its performance does not increase with the same rate as the other policies. With five vehicles this policy even performs worse than the benchmark. The most number of

vehicles tested in the site was limited to five as we saw that exceeding this number caused a lot more simulations to fail.

6.3 Design choices

When modelling the charging decision problem as an MDP some simplifications were made. It is not certain that a time discretized model would translate well to a real-world scenario. In the model used in this thesis it was assumed that the time between node transitions was fixed, independent of the distance between these nodes. The reason for such an assumption was that the training and evaluation were carried out on a simulation model provided by VAS. Additionally, the distance and time between nodes was in no way incorporated in the evaluation of the results, only the number of completed trips and queue times.

When defining the specific components of an MDP, some different approaches could be taken when choosing the state-space and the reward function. Since an action was binary there was no other setup for the action-space. A state could be defined in several ways. One idea that was considered was to let a state be observed each time an update in the system happened, and not only when a vehicle arrives at the decision node. This would give the algorithm more detailed information about what happens right after an action. Thus, the transition and reward functions would not be very complex. But the long-term consequences of an action would be more difficult to analyze since there could be many states before something interesting happens, such as battery depletion or queuing. This would require a large γ and might affect the learning. The state configuration used in this thesis instead have more complex transition and reward functions. Their advantage comes from having more linkage between state-action pairs and the actual consequences they cause.

The reward function should resemble the main goal but does not necessarily need to depend on the same metrics which are used to evaluate the policy. In this thesis work, the reward function used depended on the queue times, while the main evaluator was the number of completed trips. The two are of course correlated and therefore good policies were able to be produced. Other reward functions were investigated, especially a reward function rewarding forward movements was thoroughly tested. Using such a reward function, the generated policies were able to score better than the benchmark, but still worse than the policies produced using the queue time-based reward function. The main problem seemed to be that the long-term benefits of charging was hard to learn, and thus the agent focused more on the immediate loss of reward if they decided to charge. Another candidate reward function considered was to only give a penalty for battery depletion. This produced policies where the vehicles tried to charge all the time and there were instead queues most of the time steps.

6.4 Future work

To investigate the potential for Q-learning, more complex dynamics and interesting sites could be used. One such change in the dynamics could be non-linear charging output. In this thesis work the charging efficiency was always the same, regardless of the SoC. In practice it is common that the charging efficiency increases as the battery's SoC gets lower, and vice versa. This would certainly have effects on the optimal charging policy. While it would still be true that a good charging policy would avoid low SoC in fear of depletion, it would also want to charge at as low SoC as possible, to maximize charging output. This change could potentially make good policies even more important.

7
Conclusion

The aim of this thesis work, as declared in Section 1.2, was to find ways to optimize a charging policy for a fleet of autonomous electric vehicles, using reinforcement learning. The results show that the chosen methods are capable of learning a good behavior, see Figure 5.4 for the main comparison against the benchmark. For this specific site the generated policies could perform around 2.5% more completed trips than the benchmark. By modelling the system as a Markov decision process, and then use different variants of Q-learning, we tried to approximate the optimal action-value function $Q^*(s, a)$. This approximation was then used to generate a policy, and a simple benchmark policy was used to evaluate the generated policies. The methods used to approximate $Q^*(s, a)$ was the classic tabular Q-learning and deep Q-learning, which uses neural networks as function approximator. Two state representations were used, one of them for tabular Q-learning and both for DQL.

By doing small adjustments in the site, such as adding more vehicles, varying the charging duration, and altering the stochasticity, the generalizability of the generated policies was compared. The comparisons, presented in Figure 5.5, Figure 5.6, and Figure 5.7, show that the policies generated by both Q-learning and DQL could handle small changes well, in some cases up to 13% more completed trips than the benchmark. These results also show that having a state configuration with less features generally generated policies that are better at handling such changes, i.e., that generalize better.

Although all the training and simulations were executed on one small site model, the methods used are plug-and-play and should thus be able to handle other sites and setups as well. However, we want to note that parameter tuning was a non-negligible part of producing the policies. Therefore, we expect that a certain amount of work needs to be put into tuning these when adopting the framework for learning charging policies for other sites and underlying models.

Bibliography

[1] Volvo Autonomous Solutions, "Why autonomous transport will happen in quarries and light mining first," 2020, (Accessed: 24-May-2022). [Online]. Available: https://www.volvoautonomoussolutions.com/en-en/news/press-releases/2020/sep/why-autonomous-transport-will-happen-in-quarries-and-light-mining-first.html

[2] ——, "Full speed ahead on autonomous transport solutions for ports," 2021, (Accessed: 24-May-2022). [Online]. Available: https://www.volvogroup.com/en/news-and-media/news/2021/may/full-speed-ahead-on-autonomous-transport-solutions-for-ports.html

[3] ——, "Optimizing productivity through complete autonomous solutions," 2022, (Accessed: 24-May-2022). [Online]. Available: https://www.volvoautonomoussolutions.com/en-en/our-solutions.html

[4] S. Bahrami and M. Parniani, "Game theoretic based charging strategy for plug-in hybrid electric vehicles," *IEEE Transactions on Smart Grid*, vol. 5, no. 5, pp. 2368–2375, 2014.

[5] F. Parise, M. Colombino, S. Grammatico, and J. Lygeros, "Mean field constrained charging policy for large populations of plug-in electric vehicles," in *53rd IEEE Conference on Decision and Control*, 2014, pp. 5101–5106.

[6] S. Sachan, S. Deb, and S. N. Singh, "Different charging infrastructures along with smart charging strategies for electric vehicles," *Sustainable Cities and Society*, vol. 60, p. 102238, 2020.

[7] I. Fernández, C. Calvillo, A. Sánchez-Miralles, and J. Boal, "Capacity fade and aging models for electric batteries and optimal charging strategy for electric vehicles," *Energy*, vol. 60, pp. 35–43, 2013.

[8] S. Mehar, S. M. Senouci, and G. Rémy, "Ev-planning: Electric vehicle itinerary planning," in *2013 International Conference on Smart Communications in Network Technologies (SaCoNeT)*, vol. 01, 2013, pp. 1–5.

[9] M. Baum, J. Dibbelt, L. Hübschle-Schneider, T. Pajor, and D. Wagner, "Speed-Consumption Tradeoff for Electric Vehicle Route Planning," in *14th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems*, ser. OpenAccess Series in Informatics (OASIcs), S. Funke and M. Mihalák, Eds., vol. 42. Dagstuhl, Germany: Schloss

Dagstuhl–Leibniz-Zentrum fuer Informatik, 2014, pp. 138–151. [Online]. Available: http://drops.dagstuhl.de/opus/volltexte/2014/4758

[10] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. v. d. Driessche, T. Graepel, and D. Hassabis, "Mastering the game of Go without human knowledge," *Nature*, vol. 550, no. 7676, pp. 354–359, 2017.

[11] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep reinforcement learning: A brief survey," *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 26–38, 2017.

[12] Y. Li, "Deep reinforcement learning: An overview," *CoRR*, vol. abs/1701.07274, 2017. [Online]. Available: http://arxiv.org/abs/1701.07274

[13] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*, 2nd ed. MIT press, 2020.

[14] C. J. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, no. 3, pp. 279–292, 1992. [Online]. Available: https://doi.org/10.1007/BF00992698

[15] D. Silver, A. Huang, C. Maddison, A. Guez, L. Sifre, G. Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, pp. 484–489, 01 2016.

[16] A. Krenker, J. Bester, and A. Kos, "Introduction to the artificial neural networks," in *Artificial Neural Networks*, K. Suzuki, Ed. Rijeka: IntechOpen, 2011, ch. 1. [Online]. Available: https://doi.org/10.5772/15751

[17] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, no. 5, pp. 359–366, 1989.

[18] O. I. Abiodun, A. Jantan, A. E. Omolara, K. V. Dada, N. A. Mohamed, and H. Arshad, "State-of-the-art in artificial neural network applications: A survey," *Heliyon*, vol. 4, no. 11, p. e00938, 2018.

[19] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014.

[20] H. v. Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, ser. AAAI'16. AAAI Press, 2016, p. 2094–2100.

[21] V. Mnih, K. Kavukcuoglu, D. Silver, A. Rusu, J. Veness, M. Bellemare, A. Graves, M. Riedmiller, A. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–33, 02 2015.

DEPARTMENT OF MATHEMATICAL SCIENCES
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY