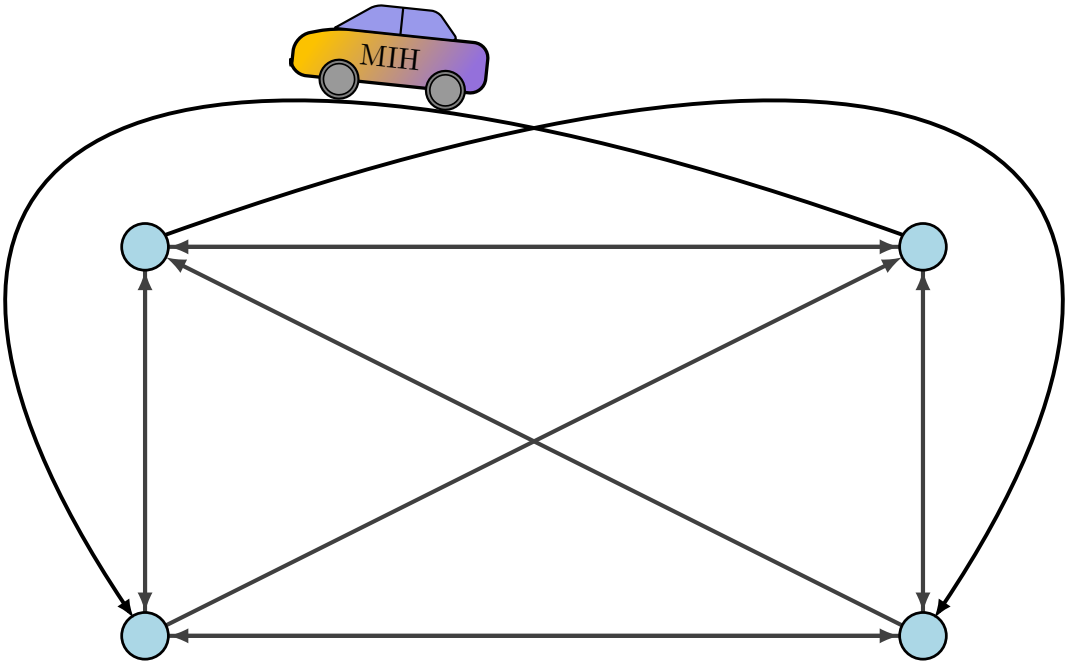


Bayesian Reinforcement Learning on Semi–Markov Decision Processes

Exploring Risk–Averse Decision–Making in Dynamic Vehicle Routing Problems

Master's Thesis in Complex Adaptive Systems
MARCUS HILDING SÖDERGREN

Master's Thesis in Engineering Mathematics and Computational Sciences
SAMUEL VREDE

DEPARTMENT OF MATHEMATICAL SCIENCES
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2020
www.chalmers.se

Master's thesis 2020

Bayesian Reinforcement Learning on Semi–Markov Decision Processes

Exploring Risk–Averse Decision–Making in Dynamic Vehicle Routing Problems

MARCUS HILDING SÖDERGREN
SAMUEL VREDE

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Mathematical Sciences
Chalmers University of Technology
Gothenburg, Sweden 2020

Bayesian Reinforcement Learning on Semi–Markov Decision Processes
Exploring Risk–Averse Decision–Making in Dynamic Vehicle Routing Problems
MARCUS HILDING SÖDERGREN
SAMUEL VREDE

Supervisors:
Gustaf Ehn, Syntronic
Marina Axelson–Fisk, Mathematical Sciences, Chalmers University of Technology
Examiner:
Marina Axelson–Fisk, Mathematical Sciences, Chalmers University of Technology

Master's Thesis 2020
Department of Mathematical Sciences
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: A graph representing the model developed in this thesis.

Typeset in LaTeX
Gothenburg, Sweden 2020

Bayesian Reinforcement Learning on Semi–Markov Decision Processes
MARCUS HILDING SÖDERGREN
SAMUEL VREDE
Department of Mathematical Sciences
Chalmers University of Technology

Abstract

Automated decision making is a highly relevant research topic in the context of supply chains as the usage of automated vehicles is increasing. Further, utilising historical data and domain knowledge to increase the performance and ensure robust behaviour of the decision–making agent is valuable from an industry point of view. The *dynamic vehicle routing problem* (DVRP) is a routing problem where costs of stochastically and dynamically appearing tasks in a system are minimised. Further, the *semi–Markov decision process* (SMDP) is an extension of the, in *reinforcement learning* commonly used, *Markov decision process* (MDP), that allows for modelling systems with stochastic state and time transition dynamics. In *Bayesian reinforcement learning*, ideas from Bayesian statistics are incorporated with reinforcement learning as a method to obtain better models based on historical data. In this thesis, we study how SMDPs can be applied, within the context of Bayesian reinforcement learning, to the DVRP, while considering risk–aversion. We develop an SMDP model of the DVRP and a Bayesian reinforcement learning solver for SMDPs, and show that our solver is able to outperform a naive routing strategy such as *first in, first out* (FIFO). Our results show that the, to our knowledge, novel idea of applying SMDPs in a Bayesian reinforcement learning context to the DVRP is promising, though further work is needed. In addition, while we have incorporated risk–aversion into our solver we believe that the topic of risk–aversion needs further study. Based on the results, and by the development of the research fields, we believe that the ideas covered in this thesis deserve, and will get, more research attention. More autonomous decision making in the global supply chains is interesting from multiple perspectives. Improved decision making may result in more rapid supply chains while improving the efficiency resulting in reduced resource consumption. Further, incorporating risk–aversion into the decision making may lead to less fragile supply chains, potentially reducing the impacts caused by unexpected events and disturbances.

Keywords: Bayes–adaptive Monte–Carlo planning, Bayesian reinforcement learning, dynamic vehicle routing, Monte–Carlo tree search, risk measure, robustness, semi–Markov decision process.

Acknowledgements

First and foremost, we wish to thank our supervisors, Gustaf Ehn at Syntronic, and Marina Axelson–Fisk at Chalmers University of Technology. Without your continuous support and valuable advice this thesis would not have been possible. Also, we are very grateful for the support we have received from Syntronic and all our co–workers. During the time that we got to spend with you at the Gothenburg office we were surrounded by a positive and stimulating atmosphere.

The computational resources provided by the department of Mathematical Sciences at Chalmers University of Technology proved invaluable, and for that we are thankful. In addition, we want to reach out and show our full gratitude to Petter Mostad, for answering our Bayesian related questions, to Daniel Westerlund, for your IT–support, and to Victor Wåhlstrand Skärström, for being there when our LaTeX–skills were not sufficient.

Lastly, a great thanks to everyone that have contributed to making our years at Chalmers to the great fun time it has been. Hurra!

Marcus Hilding Södergren and Samuel Vrede, Gothenburg, June 2020

Contents

List of Figures

List of Tables

List of Symbols

a	An action in $\mathcal{A}$.
$\mathcal{A}$	Set of actions.
α	Weight parameter of a risk measure.
$b(\cdot)$	Believed probability to be in a certain state.
β	Learning rate (step size). Takes values in $(0, 1]$.
c	The UCT constant used in MIH.
γ	Discount factor. Takes values in $[0, 1]$.
$\mathsf{E}_{P,\mu}$	Expected value given state transition probability P and policy μ.
ϵ	The probability of selecting a random action in a roll-out in MIH.
h	A history of states $s \in \mathcal{S}$ and actions $a \in \mathcal{A}$ observed. Belongs to $\mathcal{H}$.
$\mathcal{H}$	The set of histories.
$\mu(\cdot\|s)$	Policy for action–choice given a state $s \in \mathcal{S}$.
n	The number of simulations performed per decision epoch in MIH.
N	The set of natural numbers. Includes 0.
o	A (partial) observation in $\mathcal{O}$ of a state.
$\mathcal{O}$	The set of observations.
$\Omega(\cdot\|s, a)$	Probability distribution over observations given that action $a \in \mathcal{A}$ lead to state $s \in \mathcal{S}$.
$\mathcal{P}(\cdot\|s, a)$	State transition probability given state $s \in \mathcal{S}$ and action $a \in \mathcal{A}$.
$P_{prior}(\mathcal{P})$	Prior distribution over transition probabilities $\mathcal{P}$.
$P(\mathcal{P}\|h)$	Posterior distribution over transition probabilities $\mathcal{P}$ given a history $h \in \mathcal{H}$.
$\mathcal{P}_0(\cdot)$	Probability distribution of initial state.
$\mathcal{P}^+(\cdot\|\ s, h\ , a)$	Augmented state transition probability given augmented state $s, h \in \mathcal{S}^+$ and action $a \in \mathcal{A}$.
$\mathsf{P}(B)$	The probability distribution over Borel subsets of a set B.
$Q^\mu(s, a)$	(Discounted) Action–value function of the policy μ.
$Q\ (s, a)$	(Discounted) Optimal action–value function of the policy μ.
$r(s, a)$	Reward rate of the state–action pair (s, a) in an SMDP.
$R(s, a)$	Instant reward of the state–action pair (s, a) in an SMDP.
$\mathcal{R}(s, a)$	Reward obtained from taking action $a \in \mathcal{A}$ when in state $s \in \mathcal{S}$.

$R^+(s,h,a)$	Augmented reward obtained from taking action $a \in A$ when in augmented state $s,h \in S^+$.
$\mathbb{R}$	The set of real numbers.
ρ	A risk measure that maps the set of possible rewards to a real number.
s	A (current) state in S.
(s,a)	State–action pair.
s,h	An augmented state in S^+.
S	Set of states.
S^+	Set of augmented states $s,h \in S \times H$.
t_{max}	The tree depth of each simulation in MIH, expressed in time units.
$V^\mu(s)$	(Discounted) Value function of the policy μ.
$V^*(s)$	(Discounted) Optimal value function of the policy μ.
Υ_m	Sojourn time between decision epoch $m-1$ and m in an SMDP.
Z	The cost related to an outcome. Can be interpreted as a negative reward.
$\mathbb{1}$	The indicator function.

List of Abbreviations

BAMCP	Bayes–Adaptive Monte–Carlo Planning
BAMDP	Bayes–Adaptive Markov Decision Process
CTMC	Continuous–Time Markov Chain
CVaR	Conditional Value at Risk
DVRP	Dynamic Vehicle Routing Problem
DTMC	Discrete–Time Markov Chain
DTRP	Dynamic Travelling Repairman Problem
FIFO	First In, First Out
MDP	Markov Decision Process
MIH	Make It Happen
MSD	Mean–SemiDeviation
POMDP	Partially Observable Markov Decision Process
RAMCP	Robust Adaptive Monte Carlo Planning
SMART	Semi–Markov Average Reward Technique
SMDP	Semi–Markov Decision Process
TD	Temporal–Difference
TSP	Travelling Salesman Problem
UCT	Upper Confidence bounds applied to Trees
VaR	Value at Risk
VRP	Vehicle Routing Problem

1 Introduction

In a highly interconnected global society, route planning plays an important role in global supply chains. Traditionally, vehicle routing problems have been studied as static problems and models in the optimisation and operation research domains. However, the great increase in available computational power has during the last decades enabled the inclusion of more realistic aspects into these models. Examples of such aspects are dynamism and stochasticity, which has resulted in, for example, the *dynamic vehicle routing problem* (DVRP). In the DVRP, we seek to maximise some utility in an environment where all future tasks are not necessarily known at the time of a decision but are instead revealed as time progresses. In addition, the system is stochastic in the sense that actions that we may perform, have uncertain outcomes associated with them.

A solution method that has not been applied to DVRP traditionally is *reinforcement learning*. Reinforcement learning, as a technique, aims to mimic the way humans and animals are believed to learn. It is a goal–oriented approach where an agent attempts to maximise the obtained reward while acting in, and learning about, a stochastic environment (Dearden and Andre 1999). It is common for reinforcement learning methods to maximise the average reward obtained. This may not be a problem, though it clearly exists scenarios when it is undesirable. For instance, if maximising the average reward in the context of supply deliveries results in that most days the delivery is made quickly, but some days are not made at all, it may not be a tolerable variance in some cases. This problem could be alleviated by including risk assessment in the planning, resulting in a so–called *robust* solution. In our work, risk awareness will be one of the goals when training an agent. To do so means that we will try to minimise the risk of bad outcomes occurring due to the actions of an agent. For this, inspiration is taken from Majumdar and Pavone (2017) that argues for the incorporation of *risk measures*, widely used in economics, into technical applications. The possibility to measure how risky a move is in a given situation opens for the possibility to either optimise for minimal risk exposure or to avoid making moves that are associated with a high risk.

1.1 Purpose

The DVRP is of great interest since the planning of dynamic processes is becoming more important with the increasing interest in autonomous agents. As reinforcement learning has not yet been applied to this topic to a great extent, it is of interest to study this application. Further, from an industry point of view, it is interesting to include risk awareness to produce algorithms with known uncertainty characteristics. Solving specific DVRP problems using such techniques could potentially allow for more sound autonomous decision making than currently possible, especially in environments with a high degree of stochasticity and dynamism.

1.2 Scope and Outline

In this work, we limit ourselves to Bayesian reinforcement learning methods utilising semi–Markov decision processes for finding solutions to a specifically defined variant of the DVRP. The semi–Markov decision processes are limited to the time homogeneous case with discrete state–space and action–space. Further, the computational complexity of the algorithms developed is not studied. Lastly, the scope of this work is limited to producing numerical proof of concepts, rather than providing theoretical proofs of convergence.

In Chapter 2, we present the theoretical framework of this report. First, we give an introduction to the topic of reinforcement learning, starting by covering Markov processes and gradually extending these to semi–Markov decision processes. Following this, we cover some existing reinforcement learning solvers relevant to this work. Then, an introduction to robustness and some specific risk measures are covered, followed by an introduction to the topic of dynamic vehicle routing. In Chapter 3, we start by describing our DVRP model, followed by an in–depth explanation of our developed algorithm and different configurations of it that has been evaluated. Lastly, we cover some policies used as a baseline reference to compare our algorithm with. The results in terms of the performance of our solver is presented in Chapter 4 which is followed by a discussion and suggestions for future work in Chapter 5. Lastly, we present our conclusions in Chapter 6.

2 Theory

This chapter will introduce the concepts and techniques that are used throughout this thesis. To begin with, one central idea in reinforcement learning, namely the Markov decision processes, is presented. It is then expanded upon into more specific versions that are of interest in our applications. Following this, existing methods for solving problem formulations similar to ours are explored. One aspect that we are interested in is robust decision making. Thus, we continue by introducing and explaining some risk measures. Finally, to put this theory into a relevant problem formulation, the field of dynamic vehicle routing is covered.

2.1 Reinforcement Learning

Reinforcement learning is a machine learning technique that has become popular in recent years, with the increase in computational capacity. A key aspect of reinforcement learning is to employ a step by step approach to make an algorithm perform better and better decisions as it explores more of its search space. A common approach when building models for reinforcement learning algorithms is to create isolated decision events that combined form sequences. By assuming that these events are independent of earlier events, the sequences form Markov chains, that have the property of not being history–dependent. This condition is somewhat restricting in some applications, and the goal of this section is to build up to a more general framework that allows for some relaxation of this property.

2.1.1 Markov Processes

When studying a system, it is not uncommon to view it as a set of states with probabilities of transitioning between the states. A reasonable start when modelling this kind of systems is to assume that the *states* s belong to a *state–space* S. In addition, if we assume that transitions between states follow some distribution, we may characterise the state of the system at time t by the random variable $X_t, t \quad T$, where T is an index set which may be discrete or continuous. Let $X = \{X_t\}_{t \ T}$. The stochastic process X is called a *Markov process* if, given the current state X_t, the future of the process is independent of past states. In other words, X is called a Markov process if it fulfils the so–called *Markov property* (also known as *Markov condition* or *Markovian assumption*), which for $T = \mathbb{N}$ is defined as

$$P(X_{t+1} = s | X_0 = s_0, \ldots, X_t = s_t) = P(X_{t+1} = s | X_t = s_t) \qquad t \geq 1, s, s_0, \ldots, s_t \quad S. \tag{2.1}$$

While the naming varies in the literature we choose to call a Markov process with discrete and finite state–space S a *Markov chain*. Further, if T is a discrete set, as in Equation (2.1), we have a *discrete–time Markov chain* (DTMC). For a continuous set T, we have a *continuous–time Markov chain* (CTMC). We choose the initial state of a Markov process according to some initial probability distribution $P_0(\cdot) \quad \mathbb{P}(S)$, where $\mathbb{P}(B)$ denotes the set of probability distributions over subsets of a set B. We assume that both the transition probability, given in Equation (2.1), and P_0 are *stationary distributions*, that is,

the distributions will not change as time evolves in the system. (Grimmett and Stirzaker 2001)

2.1.2 Markov Decision Processes

A structure that is commonly used to model sequential decision making is the *Markov decision process* (MDP). The concept expands on Markov processes by letting the next state not only depend on the current state, but also on a decision or *action* made in the current state. Each moment where a new decision is required is called a *decision epoch*. More formally we say that the current state s, takes values in the state–space S, as in a Markov process. Similarly, an *action*, $a \quad A$, takes values in the set of possible actions A. A pair consisting of a state and an action is called a *state–action pair* (s, a). In this report, we limit ourselves to finite sets of states and actions, although the theory of MDPs support both infinite state– and action–spaces.

Transitions between states are determined by a probability distribution $P(\cdot|s, a) \quad \mathsf{P}(S)$ that maps a state–action pair to a probability distribution over the next state. For every state–action pair, we also have a bounded reward function $R(s, a) : S \times A \rightarrow \mathsf{R}$. This reward poses a central part in the modelling of MDPs and is used for specifying what, but not how, an agent should learn (Sutton and Barto 2018). As for the Markov process, we also include a probability distribution over the initial state $P_0(\cdot) \quad \mathsf{P}(S)$. To summarise, an MDP is characterised by five properties (Dimitrakakis and Ortner 2019; Ghavamzadeh et al. 2015; Mcmahon 2008):

- S the set of states,
- A the set of actions,
- $P(\cdot|s, a) \quad \mathsf{P}(S)$ the probability distribution over the next state if action a is taken in state s,
- $R(s, a) : S \times A \rightarrow \mathsf{R}$ the bounded reward function for state–action pair (s, a),
- $P_0(\cdot) \quad \mathsf{P}(S)$ the probability distribution of the initial state,

where $P(\cdot|s, a)$ and $P_0(\cdot)$ are assumed to be stationary distributions.

2.1.3 Policies

The process of determining what action to take in a given state is governed by a decision rule called a *policy*. We call the policy that always performs the best action in every state an *optimal policy*. Moreover, we call an optimal policy a *solution* to the MDP as it determines how to navigate through the states and actions in the best possible way. When considering policies for MDPs we generally focus on *stationary Markov policies*, meaning that they will not change with time (stationary) and only depend on the current state (Markov). Formally, we denote a policy $\mu(\cdot|s) \quad \mathsf{P}(A)$, meaning that for a given state it determines a probability distribution over the possible actions. The optimal policy is denoted by μ . In the special case when this distribution is degenerate and only gives a specific action for each state we have a *deterministic policy*. (Dimitrakakis and Ortner 2019; Ghavamzadeh et al. 2015)

2.1.4 Bellman Equations

To find a solution to an MDP, that is to find an optimal policy, one needs to define a measure of how good it is to be in a specific state. An important such measure is the *value function*, which, given a state s and a policy $\mu(\cdot|s)$, is defined as the expected value of the

sum of future rewards,

$$\mathsf{E}_{P,\mu}\left[\sum_{t=0}^{\infty} R(S_t, A_t)\Big|S_0 = s\right]. \tag{2.2}$$

Then, in order to ensure convergence of the series in Equation (2.2), it is common to add a *discount factor*, $\gamma \quad [0,1]$, to the reward in each iteration. This results in the *discounted value function* (sometimes referred to as only the value function),

$$V^{\mu}(s) = \mathsf{E}_{P,\mu}\left[\sum_{t=0}^{\infty} \gamma^t R(S_t, A_t)\Big|S_0 = s\right]. \tag{2.3}$$

The discount factor may, in addition of being viewed as a tool to ensure convergence, be interpreted to capture the aspect of uncertainty regarding distant rewards. This uncertainty is due to that the next state, given a state and an action, is stochastic and thus, by adding a discount factor, we reduce the impact of more distant rewards in the decision making. As mentioned, an optimal policy is a policy that takes the best action in every state. This means that a policy that maximises the value function is optimal. In return, we get the *optimal value function* (Ghavamzadeh et al. 2015)

$$V^{\ }(s) = \sup_{\mu} V^{\mu}(s), \qquad s \quad S. \tag{2.4}$$

Another useful measure, that values the state–action pair, is the *action–value function*, $Q^{\mu}(s,a)$. With discount this measure is defined as the expected sum of discounted future rewards given that action a is taken in state s, and can be written as

$$Q^{\mu}(s,a) = \mathsf{E}_{P,\mu}\left[\sum_{t=0}^{\infty} \gamma^t R(S_t, A_t)\Big|S_0 = s, A_0 = a\right]. \tag{2.5}$$

The difference from the value function in Equation (2.3) is that the first action that is taken, a_0, is not (necessarily) drawn from the policy $\mu(\cdot|s)$, but can be an arbitrarily chosen action $a \quad A$. Similarly, as for the value function, we define the *optimal action–value* function as

$$Q^{\ }(s,a) = \sup_{\mu} Q^{\mu}(s,a), \qquad (s,a) \quad S \times A. \tag{2.6}$$

As these measures can be computationally complex, we need to utilise some kind of algorithm to compute them. We start with the discounted value function from Equation (2.3) and expand the sum one step ending up with the following expression

$$\begin{aligned} V^{\mu}(s) &= \mathsf{E}_{P,\mu}\left[\sum_{t=0}^{\infty} \gamma^t R(S_t, A_t)\Big|S_0 = s\right] \\ &= \sum_{a \ A} \mu(a|s)\left(R(s,a) + \mathsf{E}_{P,\mu}\left[\sum_{t=1}^{\infty} \gamma^t R(S_t, A_t)\Big|S_0 = s, A_0 = a\right]\right) \\ &= \sum_{a \ A} \mu(a|s)\left(R(s,a) + \gamma\mathsf{E}_{P,\mu}\left[\sum_{t=1}^{\infty} \gamma^{t-1} R(S_t, A_t)\Big|S_0 = s, A_0 = a\right]\right) \end{aligned} \tag{2.7}$$

$$= \sum_{a \in A} \mu(a|s) \left(R(s,a) + \gamma \sum_{s \in S} P(s'|s,a) \mathbb{E}_{P,\mu} \left[\sum_{t=1}^{\infty} \gamma^{t-1} R(S_t, A_t) \Big| S_0 = s, A_0 = a, S_1 = s' \right] \right)$$

$$= \sum_{a \in A} \mu(a|s) \left(R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) V^{\mu}(s') \right). \tag{2.7}$$

This gives us a recursive relation that is called the *Bellman equation for* V^{μ}. The recursion is an important property as it relates the current value function to the value functions in coming states.

From the second line of Equation (2.7), we get that the value function, V^{μ}, may be expressed in terms of the action–value function, Q^{μ}, as follows

$$V^{\mu}(s) = \sum_{a \in A} \mu(a|s) Q^{\mu}(s,a). \tag{2.8}$$

Similarly, we may express the action–value function in terms of the value function as

$$Q^{\mu}(s,a) = R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) V^{\mu}(s'). \tag{2.9}$$

We shall now derive a recursion for the optimal value function. Assuming Q is known we get that

$$V^{*}(s) = \max_{a \in A} Q^{*}(s,a). \tag{2.10}$$

Comparable to Equation (2.9) we get that the optimal action–value function may be written as

$$Q^{*}(s,a) = R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) V^{*}(s') \tag{2.11}$$

and by combining Equation (2.10) and Equation (2.11) we can write the optimal value function of a state s as a function of the optimal value function in the following states. This gives us another recursive equation

$$V^{*}(s) = \max_{a \in A} \left[R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) V^{*}(s') \right] \tag{2.12}$$

called the *Bellman optimality equation* (Ghavamzadeh et al. 2015; Sutton and Barto 2018).

2.1.5 Partially Observable Markov Decision Processes

When solving an MDP we generally assume that the whole state is observed and, thus, it is perfectly known what the current state is. This is, however, not the case in most real–world applications (Dimitrakakis and Ortner 2019). The observed variables can often be a measurement of the state or just parts of the full state. In those cases, we do not have perfect information. To generalise the MDP to such situations we introduce the *partially observable Markov decision process* (POMDP).

The underlying dynamics defined for the MDP remains for the POMDP. In addition, we introduce $o \in O$, which is a, possibly incomplete, *observation* of the state from the set of possible observations O. Further, we introduce $\Omega(\cdot|s,a) \in P(O)$ which is the probability distribution over possible observations given the current state s and the action a that was taken to reach that state. As a consequence, the agent needs to make decisions based on incomplete information given by the observation. We define a POMDP, as in

Ghavamzadeh et al. (2015), by these seven properties:

- S the set of states,
- A the set of actions,
- O the set of observations,
- $P(\cdot|s,a)$ $\mathsf{P}(S)$ the probability distribution over the next state if action a is taken in state s,
- $\Omega(\cdot|s,a)$ $\mathsf{P}(O)$ the probability distribution over possible observations given that action a was taken to get to state s,
- $R(s,a) : S \times A \to \mathsf{R}$ the bounded reward function for state–action pair (s,a),
- $P_0(\cdot)$ $\mathsf{P}(S)$ the probability distribution of the initial state.

The choices made by the agent in a POMDP are based upon a *belief* $b(\cdot)$ $\mathsf{P}(S)$, stored and updated by the agent, describing the probability of being in a particular state. By updating the beliefs, the agent can include information from past events in order to form more well–informed decisions. In each iteration, the belief is updated with the new information acquired from the observation by applying Bayes' rule as

$$b_{t+1}(s\ ) = \frac{\Omega(o_{t+1}|s\ , a_t)\sum_{s\ \ S} P(s\ |s,a_t)b_t(s)}{\sum_{s\quad S}\Omega(o_{t+1}|s\ \ , a_t)\sum_{s\ \ S} P(s\ \ |s,a_t)b_t(s)}. \tag{2.13}$$

This updated belief is used in the Bellman optimality equation which for POMDPs is defined as

$$V\ \ (b_t) = \max_{a\ \ A}\left[\sum_{s\ \ S} b_t(s)R(s,a) + \gamma\sum_{o\ \ O} P(o|b_t,a)V\ \ (b_{t+1}(b_t,a,o))\right]. \tag{2.14}$$

Here we use the notation $b_{t+1}(b_t,a,o)$ to denote the updated belief after performing action a and observing o with the old belief b_t. (Ross et al. 2008)

2.1.6 Bayes–Adaptive Markov Decision Processes

In *Bayes-adaptive Markov decision processes* (BAMDP) the key distinction compared to MDPs is that we consider a distribution over transition models, $P(P|h)$, instead of considering one specific model. In a sense, we have a distribution over MDPs. Given a *history*[1], h H, of states and actions we update the distribution over models in a Bayesian fashion to obtain the *posterior* $P(P|h)$. This allows us to include domain knowledge through a *prior*, $P_{prior}(P) = P(P|\ )$, hopefully resulting in quicker convergence towards the true transition probability. The fact that we have a distribution over transition models opens up the possibility of performing more robust decisions with respect to this distribution using risk measures (Sharma et al. 2019), which is not possible for the standard MDP. (Ghavamzadeh et al. 2015; Guez et al. 2013)

We introduce the *augmented state–space* $S^+ = S \times H$ and define the *augmented transition probabilities* as

$$P^+(\ s\ , h\ \ |\ s, h\ , a) = 1[h\ = has\ ]\int_P P(s\ |s,a)P(P|h)dP \tag{2.15}$$

and the *augmented rewards* as

$$R^+(\ s, h\ , a) = R(s,a). \tag{2.16}$$

[1] For instance $h_t = s_0a_0s_1a_1\ldots s_{t-1}a_{t-1}s_t$.

The BAMDP is defined by (Guez et al. 2013)

- S^+ the set of augmented states,
- A the set of actions,
- $P^+(\cdot \mid s, h\ , a)$ $\mathsf{P}(S^+)$ the augmented probability distribution over the next augmented state if action a is taken in augmented state s, h S^+,
- $R^+(\ s, h\ , a) = R(s, a) : S \times A \to \mathsf{R}$ the bounded reward function for the augmented state–action pair $(\ s, h\ , a)$,
- $P_0(\cdot)$ $\mathsf{P}(S)$ the probability distribution of the initial non-augmented state,
- $P_{prior}(P) = P(P \mid\ )$ the prior distribution over transition probabilities,
- $P(P|h)$ the posterior distribution over transition models P $\mathsf{P}(S)$ given the history h H.

A core concept of reinforcement learning that has not been covered so far is the *exploration–exploitation* dilemma. How an agent handles this problem will greatly affect its performance. The agent may *exploit* the current information it has about the environment. That is to perform what it believes to be the best action, given its current knowledge, in order to maximise the reward obtained. It may also choose to *explore* the environment. Doing so may lead to discoveries of better actions, previously unknown to the agent, resulting in higher long term reward. In the context of Bayesian reinforcement learning (see Section 2.3), which is what we have for BAMDPs, it is common to use so called *exploration–exploitation policies* that balances exploration and exploitation, and utilises the problem history (Guez et al. 2013). The best such policy is called a *Bayes–optimal policy* and is also the optimal policy of a BAMDP (Guez et al. 2013).

2.1.7 Semi–Markov Decision Processes

Thus far we have covered MDPs and some extensions to them. All of these processes have had the property that transition times between states are constant and equal to one time unit (and thus are discrete). Let us instead consider the case where the transition times are continuous and follow some distribution, which may or may not be different depending on the current state, action and the next state. These assumptions are included in the *semi–Markov decision process* (SMDP), which we in this report consider in the time–homogeneous setting and with finite state– and action–spaces.

Similarly to MDPs, the state transition probabilities $P(\cdot|s, a)$ $\mathsf{P}(S)$ in SMDPs are only dependent on the current state s S and the action a A. However, as mentioned, each decision has an associated *sojourn time* (also known as *waiting time* or *holding time*) which between decision epoch $m - 1$ and m is represented by the random variable Υ_m. As the sojourn time need not be exponentially distributed, that is memoryless distributed, the Markov property does not hold, and this makes the SMDP *semi–Markov*. Moreover, given that the new state is s S, Υ_m has the cumulative distribution function $F_{sas}(t)$, where we assume that $F_{sas}(0) < 1$. The difference in time–space between MDPs and SMDPs is visualised in Figure 2.1.

An SMDP is defined by

- S the set of states,
- A the set of actions,
- $P(\cdot|s, a)$ $\mathsf{P}(S)$ the probability distribution over the next state if action a is taken in state s,
- $F_{sas}(t)$ the cumulative distribution function for the sojourn time given the old state s, the action a and the new state s ,
- $R(s, a) : S \times A \to \mathsf{R}$ the bounded reward function for the state–action pair (s, a),
- $P_0(\cdot)$ $\mathsf{P}(S)$ the probability distribution of the initial state.

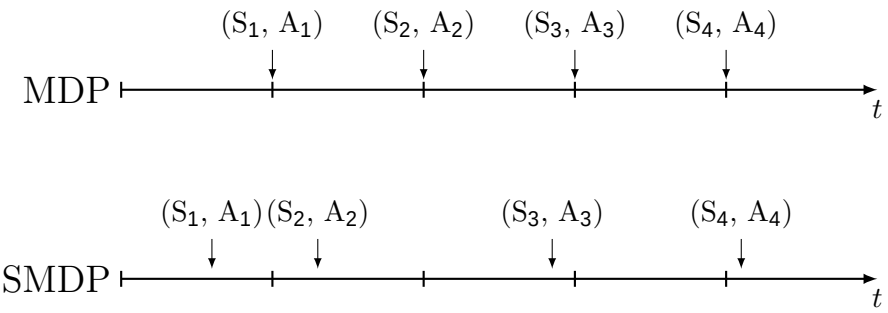


Figure 2.1: *Visualisation, in the time–space, of the di erence between when decision epochs occurs in MDPs and SMDPs.*

Given a state–action pair (s, a) the reward, $\mathsf{R}(s, a)$, may be decomposed into an ***instant reward***, $R(s, a)$, and a ***reward rate***, $r(s, a)$. Thus between epoch $m - 1$ and m the reward is given by

$$\mathsf{R}(s, a) = R(s, a) + r(s, a)\Upsilon_m. \tag{2.17}$$

As we are now dealing with continuous time the definitions of the discounted value function and the discounted action value function given in Equation (2.3) and Equation (2.5) need to be modified. Instead of multiplying a discount factor in each step we now integrate over the reward and the discount factor, yielding the discounted action value function (Baykal-Gürsoy 2010)

$$Q^{\mu}(s, a) = \mathsf{E}_{P,\mu}\left[\int_{t=0}^{\infty} \gamma^t \mathsf{R}(S_t, A_t)dt \Big| S_0 = s, A_0 = a\right]. \tag{2.18}$$

If we by T_m denote the time that the decision of epoch m is made, and we wish to compute the discounted reward observed from the transition between epoch m and $m + 1$, we get that[2]

$$\mathsf{R}_{discounted}(s_{T_m}, a_{T_m}) = \gamma^{T_m}\left(R(s_{T_m}, a_{T_m}) - \frac{r(s_{T_m}, a_{T_m})}{\ln \gamma}(1 - \gamma^{\upsilon_{m+1}})\right), \tag{2.19}$$

where υ_{m+1} is the observed transition time between the epochs.

2.2 Model–Free Solvers

Here we give a brief introduction to the key aspects of model–free reinforcement learning, followed by a description of a model–free method for MDPs and one for SMDPs. In model–free reinforcement learning the solver does not contain transition probabilities of our model. As no transition models are kept, these methods act only based on what they have previously learnt by acting in the environment and observing the outcomes. Keeping no model may be a benefit as it is a simpler approach. However, it may also be limiting as the solver can not utilise information about how the environment changes due to an action (Sutton and Barto 2018).

[2] Note that Baykal-Gürsoy (2010) has a misprint in the index of υ in the equation corresponding to Equation (2.19).

2.2.1 Q–Learning

Q–learning is a model–free method for MDPs introduced originally by Watkins (1989). It is a *temporal–di erence* (TD) based method, meaning that the estimates obtained from the algorithm are bootstrapped throughout a simulation based on the current estimates (Sutton and Barto 2018). As the name suggests, it revolves around estimating the optimal action–value function, Q , of an MDP. By iteratively simulating the MDP at hand, estimates of the optimal action–value function for each state–action pair is updated based on the reward obtained in a transition and the best current estimate in the new state.

In Q–learning, we start by initialising $Q^{\mu}(s,a)$ (s,a) $S \times A$, arbitrarily (though often to 0). Given the current state s and the current Q, we select an action a through some policy based on Q. Such a policy could be an *ϵ–greedy policy*, that is to often perform the action that maximises the current Q–value, but with probability ϵ select the action randomly. Assuming that the next state is s , the update rule for Q is given by

$$Q(s,a) \leftarrow Q(s,a) + \beta \left(R(s,a) + \gamma \max_{a} Q(s\ ,a\ ) - Q(s,a) \right), \tag{2.20}$$

where β $(0,1]$ is the *learning rate* (sometimes called *step size*). This procedure is repeated either until a terminal state is reached, or the problem reaches the end of the problem time horizon. Once a terminal state or the end of the time horizon is reached, we start over in the initial state without resetting Q, that is to start a new *training episode*. This is done over several training episodes to gain the final estimate of Q .

2.2.2 Semi–Markov Average Reward Technique

Semi–Markov average reward technique (SMART) is another model–free method, originally introduced by Das et al. (1999). It resembles Q–learning a lot but is applied to SMDPs instead of MDPs. As each reward of an SMDP has an associated sojourn time, Equation (2.20) has to be modified slightly. We start by keeping track of the total reward obtained and the total time elapsed, and then form the average reward, g, using these values. Given a state–action pair, (s,a), a new state s and the observed associated sojourn time, υ, we get the update formula

$$Q(s,a) \leftarrow Q(s,a) + \beta \left(R(s,a) - g\upsilon + \max_{a} Q(s\ ,a\ ) - Q(s,a) \right), \tag{2.21}$$

where the average reward, g, is updated after computing the new optimal action–value function estimate, $Q(s,a)$. By subtracting $g\upsilon$ from $R(s,a)$, this method captures the time dynamics of the SMDP.

2.3 Model–Based Solvers

In this section, we cover the characteristics of model–based solvers for MDPs by first giving some general background and then look into a particular method. In *model–based reinforcement learning* at large, the solver needs to know the state transition probabilities, $P(\cdot|s,a)$, of the model we wish to solve. Related to model–based reinforcement learning is *model–based Bayesian reinforcement learning*, where the model is not fully known, but rather is estimated from observations. Model–based Bayesian reinforcement learning can, in turn, be applied to BAMDPs that have an uncertainty over models as its characteristic property.

2.3.1 Bayes–Adaptive Monte–Carlo Planning

Bayes–adaptive Monte–Carlo planning (BAMCP) is a model–based Bayesian reinforcement learning algorithm introduced by Guez et al. (2013) that produces a Bayes–optimal policy. As it deals with BAMDPs (see Section 2.1.6), we maintain a distribution over transition models, $P(\boldsymbol{P}|h)$, which initially is defined by the prior, $P_{prior}(\boldsymbol{P}) = P(\boldsymbol{P}|\ )$. For each step an agent is to take in the real environment, a search for an appropriate action is preformed. We begin by sampling transition probabilities $\boldsymbol{P}$ from $P(\boldsymbol{P}|h)$, given the current history h, multiple times. Based on these samples, we perform a *Monte–Carlo tree search* to estimate the optimal action–value functions. More specifically, the Monte–Carlo tree search method used is *upper confidence bounds applied to trees* (UCT) for which we refer to Kocsis and Szepesvári (2006). Using the estimates of the optimal action–value functions, the action corresponding to the maximal action value is selected and performed in the real environment.

This algorithm serves as a basis for our own work, and aspects of it are thus covered more in the method, see Section 3.2. For a more in–depth presentation of BAMCP, we refer to Guez et al. (2013).

2.4 Robustness

In this section, we are going to explore methods to obtain robust solutions. This will be done by exploring the field of *risk measures* and investigate how they can be used in the framework of reinforcement learning.

2.4.1 Risk Measures

To asses and manage risks is crucial in many fields. In reality, this means to evaluate a situation according to some measures, estimate how likely it is that something bad happens and the actual cost of that event, and then act accordingly to avoid possible bad outcomes. A field where risk management is of high importance is the financial sector that has included risk measures in their models for a long time. In economics, the goal is quite easy to understand as one tries to maximise the yield while still having an acceptable risk level. A way of dealing with this more qualitatively is to put a price on risk by including it in predictive models, see for instance Wang (2000). These ideas from economics have later been applied in other fields where risk management is crucial, for example in robotics (Majumdar and Pavone 2017). In the risk assessment field, one talks about costs rather than rewards as it is more easily understood that a cost brings a risk. However, the terms are interchangeable as a cost can be interpreted as a negative reward.

When beginning to apply the framework of risk measures the essential question is how they should be formulated in order to have the desired effect. We define a risk measure ρ as a measure that maps a (cost) random variable to a real number quantifying the risk. This is a quite general definition that needs to be specified further to be of any use. There are two quite naive risk measures that we consider as the extreme boundaries of risk aversion. The lower boundary is to use the expected cost. This measure is neutral to risk and any measure resulting in lower values than this would seek risk rather than avoid it, making it a somewhat stupid measure of risk. In the other end of the spectrum is the worst–case assessment. This means that we look for the worst possible outcome and use it as a measure of the risk. Neither of these two measures is very helpful when one should make a decision and take the risk into account in a nuanced and rational way. Therefore,

we look for risk measures somewhere between these extremes. (Majumdar and Pavone 2017)

In Majumdar and Pavone (2017), six axioms are proposed which the authors suggest should be satisfied by any risk measures used in high–level decision making. Risk measures fulfilling these axioms are called *distortion risk measures*. Four of these axioms, namely axiom 1 to 4, are the foundation of what we call *coherent risk measures*. The coherent risk measures are a well known standard and the additional axioms provided in Majumdar and Pavone (2017) are their proposal to additional properties that they argue should be present in high–level decision making. We refer back to Majumdar and Pavone (2017) for the full definitions and a more exhaustive description of the risk measure axiom framework and settle with a brief description in this report. Here we assume that Z and Z are random cost variables and ω and ω are outcomes, which for instance in an MDP could be states or state–action pairs, such that $Z(\omega)$ is the random cost variable associated with the outcome ω. For a risk measure, ρ, the axioms then state that:

1. **Monotonicity:** If $Z \leq Z$, then $\rho(Z) \leq \rho(Z)$.
2. **Translation invariance:** If c R, then $\rho(Z + c) = \rho(Z) + c$.
3. **Positive homogeneity:** If $\beta \geq 0$, then $\rho(\beta Z) = \beta\rho(Z)$.
4. **Subadditivity:** $\rho(Z + Z) \leq \rho(Z) + \rho(Z)$.
5. **Comonotone additivity:** If Z and Z are comonotone, i.e. $(Z(\omega) - Z(\omega))(Z(\omega) - Z(\omega)) \geq 0 \quad \omega, \omega$, then $\rho(Z + Z) = \rho(Z) + \rho(Z)$.
6. **Law invariance:** If Z and Z are identically distributed, then $\rho(Z) = \rho(Z)$.

These axioms are argued to provide properties that intuitively are desirable. Further, in Majumdar and Pavone (2017), the authors conclude that the process of posing these axioms is not completed yet and that further discussion can lead to a standardised set of axioms for decision measures. Worth noting is that several popular risk measures do not meet the criteria of these axioms, resulting in unexpected and obvious errors when assessing risk (Majumdar and Pavone 2017). Because coherent risk measures, those that fulfil axioms 1 to 4, have properties that are considered to be necessary for any stable risk measure (Majumdar and Pavone 2017; Rockafellar and Uryasev 2002; Tamar et al. 2017), those are the criteria that we put on the risk measures we use in this report.

2.4.2 Value at Risk

One of the most common risk measures, at least from a historical perspective, is the *value at risk* (VaR). It has been used in many applications and has even been written into industry regulations (Rockafellar and Uryasev 2002). Despite being popular, VaR does not, in general, fulfil all axioms stated above as it does not satisfy subadditivity (Majumdar and Pavone 2017) and is thus neither a distortion or coherent risk measure. To not have the subadditivity property simply means that, for example, two independent investments together can result in a greater loss than the sum of the same two investments each on their own. However, we will present it here as the next risk measure that we present expands on it.

We define VaR at α–level as the α–quantile of the cost random variable Z, that is

$$\mathrm{VaR}_\alpha(Z) = \min_\zeta \{ z \,\big|\, \mathsf{P}[Z \leq \zeta] \geq \alpha \}. \tag{2.22}$$

Here, we wish to inform the reader that there exists two contrary definitions. We define VaR as the α–quantile, meaning that there is α probability for the cost to be lower than the risk measure. This is in line with for example Rockafellar and Uryasev (2002). The other definition uses α as the probability for the cost to excess the VaR measure, making

it the $(1 - \alpha)$–quantile, see for example Chapman et al. (2019) and Majumdar and Pavone (2017).

2.4.3 Conditional Value at Risk

A risk measure that does fulfil all six axioms listed in Section 2.4.1 is the *conditional value at risk* (CVaR), which is the extension to VaR. Its definition can simply be interpreted as the mean value of the probability distribution above the VaR_α–level (Majumdar and Pavone 2017). Another perspective is to say that it is the expected cost in the case of a bad outcome. Here, the definition of a bad outcome is that it should be larger than the VaR–level at a specified α–level. This view has given rise to other names of the risk measure used throughout the literature, for example, *expected shortfall* and *mean shortfall*. Note, however, that these names are not always defined interchangeably. Alternatively, as we are talking about expected values, CVaR can also be seen as an integral over the probability density in this region,

$$\text{CVaR}_\alpha(Z) = \frac{1}{1 - \alpha} \int_\alpha^1 \text{VaR}_\tau(Z)\, \mathrm{d}\tau\,. \tag{2.23}$$

CVaR is visualised, together with VaR, in Figure 2.2. One can see that VaR and CVaR gives a more nuanced measure than the previously mentioned more naive measures. Further, CVaR is a more conservative risk measure than VaR as it will, by construction, always be larger or equal to VaR.

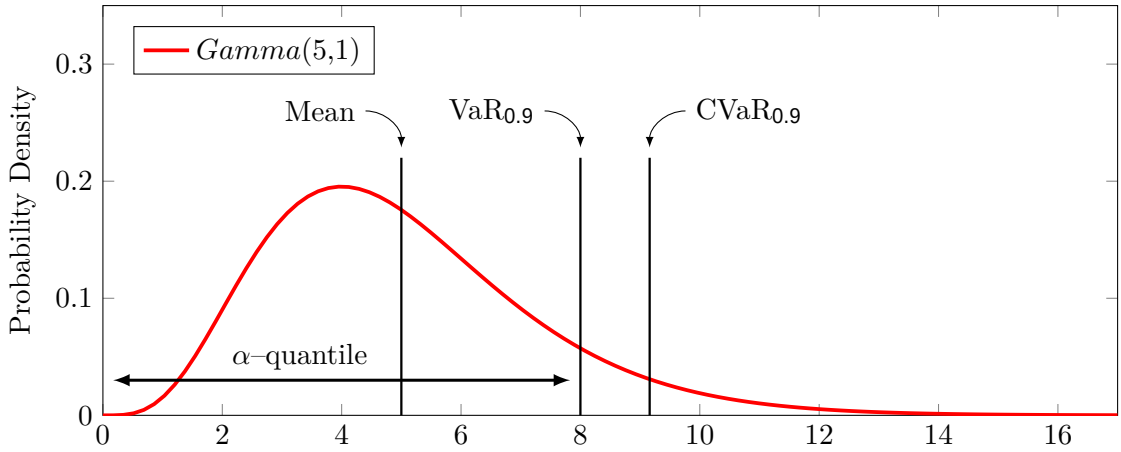


Figure 2.2: *In this figure, the VaR and CVaR risk measures are visualised. VaR is the $\alpha = 0.9$–quantile and CVaR is interpreted as the mean in the upper $(1 - \alpha) = 0.1$–quantile. The red curve is the density of a gamma distribution with shape $= 5$ and rate $= 1$.*

2.4.4 Mean–Semideviation

Another useful risk measure is the *mean–semideviation* (MSD). It is a risk measure that not only takes the tail of the distribution into account but also includes the mean value. As the name suggests, this risk metric uses both the mean, as well as the *semideviation*, of the cost. The semideviation is the standard deviation in one direction from the mean. The upper semideviation is given by the standard deviation, from the mean, of all observations above the mean. Likewise, the downside semideviation is defined as the standard deviation,

from the mean, for all observations below the mean. The definition of the upper mean–semideviation of the cost random variable, Z, becomes

$$\mathrm{MSD}_{\alpha}^{+}(Z) = \mathsf{E}[Z] + \alpha \mathrm{SD}^{+}(Z) = \mathsf{E}[Z] + \alpha \sqrt{\mathsf{E}\left[(Z - \mathsf{E}[Z])_{+}^{2}\right]}, \tag{2.24}$$

where SD^{+} is the upper semideviation and $\alpha \geq 0$ is a weight parameter that can be tuned (Shapiro et al. 2009). The MSD^{+}, and its relation to the SD^{+}, is visualised in Figure 2.3. The $+$ in the notation means that only the positive contributions should be kept. This is the same as $(Z - \mathsf{E}[Z])_{+}^{2} = (max\{0, Z - \mathsf{E}[Z]\})^{2}$. The upper mean–semideviation is useful when one seeks to minimise a cost, as well as the risk of an extremely costly outcome, jointly. Similarly, when dealing with rewards, R, that should be maximised, the lower mean–semideviation is used since one like to have as high mean as possible while keeping the lower tail of the distribution as short as possible. The risk measure then becomes

$$\mathrm{MSD}_{\alpha}^{-}(R) = \mathsf{E}[R] - \alpha \mathrm{SD}^{-}(R) = \mathsf{E}[R] - \alpha \sqrt{\mathsf{E}\left[(\mathsf{E}[R] - R)_{+}^{2}\right]}. \tag{2.25}$$

MSD is a coherent risk measure if, and only if, $\alpha \quad [0, 1]$ (Shapiro et al. 2009), which is essential to keep in mind when choosing parameter settings. The MSD measure is, together with CVaR, discussed and evaluated in the reinforcement learning framework in work by Tamar et al. (2017). In the study, the authors concluded that these coherent risk measures could create robustness to both stochastic dynamics and modelling errors.

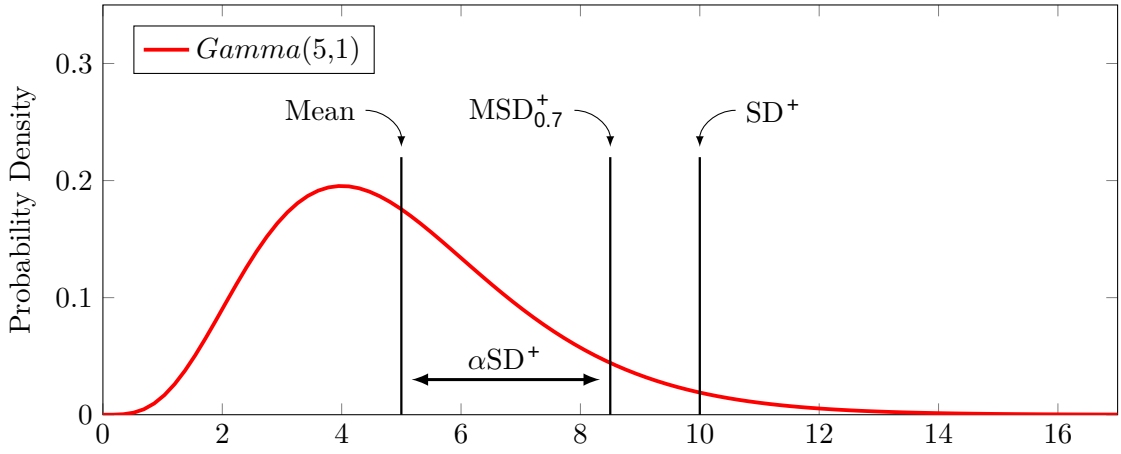


Figure 2.3: *In this figure, the MSD^{+} risk measure is visualised at $\alpha = 0.7$–level. The risk measure is the sum of the mean value and α times the upper semideviation. The red curve is the density of a gamma distribution with shape $= 5$ and rate $= 1$.*

2.4.5 Using Risk Measures for Robustness

To create robustness against both stochastic dynamics and errors in modelling, a risk measure can be integrated into a solver. One way to achieve model robustness, that is to hedge against an incorrect model, in a Bayes–adaptive approach, is to make sure that the model chosen to represent the problem have a lower risk of a significant error. This is the approach that, for example, Sharma et al. (2019) takes as they apply CVaR to the model uncertainty, followed by computing an optimal policy according to this risk measure. Another approach, used for example by Chow et al. (2018), is to use the risk measure as

a constraint when optimising. They search for a policy by optimising the value–function while applying CVaR as a constraint by always keeping it below a set threshold. A third approach is to optimise with respect to a risk measure that is applied to the value or action–value function itself. These three approaches have similarities in the way that they all utilise a risk measure to avoid risks. However, there are significant differences as the first approach minimises the risk induced by model errors, the second ensures that we do not choose a too risky policy while optimising the expected return, and the last approach minimises the risk for the policy according to the risk measure applied directly to the action–value function.

As mentioned, we differentiate between model and dynamic robustness. It is important to realise that if an agent is dynamically robust, that is to not make too risky actions according to its model, but lacks model robustness, that is to not consider model incorrectness, it will perform believed safe actions based on a model that might not correspond to the true environment. This means that the agent will not act in a risk–averse manner. It is thus essential to have both model and dynamic robustness jointly to ensure a generally robust solver. To obtain model robustness, one can choose the approach of Sharma et al. (2019) previously mentioned. Another approach, that might result in model robustness, is to use a coherent risk measure, such as the MSD risk measure, applied to the action–value function. Tamar et al. (2017) argues that with such risk measures it should be possible to obtain model robustness, as well as dynamic robustness, if it is shaped properly.

2.5 Vehicle Routing

The aim of this section is to introduce the *dynamic vehicle routing problem* (DVRP) by first looking at the precursory routing problems, before exploring more details. The *travelling salesperson problem*[3] (TSP) is a well–known optimisation problem where one seeks to find the shortest route that visits all nodes in a graph exactly once and then returns to the first node. This problem generalises to the *vehicle routing problem* (VRP, sometimes referred to as the *static vehicle routing problem*), where the goal is the same, but with multiple vehicles at disposal. These problems have historically been important optimisation problems, and different versions have emerged to represent more specific cases. Some examples are multiple depot nodes, time window constraints and limited vehicle capacities.

In both the TSP and the VRP, the complete system is known beforehand and will not change as time evolves during the route execution. However, more realistically, it is common that not all problem information, such as all customer demands during a day, is known at time of planning. Instead, it is revealed as time progresses. In this case we get the dynamic vehicle routing problem, which is presented in more detail below, followed by an introduction of the special case with only one vehicle called the *dynamic travelling repairperson problem*[4]. (Larsen 2000)

2.5.1 Dynamic Vehicle Routing

Here the DVRP is introduced and compared to the VRP by exploring the different information flows in the two cases. As stated above, all relevant information is known in the VRP when planning occurs, and this information does not change as time evolves and the

[3] Sometimes referred to as the travelling salesman problem.
[4] Sometimes referred to as the dynamic travelling repairman problem.

routes are executed. Relaxing these conditions opens up for the possibility to have incomplete information at the beginning of the routes as well as changing information during the problem horizon. This usually takes the form of tasks appearing at a random position as time evolves, forcing the vehicles to adapt and adjust the routes to find a solution with these new conditions. A solver to this dynamic problem formulation produces a policy that is used to choose actions, described in more detail in Section 2.1.3, rather than fixed routes that are strictly followed. (Larsen 2000)

This new formulation somewhat changes the challenges that occur. First, the computation times must be short as the policy often is calculated online, while the process is running. This is done to utilise new problem information obtained. Secondly, the objective function, that is what is optimised, may need to be adjusted. The total travel distance might not be as important in the DVRP as in the VRP as the problem might be open–ended, with an infinite time horizon. Moreover, the tasks could represent customers that can not accept too long waiting times. Further, other factors might be essential to keep in mind. For instance, the problem may have infinite time horizon, the current location of the vehicles are more important to know in detail, requests of tasks might be rejected due to unfavourable positioning, and so on. (Larsen 2000)

To summarise this far, the DVRP is a routing problem formulation where not all relevant information is known before the route execution starts. Information is instead added or changed as time evolves during the route execution. How much the problem changes during the process is called the problem's *degree of dynamism* (Adewumi and Adeleke 2018). Further, it is essential to note that the formulations that we present here are not real–world problems, but rather abstract representations thereof (Psaraftis et al. 2016). This is also reflected in the fact that, when studying the DVRP, one uses simulated data instead of real–world data. This is mainly because of two reasons. Firstly, it can be hard to capture all data required to represent the system in real–time. The other reason is that simulated data enables for more in–depth analysis as the underlying distributions are known (Larsen 2000).

Dynamic Vehicle Routing is a very active research field as seen by the large amount of published material on the subject, especially since the year 2000 and forward (Psaraftis et al. 2016). Hence, it can be hard to get a good grasp on the field as many factors can be altered to gain a new perspective. To present and categorise these versions of sub–problems are out of the scope of this thesis. However, the interested can read more about the theoretical foundations in Larsen (2000) and surveys of later research in Pillac et al. (2013), Psaraftis et al. (2016), and Ritzinger et al. (2016).

2.5.2 Dynamic Travelling Repairperson Problem

In this section, we are going to present a particular version of the DVRP, called the *dynamic travelling repairperson problem* (DTRP). This model is the foundation for the model later used in this thesis. The DTRP was first introduced by Bertsimas and van Ryzin (1991) and have since been one of the more common DVRP formulations in the literature. The DTRP is a special case of the DVRP, in which the aim is to minimise the waiting time of the tasks appearing in the system. Specifically in the DTRP we deal with one vehicle, though a generalisation to multiple vehicles exists (Bertsimas and van Ryzin 1993).

More specifically, the DTRP takes place on a convex, often quadratic, subset of the euclidean plane where tasks appear uniformly distributed in space and follow a Poisson process in time. The time to complete a task does in turn follow a known distribution with known first and second moment. Then, the objective in the DTRP is to minimise

the total waiting time of uncompleted tasks in the system. This minimisation should ideally be done with a solver that reacts on the information in real–time, and thus does not (solely) depend on precomputed routes. The vehicle moves with constant speed in this system, and thus, the travel time between any two tasks is deterministic. There is also a variant to this formulation where a graph replaces the euclidean plane. On this graph, tasks can appear and form a queue on the nodes. In this setting, it becomes more apparent that a queuing phenomenon occurs as the Poisson process places the tasks in queues at the nodes. This network formulation is also the approach that we use in this thesis. (Bertsimas and van Ryzin 1991)

A drawback of this formulation, where one minimises the total waiting time, is that all queued tasks have the same priority regardless of how long time they have spent in the system. To not prioritise older tasks might not be a problem in some applications, but it would be unwanted behaviour in many real–world scenarios. Though studied extensively, Larsen (2000) states that the DTRP is not used in many real–world applications. He argues that this could be due to the simple structure and general behaviour of the model, together with the heavy computation required for a solution, as the problem is NP–hard. Despite this, the problem formulation is still interesting as a foundation that can be extended upon, as we do later in this thesis.

3 Method

We like to investigate how existing solvers for MDPs, or more specifically BAMDPs, can be modified and adapted to generalise into the SMDP domain. This is done by defining a DVRP model, more specifically, a variant of the DTRP, as an SMDP that is used to test different solvers. In turn, the solvers are further developed to work with the new prerequisites that arise when dealing with continuous time and stochasticity in transition times. In this chapter, we first present the model that we have defined and motivate the design choices. Then, our solver, *make it happen* (MIH), and a robust modification of it, *robust MIH*, is presented. Finally, we introduce three rule–based baseline policies that are used as a reference in the evaluation of our solvers.

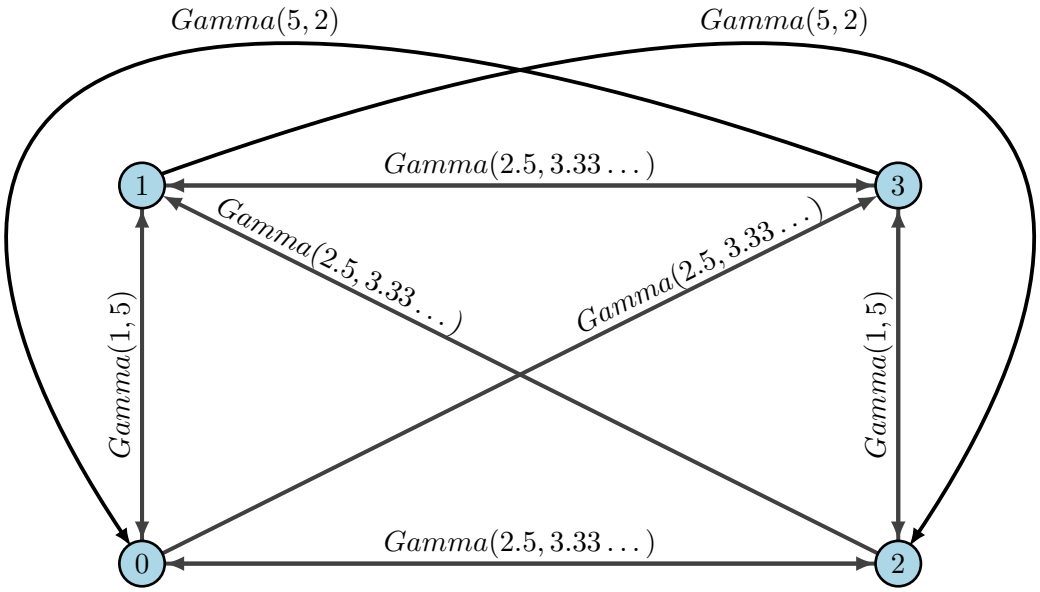
3.1 Model

The aim of this section is to construct a general model that fits within the SMDP framework, as well as having properties of the vehicle routing problems covered in Section 2.5. We do not try to capture the properties of a specific real–world problem. Instead, the model is constructed to be an interesting test case that can give insight about the behaviour and characteristics of SMDP solvers.

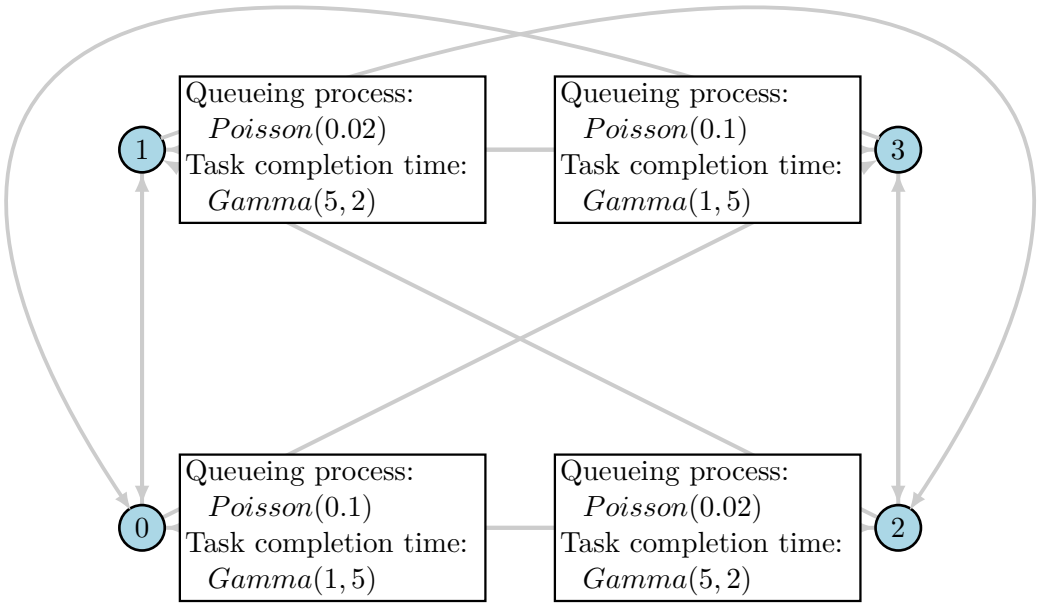
3.1.1 Model Definition

Our model is defined by a fully connected graph with four nodes, where each node has a queue of tasks. These queues have max length one, and new tasks arrive according to independent Poisson processes. The parameters of these Poisson processes are tuned to make two nodes, node 0 and 3, have tasks appear more frequently. The other two, node 1 and 2, have a lower task frequency. The rates used are presented in Table 3.1, where the parameter in a Poisson process represents the number of tasks per unit time. To maintain a time homogeneous SMDP, the Poisson parameters are assumed to be known by the agent acting in this model.

Tasks are completed by an agent when it stays in the same node between two decision epochs (two consecutive decision times). From the moment an agent performs an action to serve a queue, the queue is considered empty. This means that a new task may arrive to the queue while the agent is still serving the previous task. The sojourn time, the time between two epochs, are, together with the queueing processes and the stochastic outcomes (described later), what induces dynamism to the model and makes it interesting to study. Three different classes of sojourn times are used, which we call idle time, task completion time and travel time. Idle time is the transition time between two epochs when staying in a state with an empty queue. The idle time is kept constant and is short relative to other times in the model. The agent can, for example, use this option to let time pass. The idle time is 0.1 in our model. The second class is what we call task completion time. This is the time required to complete a task. The completion times are random variables following gamma distributions with parameters as given in Table 3.2. Finally, the third class, the travel times, are also random variables following gamma distributions that determines the sojourn time when transitioning between different states. The parameters used for the



(a) *The distributions that the travel time random variables for the di erent transitions follow.*



(b) *The distributions in each node that defines the queue–process and the task completion times.*

Figure 3.1: *A visualisation of the graph representation of the model with the distributions defining the di erent processes. The gamma distribution is defined by the first parameter as shape and the other as rate.*

travel times in our model is given in Table 3.3. This network is shown in Figure 3.1, where the travel time distribution is presented next to the arrow representing a path.

Table 3.1: *Parameters for the rate at which tasks arrive to the queues. The rate is expressed in average number of new tasks per time unit and defines the Poisson processes managing the queues.*

State	Rate (tasks/time unit)	Distribution
0 and 3	0.1	Poisson(0.1)
1 and 2	0.02	Poisson(0.02)

Table 3.2: *Parameters for the task completion time distributions in the model. The gamma distribution is defined by the first parameter as shape and the other as rate.*

State	Mean	Variance	Distribution
0 and 3	1	0.2	Gamma$(5, 5)$
1 and 2	5	0.5	Gamma$(50, 10)$

For the model to be as general as possible, the outcome when taking an action is stochastic. This means that the current state of the model and the action taken influences a probability distribution representing different outcomes. It is always more likely that the agent ends up where the action is meant to take it. However, this probability differs between different nodes and actions, thus making some nodes riskier to operate in than others. The probabilities used in our model is presented in Table 3.4. Worth noting, the agent is service–minded and thus has no option to stay idle in a state with tasks queued. The only two possible outcomes when deciding what action to take in a state with a non-empty queue is that the task is being worked on or the agent leaves that node.

Table 3.3: *Parameters for the travel time distributions in the model. The gamma distribution is defined by the first parameter as shape and the other as rate.*

Transitions	Mean	Variance	Distribution
$0 \rightleftarrows 1$ $2 \rightleftarrows 3$	1	0.2	Gamma$(5, 5)$
$0 \rightleftarrows 2$ $1 \rightleftarrows 3$	2.5	0.3	Gamma$(20.8, 8.33\ldots)$
$0 \rightarrow 3$ $2 \rightarrow 1$	2.5	0.3	Gamma$(20.8, 8.33\ldots)$
$0 \leftarrow 3$ $2 \leftarrow 1$	5	0.5	Gamma$(50, 10)$

Table 3.4: *Here, the probability to end up in the node that represents the action you took is presented. The other nodes share the remaining probability (up to 1) equally.*

	New Node Probability			
Current Node	**0**	**1**	**2**	**3**
0	0.94	0.85	0.85	0.85
1	0.85	0.94	0.7	0.7
2	0.85	0.85	0.94	0.85
3	0.7	0.7	0.85	0.94

When making decisions, the agent has complete information of the model state. That means that it is aware of both its position and at what nodes tasks are queued in. Moreover, it has four possible actions every time a decision should be made. One is to stay in the current node, and the other three are to aim for any of the other three nodes. Note, however, that the outcome might not correspond to the action taken because of the stochastic outcomes discussed earlier. The agent aims to minimise the total time that tasks exists in the system. That means that each task is generating a negative reward proportional to its time spent in the model, from the time a Poisson process creates it until the agent starts to process the task. The reward is presented to the agent in each epoch as the total time that tasks have spent in queues since the last epoch, times minus one. This means that the reward is bounded between -4 and 0 per unit time, with the upper bound actually being lower as it is not possible to have empty queues at all times.

3.1.2 Model Motivation

We will now try to motivate the design choices made when deciding on the parameters for the model. The number of nodes is chosen to be small because of how the complexity scales. The total number of states in the model is given by

$$\#\text{states} = \#\text{nodes} \cdot (1 + \text{queue length})^{\#\text{nodes}}. \tag{3.1}$$

The parameter settings we use with four nodes and queue length one thereby result in 64 different states in total. A larger graph than this is computationally hard to handle as the search space for a solver increases exponentially with regards to the number of states and actions. As computational performance of the implementations and ability to scale up the model size have not been the main priority of this project, but rather exploratory development, the model has been kept small.

The graph is constructed in a way that makes it semi–symmetric, meaning that the left and right side of the graph, visualised in Figure 3.1, share the same sojourn times and queue properties, but combined in different ways. To begin with, the transitions times have a structure where transitions between node 0 and 1, as well as transitions between node 2 and 3, are quick. Then there are medium timed transitions between node 0 and 2, node 1 and 3, from node 0 to 3, and from node 1 to 2. The travel times from node 1 to 2 and node 3 to 0 are slow transitions that take a long time. The exact parameters for these transition times are found in Table 3.3. Regarding the uncertainty of outcomes, the nodes 1 and 3 have higher uncertainty, meaning that actions taken towards node 2 and 3

from node 1, and towards node 0 and 1 from node 3, are more likely to result in a different outcome. Node 0 and 2 are less uncertain as the probability of an unexpected outcome is lower. Finally, the rate at which tasks arrive and the time to complete tasks are paired so that node 0 and 3 share the same behaviour of shorter tasks with high arrival rates and node 1 and 2 have longer tasks with lower arrival rates.

The idea with this model is that it should be possible to have cases where the direct path is not necessarily the best because of travel time and uncertainty of the outcome. This feature can favour agents that are risk–averse if they choose safer, less naive, actions. Moreover, we choose to have "faster" nodes, with shorter tasks and high arrival rates, and "slower" nodes, with longer tasks and lower arrival rates, to determine if the behaviour of the agent favours one type of tasks above others. This setting is likely a non–trivial environment where we believe that a human will not necessarily perform well.

3.1.3 Relation to DTRP

We have used many features of the dynamic travelling repairperson problem described in Bertsimas and van Ryzin (1991), and explored in Section 2.5.2. The main characteristics are that one agent uses a policy to make decisions to travel to, and complete, tasks that emerge randomly according to Poisson processes. Also, the time to complete a task follows a distribution, and the objective is to minimise the time that tasks spend in the system. All these properties are from the DTRP formulation that we have chosen to use in a network setting.

The extensions that we have chosen to add to this framework are primarily the stochastic travel times and the uncertain outcomes. The latter is a natural part in using an MDP formulation that allows for uncertainty in decision making which generalises the model to have more real–word properties. The stochastic travel times can be motivated with the same argument and fits nicely within the SMDP–framework that is necessary to use because of the continuous–time characteristics of the DTRP. To summarise, our model combines the problem formulation from DTRP with the SMDP decision making framework. In this setting, we have defined a graph with specific parameters in which questions about the behaviour of different solvers can be investigated.

3.2 Solvers

In this section, we cover the solvers we have developed, which build upon BAMCP (see Section 2.3.1). The first solver covered is *make it happen*, MIH, which is an extension of BAMCP from the MDP domain to the SMDP domain. Following this, we describe a modification of MIH, called robust MIH, which is meant to introduce robustness to the decision making process.

3.2.1 MIH

MIH is an extension of the BAMDP solver BAMCP, introduced by Guez et al. (2013) (see Section 2.3.1), from the MDP domain to the SMDP domain. The algorithm is presented as pseudocode in Algorithm 1 and in text below, where the pseudocode presented in Algorithm 1 is based the BAMCP pseudocode from Guez et al. (2013). All hyperparameters of the algorithm are listed in Table 3.5. Briefly, the algorithm assumes that the agent has full information on the current state of the environment it acts in. Then, in each decision epoch, a Monte–Carlo tree search is performed and to obtain estimates of the optimal

action–value functions for the current state. Based on this, a greedy choice of action is performed.

Table 3.5: *Table of hyperparameters in MIH.*

Variable	Description
c	UCT constant
γ	Reward discount
β	Learning rate of SMART
ϵ	Probability of random action in the rollout
n	Number of simulations per epoch
t_{max}	Time each sampled SMDP should be simulated

As the algorithm is a Bayesian reinforcement learning algorithm, where we maintain distributions over transition models, it is initialised by setting a prior distribution over the state transition probabilities, as well as a prior over the sojourn time distributions, as we are now in the SMDP domain that includes stochastic sojourn times. For every action the agent makes in the real environment these priors are updated in order to obtain new posterior distributions. Instead of using a global Q–learning scheme (see Section 2.2.1), for guiding action selections when a rollout is performed on new nodes, as is done in BAMCP, we employ SMART (see Section 2.2.2), as we are dealing with an SMDP. Our SMART–estimates of the optimal action–value functions, are also updated based on the actions performed in the real environment and are initialised to 0 for each state–action pair. This global SMART scheme is used for an ϵ–greedy rollout policy (explained below) in the tree search and should not be confused with the local action–value estimation, denoted by Q in the algorithm. The local action–value estimation scheme, which is averaged in each node, is based on the results of the tree search and resets with each decision epoch. Last, we initialise the maximum time a tree search simulation is performed and a parameter for UCT action selection that affects the trade off between exploration and exploitation in the tree search.

For each decision epoch in the real environment, we sample our posterior over state transition probabilities and our posterior over sojourn time distributions multiple times. We also initialise a local action–value estimation scheme, Q, and node visitation counts, N, to 0. Each pair of sampled transition probabilities and sojourn time distributions constitute an SMDP together with a known reward structure. Over the sampled SMDPs, a Monte–Carlo tree search is performed. The result of the tree search is estimates of the optimal action–value functions, obtained through the local action–value estimation scheme, and the agent selects the action to perform based on which action maximises these optimal action–value functions. This covers the Search() function of the algorithm.

The tree search is done through recursive Sim() calls, that traverses the state and action nodes constituting the tree. If a state node is visited for the first time, given the current epoch, a call to Sim_new_node() is made. In this function a ϵ–greedy Rollout() is performed. In such rollout, the next action on the tree is selected with probability ϵ according to a uniform distribution over all possible actions, and is otherwise selected according to which action that maximises the action value functions of the global SMART scheme. Then, the next state and sojourn time are sampled from the sampled SMDP we

are doing the tree search on. This is repeated until the accumulated sojourn times exceeds the maximum time of the simulation. The discounted rewards (see Equation (2.19)) are then propagated backwards through the tree along with counts of which nodes that have been visited, where they are used to update the local action–value estimation scheme.

In Sim(), if a state node has already been visited, actions are instead selected based on UCT (Kocsis and Szepesvári 2006). In UCT, the action, a , is selected based on the current local action–value estimate of a state–action pair, $Q(\ s, h\ , a\ )$, as well as on a term depending on the number of times child action nodes have been visited in relation to the state node. We have that the action, a , is selected to maximise

$$Q(\ s, h\ , a\ ) + c\sqrt{\frac{\ln(N(\ s, h\ ))}{N(\ s, h\ , a\ )}}, \tag{3.2}$$

where c is the UCT constant and N denotes node visitation counts. The UCT method is meant to balance exploration and exploitation. Then, based on the selected action, again, the next state and sojourn time are sampled from the sampled SMDP we are currently simulating, and the discounted rewards are propagated backwards through the tree along with node counts, in order to update the local action–value estimation scheme.

Once the tree search has been performed over all sampled SMDPs, and the next action has been selected, the agent proceeds to perform this action in the real environment. This results in a transition to a new state in the real environment, which is used to update the global SMART scheme and to obtain new posterior distributions for the state transitions and time transitions. Then Search() is called again with the new state as a root node, and the process is repeated until the agent does not query a new action.

3.2.2 Robust MIH

Our modification of MIH, made to obtain robust decision making, is quite small. It is based on the risk measure mean–semideviation (see Section 2.4.4), with which we replace the local action–value estimation scheme of MIH. Hence, instead of selecting the next action in the real environment based on action–value estimate, it is selected based on the maximal mean–semideviation estimate in order to hedge against actions with high risk. That is to hedge against actions with large lower semideviation, or semivariance. In addition to computing the action–value estimate on line 44 and line 55 of Algorithm 1, we compute estimates of the semivariance with rolling updates according to

$$SV(\ s, h\ , a) \leftarrow SV(\ s, h\ , a) \cdot (N(\ s, h\ , a) - 1) + \frac{\max(0, Q(\ s, h\ , a) - r_{tot})^2}{N(\ s, h\ , a)}. \tag{3.3}$$

Then, instead of using the mean reward estimates, $Q(\ s, h\ , a)$, on line 12 and line 37, we use the mean–semideviation,

$$Q(\ s, h\ , a) - \alpha\sqrt{SV(\ s, h\ , a)}, \tag{3.4}$$

where $\alpha \geq 0$ is the parameter of the mean semideviation risk measure.

When using Equation (3.3) the contribution of the i:th observation to the total semivariance depends on the mean estimate based on the i first observations. However, in the limit, this does not pose a problem as when the mean converges, so does the semivariance estimate, as the contribution of the errors of the first terms diminishes.

Algorithm 1: *Make It Happen (MIH)*

Input: The augmented state of current epoch, s_c, h_c , in the real environment, which is passed to `Search()`.

Output: An action a to be performed by the agent in the real environment. The result of this action is used for updating the global SMART scheme which is the basis of $rollout\_policy(\cdot)$.

Variables: P - state transition probabilities, T - sojourn time distributions, c - UCT constant, γ - reward discount, β - learning rate of SMART, ϵ - probability of random action in `Rollout()`, n - number of simulations per epoch, s, h - augmented state, a and a - action, s and s - state, h - history, has - history when action a is taken from history h to state s, t - time, t_{max} - time each sampled SMDP should be simulated, Δt - sojourn time, Δr - discounted reward from a transition (see Equation (2.19)), r_{tot} - total backpropagated discounted reward, $N(\ s, h\ )$ - counts of times node s, h visited in the current epoch, $N(\ s, h\ , a)$ - counts of times node $(\ s, h\ , a)$ visited in the current epoch, $Q(\ s, h\ , a)$ - optimal action value functions estimates from the local action–value estimation scheme in the node $(\ s, h\ , a)$ in the current epoch.

```
1  Function Search( s_c, h_c ):
2      N( s,h ) ← 0    s,h
3      N( s,h ,a) ← 0   ( s,h ,a)
4      Q( s,h ,a) ← 0   ( s,h ,a)
5      i = 0
6      while i < n do
7          i ← i + 1
8          P    P(P|h_c)
9          T    P(T|h_c)
10         Sim( s_c,h_c , 0, P, T )
11     end
12     a ← arg max_a Q( s_c,h_c ,a )
13     return a
14 end
15 Function Rollout(s, t, P, T ):
16     if t > t_max then
17         return 0
18     else
19         r_tot ← 0
20         while t < t_max do
21             a    rollout_policy(s)
22             s    P(·|s,a)
23             Δt, Δr    T(·|s,a,s )
24             s ← s
25             r_tot ← r_tot + Δr
26             t ← t + Δt
27         end
28         return r_tot
29     end
30 end
31 Function Sim( s,h , t, P, T ):
32     if t > t_max then
33         return 0
34     else if N( s,h ) == 0 then
35         return Sim_new_node( s,h , t, P, T )
36     else
37         a ← arg max_a Q( s,h ,a )+c sqrt( ln(N( s,h )) / N( s,h ,a ) )
38         s    P(·|s,a)
39         Δt, Δr    T(·|s,a,s )
40         t ← t + Δt
41         r_tot ← Δr+Sim( s ,has , t, P, T )
42         N( s,h ) ← N( s,h ) + 1
43         N( s,h ,a) ← N( s,h ,a) + 1
44         Q( s,h ,a) ← Q( s,h ,a) + (r_tot − Q( s,h ,a)) / N( s,h ,a)
45         return r_tot
46     end
47 end
48 Function Sim_new_node( s,h , t, P, T ):
49     a    rollout_policy(s)
50     s    P(·|s,a)
51     Δt, Δr    T(·|s,a,s )
52     r_tot ← Δr+Rollout(s , t + Δt, P, T )
53     N( s,h ) ← N( s,h ) + 1
54     N( s,h ,a) ← N( s,h ,a) + 1
55     Q( s,h ,a) ← Q( s,h ,a) + (r_tot − Q( s,h ,a)) / N( s,h ,a)
56     return r_tot
57 end
```

3.2.3 Solver Configurations

Here, we cover the different configurations and hyperparameter combinations of the solvers that we have used in the simulations. Note that the algorithms do not require specific priors and likelihoods to be used per se.

Priors

Starting with the distribution over state transition models, we note that we may decompose a state into the position of the agent in our network and the status of the queues, that is whether the queues are empty or not. As the transitions of the latter are determined by the Poisson parameters, that are assumed to be known by the agent in our model, the distribution over state transition models reduces to a distribution over position transitions. Given a position and an action, the likelihood of the next position is multinomial. For our prior, we use a Dirichlet distribution, where we set all concentration parameters to 1 initially, to have an uninformative prior. The benefit of using the Dirichlet distribution together with the multinomial likelihood is that the Dirichlet is a conjugate prior in this setting, which is beneficial from a computational point of view.

For the sojourn time models, we select a gamma likelihood with known, and true, shape parameter, and for the rate parameter, we use a gamma prior. These choices allow us to utilise that the gamma distribution is a conjugate prior with a gamma likelihood which, again, is beneficial from a computational point of view. We evaluate three different initialisations of the gamma prior. As a base case we use the true mean with unit variance. The idea of using this prior is that it allows us to verify if the algorithm can perform well given a good understanding of the problem at hand. In addition we test the case of initialising the mean to half of the true mean while maintaining unit variance. The reasoning behind this is to evaluate how a more pessimistic prior behaves. The prior is pessimistic in the sense that a lower rate in the likelihood increases the mean and variance. Lastly, we evaluate the impact of maintaining the true mean in the prior but increasing the variance from 1 to 5 to see if this greater uncertainty has any impact on the results. The two latter priors are only evaluated using the specific hyperparameter setup described in Table 3.7, with a total of 10 simulation samples performed for each prior and hyperparameter combination.

Hyperparameters

As the solvers have multiple hyperparameters we perform an initial parameter sweep to gain a better understanding of how the parameters impact the outcome. The hyperparameter values are described in Table 3.6 and for the sweep the uninformative state prior and the sojourn time prior with true mean and unit variance, as described above, are used. For each combination of hyperparameters, 5 simulation samples are performed[1] in order to limit the computation times.

Based on the results of the initial hyperparameter sweep, the hyperparameter configuration presented in Table 3.7 is used as basis for further simulations. We refer to these as our *standard parameters*. Moreover, due to the outcome of the initial parameter sweep, a more thorough sweep of the UCT constant, c, is performed. Using the standard parameters in Table 3.7, we vary $c \quad \{0, 0.5, 2, 5, 7.5, 12.5, 15, 17.5, 20, 50\}$, with 5 simulation samples and the same priors as previously used. Also, using the standard parameters in

[1] Originally, we intended to increase the simulation sample count. This was not be performed due to computational limitations in the project.

Table 3.6: *Hyperparameter values that are used in the initial sweep. Five simulation samples are performed for each combination of parameters within the two* ***Value*** *columns.*

Variable	Value	Value	Description
c	$1, 3, 10$	$1, 3, 10$	UCT constant
γ	0.95	$0.8, 0.95$	Reward discount
β	$0.2, 0.9$	$0.2, 0.9$	Learning rate of SMART
ϵ	$0.1, 0.5, 0.9$	$0.1, 0.5, 0.9$	Probability of random action in the rollout
n	$100, 250, 500, 1000$	500	Number of simulations per epoch
t_{max}	114	$18, 24, 87, 114$	Time each sampled SMDP should be simulated
α	$0.5, 1$	$0.5, 1$	MSD parameter (only used in robust MIH)

Table 3.7, but with $t_{max} \approx 1000, \gamma = 0.99, n = 2000$, 10 simulation samples, and the same priors as before, we investigate the impact of a deeper tree search.

Pretraining

As it is an imaginable scenario that a business has historical data from a deterministic policy, we wish to evaluate how the solvers perform if given access to data from the baseline policies introduced in Section 3.3. The hyperparameter setup used is the one described in Table 3.7, but with $\epsilon \in \{0.01, 0.1, 0.5\}$, using the uninformative state prior and the sojourn time prior with true mean and unit variance, as described above. We evaluate scenarios where data is available for $\{10^3, 10^4, 10^5, 10^6\}$ time units of following the policies, and the policies evaluated were FIFO, extended two–node focus and a combination of the two where FIFO is used for the first half of the time, and two–node focus was used for the remaining time. FIFO is selected as it should perform all transitions available and extended two–node focus as it performs the best of the baseline policies. Based on the historical data, we update the priors and SMART–estimates of our solvers. Following the updates, we let the solvers start with an empty system. For each scenario 10 simulation samples are performed.

3.3 Baseline Policies

In this section, we introduce three baseline policies. These are used to quantify how well our solver performs in the model as we do not have any other solvers with known performance to use as a comparison.

Table 3.7: *Hyperparameter values selected from the initial sweep. The two $alpha$–values denotes that all other parameters were applied to MIH, and robust MIH using both of the α–values.*

Variable	Value	Description
c	10	UCT constant
γ	0.95	Reward discount
β	0.2	Learning rate of SMART
ϵ	0.1	Probability of random action in the rollout
n	500	Number of simulations per epoch
t_{max}	87	Time each sampled SMDP should be simulated
α	$0.5, 1$	MSD parameter (only used in robust MIH)

3.3.1 First In, First Out

First in, first out (FIFO) is a commonly known and quite often used policy, that performs well in low–density systems (Bertsimas and van Ryzin 1991). The principle is as simple as that the oldest remaining task in the network is prioritised. In our implementation of FIFO, the agent does not observe or use any feedback from the system but rather takes the action related to the oldest task in the queues until the task has been completed, and then continues with the second oldest task.

This kind of policy performs quite well as long as the workload is not too high within the system. However, with a heavier workload, this policy is not ideal as it blindly takes the path that leads to the next queued task. This results in traversing edges in the graph that are unfavourable, as well as not prioritising tasks that are close, potentially even ignoring tasks present in the current node.

3.3.2 Two–Node Focus

The *two–node focus* policy is a naive policy that only cares about half of the network, namely node 0 and 1. If there is a task in the current node, it tries to complete it by performing the appropriate action. If not, it checks if there is a task in the other node, and performs an action to move there if so is the case. In case there is no task either in node 0 or 1, the agent performs actions to return to node 0, as that is where tasks appear most frequently.

This naive approach to the problem should perform quite well if the system has a heavy workload, as these two nodes are enough to keep the agent working most of the time. But because two nodes are almost never visited, this policy will, with high probability, never perform better than -2 reward per unit time once tasks have arrived in node 2 and 3. Node 2 and 3 can only be visited by accident if the uncertain outcomes turn out to put them there, and only complete tasks there if there are two unlikely outcomes in a row.

3.3.3 Extended Two–Node Focus

The *extended two–node focus* policy builds on the same principle as the two–node focus policy and does only differ in one way. If there are no tasks in node 0 or 1, it changes from

focusing on these two nodes to focus on node 2 and 3 instead. That is done by attempting to travel to node 2 from 0 or to node 3 from node 1 if the queues in both node 0 and 1 are empty. Likewise, when node 2 and 3 are empty, the agent attempts to travel back to 0 and 1 and focus on these two nodes again.

This approach could be more favourable in a model with a medium and high workload. This is because when many tasks are present, it behaves like the two–node policy. If the number of tasks is lower, this extended policy performs better than the two–node policy as it can reach and perform tasks in all parts of the graph. In a graph with a low workload, this policy could perform decently as well. It will not be optimal as it does not consider the different travel times and uncertainty between the nodes. However, an agent following this policy will, with high probability, reach any task in the system within two decision epochs if incitement exists.

4 Results

In this chapter the results obtained from our simulations are presented. The different solver configurations are presented for both MIH and robust MIH at the same time. We start off by presenting the results from the different hyperparameters evaluated, followed by a section on the impact of priors. Lastly, we present the results of pretraining the solvers.

4.1 Hyperparameters

Here we cover the results of varying the hyperparameters of the solver. The results presented are based on the standard parameters, described in Table 3.7, where one parameter at a time is varied according to Table 3.6. In Figure 4.2 the cumulative reward per time unit is presented for the non–robust case. The results of the robust scenario did not differ notably. In addition, in Table 4.1 we present the fraction of actions selected for our standard parameters, described in Table 3.7. These fractions did not notably differ in the robust case. Lastly, the results of the extra UCT parameter sweep and the run with search tree depth $t_{max} \approx 1000$ and 2000 simulations per epoch did not provide any different results than already presented above.

4.2 Baseline Policies

In Figure 4.1 the performance of the baseline policies, in terms of cumulative reward per time unit, is compared with the standard parameter setup. We note that our solver, MIH, performs better than FIFO, and that both two–node focus and extended two–node focus, in turn, outperforms MIH.

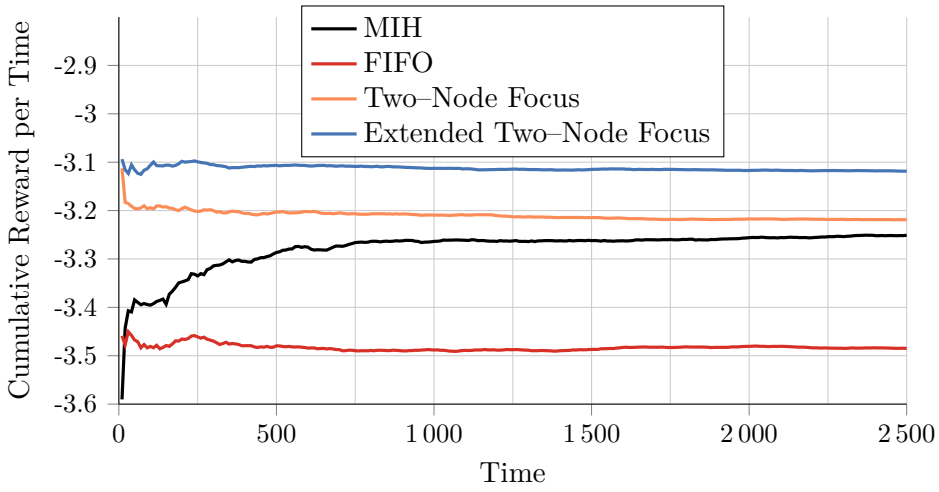
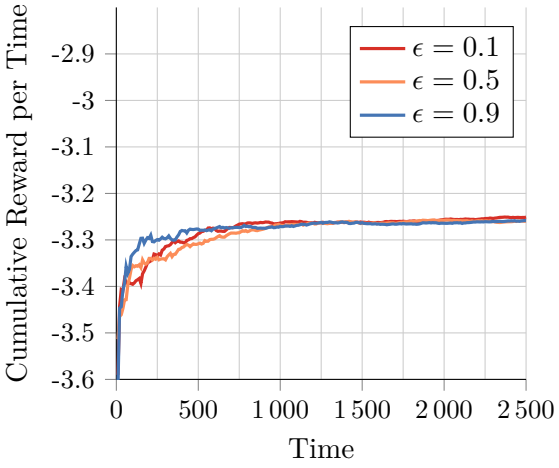
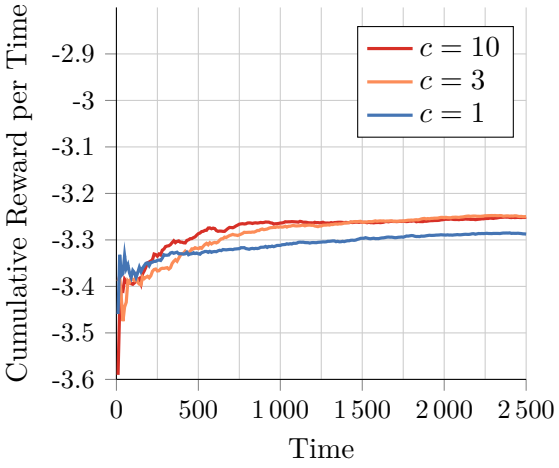


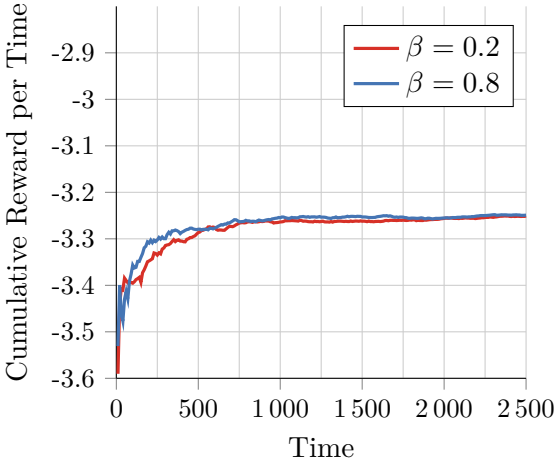
Figure 4.1: *In this figure, the performance of the MIH solver is compared to the performance of our baseline policies described in Section 3.3. The baseline policies are averaged over 20 samples, and the MIH solver is averaged over 5 samples.*



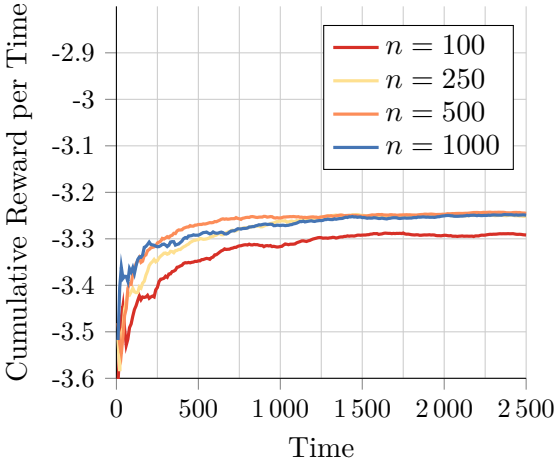
(a) *Rollout epsilon, the proportion of random moves.*



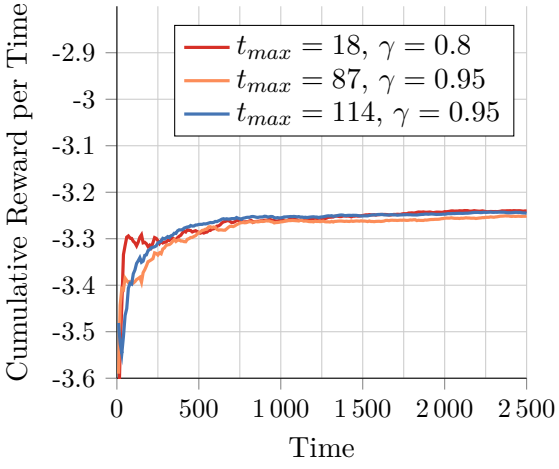
(b) *UCT constant, monitors the exploration.*



(c) *The SMART learning rate.*



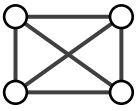
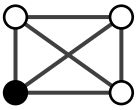
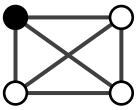
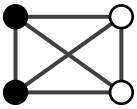
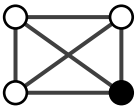
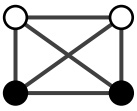
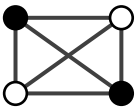
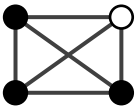
(d) *The number of simulations in each decision epoch. Here,* $t_{max} = 114$.

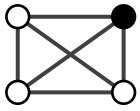
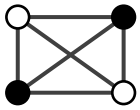
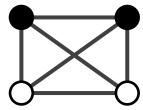
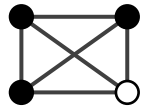
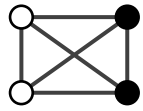
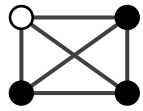
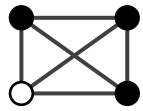
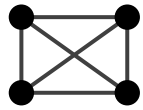


(e) *The time each rollout should be simulated.*

Figure 4.2: *In these figures, the performance when varying di erent parameters are shown. The parameters that are not altered are set to our standard parameters from Table 3.7, with one exception in Figure 4.2d. There are notable performance di erence for the UCT constant,* c*, and the number of searches,* n*. The results are averaged over 5 samples per parameter setting.*

Table 4.1: *The heatmap shows the fraction of times each action has been taken in each state, when excluding idle moves. The network represents the status of the queues, white for an empty and black for a non–empty queue. The position is the agents position in the network, where zero is the bottom left node, one top left, two bottom right, and three top right, as in Figure 3.1. This is based on the standard parameters from Table 3.7, with standard prior, and no pretraining, averaged over 5 samples.*

Network	Position	Action taken 0	1	2	3
0: 	**0**	0	0.00	0	0.00
	1	0	0	0	0
	2	0	0	0	0
	3	0	0	0	0
1: 	**0**	0	0	0	0
	1	0	0	0	0
	2	0	0	0	0
	3	0	0	0	0
2: 	**0**	0	0	0	0
	1	0	0	0	0
	2	0	0	0	0
	3	0	0	0	0
3: 	**0**	0	0	0	0
	1	0	0	0	0
	2	0	0	0	0
	3	0	0	0	0
4: 	**0**	0	0	0	0
	1	0	0	0	0
	2	0	0	0	0
	3	0	0	0	0
5: 	**0**	0	0	0	0
	1	0	0	0	0
	2	0	0	0	0
	3	0	0	0	0
6: 	**0**	0	0	0	0
	1	0	0	0	0
	2	0	0	0	0
	3	0	0	0	0
7: 	**0**	0.00	0	0	0
	1	0.00	0.00	0.00	0
	2	0.00	0.01	0.01	0.00
	3	0.00	0.00	0.02	0

Network	Position	Action taken 0	1	2	3
8: 	**0**	0	0	0	0
	1	0	0	0	0
	2	0	0	0	0
	3	0	0	0	0
9: 	**0**	0	0	0	0
	1	0	0	0	0
	2	0	0	0	0
	3	0	0	0	0
10: 	**0**	0	0	0	0
	1	0	0	0	0
	2	0	0	0	0
	3	0	0	0	0
11: 	**0**	0.02	0.02	0.01	0.00
	1	0.02	0.04	0.04	0.01
	2	0.02	0.06	0	0.00
	3	0.00	0.00	0.01	0.01
12: 	**0**	0	0	0	0
	1	0	0	0	0
	2	0	0	0	0
	3	0	0	0	0
13: 	**0**	0.03	0.02	0.01	0.00
	1	0.02	0	0.02	0.01
	2	0.01	0.02	0.02	0.00
	3	0.00	0.00	0.03	0.01
14: 	**0**	0	0.03	0.01	0.00
	1	0.01	0.02	0.01	0.01
	2	0.00	0.01	0.01	0.00
	3	0.00	0.00	0.01	0.00
15: 	**0**	0.03	0.02	0.01	0.00
	1	0.02	0.05	0.05	0.01
	2	0.02	0.06	0.07	0.00
	3	0	0.00	0.02	0.01

4.3 Priors

In total three different setups of priors were evaluated, as covered in Section 3.2.3. In Figure 4.3 the cumulative reward per time of the non–robust solver is presented, and in Figure 4.4 the same results are presented for the robust case. No large differences is observed in the performance. For these scenarios no notable difference was observed in action selection, compared to the action frequency of the standard parameter setup presented in Table 4.1.

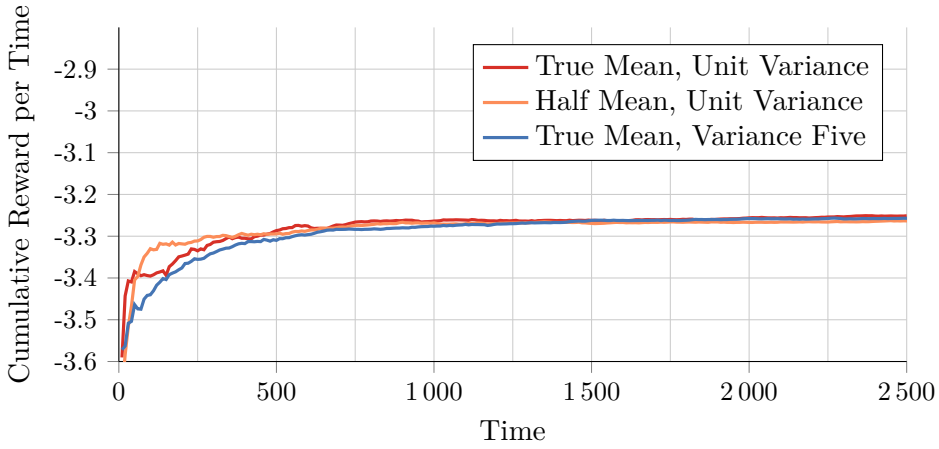
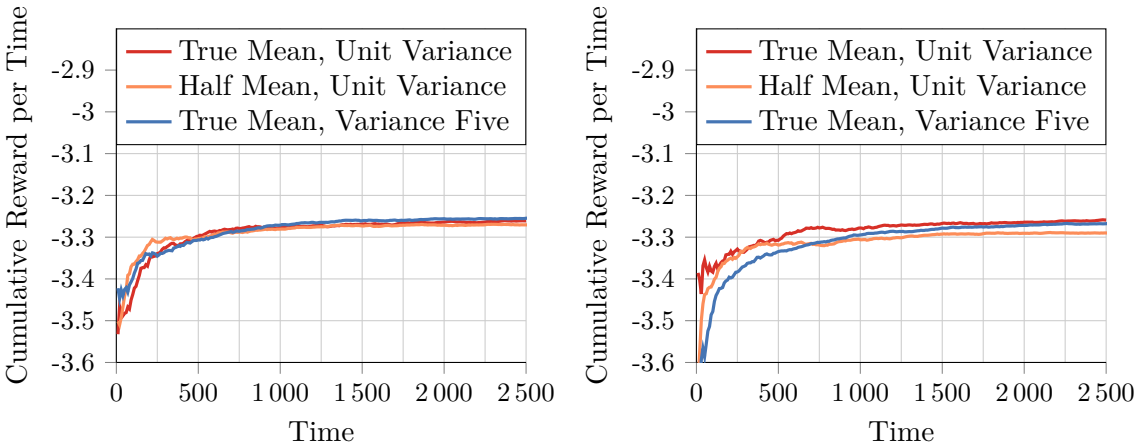


Figure 4.3: *The standard parameters from Table 3.7 are combined with three di erent priors. The true mean and unit variance is what we consider as standard. The half mean with unit variance, and true mean with variance five does not, however, result in a significantly di erent outcome. The true mean with unit variance is averaged over 5 samples, and the other two are averaged over 10 samples.*



(a) *Di erent priors for the robust solver with MSD parameter* $\alpha = 0.5$.

(b) *Di erent priors for the robust solver with MSD parameter* $\alpha = 1.0$.

Figure 4.4: *In these figures, we compare di erent priors for the robust solver with two di erent MSD parameters. The half mean with unit variance performs slightly worse than the other two. In both graph, the true mean, unit variance case is averaged over 5 samples and the other two are averaged over 10 samples.*

4.4 Pretraining

Here we cover the results of pretraining the solver. The cumulative reward per time unit, based on 1000 time units of pretraining, is prestented in Figure 4.5 for the non–robust case and in Figure 4.6 for the robust case. No difference is noted between the different pretraining, and the results remained the same when the pretraining time was increased to $10^4, 10^5$ and 10^6 time units. For these scenarios no notable difference was observed in action selection, compared to the action frequency of the standard parameter setup presented in Table 4.1.

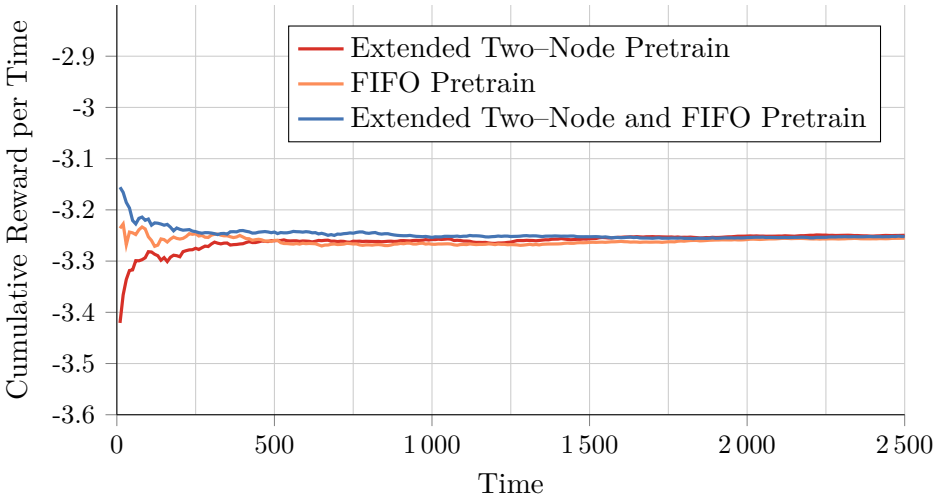
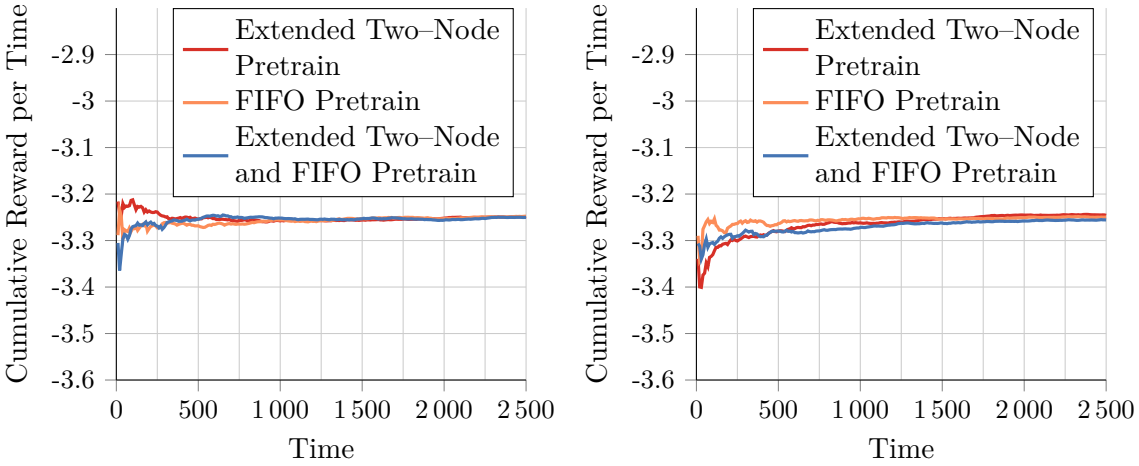


Figure 4.5: *In this figure, we visualise the performance of a solver, with the standard parameters from Table 3.7, that have been pretrained with three di erent methods for 1000 time units. All cases are averaged over 10 samples.*



(a) *Di erent versions of pretraining, with MSD parameter* $\alpha = 0.5$

(b) *Di erent versions of pretraining, with MSD parameter* $\alpha = 1.0$

Figure 4.6: *In this figure, we visualise the performance of the robust solver, with two di erent MSD parameters and the standard parameters from Table 3.7, that have been pretrained with three di erent methods for 1000 time units. All cases are averaged over 10 samples.*

5 Discussion

In this chapter, the aim is to break down the work that we have done. We will discuss aspects of the field, our contributions, the performance of our solver, and examples and suggestions of future work related to this thesis.

5.1 Characteristics of the Model

Here we discuss the limitations and characteristics of the model developed in this report, including the performance of the baseline policies. First, the time homogeneity of the model, and implications of it, is discussed. Following this, we discuss why we settled with this precise model. Lastly, we go into the performance of the baseline policies proposed for our model and discuss various aspects of deterministic policies developed by humans.

5.1.1 Time Homogeneity

The DVRP model described in Section 3.1 is, as previously stated, time homogeneous. This means that our solver does not need to keep track of when specific events occurred historically once that information has been included by updating the posterior distributions of the solver. Instead, the solver only needs to keep track of the current state of the environment when making decisions, which is a much less complicated task to perform. However, the limitations in modelling imposed by the time homogeneity can be quite severe. For instance, we are limited to a linear reward for the tasks, with respect to time, meaning that new and old tasks will always have the same reward associated with them. This is problematic as it could lead to tasks being left untouched indefinitely.

Another limitation imposed by the time homogeneity is that it limits the choice of queueing processes to Poisson processes. If a queue does not follow a Poisson process, it would require both the model and the solver to keep track of when tasks have arrived to a queue, thus requiring time inhomogeneity. Using a Poisson process, one assumes that arrivals are independent and the times between them are exponentially distributed. This is limiting in cases where, for instance, there are seasonal variations in task arrivals, or if there is correlation between arrival times of two or more queues.

Further, generalising the model by making it time inhomogeneous would allow for a model with distributions that change with temporal cycles, such as time of the day and seasons. Allowing such behaviour would be interesting as it enables modelling of, for instance, rush hour and light traffic, as well as seasonal changes in travel times, to mention a few examples.

In conclusion, from a modelling perspective, it would be of great interest to relax the time homogeneity and instead have a time inhomogeneous model. This would, however, require a more complex solver.

5.1.2 Aim of the Model

Our model, as it stands, was developed with a focus on three key aspects. The first aspect is that we, with this thesis, seek to evaluate how risk aversion and robustness

can be integrated with Bayesian reinforcement learning. Due to this, the model is not deterministic, in terms of outcomes, when performing an action. Having a stochastic model is not only a realistic modelling choice, but also yields possibilities for solvers trying to hedge against this stochasticity to act differently, compared to solvers not hedging against the stochasticity.

The second aspect is that we have sought to have a small, albeit representative, model to limit simulation times. This is important as we have prioritised being able to evaluate and explore different ideas, rather than focus, actively, on producing code optimised for high computational speed. As a result of opting for a small model the model only considers queues with space for a single task. This is not a theoretical limitation and longer queues are therefore easy to incorporate into the model. In addition, limiting the model size has made the result easier to analyse as the number of possible combinations of networks, positions and actions are somewhat comprehensible.

The third and last aspect considered during the model development is that the model should correspond well to the DVRP and the DTRP, as covered in Section 2.5, while fitting into the SMDP context, introduced in Section 2.1.7. This aspect has already been covered to some extent in Section 3.1.3, and how our work relates to the field of the DVRP, as studied from a historical point of view, is discussed further in Section 5.4.

5.1.3 Deterministic Policies

As an aim of this work is to enable autonomous decision making, possibly replacing problem–specific deterministic policies, it is interesting to discuss how our baseline policies perform in our model. This can, in turn, be linked to limitations of using problem–specific deterministic policies in more complex models. We start by noting that, while the best result we are able to produce with our solver outperform the FIFO policy, our solver is still surpassed by the results obtained using the two–node focus and extended two–node focus policies. As our model is quite small, and not overly complex, developing the two–node focus and extended two–node focus policies was not a hard task. However, realistically, a problem will likely be a lot more complex which raises the question of how well human developed deterministic policies will be able to perform. In complex scenarios, a good policy might not feasibly be constructed by a human, while a solver, such as ours, could still perform at the level it does in our current model. This motivates the usage, and further development, of approaches applying the same, or similar, techniques to the ones we employ.

5.1.4 Model Crowdedness

An important aspect of the model is how crowded it is. By a crowded model, we mean a scenario where policies are unable to keep the number of tasks at bay, which can be related to heavy traffic in the DTRP problem as discussed in Bertsimas and van Ryzin (1991), resulting in a model where the queues are full most of the time. Based on the state–action table, Table 4.1, and the performance of our baseline policies, see Figure 4.1, we characterise our model as quite crowded. The crowdedness may pose a limitation for reinforcement learning methods that require observations of transitions, as states with few tasks rarely occurs, and the solver thus has no opportunity to differentiate between these states.

5.2 Characteristics of the Solver

In this section, we discuss the characteristics and performance of our solver, MIH, in the scenarios we have evaluated. We start by discussing how the different priors we employed affected the solver, followed by a discussion on our hyperparameters and the effects of pretraining the solver. Lastly, we discuss the computational requirements of the algorithm and potential modifications of the algorithm.

5.2.1 Priors

Based on the results presented in Section 4.3, we see very little impact from the modified sojourn time priors. In the non–robust case the variations observed between the different priors are so small that we can not, with our replication count, claim with certainty that they are not due to variation between replicate simulations. In the robust case, we note some indications that may point towards a negative performance impact from the half mean, unit variance prior. These differences are, however, still small and it is unclear if the results would stand with more replicate simulations.

An interesting measure when studying the effect of the priors is the Kullback–Leibler divergence, that is a measure of how close two distributions are. Initially, we planned to include this measure for the posterior predictive and the true distributions of the position transition probabilities and the sojourn time distributions. However, due to lack of time this was not possible. Still, we wish to point out that the Kullback–Leibler divergence of the position transition probabilities was concluded to tend toward 0 quickly, as a part of a debugging process carried out during the project. We find this promising as it indicates that our solver is able to learn the position transition probabilities based off a uninformative prior. Further, we suggest that the sojourn time distributions should be studied using the Kullback-Leibler divergence as we believe this could provide valuable insight.

5.2.2 Hyperparameters

As made clear in Section 4.1, and by Figure 4.2, the only hyperparameters, with regards to the ranges we have investigated, that has a large impact on the results are the UCT constant, c, and the number of simulations per epoch, n. For the UCT constant, we conclude, based on the initial sweep, that values above 2 yield the best obtained performance of our solver. For the number of simulations per epoch, 250 or more simulations also results in the best obtained performance of our solver. The impact of introducing the robust modification to our algorithm was negligible. Also, using a search tree with max depth $t_{max} \approx 1000$ and 2000 simulations per epoch provided no performance increase.

The central question regarding these results is what limits our solver from performing at the level of, for instance, the baseline extended two–node focus policy. It may be the case that our algorithm is inadequate for some reason, but we believe that further increasing the number of simulations per epoch is warranted before drawing conclusions regarding this. What might be worrisome is that the algorithm appears agnostic to the majority of hyperparameters. In addition, to investigate an increased number of simulations per epoch, we suggest that the solver is applied to a less crowded model, as crowdedness might impact a solver severely, to see if the agnosticism prevails.

5.2.3 Pretraining

As seen by the results presented in Section 4.4, no version of the pretraining leads to improved solver performance. We hoped that pretraining the solver using data from the

extended two–node focus policy would have resulted in improved solver performance, as the extended two–node focus policy performs better than the solver without pretraining. However, this was not the case and we can not provide any clear motivation for this other than that it may be due to failing to identify better hyperparameters, model crowdedness, or unidentified errors in our implementation. No notable change in action selection was visible compared to the values presented for the standard parameters in Table 4.1.

What we do see from the results of the pretraining is that already with 1000 time units of pretraining, we almost completely remove the early performance dip of the solver, which occurs in the scenario without pretraining as our solver learns the system. These are promising results as they indicate that available historical data can be used to initialise the solver.

5.2.4 Robustness

Regarding the overall results of the robust solver, we note that, generally, using robust MIH leads to no, or a small, decrease in solver performance, compared to using the non–robust solver. As introducing robustness is expected to decrease the performance, in terms of reward obtained, these results may be interpreted as promising. However, we have not been able to verify, nor refute, that our solver behaves generally robust, that is to be robust with respect to dynamism and the model. As a result, we do not believe that too extensive conclusions should be drawn from these results. Instead, we suggest that the topic of robustness should be studied further to ensure a generally robust solver.

5.2.5 Computational Complexity and Algorithmic Modifications

Our algorithm, as it stands, is quite computationally demanding. While studying the computational complexity is not included in the scope of the thesis, we wish to point out that there exist multiple works on parallelisation of Monte–Carlo tree search. However, we refrain from mentioning any specific sources as we have not studied this topic in depth.

We also wish to point out that an alternative to using rollouts for action–value function estimation in new nodes, often encountered in modern literature, is to train a neural network to estimate this action–value function. We actively avoided this approach as using neural networks, currently, is not a good choice when pursuing theoretical robustness, due to the lack of convergence guarantees of the estimates.

Lastly, in our algorithm, we compute a tree with action–value estimates for each decision epoch. Something we believe is worth further consideration is to not compute a new tree for each decision epoch, but rather prune the tree according to the new history after a transition has been performed in the real environment. A question that arises with such an approach is how the older action–value estimates, that are sampled based on previous posterior distributions, and rollout policies, bias the action–value estimates in the current decision epoch. We have not had the time to consider this during our thesis, but nevertheless, find the question interesting as it may allow for a reduction in the number of search iterations applied in each decision epoch, resulting in shorter computation times.

5.3 Tried and Considered Techniques

Throughout this project, we have worked with, and considered, theory, techniques, and methods, that are not included in the algorithm we have developed. Most notably, two ideas were considered. We considered extending the *robust adaptive Monte Carlo planning* (RAMCP), a robust BAMDP solver developed by (Sharma et al. 2019), to the SMDP

domain. Secondly, we considered using the risk measure CVaR, described in Section 2.4.3. Below, we discuss why these concepts are interesting and why they were not included in the final algorithm.

5.3.1 Robust Adaptive Monte Carlo Planning

RAMCP is a robust BAMDP solver with many similarities to the BAMCP algorithm, that our solver extend. It is based on Monte–Carlo tree search, but the planning occurs over a fixed time horizon instead of a rolling time horizon as in BAMCP and our algorithm, MIH. It uses the risk measure CVaR to minimise model risk, that is the risk induced by modelling errors, and does so by reformulating the optimisation over CVaR to a two–player zero–sum game (Sharma et al. 2019). The reformulation is done to enable utilisation of ideas from game theory to compute the optimal policy. We find RAMCP interesting as it ensures model robustness. In addition, the game–theoretical approach to the optimisation of the risk measure is an interesting idea.

However, there are negative aspects of this method as well. A large drawback of the method is that for each state node encountered, it expands into all available action nodes when performing the Monte–Carlo tree search. This can be compared to BAMCP and MIH that only samples one action per encountered state node. The result is that the search tree grows exponentially with the number of actions, which increases the computational load dramatically. A possible solution to this problem could be to employ UCT, as used in BAMCP, and sample the actions based on this, which is something the authors mention as an idea for further work in their paper. In addition, we have found one effect of the fixed time horizon problematic. Namely, once a policy has been computed for the planning period, if the agent ends up in a sequence of states and actions that have not been encountered during the simulation, the proceeding actions until the end of the time horizon will be randomly selected. A quite simple solution to this could be to rerun the tree search from the current state if such scenario would occur. This has, however, not been evaluated. Given that we wish to develop a risk–averse solver, we find that this kind of designed behaviour is quite concerning and a large flaw. Lastly, the game–theoretic approach makes the extension to the SMDP domain more difficult than in the BAMCP case. Primarily based on the two former concerns, we decided not to pursue extending RAMCP further, despite the method having interesting aspects.

5.3.2 Conditional Value at Risk

In robust MIH, we use MSD as the risk measure. However, CVaR was also considered for this purpose. Both CVaR and MSD are coherent risk measures, under the conditions they are considered in this thesis. A difference between the two metrics is that CVaR satisfies all properties of a distortion risk measure, while MSD does not, as comonotone additivity does not hold for MSD (Tasche 2002). Majumdar and Pavone (2017) argues that, for high–level decision making, having a distortion risk measure is desirable, and as DVRP can be considered to be a high–level decision making task, in contrast with for instance robotic motion control, it makes CVaR interesting. Despite this, there are, primarily, two reasons which led us to select MSD instead of CVaR. The first is that CVaR is more complicated to compute and integrate into the solver, especially in the context we apply the risk measure in. Secondly, when optimising over CVaR, one solely seeks to reduce the risk of undesirable outcomes as only the values above the α–quantile are considered. On the other hand, when optimising over MSD, we both optimise with respect to the mean and the one–sided deviation. Thus, we not only consider reducing the risk, but also

increasing the overall performance.

5.4 The Dynamic Vehicle Routing Field

As SMDPs, to our knowledge, has not been widely applied to the DVRP, we dedicate this section to relating our work to the field of the DVRP as a whole, including proposing some ideas that we believe would benefit further research in this area.

5.4.1 Context of the Thesis

The field of dynamic vehicle routing is relatively young and has seen a large growth during the two most recent decades (Psaraftis et al. 2016). From what we have been able to find in our literature research, the application of SMDPs to the DVRP has not been widely studied. There exists some literature where MDPs are applied to the topic, though this is by far not the only angle of approach (Pillac et al. 2013; Psaraftis et al. 2016; Ritzinger et al. 2016). The most common approach is heuristic algorithms, and some other approaches are, for instance, nature–inspired algorithms and dynamic programming (Psaraftis et al. 2016). Thus, as the work in this thesis shows promise for the application of SMDPs to the DVRP, we believe that further research on this topic is warranted.

5.4.2 Benchmark Models

In this thesis, we have developed a model for evaluation of our SMDP–based method, not with a real use case scenario in mind but with the intent to include core aspects of the DVRP. As stated in the previous section, to our knowledge, the approach of using SMDPs to model the DVRP has not been widely applied. If more research is to be performed within this subject we, thereby, believe it would be of great value to have one or multiple benchmark models, based on real use cases, for evaluating the performance of different algorithms.

A suggestion for such benchmark models is to have a suite of models, based on one or multiple use cases, where more stochasticity and dynamism is added gradually in a well–defined manner. We believe that having a suite of models with incremental levels of complexity would be beneficial for research and development of solution methodologies. The benefit is that this makes identifying where approaches fail and succeed easier. Ideally, the models should map well enough to real use cases, such that research results benchmarked with the models can be compared with current industry level solutions, applied to the real use cases. This would enable a good evaluation of research results and possibly open possibilities for more collaborations between research and industry.

5.5 Future Work

Previous discussion sections have touched upon some ideas that we believe are worthwhile studying further. Below, we present these, and additional, ideas grouped by whether the idea is related to the model, the solver, or none of these specifically.

5.5.1 Model

As discussed above, our model is time homogeneous, which imposes some limitations. We, therefore, believe it would be interesting, as future work, to generalise the model by making it time inhomogeneous.

In part linked to, but not necessarily requiring, the inclusion of time inhomogeneity in the model, we think that adding drift to the model distributions would be an interesting case to study. This would be interesting as, realistically, there should exist applications where model distributions drift.

Other aspects that we believe could be interesting to study, which are partially related to the discussion in Section 5.1, are expanding the state space, both with more nodes and longer queues, looking into other queue and transition distributions, and lastly, using non–linear task rewards that increase with time.

5.5.2 Solver

When it comes to the solver, a quite clear proposal for future work is studying convergence from a theoretical point of view. Further, an interesting, albeit computationally even more complex, scenario to study would be to introduce partial observability, as discussed for MDPs in Section 2.1.5. In this setting, the agent does not, necessarily, know what state it is in, but instead has a belief over states. Also, in our approach, the queue parameters are known by the solver. Removing this would require introducing time inhomogeneity, which in itself is interesting. Moreover, it would be interesting to study how an approach such as ours could handle having to learn the queue distributions in addition to the state and time transitions already it is already required to learn.

Regarding the priors, we believe there are multiple interesting further topics for research. First, a deeper analysis of how sensitive our algorithm, MIH, is to the prior knowledge would be interesting. In this thesis, we have only varied the prior in the same way for all state–action pairs. We, therefore, suggest that varying the prior differently for the different state–action pairs and using a prior with incorrect mean and large variance at the same time. Secondly, we have only used the true likelihoods in this thesis. Thereby, studying how, and when, selecting incorrect likelihoods has an impact would be interesting. Thirdly, we believe it would be interesting, especially if the method developed is to be applied to a real use case, to study more complex priors such as hierarchical models, or models with multiple unknown parameters. Lastly, it would also be of interest to study how the solver would be affected by introducing noise, such as occasionally feeding the posterior and SMART updates with incorrectly labelled actions.

As states above, in the solver discussion, we have not been able to verify that our solver is robust in terms of dynamism and with respect to the model. Therefore, we believe that studying the topic of robustness further, to provide clear evidence for these types of robustness, is justified. In addition to this, we believe that applying the solver to a less crowded model would be of interest as our current model is quite crowded which might limit the possibility of the solver to learn the system properly.

5.5.3 Other

A further suggestion for future work, that is not directly model or solver related, is the application of our algorithm to a real–world use case. The work carried out in this thesis has not been done with a certain real–life use case in mind. Therefore, it would be interesting to see how MIH would perform in a real–world application, and how it might need to be modified.

Lastly, there are several problems studied within optimisation that have similarities with the DVRP. Examples of such problems are scheduling and maintenance optimisation. As the SMDP approach that we have applied is quite general, we believe that the ideas employed in this thesis could be applied to other topics than DVRP. It would thus be

interesting to look further into these related topics to see if similar approaches have been applied in these contexts, as well as trying to apply results from our work within these topics.

6 Conclusion

The goal of this thesis has been to explore dynamic vehicle routing, define an SMDP model within this context, and develop a solver for this model. The work has consisted of an extensive literature study as the fields in consideration are subject to active research. Following this, the model and the solver development was performed in parallel, with more complexity added incrementally. Initially, a substantial focus was placed on understanding how the different parts of the project were to be defined and used. Then, we began to find a way to comprise these individual parts into one entity. To conclude the work in this thesis, we are going to touch upon the three main areas of this thesis.

We begin with summarising the work relating to the model. The goal was to develop a model based on principles from SMDPs while trying to frame it as a DVRP. This has been successful as we present a model that we believe represent a good example of the desired properties. This model, that purposely has been kept small, has a great potential for further development as it can be modified to represent larger problems, or tuned to imitate a specific behaviour. The only concern we have regarding the model is that it, in its current configuration, might be too crowded, with respect to task arrivals, for a solver to learn to handle it well. However, we think that this scenario is better than the opposite, with a too empty model, as the latter tends to be a simpler problem.

Secondly, we want to draw some conclusions regarding our solver, MIH. The aim was to develop a Bayesian reinforcement learning algorithm that solves time homogeneous semi–Markov decision processes. This aim has been fulfilled as we present a functioning solver for the proposed model. Still, the performance of this solver has, in our simulations, not been as good as we had hoped. This could be contingent on the functionality of the solver not being sufficient, that we have not pinpoint good enough hyperparameters, or some other factor that we have failed to identify. However, we are satisfied with the result as we have been able to prove that the MIH algorithm learns to make decisions and that it has shown better performance than, for example, the baseline FIFO policy.

The third area that we have studied and worked with is robustness. The goal was to evaluate whether a risk measure could be incorporated into our solver to make the policies more risk–averse. We have concluded that it is possible to incorporate risk measures into our solver and that the robust version of the MIH solver does not perform any worse than the non–robust version. Although, we have not been able to validate that the robust solver actually makes more safe decisions as this was a too extensive task to fit into the scope of this thesis. Assessing the robustness of the solver requires further clever development of the model to separate the behaviour of the robust solver from the non–robust case. This, together with an evaluation of whether coherent risk measures are sufficient, or if the extension into distortion risk metrics is necessary, are good starting points for further work.

To summarise, this project has allowed us to explore new territory with respect to SMDPs and the DVRP. We have been able to produce methods and results that fulfil the aim of this thesis as we have presented a Bayesian reinforcement learning algorithm that finds solutions in our SMDP model. What is yet to explore is methods for robustness validation, application to real–world problems, and tuning of the MIH solver for better performance.

References

Adewumi, A. O., & Adeleke, O. J. (2018). A survey of recent advances in vehicle routing problems. *International Journal of Systems Assurance Engineering and Management*, *9*(1), 155–172.

Baykal-Gürsoy, M. (2010). Semi-Markov Decision Processes. In *Wiley encyclopedia of operations research and management science*. American Cancer Society.

Bertsimas, D. J., & van Ryzin, G. (1991). A Stochastic and Dynamic Vehicle Routing Problem in the Euclidean Plane. *Operations Research*, *39*(4), 601–615.

Bertsimas, D. J., & van Ryzin, G. (1993). Stochastic and Dynamic Vehicle Routing in the Euclidean Plane with Multiple Capacitated Vehicles. *Operations Research*, *41*(1), 60–76.

Chapman, M. P., Lacotte, J. P., Smith, K. M., Yang, I., Han, Y., Pavone, M., & Tomlin, C. J. (2019). Risk-sensitive safety specifications for stochastic systems using conditional value-at-risk.

Chow, Y., Ghavamzadeh, M., Janson, L., & Pavone, M. (2018). Risk-constrained reinforcement learning with percentile risk criteria. *Journal of Machine Learning Research*, *18*, 1–51.

Das, T. K., Gosavi, A., Mahadevan, S., & Marchalleck, N. (1999). Solving semi-Markov decision problems using average reward reinforcement learning. *Management Science*, *45*(4), 560–574.

Dearden, R., & Andre, D. (1999). Model based Bayesian Exploration, In *Proceedings of the fifteenth conference on uncertainty in artificial intelligence*.

Dimitrakakis, C., & Ortner, R. (2019). Decision Making Under Uncertainty and Reinforcement Learning.

Ghavamzadeh, M., Mannor, S., Pineau, J., & Tamar, A. (2015). Bayesian reinforcement learning: A survey. *Foundations and Trends in Machine Learning*, *8*(5-6), 359–483.

Grimmett, G., & Stirzaker, D. R. (2001). *Probability and random processes.* Oxford University Press.

Guez, A., Silver, D., & Dayan, P. (2013). Scalable and efficient bayes-adaptive reinforcement learning based on Monte-Carlo tree search. *Journal of Artificial Intelligence Research*, *48*, 841–883.

Kocsis, L., & Szepesvári, C. (2006). Bandit based monte-carlo planning, In *Machine learning: Ecml 2006*, Springer Berlin Heidelberg.

Larsen, A. (2000). *The Dynamic Vehicle Routing Problem* (Doctoral dissertation). Technical University of Denmark.

Majumdar, A., & Pavone, M. (2017). How Should a Robot Assess Risk? Towards an Axiomatic Theory of Risk in Robotics.

Mcmahon, J. J. (2008). *Time–Dependence in Markovian Decision Processes* (Doctoral dissertation). The University of Adelaide.

Pillac, V., Gendreau, M., Guéret, C., & Medaglia, A. L. (2013). A review of dynamic vehicle routing problems. *European Journal of Operational Research*, *225*(1), 1–11.

Psaraftis, H. N., Wen, M., & Kontovas, C. A. (2016). Dynamic vehicle routing problems: Three decades and counting. *Networks*, *67*(1), 3–31.

Ritzinger, U., Puchinger, J., & Hartl, R. F. (2016). A survey on dynamic and stochastic vehicle routing problems. *International Journal of Production Research*, *54*(1), 215–231.

Rockafellar, R. T., & Uryasev, S. (2002). Conditional value-at-risk for general loss distributions. *Journal of Banking and Finance*, *26*(7), 1443–1471.

Ross, S., Chailb-draa, B., & Pineau, J. (2008). Bayes-Adaptive POMDPs, In *In proceedings of the advances in neural information processing systems*.

Shapiro, A., Dentcheva, D., & Ruszczyński, A. (2009). *Lectures on Stochastic Programming*. Society for Industrial; Applied Mathematics.

Sharma, A., Harrison, J., Tsao, M., & Pavone, M. (2019). Robust and adaptive planning under model uncertainty, In *Proceedings of the twenty-ninth international conference on automated planning and scheduling*.

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement Learning, An introduction*. MIT Press.

Tamar, A., Chow, Y., Ghavamzadeh, M., & Mannor, S. (2017). Sequential Decision Making with Coherent Risk. *IEEE Transactions on Automatic Control*, *62*(7), 3323–3338.

Tasche, D. (2002). Expected shortfall and beyond. *Journal of Banking & Finance*, *26*(7), 1519–1533.

Wang, S. S. (2000). A Class of Distortion Operators for Pricing Financial and Insurance Risks. *Journal of Risk & Insurance*, *67*(1), 15–36.

Watkins, C. (1989). *Learning from delayed rewards* (Doctoral dissertation). King's College.

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

CHALMERS
UNIVERSITY OF TECHNOLOGY