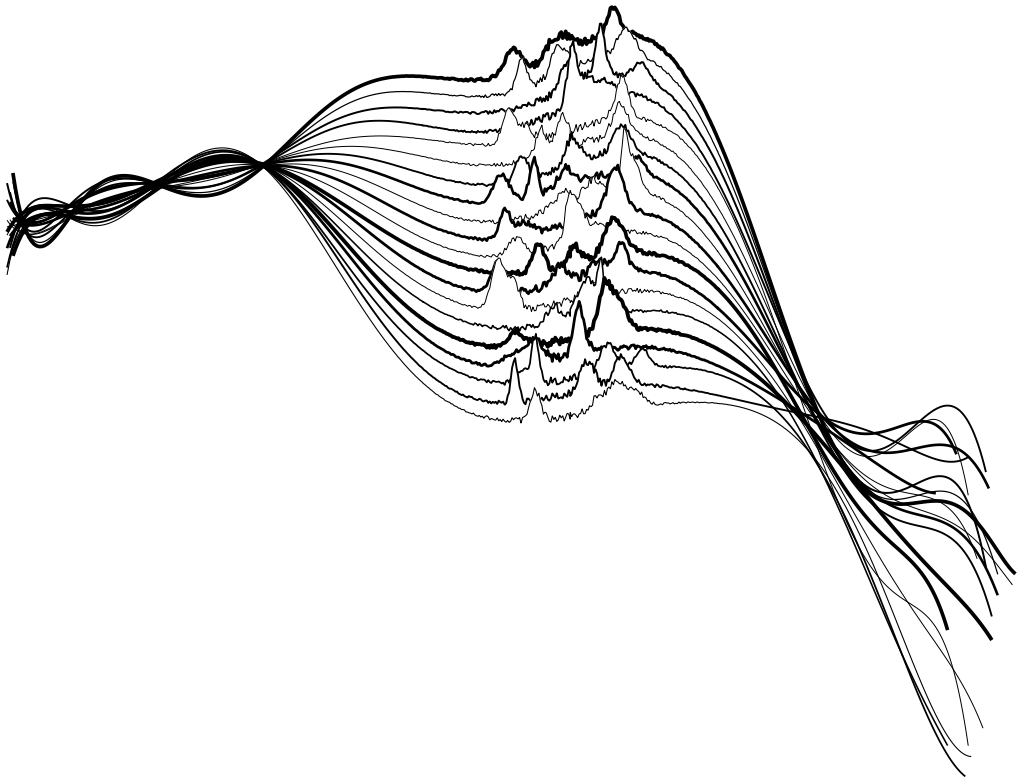
CHALMERS
UNIVERSITY OF TECHNOLOGY



Temporal Logic Specifications in Reinforcement Learning

Methods for imposing constraints

Master's thesis in Systems, Control and Mechatronics

CARL-JOHAN HEIKER

Department of Electrical Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2021

Master's thesis 2021

Temporal Logic Specifications in Reinforcement Learning

Methods for imposing constraints

CARL-JOHAN HEIKER

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Division of Systems and Control
Automation Research Group
Chalmers University of Technology
Gothenburg, Sweden 2021

Temporal Logic Specifications in Reinforcement Learning
Methods for imposing constraints
CARL-JOHAN HEIKER

Supervisor and Examiner: Bengt Lennartson, Chalmers University of Technology

Master's Thesis 2021
Department of Electrical Engineering
Division of Systems and Control
Automation Research Group
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Interpretation of a constrained behaviour visualised as a modified radio pulsar.

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2021

Temporal Logic Specifications in Reinforcement Learning
Methods for imposing constraints
CARL-JOHAN HEIKER
Department of Electrical Engineering
Chalmers University of Technology

Abstract

This work, consisting of two parts, examines the possibilities and issues with enforcing specified behaviors when performing reinforcement learning (RL). For a discrete event system, modelled as a Markov decision process (MDP), a specification dictating the present and future system behavior can be formulated as a linear temporal logic (LTL) formula, which is then realised by a Büchi automaton. A constrained MDP control policy can then be obtained through RL, performed on a product system constructed by a form of synchronous composition of the MDP and the Büchi automaton.

In the first part of this thesis, a small research platform is constructed where discrete time MDPs and LTL realising Büchi automata can be implemented. Path planning control problems for so called slippery grid worlds are formulated, and the chosen specifications test the limits of TL constrained RL. Specifically, constraints that are hard to fulfill with classic RL are considered, such as sequential state visits and liveness specifications.

Three different composite algorithms are proposed, tested and evaluated, and they draw inspiration from existing methods for TL constrained RL. Firstly, a standard TL constrained Q-learning is considered. Secondly, a detached accepting frontier extension that rewards reaching specific intermediate automaton states is formulated for the standard algorithm. Thirdly, a potential function that rewards the RL agent, depending on the direction of travel, is fitted to the aforementioned second algorithm.

In the second part of this thesis, a method to reduce the computational burden for the TL constrained RL algorithm is considered. By the use of modular analysis, a jointly controlled path planning and admission control problem modelled as a continuous time MDP is divided into two partitions, where one part is modelled as a queue for which a threshold type admission control policy can be found analytically. This reduces the RL exploration needed while also reducing the infinite joint state space, making it possible to find the path planning policy through tabular TL constrained RL.

The results of the first part show that there are situations, such as when the state space is large, in which additional methods are needed to solve temporal logic constrained reinforcement learning problems. The experiments presented in the second part entail that modular analysis successfully reduces the amount of exploration needed for the learning agent to find a path planning policy that agrees with a formulated specification.

Keywords: Reinforcement learning, Temporal logic, Reward shaping, Büchi automata, Safety, Liveness, Fairness

Acknowledgements

The author would like to thank Bengt Lennartson for his guidance throughout this project, without which this thesis would not have been possible. While this work was conducted with a large degree of freedom in terms of research direction, many long discussions provided all of the necessary inspiration and were crucial to find a clear path to the goal when this was not apparent.

Furthermore, Constantin Cronrath has had a keen eye for when motivation and moral needed to be injected into the project. His ability to provide the right perspective and put things in the right light has been very much appreciated.

Many thanks to my family for supporting me throughout this project and for always believing in me.

Carl-Johan Heiker, Gothenburg, January 6, 2021

Contents

List of Figures **xiii**

List of Tables **xvii**

1 Introduction **1**
1.1 Background . . . 2
1.2 Aim . . . 3
1.3 Limitations . . . 4
1.4 Research Questions . . . 5
1.5 Contributions . . . 5
1.6 Structure . . . 7

I Temporal Logic Constrained Reinforcement Learning **11**

2 Reinforcement Learning **13**
2.1 Discrete Event Systems . . . 13
2.2 Markov Decision Processes . . . 13
2.3 Reinforcement Learning Fundamentals . . . 17
2.4 Q-learning . . . 21
2.5 Summary . . . 23

3 Temporal Logic Specifications **25**
3.1 Temporal Logic and Predicate Logic . . . 25
3.2 Linear Temporal Logic . . . 26
3.3 Formal Languages . . . 26
3.4 Automata . . . 28
3.5 Translating LTL Formulae to Automata . . . 29
3.6 Büchi Automata . . . 30
3.7 Summary . . . 34

4 Algorithms and Methods **35**
4.1 Algorithm 1: Temporal Logic Constrained Reinforcement Learning . . . 35
4.2 Algorithm 2: LDBA Constrained Reinforcement Learning . . . 38
4.3 Algorithm 3: Shielded Reinforcement Learning . . . 41

4.4 Reward Shaping . . . 42
4.5 Advice Based Exploration . . . 44
4.6 Comparing and Combining Algorithm Features . . . 45
4.7 Summary . . . 48

5 Problems and Measurement Techniques 51
5.1 Evaluating LTL Constrained Reinforcement Learning . . . 51
5.2 Reinforcement Learning Problem Categories . . . 52
5.3 Measuring Algorithms . . . 54
5.4 Summary . . . 57

6 Implementation of Research Platform 59
6.1 Environment . . . 59
6.2 Automata . . . 63
6.3 Statistics . . . 65
6.4 Experiment Implementation . . . 66
6.5 Summary . . . 66

7 Experiments 67
7.1 Experiment Structure . . . 67
7.2 Safe Navigation to Destination . . . 69
7.3 Sequential State Visits Experiment 1 . . . 74
7.4 Sequential State Visits Experiment 2 . . . 78
7.5 Liveness and Fairness Experiment 1 . . . 84
7.6 Liveness and Fairness Experiment 2 . . . 88
7.7 Sequential State Visits Experiment 3 . . . 93
7.8 Potential for Initial Guiding Experiment . . . 97
7.9 Summary . . . 101

8 Conclusions of Part I 103
8.1 Research Platform Evaluation . . . 103
8.2 Conducted Experiments . . . 104
8.3 Answers to Research Questions . . . 106
8.4 Additional Conclusions and Suggestions for Future Work . . . 108

II Modular Analysis 111

9 Introduction to Part II 113
9.1 Background . . . 113
9.2 Problem Formulation . . . 113
9.3 Limitations . . . 115
9.4 Changes to Implementation . . . 115

10 Generalised Semi Markov Processes and the Poisson Distribution **117**
10.1 Stochastic Timed Automata and GSMP . . . 117
10.2 Poisson Counting Process . . . 119
10.3 The Poisson Distribution . . . 123
10.4 Superposition of Multiple Poisson Processes . . . 126
10.5 Summary . . . 128

11 Modelling with Discrete and Continuous Markov Chains **129**
11.1 Continuous Time Markov Chains . . . 130
11.2 Discrete Time Markov Chains . . . 134
11.3 Uniformisation . . . 137
11.4 Joint Markov Chains . . . 139
11.5 Summary . . . 145

12 Queuing Theory **147**
12.1 Concepts and Notation in Queueing Theory . . . 148
12.2 Markovian Queueing Systems . . . 150
12.3 Markov Decision Processes and Analytical Solutions to the Optimality Equation . . . 156
12.4 Summary . . . 161

13 Control Problems **163**
13.1 Admission Control for M/M/1 Queues . . . 163
13.2 Two Methods of Finding the Optimal Threshold . . . 168
13.3 Path Planning and Admission Control in Continuous Markov Processes . . . 172
13.4 Summary . . . 181

14 Experiments **183**
14.1 M/M/1 Queues . . . 183
14.2 M/M/1/K Queues . . . 192
14.3 Optimal Admission Control Threshold . . . 198
14.4 Temporal Logic Constrained Q-learning of Joint Path Planning and Admission Control Process . . . 203
14.5 Summary . . . 212

15 Conclusions of Part II **213**
15.1 Evaluation of Method and Implementation . . . 213
15.2 Conducted Experiments . . . 216
15.3 Answers to Research Questions . . . 218
15.4 Final Project Conclusions and Suggestions for Future Work . . . 219

Bibliography **221**

List of Figures

2.1 A deterministic transition from s_4 to s_1 via the action N. 16
2.2 Probabilistic transitions resulting from selecting the action N. 17
2.3 The interaction based reinforcement learning principle. 18

3.1 DFA with the marked state q_3 and the forbidden state q_2. 28
3.2 Automaton from [1] realising $\varphi = \Diamond p \wedge \Box \neg q$. 30
3.3 NBA from [2] that satisfies the LTL property $\varphi = \Box b \wedge \Box \Diamond a$. 31
3.4 DBA from [2] that satisfies the LTL property $\varphi = \Box b \wedge \Box \Diamond a$. 32
3.5 NBA from [2] representing the language of $(a + b)^* b^\omega$. 32

4.1 Visualized example from [1] of the product $\mathcal{M} \otimes \mathcal{B}_\varphi$ between an MDP $\mathcal{M}$ and a Büchi automaton $\mathcal{B}_\varphi$. 36
4.2 Temporal logic constrained reinforcement learning. 37
4.3 LDBA constrained reinforcement learning. 40
4.4 The shielding principle from [3]. 41

5.1 Extracting the optimal policy from a converged Q-table. 56

6.1 Example of indexed coordinate states. 60
6.2 The new transition probability assignment method. 63

7.1 Grid world representation of the safe navigation to destination problem. 69
7.2 Colored deterministic Büchi automaton specifying safe navigation to a destination. 70
7.3 Statistics from the safe navigation to destination experiment. 72
7.4 Representative state sequences derived from the converged Q-tables. . . 73
7.5 Grid world MDP visualisation for the first sequential experiment. . . . 74
7.6 Deterministic Büchi specification for the first sequential state visits experiment. 75
7.7 Statistics from the first sequential state visits experiment. 77
7.8 Representative state sequences derived from the converged Q-tables. . . 78
7.9 Grid world MDP for the second sequential problem. 79
7.10 Deterministic Büchi automata with one set of colored states per specified sequence. 80
7.11 Statistics related to the second sequential state visits experiment. . . . 82
7.12 Representative state sequences for the second sequential state visits experiment. 83
7.13 Grid world MDP for the first liveness and fairness problem. 84

7.14 DBA for liveness and fairness suggested in [4] with generalized deterministic Büchi automaton proposed in [2]. 85
7.15 The statistics for the first liveness experiment. 87
7.16 Representative state sequences for the liveness experiment. 88
7.17 A modified version of the GBA from [2]. 89
7.18 Statistics for the second liveness experiment. 91
7.19 Representative state sequences derived from the converged Q-tables. . . 92
7.20 16×16 grid world MDP designed for the large scale sequential problem. 93
7.21 Deterministic Büchi realization of a sequential LTL specification. 94
7.22 Statistics for the third sequential experiment. 96
7.23 Representative state sequences derived from the converged Q-tables. . . 97
7.24 Statistics for the initial potential guiding experiment. 99
7.25 Representative state sequences derived using the Q-tables of the initial potential guiding experiment. 100

10.1 The Poisson distribution for different λ and t. 126

11.1 Continuous Markov chain describing a pure birth process. 134
11.2 Oscillatory versus stationary state probability development. 137
11.3 State transition rate and state probability diagrams for the continuous time pure death Markov chain and its uniformised discrete counterpart. 139
11.4 Two independent continuous Markov processes that form one joint system. 142
11.5 Example of a uniformised joint system. 145

12.1 A basic single server queueing system. 147
12.2 State transition rate diagram for the $M/M/1$ queueing system. 151
12.3 State transition rate diagram for the $M/M/1$ queueing system. 153
12.4 State transition diagram for the uniformised $M/M/1/K$ system. 154
12.5 State transition diagram for the uniformised $M/M/1/K$ system. 155

13.1 Example state space of the continuous time Markov grid world. 174
13.2 Intended direction transitions λ and slip transitions ω. 175
13.3 Possible transitions when $a(t) = \begin{bmatrix} 0 & 1 & 0 & 0 \end{bmatrix}^T$ 176
13.4 The joint process ordered as a tower of path planning state sets. 177
13.5 Joint state with admission control over the queue transitions and path planning controlled directional transitions with slip. 178

14.1 State transition rate diagram for the $M/M/1$ queueing system. 183
14.2 Relationship between probability and time interval through the exponential CDF. 184
14.3 $M/M/1$ queueing system simulation with recorded state time distributions. 187
14.4 State transition diagram for the uniformised $M/M/1$ queueing system. 188
14.5 Results for the uniformised $M/M/1$ simulation experiment. 189
14.6 Stationary state time distributions for the $M/M/1$ and uniformised $M/M/1/K$ systems. 191
14.7 State transition rate diagram for the $M/M/1/K$ queueing system. . . . 192
14.8 Results for the $M/M/1/K$ simulation experiment. 194

14.9 State transition diagram for the uniformised $M/M/1/K$ system. 195
14.10 Results for the uniformised $M/M/1/K$ simulation. 196
14.11 State time distributions for the $M/M/1/K$ and the uniformised $M/M/1/K$ systems. 198
14.12 Expected total undiscounted cost of admission controlled queues with different thresholds. 200
14.13 Simulated discounted cost of admission controlled queues for different limits K. 202
14.14 Principle sketch of the two ways of viewing a process. 205
14.15 Generalised Büchi automaton for the final experiment. 206
14.16 Q-table development. 210
14.17 Recommended actions, denoted by arrows, for all joint states. 211

List of Tables

3.1 Predicate logic propositions with examples. 25
3.2 LTL modalities with examples. 26

7.1 Parameters for the safe navigation to destination experiment. 71
7.2 Parameters for the first sequential state visits experiment. 76
7.3 Parameters for the second sequential experiment. 80
7.4 Parameters for the first liveness and fairness experiment. 85
7.5 Parameters for the second liveness experiment. 90
7.6 Parameters for the first sequential experiment on a 16×16 grid world MDP. 94
7.7 Parameters for the potential guiding experiment. 98

14.1 Rate parameters for the joint path planning and admission control experiment. 205
14.2 Cost parameters for admission control. 207
14.3 Temporal logic constrained RL parameters. 208

1
Introduction

A *Markov decision process (MDP)* is a common way of modelling *discrete event systems (DES)* which, in essence, are systems consisting of states between which transitions, triggered by actions, are made. MDPs are versatile, since most systems to some extent can be modelled as deterministic or probabilistic transition processes.

Environment modelling using Markov decision process is a standard approach in *reinforcement learing (RL)*, in which an agent shall interact with an environment MDP by the use of actions. The RL problem aims to derive a way of selecting the actions, and the derived strategy is called a policy which is optimal with respect to a defined cost function.

An MDP model may include probabilistic transitions between states, and this is often a reasonable abstraction for unmodelled parts of an environment. An optimal control policy can be obtained even for models with stochastic properties by the use of RL. This is due to the *Markov property*, which says that all information of the previous states is encapsulated in the current state of the system. In RL, this property is combined with the fundamental *Bellman optimality principle* and *dynamic programming*, to form an iterative policy deriving algorithm that assigns values to the observed states and actions of the environment.

Although RL can find the optimal policy, this can understandably not happen at any cost in practical applications. Often, there is a need to develop a solution that is not only optimal, but also safe. As an example, if an optimal way to walk through a city is to be derived, it is reasonable to avoid walking on highly trafficked and dangerous roads.

In this setting, *temporal logic (TL)*, and in particular *linear temporal logic (LTL)*, can be used to impose constraints on a learning problem. LTL is a type of modal logic, dictating both the present and the future behaviour of a system. For example, a requirement may be that a certain state shall always be reached *eventually*. Naturally, the opposite constraint may also be formulated, if avoiding certain states is desired. Specific LTL formulae can be realised as *regular languages* for which it is possible to formulate a corresponding automaton that embodies that language in its transitions. Depending on the language, different types of automata are required.

This work concerns ways of solving reinforcement learning problems on systems that are constrained by LTL specifications, formulated as automata. These specifications are inspired by realistic and practically applicable constraints that may be necessary to formulate for a wide range of systems.

1.1 Background

Machine learning is becoming an increasingly popular topic in industry, and the method is being utilized due to its versatility and availability; many different problems can be solved by methods that are easily accessible for everyone. With basic knowledge in reinforcement learning, it is easy to adapt the formality of minimizing a cost function to economic efficiency or resource management optimisation. However, this perspective is not always compatible with the ethical aspects of machine learning. Topics such as safety may become less interesting, while remaining very important as the complexity and extent of systems increase with time.

Due to the interest of the industry and the vast number of problems that are being solved with reinforcement learning, a currently active area of research is to derive new ways of ensuring that different types of specifications can be fulfilled. The possibilities of developing a framework for which increasingly complicated specifications, describing many different system behaviors, thus serve as motivation for both industry and research.

While there are many different reasons to constrain a system, one of them is purely based on the physical limitations of the system that is modelled. However, to some extent it is possible to motivate the imposing of constraints to a deeper level. For certain systems, solely ethical dilemmas are not very far fetched. For example, take the popular problem in which an RL based strategy for navigating a self driving car is to be derived. It might be hard to guarantee that the algorithm will produce an optimal policy that does not include dangerous states, which in this case could mean life or death, and ensuring a safety specification may in this case save lives. Therefore, while not pretending to solve ethical dilemmas, this project is in a position to help increase safety in a vast variety of processes and thus contribute to increased safety while still maintaining a focus on optimization, keeping it attractive in industrial applications. More information about the work towards safety in machine learning can be gathered from the Institute for Ethical AI & Machine Learning [5].

To motivate this work further, one of the major topics in reaching the current national climate goals (available from the Swedish Energy Agency [6] where the national goals are put in context to the established European climate goals for 2020 and 2030) is to minimize the energy consumption. Sectors such as trade, transportation and industry are responsible for a majority of the Swedish energy consumption [7], and as any measurement of pollution or CO2 emission could in theory be turned into an RL performance criterion, constrained reinforcement learning may be a useful investment to combat the climate change challenges that we stand before.

Regardless of the application of this broad theory, the question remains the same: what are the possible benefits and challenges with trying to impose logical constraints on reinforcement learning problems?

1.2 Aim

This is a two part Master's thesis, and in the first half, focus lies on exploring different methods for imposing specifications formulated in LTL expressions on reinforcement learning problems. The work done in Part II, where the method of modular analysis is used to solve a highly specific control problem, is very much dependent on the concepts developed in Part I. Therefore, the aim presented here can be applied to the entire project.

The goal of the project is to demonstrate how one can benefit from formulating constraints for reinforcement learning problems in temporal logic, and to investigate different situations and special cases in which modifications to a particular method is needed. Furthermore, it is of interest to describe, nuance and possibly solve the aforementioned potential contradictions that may emerge. For example, integrating the LTL constraints into RL problems may include additional performance measurements that are to be minimized. Therefore, the task runs the risk of introducing contradictions and trade offs between maximising performance reward and ensuring that a safety specification is not violated. Thus, exposing, exploring and discussing challenging and contradictory situations is a part of this project, and the aim here is to propose ways of handling these difficulties.

In the first part of this project, the performance of the main algorithm, proposed in [1], is tested on different challenging specifications. To develop solutions to the problems that arise, an extensive literary study is conducted where other variants of logically constrained reinforcement learning are investigated. The purpose of this is to draw inspiration from other methods, and integrate the key features of these into the basic version of the main algorithm as improvements, which may make it possible to solve more challenging problems. In essence, this project investigates the possibilities, but also the limitations of methods for imposing temporal logic constraints on reinforcement learning. When a limitation is discovered, the aim is to both highlight the challenges that one should expect will arise in the particular scenario, and propose ideas for how to handle them.

In the second part of the project, the results of the first part of the project are complemented by the formulation of the previously mentioned modular analysis, which is one of the proposed future research directions formulated in [1]. The purpose of this is to evaluate if the method of modular analysis is an efficient way of solving temporal logic constrained reinforcement learning problems when the environment can be modelled as distinct partitions working together. Specifically, by identifying control policies for each partition of a jointly controlled path planning and queueing process, the goal is to find situations in which the modular analysis method reduces the complete reinforcement learning problem such that it can be solved more efficiently than if reinforcement learning was used to find the solution to the whole problem.

1.3 Limitations

In this section, limitations are formulated for the first part of this project. Limitations for the second part are found in Section 9.3 of Part II, and are formulated there due to the fact that they depend upon the evaluation of Part I of this thesis.

To evaluate the performance of the algorithms under different circumstances and with different additional modifications, it is in the first part of this project necessary to develop a platform where RL problems for different specification automata and MDPs can be formulated. There are certain demands on this implementation that must be met, but it is also important to keep it as simple and stable as possible in order to produce quality results.

The environment with which the algorithms interact can thus be considered the core of the platform. A scalable and easily modified grid-world environment is sufficient to explore basic problems and is also one of the most common environments in experiments with reinforcement learning. An effort is first made to implement this type of environment in a minimal way from the ground up, but if it proves to be a more stable solution, an open source environment will be used instead.

The LTL formulae needed for this project are deemed simple enough to be manually translated to corresponding automata. However, if the automaton representation proves to be difficult to find, there is a software tool called *OWL* [8] which outputs an automaton realizing a given LTL specification. Implementing the translation of an LTL formula to an automaton itself is thus not in the scope of this project.

In Part I, there is a need to compare the main algorithm in [1] to some of the similar algorithms in the field. Outside of the standard Q-learning method which is the basic RL method that most of the other algorithms operate with, an action supervising reactive shield system proposed by Alshiekh et. al [3] is studied. In addition to this, the method proposed by Hasanbeig et. al. in [9] and [4] is also of interest. Lastly, it is interesting to investigate additional methods designed to solve highly specific issues. Therefore, methods such as advice based learning [10] and reward shaping [11] are within the scope of this work.

Furthermore, as the focus lies on developing the method in [1], the exploration of the other implementations is kept to a minimum, as their role is to act as references. With this in mind, implementing the key features of these algorithms is highly relevant. The idea is to regard these algorithms as fully working final products, allowing them to be used for reference and comparison.

When evaluating performance, the most important metrics are considered. For example, although computational complexity is a common measurement to make when evaluating an algorithm, it is not a common way of evaluating reinforcement learning algorithms. Measuring the number of iterations through a learning process, or the agents interactions with the environment before convergence to an optimal policy, is more interesting. Therefore, computational complexity is not used as a measurement, and a slightly more detailed motivation to why this is the fact is provided in later sections.

The research questions in the next section are formulated for both Part I and Part II of this thesis, and are also restricted by certain limitations. They are formulated

to mainly focus on the potential conflicts that may arise between the specification, environment and additional performance goals in constrained reinforcement learning. The purpose of this is to highlight how the performance of the reinforcement learning method is affected when introducing temporal logic constraints.

1.4 Research Questions

This project aims to answer the following research questions.

1. By imposing LTL constraints on an already existing reinforcement learning problem, how can potential conflicts that may arise within the combined reinforcement learning cost function be handled, in order to ensure a mutually beneficial relationship between the LTL constraints and the original optimization problem?

2. Most often, a model prediction may be enhanced by observations gathered from simulation, measurements or information gathered elsewhere. What are the possible improvements on estimation that can be made by assuming partial knowledge of a large system, in terms of additional knowledge of the MDP, and how can they be integrated into the basic method for temporal logic constrained reinforcement learning?

3. How does the temporal logic constrained reinforcement learning perform in terms of constraint violation, optimality, and computational efficiency compared to classical Q-learning and other constrained reinforcement learning methods, where safety is imposed in other ways? Are there solutions that are more efficient for certain problems?

1.5 Contributions

The following contributions are made in Part I and Part II of this project:

- Complex LTL specifications can easily be formulated as automata. However, for specifications of relatively low complexity the rewards or punishments supplied by the accepting and forbidden states run the risk of being too sparse. This implies that even with high exploration and an extensive number of episodes, rewards will not be handed out until the automaton reaches a marked state. This is unlikely to happen even in one episode during the course of learning, making it impossible for the agent to find a correct path through the MDP. The solution to this is the accepting frontier function, described in Section 4.2.2. The method uses the concept of colored automata states to hand out intermediate rewards that effectively guide the agent along an MDP trace that does not break the LTL specification. Experiments showing this are found in Sections 7.3 and 7.7.

- The problem of sparse rewards described above can be induced in any problem by scaling up the state space dimensionality of the MDP. In this case, even

the accepting frontier cannot lower the exploration time needed to find a path that satisfies the LTL formula of a problem. It is very unlikely that complex LTL specifications can be realised in these settings, solely due to the size of the MDP. A solution to this is to use the potential function, and the success of this function is independent on distance between MDP states. This type of function is a reward shaping technique designed to reward the learning agent for going in the direction of one or more goal states, and the effects of this can be observed in the experiment conducted in Sections 7.7 and 7.8.

- When introducing different functions such as potential and the accepting frontier, tuning is required between the reward sources. This is due to the potential functions inability to grasp dangerous states, and it will override the punishments given by the marked and forbidden states of the automaton or the recommendations from the accepting frontier. These conclusions can be drawn from many of the experiments, but a particularly sensitive situation is found in Section 7.4.

- Theoretically correct automata implementations of LTL formulae do not necessarily work in the practical context of temporal logic constrained tabular Q-learning. In Section 7.5 it is shown that for a liveness specification, the mappings between rewards, states and actions done in the Q-function will effectively be overwritten due to that the automaton always returns to the same state after parts of the specification has been fulfilled. A solution to this is shown in Section 7.6 where a slight modification to the problematic automaton from the failed experiment in Section 7.5 is made.

- The last experiment of Part I, found in Section 7.8, finds that the potential function can be used on the first few episodes of a large scale MDP sequential problem and after this be turned off to let an accepting frontier extended TL constrained RL method take over. Essentially, the potential function is used as a boost rocket for the agent, which then uses a method designed for small scale problems to converge to a policy that satisfies the LTL formula.

- In Part II, the main contribution is the evaluation of the modular analysis method with which the specific problem of joint path planning and admission control is solved. This method consists of first formulating a continuous time Markov decision process that can be split into two independent processes, and then deriving control policies for each of these sub-processes both analytically and through reinforcement learning. The procedure is described in its entirety in Chapter 13, and solved for specific numerical parameters in Chapter 14.

- For the sub-process that can be modelled as a queue, two successful methods to find threshold type solutions to the admission control problem are provided in Section 13.2.1 and Section 13.2.2.

1.6 Structure

The first part of this thesis explores the concept of temporal logic constrained reinforcement learning, while the second part focuses on the concept of modular analysis.

Part I

The first part of this thesis consists of Chapter 2 through Chapter 8, in which temporal logic constrained reinforcement learning is described and evaluated, starting with basic concepts.

Reinforcement learning

In the second chapter, the theory behind reinforcement learning is covered, as this is one of the corner stones of this work. From a basic definition of discrete event systems, topics such as

- Discrete time Markov decision processes
- Dynamic programming
- Value function iteration
- Q-learning

are explained.

Temporal logic specifications

Temporal logic specifications is the second par of the foundation for this thesis. This concept is presented on the basis of

- Formal languages
- Automata classes
- Automata that realise LTL formulae

and is covered in the third chapter.

Algorithms

Starting with the previous research in the field of temporal logic constrained reinforcement learning, the fourth chapter discusses three algorithms in particular:

- Temporal Logic Constrained Reinforcement Learning
- LDBA Constrained Reinforcement Learning
- Shielded Reinforcement Learning

Additionally, a discussion on the key elements of these algorithms is conducted, resulting in additional methods with which path planning problems can be solved. Three new composite algorithms are proposed:

- Online executed LTL constrained RL
- LTL constrained RL with detached accepting frontier
- LTL constrained RL with detached accepting frontier and potential based rewards

Evaluation of proposed algorithms

Starting with chapter five, the three main categories of path planning problems for reinforcement learning are presented. These are

- Safe navigation to destination
- Sequential state visits
- Liveness and fairness

Furthermore, suitable techniques for measuring the performance of the proposed algorithms are presented here.

The algorithms are evaluated through experiments on specific problems from the three path planning problem categories. The sixth chapter describes the implementation of the small research platform necessary for conducting these experiments in the first part of this project.

The seventh chapter describes the seven experiments conducted in Part I of this thesis. Through these, conclusions regarding the three proposed algorithms can be made. Among these, the most important ones are that

- The detached accepting frontier is necessary to induce initial specification compliance even in small scale systems.
- Potential methods can be used to solve problems with large state spaces.
- Potential can be used for initial exploration guidance when this information is expensive.

The conclusions are presented in the eight chapter of this thesis. Here, notes on the technical implementation and experiments, ideas for future work, and the answers to the research questions that Part I provides are also found.

Part II

The second part of the project focuses on modular analysis as a method to solve temporal logic constrained reinforcement learning problems in systems that have multiple distinct sub-processes. Specifically, the type of modular processes considered are systems consisting of a grid world and a queue.

Continuous Markov chains

The basic process that lays the foundation of Markovian queueing systems, the Poisson distribution, is introduced in the first chapter of the second part of this thesis. As this is a process in which discrete events occur at random points in continuous time, it is also necessary to introduce the concept of Generalised semi Markov processes (GSMP), in which the event timings can be described technically.

Chapter 10 starts with the definition of continuous Markov chains, and aims to highlight the differences between these and the discrete Markov chains used in the first part of this project. As reinforcement learning still requires discrete Markov decision process descriptions, an important method called *uniformisation* is introduced to discretise continuous time Markov processes. In the final part of Chapter 10, the proposed structure of joint Markov processes is formulated, along with a method for discretising those.

Queueing theory

Chapter 12 focuses entirely on queueing theory. It starts with some fundamental concepts and notation, before moving on to describe two Markovian queueing systems, the $M/M/1$ and $M/M/1/K$ queues.

The two types of queues that are discussed are continuous Markov processes. As reinforcement learning is done on discrete processes, uniformisation is described. This operation produces a discretised version of a continuous Markov process.

The chapter is finished with a few sections that explain how cost functions can be expressed for continuous and uniformised Markov processes, and how these can be used to find analytical solutions to control problems. In the end, the specific threshold type solutions are considered.

Path planning and admission control

Chapter 13 formulates the main control problem of this part, which is solving a jointly controlled path planning and admission control problem. This problem is difficult to solve with reinforcement learning as the state space is infinitely large.

Modular analysis implies deriving control policies for separate parts of a joint system, such that reinforcement learning can be used to find the policy on a finite representation of the state space. The solution is presented in several stages:

- Developing a threshold type solution to an admission control problem for $M/M/1$ queues.
- Identifying the threshold using an analytical method.
- Identifying the threshold using simulation.
- Using reinforcement learning to find the solution to the temporal logic constrained path planning problem in the joint process.

Evaluation of modular analysis method

In Chapter 14, the experiments of Part II are formulated and conducted. These are organised in four categories:

- Simulating unlimited queueing systems.
- Simulating limited queueing systems.
- Cost estimation for admission control problems. Here, a parameter that limits the infinite joint state space is found.
- Joint path planning and admission control on the joint system.

In Chapter 15, the conclusions concerning the second part of this project are presented. These are formulated in two parts, which are:

- Evaluating the method of modular analysis.
- Evaluating the implementation and conducted experiments.

After this, answers are provided for the two selected research questions that are studied in the second part of the project. In the final conclusions of this thesis, some notes on the entirety of the project are given, along with suggestions for future work.

Part I

Temporal Logic Constrained Reinforcement Learning

2
Reinforcement Learning

To describe the methods that are studied and used in this project, there is a need to explain concepts ranging from the basics of reinforcement learning to regular languages, automata and temporal logic. This chapter focuses on the first, and the structure aims to help the reader build a strong fundamental understanding of the mechanisms that work behind the fundamental reinforcement learning algorithm used in this work, called Q-learning. This is crucial to understand the problems and methods investigated in the succeeding experiments and conclusions of this work.

2.1 Discrete Event Systems

A system can, according to [12], in its fundamental meaning be intuitively defined as the concept of different entities exchanging information with each other, through some function. To describe the behaviour of a system, a system model may be constructed, describing the change of a set of parameters. Some of these may be influenced by an input and some may be measured to form an output.

When modeling dynamical systems describing for instance position and velocity of a particle, it is natural to assume that all system states are in some way dependent on time. They are continuous-time state systems, and their state transition mechanisms are time-driven. However, there are also systems where the state transition mechanism is driven by the occurrence of discrete-time events, instantaneously taking the system from one state to another, for example the pressing of a button that changes the light bulb state from off to lit.

Therefore, a practical way to view discrete event systems is a discrete state space, event driven system where the transitions between states are dependent on occurrences of asynchronous discrete events over time. Based on this, the specific discrete event system class called Markov decision processes can be defined.

2.2 Markov Decision Processes

A Markov decision process (MDP) is a system model of a discrete-time stochastic process. It can be viewed as a transition system where a probability distribution over state transitions is defined in each state, and this non-determinism is very useful for modelling an environment that may behave in an unpredictable way. A fundamental definition of an MDP $\mathcal{M}$, following in the style of [2], is

$$\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, P, p_{\text{init}}, AP, \lambda \rangle \tag{2.1}$$

where

- $\mathcal{S}$ is a finite set of discrete states.
- $\mathcal{A}$ is a finite action set such that $\mathcal{A}(s) \subseteq \mathcal{A}$ is the subset containing the actions that can be performed from the state $s \in \mathcal{S}$.
- P is a mapping over the probabilities of transitioning between states, such that $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ describes the probability to transition from state s to state s' via action a as $P(s, a, s')$.
- p_{init} describes the probability distribution that determines the starting point of the process such that $p_{\text{init}} : \mathcal{S} \rightarrow [0, 1]$. Naturally, the probability of *any* of the states being the initial state is always one.
- AP is the set of state labels, and here they are specifically called atomic propositions.
- λ is the mapping that connects each state to the atomic propositions as $\lambda : \mathcal{S} \rightarrow AP \cup \varnothing$. An important assumption embedded in this notation is that at most one element of AP can be mapped to each state. Another way of expressing this is that at most one of all possible atomic propositions can hold true in each MDP state. This is common in the grid world type MDP applications considered in this work, and is thus a special case of the general setting where more than one label from AP can be present in each state. In the general case, this relationship is then instead denoted $\lambda : \mathcal{S} \rightarrow 2^{AP}$, but this is again not the case in this work.

Furthermore, as P is a probability function it is important to note that $\sum_{s' \in \mathcal{S}} P(s, a, s') = 1$ which intuitively says that the probability of performing any transition possible from a state s is one. As mentioned before, the probability of any state being the initial state is given by $\sum_{s \in \mathcal{S}} p_{\text{init}}(s) = 1$.

However, for reinforcement learning applications the MDP description can vary. One way of modifying the definition is to consider the initial state as a deterministic entity, making the probabilistic notation p_{init} redundant. This convention is used in [9] and [4], and may be formally defined by exchanging the distribution p_{init} to a specific initial state s_{init} in the MDP definition of (2.1). It should also be noted here that even s_{init} is a somewhat redundant notation; it is often exchanged for s_0. In this work, the practical ordering of MDP states into grids is used which technically implies that each state is associated with a coordinate. This means that if each state has an index i, each i can be mapped to coordinates (x_i, y_i). To avoid confusion, when s_0 is used to denote the initial state, it does not necessarily mean that the state with coordinates $(0, 0)$ is the initial state.

In [3], another way of describing an MDP is used. Rewards are important in reinforcement learning, and they can be included in the MDP description. Formally, the environment reward convention is described by including the element $\rho : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}$ in the tuple in (2.1). The function maps a transition from $P(s, a, s')$ to a reward, received when completing the transition. Furthermore, in [3] the atomic propositions and state labeling functions are disregarded.

Lastly, a third modification to the MDP is considered. This description is used in [1], and it includes both the concept of atomic propositions with corresponding state labeling function, and that of rewards. Formally, the tuple in (2.1) is extended to $\mathcal{M} = \{\mathcal{S}, \mathcal{A}, P, s_0, AP, \lambda, \rho\}$.

With these descriptions, an MDP environment model may be modified to see fit. However, the core concept of Markov decision processes remains consistent throughout the different modifications, and is what makes the MDP a good environment representation in reinforcement learning. This concept is called the Markov property.

2.2.1 The Markov property

The Markov property, here described as in [13], is interesting as it addresses the problem of finding the probability of the environment being in a specific state. In the context of reinforcement learning, the probability of a discrete process receiving rewards r in state s at discrete time point $k+1$ depends on all previous states, rewards and actions. The joint probability distribution for the reward and state at $k+1$ is then said to be conditioned on all these different factors, and it is given by

$$Pr\{s(k+1) = s', r(k+1) = r | s(0), a(0), r(1), \ldots, r(k), s(k), a(k)\} \tag{2.2}$$

where general probability is denoted by Pr. However, if the Markov property holds, all information needed to express the next state and reward is encapsulated in the expression for the state, action and reward at the previous discrete time point k. The joint probability is then said to be conditioned on the available information at k only, such that the probability of the next state assuming some specific value s' and the next reward being r is given by

$$Pr\{s(k+1) = s', r(k+1) = r | s(k) = s, a(k) = a\} \tag{2.3}$$

The information about the process is thus propagated through the iterations of k and can be accessed without considering all previous states. To dive further into the statistical background of the Markov processes, the concept of determinism and probabilism is explained in the next section.

2.2.2 Deterministic and probabilistic Markov processes

The relationship between the states $\mathcal{S}$ and the set of actions $\mathcal{A}$ in the MDP $\mathcal{M}$ can be both deterministic and probabilistic. To understand this, consider a controller responsible for selecting the actions performed in the process. This setting can be illustrated as in [14] by the following example.

A robot is navigating through a discrete grid world. The environment is described by the MDP $\mathcal{M}_D = \langle \mathcal{S}, \mathcal{A}, P_D, s_{init}, \rho \rangle$ where $\mathcal{S} = \{s_0 \ldots s_8\}$ and the action space is $\mathcal{A} = \{N, E, S, W\}$ representing the four cardinal coordinates. The initial state is $s_{init} = s_4$, the center.

By assuming a transition probability function P_D defined as $P_D(s, a, s') \rightarrow [0, 1]$ that assumes values between 0 and 1, the action a needs to be defined in order to get the probability of transitioning from s to s'. This action is somehow determined

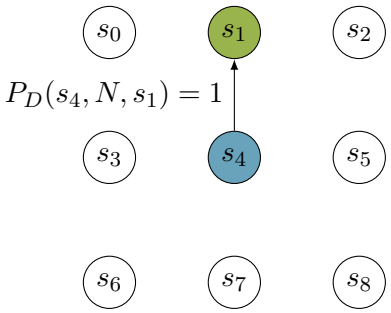


Figure 2.1: A deterministic transition from s_4 to s_1 via the action N.

by the controller, but often motivated by maximising the total number of rewards for transitioning between states, given by ρ. Having defined s and a, the probability of transitioning to *any* other state in the state space $\mathcal{S}$ is now given by P_D.

For deterministic Markov decision processes such as the robot in Fig 2.1, the probability of transitioning between states by an action is always exactly one or zero. Therefore, it is common to express deterministic Markov processes using transition functions denoted T instead of probability functions denoted P. However, as this work focuses on probabilistic Markov processes only, the probabilistic notation is used. In the example, if the robot is in state s_4 and the controller selects action N, the probability function P_D gives the values

$$P_D(s_4, N, s') = \begin{cases} 0, & s' = \{s_0, s_2, s_3, s_4, s_5, s_6, s_7, s_8\} \\ 1, & s' = s_1 \end{cases} \tag{2.4}$$

and

$$\sum_{s' \in \mathcal{S}} P_D(s_4, N, s') = 1 \tag{2.5}$$

Both the action selection and the resulting transition are deterministic in that there is a 100 percent probability of transitioning to another specific state. If the robot is in s_4 and the action selected is N, the robot will definitely move north to state s_1 at all times, according to the deterministic probabilities in P_D.

Now imagine the same setting, except that oil has been spilled on the floor, making it slippery. As the robot might slip and move unexpectedly, this must be included in the model. The function P is a probabilistic transition function, and the environment MDP is now described by the tuple $\mathcal{M}_P = \langle \mathcal{S}, \mathcal{A}, P, s_{init}, \rho \rangle$. In the same manner as in the deterministic setting, the robot is in its initial state $s_{init} = s_4$ when the action $a = N$ is chosen by the controller. Since the floor is slippery, the actual transition resulting from this is now determined by P as

$$P(s_4, N, s') = \begin{cases} 0, & s' = \{s_0, s_2, s_4, s_6, s_8\} \\ 0.1, & s' = s_3 \\ 0.1, & s' = s_7 \\ 0.3, & s' = s_5 \\ 0.5, & s' = s_1 \end{cases} \tag{2.6}$$

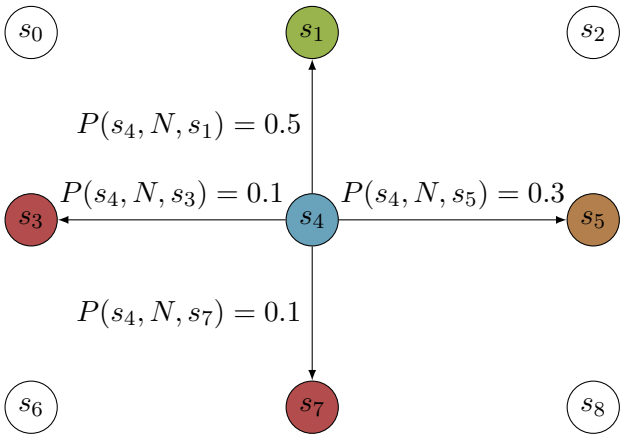


Figure 2.2: Probabilistic transitions resulting from selecting the action N.

and

$$\begin{aligned}\sum_{s' \in \mathcal{S}} P(s_4, N, s') &= P(s_4, N, s_3) + P(s_4, N, s_7) + P(s_4, N, s_5) + P(s_4, N, s_1) \\ &= 0.2 + 0.1 + 0.3 + 0.5 = 1\end{aligned} \tag{2.7}$$

Although the controller selected the northern action, the robot may slip and instead go west, south or east with the corresponding probabilities 0.1, 0.1 and 0.3, visualized in Fig 2.2. Only with probability 0.5 will the robot actually go in the northern direction. A probabilistic MDP describes the probability of transitioning from one state to another, *after* the action has been independently selected according to some external policy. The probabilistic property does not change the probability of selecting an action or what actions are available in a specific state, but it affects the model response to those actions.

With this setting explained, the concept of reinforcement learning can be described in the next section.

2.3 Reinforcement Learning Fundamentals

The principle behind reinforcement learning, as described in [13], may be defined as learning by repeated interaction with the environment and reinforcing an observed and desired behavior. In practice, the procedure can be summarized as developing a mapping from environment states and agent actions to a quantifier in order to classify each action available in a state as desired or undesired.

Reinforcement learning is considered alongside so called supervised and unsupervised learning as a separate type of machine learning. In comparison to supervised learning, in which improvements to an existing hypothesis are made if the output of the hypothesised model differs from that of the actual environment, RL does not go through labeled data sets and its learning is thus independent of any predefined answers, or "supervision". Even so, RL is not a form of unsupervised learning either, as unsupervised learning is associated with pattern finding, whereas RL is merely trying to maximise the collected total reward to achieve its goal.

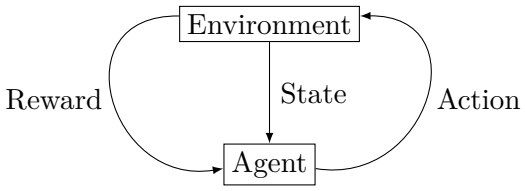


Figure 2.3: The interaction based reinforcement learning principle.

As implied before, an environment and an agent (equivalent to the previously used controller term) are the two central entities in RL, visualized in Fig 2.3. The reward, the value function and the policy are also considered main characters of the reinforcement learning setup. Although the environment is assumed to behave as an MDP, an explicit model of the environment is not always used in RL. If implemented, a model can help predict the future states of the environment based on observations of the current state, which is fitting in situations where the environment behaves according to some discretised physical dynamic process, and examples of this can be found in [13]. However, in this work, the model free variant of reinforcement learning is used.

The principle mechanism of RL, further described in the style of [13], is thus that the agent interacts with the environment, and through the structure of the MDP it receives a reward dependent on the transitions that are made. These rewards effectively serve as a guide for the agent, and immediate rewards are used to achieve a higher goal. For example, if the navigating robot is ultimately supposed to move from state s_a to s_b then rewards might be handed out for transitioning between s_a and some intermediate state s_c and then for the transition between s_c and s_b. The goal is to maximise the total reward, but this cannot be done by only considering the immediate reward.

The value function puts the concept of rewards into the context of traces through the MDP, as it maps an environment transition to a value that is not only dependent on the immediate reward signal, but also on the expectation of future rewards. For example, if the robot would get a large reward for going to a state in a neighborhood of states that all imply large negative rewards, the value function would take the risk of receiving many negative rewards into account when evaluating the transition to the positive reward state, and this state may ultimately be abandoned in order to avoid negative rewards in the future.

When the transitions have been valued, a policy can be derived. This will in part depend on the value of the transitions, but not only, as a problem in reinforcement learning is that a certain degree of exploration is needed to find out if there exists a better strategy to achieve the goal. This is in [13] described as one of the main challenges of reinforcement learning, and often implies finding a balance between exploration and exploitation.

To describe reinforcement learning technically, the starting point is to define what has already been touched upon; considering future rewards.

2.3.1 Return and discount

As mentioned before, a value function will not only take immediate reward into account, but also potential future rewards. Therefore, as done in [13], it is convenient to define the return r at discrete time points as

$$g = r(k+1) + \gamma r(k+2) + \gamma^2 r(k+3) + \cdots = \sum_{i=0}^{\infty} \gamma^i r(k+i+1) \tag{2.8}$$

where γ denotes the discount factor which is a weighting parameter that changes how far from k individual rewards are taken into consideration in the return function. The return g at point i is thus a weighted sum of all the future rewards. Using this, the optimisation target in reinforcement learning can be described as maximising the return.

2.3.2 Connection to dynamic programming and optimal value functions

Dynamic programming, originally proposed in [15] and efficiently summarised in [16], builds upon Bellmans principle of optimality which states that any part of an optimal process is also optimal. Its name comes from the application to computer programming and that it is suitable for dynamic processes, as in the following optimal control based description obtained from [16]. The next state of a discrete time system can be described by

$$s' = f(s, a) \tag{2.9}$$

where f is a function describing the system dynamics, s is the state at time k and a is the control signal at time k. A cost function for the complete process is given as

$$J(s) = S(s(K)) + V(s, a) + V(s', a') + V(s'', a'') \ldots \tag{2.10}$$

where the states and actions over a time horizon are evaluated in terms of value. Here, S denotes the cost at the final state when $k = K$ and V is the value function. If the optimal control, state and cost a^*, s^* and J^* are evaluated, the principle of optimality can be used to write the optimal cost function at stage k as

$$J^*(s) = \min_a \Big[V(s, a) \Big] + J^{*\prime}(s^{*\prime}) \tag{2.11}$$

Thus, the solution to a discrete time optimisation problem can be formulated as a recursive process. This principle is also exploited in reinforcement learning to find the optimal values of two specific functions, concerning the values of the states and the values of the actions. The optimal cost function is not necessarily deterministic, and can in the stochastic case be formulated as maximising the optimal *expected value* of the stochastic cost function.

2.3.3 Optimality in state and value functions

To use the principles of dynamic programming in a reinforcement learning setting, following in the style of [13], the starting point is to consider a function that assigns

a value to the MDP states, according to a control policy π. As dynamic programming is open for stochastic cost functions, the probabilities must now be taken into consideration, which is why the expected value of the return at a given state is considered. Let the state value at discrete time point k according to the policy π be defined as

$$V_\pi(s) = \mathbb{E}_\pi[g|s] = \mathbb{E}_\pi\Big[\sum_{i=0}^{\infty} \gamma^i r(k+i+1)|s\Big] \tag{2.12}$$

This function is thus the expected value of the return that was previously described in (2.8). Continuing in this fashion, the decision to take a specific action a in a specific state s according to π at a discrete time point k may be valued according to

$$Q(s,a) = \mathbb{E}_\pi[g|s,a] = \mathbb{E}_\pi\Big[\sum_{i=0}^{\infty} \gamma^k r(k+i+1)|s,a\Big] \tag{2.13}$$

The Bellman principle of optimality can now be used to express the state value function as the actions that maximise the action value function. This follows the same principle as (2.11), but now the notation regards MDP states s and actions a. Therefore, instead of formulating the optimality in discrete time steps, the Markov property is used to express the probability of the specific next state, which is now denoted s'. Accordingly, the next action is now denoted a'. The optimum of the value functions for the state and action selection described in (2.12) and (2.13) are now given in terms of the MDP as

$$\begin{aligned} Q^*(s,a) &= \max_\pi Q(s,a) = \sum_{s'\in\mathcal{S}} P(s,a,s')\Big[r' + \gamma \max_{a'} Q^*(s',a')\Big] \\ V^*_\pi(s) &= \max_\pi V_\pi(s) = \max_{a\in\mathcal{A}(s)} \sum_{s'\in\mathcal{S}} P(s,a,s')\Big[r' + \gamma V^*_\pi(s')\Big] \end{aligned} \tag{2.14}$$

The final expressions for the optimal state and action values in (2.14) are called the *Bellman optimality equations* for the two functions. Here, P is the MDP element that denotes the probability of transitioning to state s' from state s via action a. For illustrative purposes, the notation includes the specific reward value r' that is obtained by completing the transition, but the expression remains equivalent to the one used in the definition of the MDP.

2.3.4 Policy iteration

Previously, it has been discussed how the Bellman principle of optimality is used in the context of an MDP and how the Markov property is utilized to produce an equivalent function for a probabilistic MDP for one specific policy. Now, if two different policies π and π' are given, and their state value functions can be compared such that $V_{\pi'}(s) \geq V_\pi(s)$ in all states $s \in \mathcal{S}$, then $V_{\pi'}(s)$ is at least as good or better then than $V_\pi(s)$. One example of this is the *greedy* policy described as in [13], and given by

$$\pi'(s) = \arg\max_{a\in\mathcal{A}(s)} \sum_{s'\in\mathcal{S}} P(s,a,s')\Big[r' + \gamma V_\pi(s')\Big] \tag{2.15}$$

for which the value function is $V_{\pi'}(s)$. Since the better policy is derived from the old one, this procedure is called *policy improvement*. Evaluation and improvement

can be done consecutively until a point where $V_{\pi'}(s) = V_{\pi}(s)$ which implies that

$$V_{\pi'}(s) = \max_{a \in \mathcal{A}(s)} \sum_{s' \in \mathcal{S}} P(s, a, s') \Big[r' + \gamma V_{\pi'}(s') \Big] \tag{2.16}$$

Comparing this expression to the Bellman optimality equation (2.14), the equivalence between them becomes clear. Therefore, as $V_{\pi'} = V^*$, π' must be the optimal policy. The procedure of iteratively deriving a better policy until convergence is called *policy iteration*.

2.3.5 Value iteration

According to [13], computing the policy evaluations in the policy iteration procedure includes repeated evaluations of the whole state set. Furthermore, the convergence to the optimal value occurs only in the limit, and it may be a very beneficial trade off between computation time and accuracy to consider finish close to convergence instead of in the exact limit. *Value iteration* may thus be expressed as a combination of policy improvement and a shortened policy evaluation as

$$\begin{aligned} V_{i+1}(s) &= \max_{a \in \mathcal{A}(s)} \mathbb{E}[r' + \gamma V_i(s') | s, a] \\ &= \max_{a \in \mathcal{A}(s)} \sum_{s' \in \mathcal{S}} P(s, a, s') \Big[r' + \gamma V_i(s') \Big] \end{aligned} \tag{2.17}$$

for all states $s \in \mathcal{S}$. Here, i does not denote discrete time, but rather a general iterative index indicating that $V(s)$ is a value that is recursively improved upon. This convenient iteration is guaranteed to converge to the optimal value function, and plays a central role in a fundamental type of reinforcement learning called Q-learning.

2.4 Q-learning

Q-learning is an algorithm that produces an estimate of Q, the function that values the selection of actions in states, given previously in (2.13), hence its name [13]. The estimate is denoted $\widehat{Q}_k(s, a)$ and follows the principles of value iteration until it converges to the optimal action value function $Q^*(s, a)$ from the Bellman optimality equation (2.14). The fundamental version of Q-learning originally proposed in [17] is a model free reinforcement learning algorithm, but there are also ways of incorporating a model of the environment in the process.

2.4.1 Model free Q-learning

The formal definition of model free Q-learning follows in the style of [1], and is the type of Q-learning that is used in this work.

Consider a basic MDP $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, P, s_{init}, AP, \lambda, \rho \rangle$. The extended value function $Q : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ can be seen as a mapping from states and actions to a value and the probability expressed in the value iteration described in (2.17) is now given by the MDP transition probability map $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$, while the rewards

are given by the mapping $\rho : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$. Under the assumption that $Q^*(s, a)$ describes the action value at the optimum, the Bellman equation for optimality of the action value function in (2.14) writes as

$$Q^*(s, a) = \sum_{s' \in \mathcal{S}} P(s, a, s')[\rho(s, a, s') + \gamma \max_{a' \in \mathcal{A}(s')} Q^*(s', a')] \tag{2.18}$$

Note here that r' is replaced with the MDP element $\rho(s, a, s')$. Furthermore, given that this denotes the function at the optimum, the optimal policy is naturally given by

$$\pi(s) = \arg\max_{a \in \mathcal{A}} Q^*(s, a) \tag{2.19}$$

To further motivate that the Q-function approximate can be expressed by the principles of value iteration, consider that the estimated sample mean of a random variable X is expressed as $\hat{m}_k = \sum_{i=1}^{k} x_i/k$ where x_i are samples of the variable. As explained in [1], weighted refinements of the estimate can be expressed recursively in terms of an increasing set of samples as

$$\hat{m}_{k+1} = \hat{m}_k + \alpha(x_{k+1} - \hat{m}_k) \tag{2.20}$$

where α is the weighting. Now, let the estimated Q-function $\widehat{Q}(s, a)$ replace $\hat{m}$. Then recall that the evaluation of an action is based on both the immediate reward and the return, and replace the new sample x_{k+1} with the sum of the immediate reward and the γ weighted previous estimation of the Q-value at the next state and next action, which is $\rho(s, a, s') + \gamma \max_{a' \in \mathcal{A}(s')} \widehat{Q}_k(s', a')$. Then

$$\widehat{Q}_{k+1}(s, a) = \widehat{Q}_k(s, a) + \alpha[\rho(s, a, s') + \gamma \max_{a' \in \mathcal{A}(s')} \widehat{Q}_k(s', a') - \widehat{Q}_k(s, a)] \tag{2.21}$$

becomes the expression for the recursive Q estimate, as described in [1]. As a reminder, $0 < \gamma < 1$ is called the discount factor, and it determines how far away Q-values shall influence a new approximation. This result coincides with the original formulation of the Q-learning iteration in [17], where it is derived in a similar manner.

Finally, note that in reinforcement learning, it is well known that the Q-function is an estimate that is continuously developed in the learning process. Because of this, the estimate notation of $\widehat{Q}$ is often expressed as a plain Q.

2.4.2 Action selection

While the optimal action taken in a certain state s is said to be given by the optimal policy in (2.19), little has been said about how to select actions when developing it. As mentioned in Section 2.3, the Q-function estimate develops by the agent taking the actions with the largest Q-values, but there is also a need to keep exploring different paths through an MDP process to find different, possibly better, paths to maximise the total reward. Therefore, two concepts of action selection are considered.

As explained in [18], selecting the action a with the largest Q-value in state s is called greedy action selection. To introduce exploration, consider choosing a random action with probability ϵ from a uniform probability distribution over the

set of available actions. With probability $1 - \epsilon$, the action is selected greedily. Formally, if m actions exist to choose from, this can be expressed in terms of a probability distribution as

$$\pi(a|s) = \begin{cases} \epsilon/m + 1 - \epsilon & \text{if } a = \underset{a \in \mathcal{A}(s)}{max}\ Q(s,a) \\ \epsilon/m & \text{otherwise} \end{cases} \tag{2.22}$$

Note that π here denotes a probability distribution describing the general ϵ-greedy action selection principle, and does not denote a policy in the traditional sense.

After many iterations, when the approximation of the Q-function starts to converge to the optimum, the need for exploration generally decreases. There are many ways of ensuring that the probability ϵ decreases with the iterations, increasing the probability of selecting an action greedily. For example, exponentially or linearly decaying ϵ functions are common [19].

With the Q-learning fundamentals outlined, the field of temporal logic is described in the next chapter.

2.5 Summary

From a basic definition of discrete event systems, the concept of Markov decision processes utilising the Markov property is considered as the fundamental description of an environment in a reinforcement learning setting. MDPs are said to be either probabilistic or deterministic, which regards transitions between states after an action has been selected.

Reinforcement learning is concluded to build upon the dynamic programming principle and its implementation of the Bellman optimality principle. Basic concepts such as policy and value iteration leads up to the definition of tabular Q-learning, for which model free Q-learning and ϵ-greedy action selection are described.

3
Temporal Logic Specifications

This chapter handles the second cornerstone of the theory behind this project which, next to reinforcement learning, is temporal logic (TL).

3.1 Temporal Logic and Predicate Logic

Predicate logic presented as in [20] is used to state propositions, a form of declarative statements, about variables that can be either true or false. Common operators are negation, implication, conjunction and disjunction. In this work, it suffices to briefly summarise the commonly used propositions in the form of a table with illustrative examples.

Note that the example for conjunction in Table 3.1 can never be true. Such a statement is called a *contradiction*, and in similar manner a statement which is always true is called a *tautology*.

The difference between classical predicate logic and temporal logic as described in [21] is that temporal logic makes it possible to express logical statements in relation to time, which is why it is very useful when planning the behavior of systems. The expressions used in TL are similar to *modal logic*, where modalities such as *possibly* and *necessarily* are used to extend predicate logic statements. For example, the statement "p is possibly true if q is necessarily false" can be formulated in modal logic, while only absolute statements such as "p is true if q is false" can be formulated by the use of predicate logic. In temporal logic, different modalities are used, such as *always* and *eventually*. To elaborate on this, the specific form of temporal logic called linear temporal logic is introduced next.

Table 3.1: Predicate logic propositions with examples.

Proposition	Sign	Example	Read out
True	T	$p = T$	p is true.
False	F	$q = F$	q is false.
Negation	$\neg$	$p = \neg q$	p is not q.
Disjunction	$\vee$	$p \vee \neg p = T$	p or not p is true.
Conjunction	$\wedge$	$p \wedge \neg p = F$	p and not p is false.
Implication	$\rightarrow$	$p \rightarrow q$	If p true then q true.

Table 3.2: LTL modalities with examples.

Modality	Word	Example	Read out
$\bigcirc$	Next	$\varphi = \bigcirc p \wedge \neg \bigcirc q$	Next is p and not q.
$\Diamond$	Eventually	$\varphi = \Diamond(p \vee q)$	Eventually p or q will be true.
$\Box$	Always	$\varphi = \Box p \wedge \Box \neg q$	p is always true and q is never true.
$\bigcup$	Until	$\varphi = p \bigcup q$	p is true until q is true.

3.2 Linear Temporal Logic

Linear temporal logic, which gets its name from the linear time relation that it builds upon, is introduced and formulated in the style of [2]. To begin with, linear discrete time can be defined as a set in which any two elements, which can be regarded as points in time, are related by one of them being larger than the other. The smaller element of the two must then necessarily occur before the larger one; this is called a linear order.

Only a few fundamental modalities need to be explained in order to understand how temporal logic are used in this work, and they are as in [2] compiled in a summarising table. In Table 3.2, some of the more common modalities are presented along with illustrating examples. LTL formulae are further defined using the principle of induction according to the following statements:

- If $p \in AP$, then p is considered an LTL formula.
- If φ_1 and φ_2 are both LTL formulae, then $\neg\varphi_1$, $\varphi_1 \wedge \varphi_2$, $\bigcirc\varphi_1$ and $\varphi_1 \bigcup \varphi_2$ are also LTL formulae.

Moreover, two interesting special cases of LTL formulae arise when modalities are combined. They are $\Box\Diamond\varphi$, meaning "infinitely often φ", and $\Diamond\Box\varphi$ which means "eventually forever φ".

In connection to the MDP setting described in Section 2.2, LTL formulae can be defined for variables such as p and q from the set of atomic propositions AP. Later in this thesis, it is explained how to take advantage of the fact that atomic propositions can be assigned as state labels in the MDP formulation. First, to explain the method in which temporal logic is combined with reinforcement learning and MDPs, there is a need to briefly describe *formal languages*.

3.3 Formal Languages

Formal languages are here used to describe the behavior of certain discrete systems. It is therefore necessary to define the basic properties that make up a language.

3.3.1 Symbols, words, alphabets and languages

In accordance with [22], a symbol is considered the fundamental element, and can be for instance a letter or a digit. A sequence of symbols is called a word or string, and these annotations will be used interchangeably. For symbols and words, the following properties are defined:

- An alphabet, denoted Σ, is a finite set of symbols from which words can be formed.
- The empty word is denoted ϵ.
- The length of a word w is denoted $|w|$ and simply equates to the number of symbols in the word.
- A concatenation of two words w and v is a word defined as the first word followed by the second. Concatenations of a word w with the empty word ϵ, both before and after, results in the same word w.
- The prefix of a word is any number of leading symbols of a word. For example, the prefix of *mad* is ϵ, *m*, *ma*, or *mad*.
- The suffix of a word is any number of trailing symbols. In the *mad* example, the suffixes are ϵ, *d*, *ad* and *mad*.
- A formal language $\mathcal{L}$ is a subset of possible words that can be derived from an alphabet. For example, if the alphabet is $\Sigma = \{a, b\}$ then a language, in this case denoted $\mathcal{L} \subseteq \Sigma^*$ [1], is a set $\mathcal{L} = \{\epsilon, a, ab, b, ba, aab, bba, \dots\}$. The empty set $\varnothing$ and the set with only the empty string ϵ are also considered languages.

3.3.2 ω-languages

Extending the definition of languages derived from finite words, languages consisting of infinitely long words are also possible. This concept is here described as in [2].

Infinite length words are never ending sequences of symbols from a finite alphabet, and the notation Σ^ω describes the set of all infinite words that can be constructed using Σ. Subsets of Σ^ω are called ω-languages, and concepts such as union and concatenation that in Section 3.3.1 were used for finite sequences, apply here with infinite repetition.

To formally express the infinite repetition in the context of a language, define a language $\mathcal{L} \subseteq \Sigma^*$. The ω-language $\mathcal{L}^\omega$ is then a subset of words $\mathcal{L}^\omega \subseteq \Sigma^\omega$ formed by an infinite number of concatenations of finite words gathered from Σ, so that

$$\mathcal{L}^\omega = \{w_1 w_2 w_3 \dots | w_i \in \mathcal{L}, i \geq 1\} \tag{3.1}$$

The concept of ω-languages is central to this work, as it is used as an extension to the relation between infinite fulfillment of a specification and its automata representation, which is described next.

[1] The $*$ operator used here is the *Kleene star* [22], which since Σ is a set of symbols implies the set of *all* words over the symbols in Σ.

3.4 Automata

The basic *finite automaton (FA)*, sometimes *deterministic finite automata (DFA)* with the non-stochastic property emphasized, is a discrete model of a system according to [22]. It is, in a similar tuple fashion as the MDP, formally described by the tuple

$$\mathcal{A} = \langle Q, \Sigma, \delta, q_0, Q_m \rangle \tag{3.2}$$

where Q is a finite set of states, Σ is a finite input alphabet, $q_0 \in Q$ is the initial state and $Q_m \subseteq Q$ is the set of marked or accepting states. Sometimes, such as in [21], a set of forbidden states $Q_x \subseteq Q$ is also included. The transition function mapping $\delta : Q \times \Sigma \to Q$ describes the transitions between states via a configuration of symbols from the input alphabet, such that $\delta(q, \sigma)$ returns a state for each $q \in Q$ and $\sigma \in \Sigma$.

A language can be defined as in [22] for the automaton input alphabet Σ in the way it is described in Section 3.3.1. The sequence of inputs from the alphabet makes up a word, and since the inputs result in transitions between automaton states, the different words formed by the input alphabet can be assigned different properties depending on which states their associated sequences of transitions visit. A word w is accepted by the automaton if $\delta(q_0, w) = q_m$ for some $q_m \in Q_m$, and by association the accepted or marked language $\mathcal{L}_m$ of an automaton is the set $\{w|\delta(q_0, w) \in Q_m\}$, meaning the set of words for which the sequence of transitions finishes in a marked state. Analogously, the forbidden language can also be defined if there are forbidden states in the automaton definition.

Fig 3.1 describes an example automaton with the state set $Q = \{q_0, q_1, q_2, q_3\}$, alphabet $\Sigma = \{a, b, c, d\}$. State q_0 is the initial state, state q_2 is forbidden while state q_3 is marked, meaning the marked language is $\mathcal{L}_m = \{ad\}$ and the forbidden language is $\mathcal{L}_x = \{b, c\}$.

3.4.1 Coloured deterministic finite state automata

Marked and forbidden states of an automaton make it possible to express marked and forbidden languages in terms of input words. Sometimes, it is also necessary to characterise these words in terms of which automaton states are visited when performing the sequence of actions that the word implies, without neccessarily being associated with marked and forbidden states.

To categorize states, an addition to the automaton tuple definition can be made.

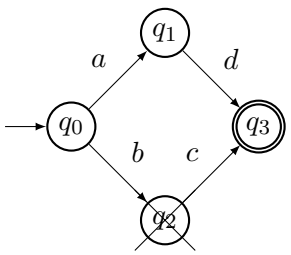


Figure 3.1: DFA with the marked state q_3 and the forbidden state q_2.

This can according to [23] be viewed as a generalization of the deterministic finite state automata class, and is called *coloured* deterministic finite state automata. This type of automaton is here formally defined as

$$\mathcal{A} = \langle Q, \Sigma, \delta, q_0, Q_m, C, \chi \rangle \tag{3.3}$$

where the function $\chi : Q \to C$ maps a state $q \in Q$ to a colour in C such that $\chi(q)$ is the color of state q. Using this notation, states can be categorized in a convenient way.

Specifically, the concept of coloured states is in this work used to label automaton states in scenarios when state categories such as marked and forbidden are insufficient. In figures describing automata, the colour of an automaton state is visualised by an actual colour, and different colours thus indicate different labels. The default color of a state is white, which is also reflected in the figures.

3.5 Translating LTL Formulae to Automata

Similar to the connection between finite languages and finite automata, described in Section 3.4, there is a connection between the infinite languages and another form of automata.

3.5.1 The key idea

The procedure of translating an LTL formula to an automaton is best explained by an example from [1]. Consider the LTL formula

$$\varphi = \Diamond p \wedge \Box \neg q \tag{3.4}$$

This is interpreted as eventually, p occurs while q must never be true. In other words, the negation of q must be repeated forever according to the formula. As described in Section 3.2, LTL formulae such as (3.4) can be defined for a set of atomic propositions, AP. For the example above, the associated set is $AP = \{p, q\}$, and it is assumed that there is a way for the automaton to observe the elements of this set and their value, which can be either true or false.

In practice, the key step is now to construct the automaton that realises (3.4) by building an automaton alphabet of temporal logic statements from the elements of AP. In the example above, a corresponding automaton is the one in Fig 3.2. It can in Fig 3.2 be observed how words can be constructed by making transitions in the automaton. For example, p and q can both be false until some point where p is true, and a transition is made to the marked state. If q becomes true in any case, a transition is made to the forbidden state, which is undesired, and this behavior agrees with (3.4). This must then hold infinitely often.

Hence, LTL formulae can be realised as languages of infinite words over the alphabet 2^{AP}. This comes from the definition of $(2^{AP})^\omega$ being the set of words that is the result of infinite concatenation of words in 2^{AP} [2]. To summarise the principle behind the procedure in theory, an LTL property can, as described in [1], be defined by the *infinite* sequences $\sigma = \sigma(0)\sigma(1)\sigma(2)\ldots$ that can be formed from the alphabet

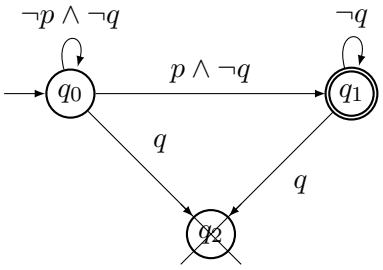


Figure 3.2: Automaton from [1] realising $\varphi = \Diamond p \wedge \Box \neg q$.

Σ. These sequences are part of the set denoted Σ^ω, and the symbols $\sigma(i) \in \Sigma$ where $\Sigma = (2^{AP})$. Thus, LTL properties are fulfilled by ω-languages based on an alphabet of atomic propositions. If LTL properties can be realised by an ω-language, there are also automata structures that can describe these languages. In this way, an automaton can be defined to represent an LTL-property.

However, the specific automata form required for this has not yet been discussed. As the automaton in the example of Fig 3.2, this class of automata is called *Büchi automata*.

3.6 Büchi Automata

Finite automata require a finite number of transitions to reach an accepting state. The words formed from the input language are finite, and all possible words that imply reaching the marked state make up a finite marked language. To express an infinite marked ω-language for an automata, this automata needs an accepting condition suitable for infinite words, according to the definition in [2].

The formal difference between an automaton that realises a finite marked language and one that realises a marked ω-language is that for the automaton with a finite marked language, it is enough that the input word leads to an accepting state once. If the marked language is an ω-language, however, the entire infinite input word must be checked so that it always leads to an accepting state. This means that the acceptance condition is in this case defined for infinite runs of the automaton, according to [2]. There are several different automata for which the acceptance condition is to visit a state an infinite number of times, but the types that are used in this work are variants of the Büchi automaton.

3.6.1 Nondeterministic Büchi automata

The non-deterministic Büchi automaton (NBA) is a general form of the Büchi automaton, and is by following the definition in [2] described by the tuple
$\mathcal{B} = \langle Q, \Sigma, \delta, Q_0, Q_m \rangle$ where Q is a finite set of states, Σ is the alphabet,
$\delta : Q \times \Sigma \rightarrow 2^Q$ is the transition function mapping, Q_0 is the set of initial states and $Q_m \subseteq Q$ is the set of marked states. Note that the transition function mapping returns a set of states, not a single state, and the initial state condition is also a set of states.

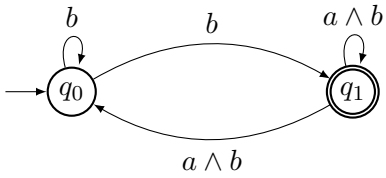


Figure 3.3: NBA from [2] that satisfies the LTL property $\varphi = \Box b \wedge \Box\Diamond a$.

The accepting ω-language of the NBA is given by

$$\mathcal{L}_\omega(\mathcal{B}) = \{\sigma \in \Sigma^\omega | \exists \text{ an accepting run for } \sigma \in \mathcal{B}\} \tag{3.5}$$

where a run is defined as a state sequence resulting from applying an input sequence from the alphabet. An infinite run is thus defined as an infinite sequence of visited states, and the accepting language is then all the infinite input words constructed from the alphabet that produce the infinite runs in which the accepting states appear infinitely often.

The behaviour of a non-deterministic Büchi automata can be illustrated by the specific example in [2] that relates linear temporal logic properties to both non-deterministic and the deterministic special case of the Büchi automata. The non-deterministic Büchi automaton in this example, illustrated in Fig 3.3, is given by $\mathcal{B} = \langle Q, \Sigma, \delta, Q_0, Q_m \rangle$ where $Q = \{q_0, q_1\}$, $\Sigma = 2^{AP}$ contains the expressions a, b and $a \wedge b$ formed from a set $AP = \{a, b\}$. Furthermore, $Q_0 = \{q_0\}$, $Q_m = \{q_1\}$ and the transitions are defined according to Fig 3.3. By inspection, it can be seen that this NBA fulfills the LTL property $\varphi = \Box b \wedge \Box\Diamond a$.

As explained in [2], it can be seen that $\mathcal{B}$ is not deterministic, because no matter what symbol is chosen, properties are fulfilled for transitions to both state; if in q_0 and b is true, it cannot be determined if a transition to q_1 shall be made or not, but the language is still that of the specified LTL formula. This automaton is a Büchi automaton since the runs that contain the marked state, such as the run $q_0 q_1^\omega$, are the result of applying infinitely long words from the accepting language. These runs can be observed to contain the accepting state an infinite number of times.

3.6.2 Deterministic Büchi automata

A Büchi automaton $\mathcal{B} = \langle Q, \Sigma, \delta, Q_0, Q_m \rangle$ is according to [2] deterministic (DBA) if the initial state set has size one, and the transition function only returns one state. The descriptions for the elements in the tuple $\mathcal{B}$ are in all other aspects the same as in the description for nondeterministic Büchi automata.

The determinism property is illustrated by the same example from [2] that is used to describe the nondeterministic Büchi automaton in Section 3.6.1. Consider the automaton $\mathcal{B}$ where $Q = \{q_0, q_1\}$, $\Sigma = 2^{AP}$ with $AP = \{a, b\}$, $Q_0 = \{q_0\}$, $Q_m = \{q_1\}$ and the transitions that are defined according to Fig 3.4. Just as the non-deterministic Büchi automaton described in Fig 3.3, the automaton describes the property $\varphi = \Box b \wedge \Box\Diamond a$.

The DBA in Fig 3.4 thus fulfills the same LTL properties as the NBA in Fig 3.3, but it is deterministic since the transition mapping from source state and action to

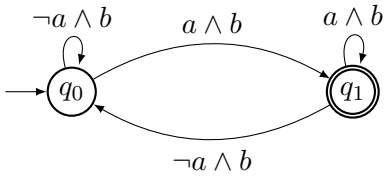


Figure 3.4: DBA from [2] that satisfies the LTL property $\varphi = \Box b \wedge \Box\Diamond a$.

a destination state only returns one destination state, and not several. For example, there are no actions available from the q_0 state that imply staying in q_0 and transitioning to q_1 at the same time, which was the case in the automaton of Fig 3.3.

A very important property is that NBAs are more general than DBAs, meaning that for every ω-language that can be represented by a DBA, there exists an NBA that represents the same language, but the converse is not true. There exist ω-languages that can be represented by an NBA, but not by any DBA; this can be proven by contradiction.

3.6.3 Some languages have an NBA representation, but not a DBA representation

The following proof is outlined from the more comprehensive version found in [2]. Consider the ω-language given by $(a+b)^*b^\omega$.

A corresponding NBA can be seen in Fig 3.5. For the word $\sigma = wb^\omega$ where w is some word from $\{a, b\}^*$, the NBA has the options to either stay in state q_0 or "guess" when the b-suffix of w starts (at the point where $w = aaaabbb\ldots$ switches from a to b repetitions) and thereby move to the accepting state q_1; this behavior is non-deterministic and can not be represented by a DBA.

As explained in the proof of Theorem 4.50 of [2], start by assuming that the language of a DBA is formed by the same expression such that
$\mathcal{L}_\omega(\mathcal{B}) = \mathcal{L}_\omega((a+b)^*b^\omega)$ for a DBA $\mathcal{B} = \langle Q, \Sigma, \delta, q_0, Q_m\rangle$. Given the deterministic property, the transition function $\delta^* : Q \times \Sigma^* \to Q$ is a mapping to one state and not to a set of states.

The word $\sigma_1 = b^\omega$ is in $\mathcal{L}_\omega(\mathcal{B})$, implying the existence of an accepting state $q_1 \in Q_m$ and $n_1 > 1$, such that
$\delta^*(q_0, b^{n_1}) = q_1 \in Q_m$. Similarly, the word $\sigma_2 = b^{n_1}ab^\omega$ is also in $\mathcal{L}_\omega(\mathcal{B})$, which results in a corresponding accepting state q_2 and natural number $n_2 > 1$ so that
$\delta^*(q_0, b^{n_1}ab^{n_2}) = q_2 \in Q_m$. The same conditions hold for the word $\sigma_3 = b^{n_1}ab^{n_2}ab^\omega$ and corresponding $q_3 \in Q_m$ and $n_3 > 1$. Iterating this relation until i and gathering

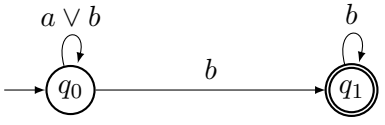


Figure 3.5: NBA from [2] representing the language of $(a+b)^*b^\omega$.

the terms results in

$$\begin{aligned}
\delta^*(q_0, b^{n_1}) &= q_1 \in Q_m \\
\delta^*(q_0, b^{n_1}ab^{n_2}) &= q_2 \in Q_m \\
\delta^*(q_0, b^{n_1}ab^{n_2}ab^{n_3}) &= q_3 \in Q_m \\
&\vdots \\
\delta^*(q_0, b^{n_1}ab^{n_2}a \dots b^{n_{i-1}}ab^{n_i}) &= q_i \in Q_m
\end{aligned} \tag{3.6}$$

As the DBA has a finite number of states, there must exist an $i < j$ such that

$$\delta^*(q_0 b^{n_1} a \dots ab^{n_i}) = \delta^*(q_0, b^{n_1}a \dots ab^{n_i} \dots ab^{n_j}) \tag{3.7}$$

implying that $\mathcal{B}$ accepts the sequence

$$b^{n_1}a \dots ab^{n_i}(ab^{n_{i+1}}a \dots ab^{n_j})^\omega \tag{3.8}$$

Since this implies an infinite repetition of a, this is a contradiction, because this feature is not described by the language $\mathcal{L}_\omega(\mathcal{B}) = \mathcal{L}_\omega((a+b)^*b^\omega)$, according to the conclusion of the proof in [2].

3.6.4 Generalized Büchi automata

The generalized Büchi automata is a modified version of the NBA, but the difference lies in the acceptance condition which states that the automaton has to visit several sets Q_m infinitely often, in accordance with the definition from [2]. The definition of a GNBA is thus the same as for NBA, but the set of accepting states Q_m is replaced with $\mathcal{Q}_m$, a set consisting of a finite number of accepting state sets $Q_m^1 \dots Q_m^k \subseteq Q$. Formally, a GNBA $\mathcal{G}$ is given by

$$\mathcal{G} = \langle Q, \Sigma, \delta, Q_0, \mathcal{Q}_m \rangle \tag{3.9}$$

where $\mathcal{Q}_m$ is a subset of 2^Q. Another note that can be made is that while a non-deterministic Büchi has been shown to be a generalization of the deterministic Büchi, a deterministic Büchi with several acceptance sets can also be defined as a special case of the non-determinstic and more general GNBA, making it a GBA.

3.6.5 Limit deterministic Büchi automata

The last class of Büchi automata that is important in this work is the limit deterministic Büchi automata (LDBA). This type is in turn a modification of the generalized Büchi automata class. Originally proposed by Sickert et al in [24], but explained in the context of reinforcement learning in [9] and [4], the GBA $\mathcal{G}$ is limit deterministic if the state set Q can be split into two sets $Q = Q_N \cup Q_D$ (where $\cup$ is the set union operator) such that

- $\delta(q, \sigma) \subset Q_D$ and $|\delta(q, \sigma)| = 1$ for every state $q \in Q_D$ and for every $\sigma \in \Sigma$.
- $Q_m^j \subset Q_D$ for every acceptance set $Q_m^j \in \mathcal{Q}_m$.

Now, sufficient theory has been covered to present algorithms that uses these concepts in reinforcement learning.

3.7 Summary

This chapter elaborates on the principles of temporal logic and starts with a distinction between that and predicate logic. Linear temporal logic is specifically defined as the central form of logic used in this work.

Moreover, formal languages and specifically infinite so-called ω-languages are defined as a fundamental concept to understand the connection between LTL-specifications and automata realisations. After this, the most fundamental type of automata is introduced, followed by the key principle between translating an LTL formula to an automaton.

For ω-languages, automata with infinite accepting conditions are required, and therefore so-called Büchi-automata are introduced. Within this group, a distinction between deterministic and non-deterministic Büchi automata is made, followed by some useful variations to these specific forms of automata.

4
Algorithms and Methods

In this chapter, results from previous research in the field of temporal logic constrained reinforcement learning is presented and thoroughly analysed. The results come in the form of three algorithms, which are presented alongside additional methods for handling large scale systems. After this, three new composite algorithms are proposed on the basis of the previous research.

4.1 Algorithm 1: Temporal Logic Constrained Reinforcement Learning

All of the algorithms try to derive an optimal policy for an unknown MDP, whilst ensuring that the optimal policy follows a specification. The specification is in all cases formulated in a formal language based on temporal logic expressions, and all algorithms use tabular Q-learning as the underlying reinforcement learning method. However, there are variations in the details of these algorithm, and this work is focused on investigating those differences in order to evaluate the potential performance differences, advantages and disadvantages with the methods.

The first algorithm is presented according to the formulation in [1], and it hereinafter referred to as the TL constrained RL algorithm. The idea behind the algorithm is to consider an MDP and a Büchi automaton that represents a temporal logic specification, and then formally compute a *Büchi weighted product MDP* on which Q-learning can be performed such that marked states in the Büchi automaton are rewarded. In this way, the learning agent is not only taught the control policy that is optimal, but also the LTL behavior specified by the Büchi automaton.

4.1.1 Büchi weighted product MDP

Though it is shown in Section 3.6.3 that there are languages that can only be represented by NBAs, it is stated in [1] that practically usable LTL formulae are most often possible to express with DBAs. Therefore, for an LTL specification φ, consider the DBA $\mathcal{B}_\varphi = \langle Q, \Sigma, \delta, q_0, Q_m \rangle$ with a language $\mathcal{L}_\omega(\mathcal{B}_\varphi)$ that fulfills φ. Next, let $\mathcal{M}$ be the MDP given by $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, P, s_0, AP, \lambda, \rho \rangle$. The product between $\mathcal{M}$ and $\mathcal{B}_\varphi$ is then defined as

$$\mathcal{M} \otimes \mathcal{B}_\varphi = \langle \mathcal{S}_\otimes, \mathcal{A}, P_\otimes, s_{0,\otimes}, AP_\otimes, \lambda_\otimes, \rho_\otimes \rangle \tag{4.1}$$

where the following holds:

- The combined state space is formed by a cross product between the two state spaces, such that $\mathcal{S}_\otimes = \mathcal{S} \times Q$.
- The initial state is $s_{0,\otimes} = (s_0, q_0)$.
- The combined transition probability from the combined state (s, q) to (s', q') is
$$P_\otimes((s,q),a,(s',q')) = \begin{cases} P(s,a,s') & q' = \delta(q, \lambda(s')) \\ 0 & \text{otherwise} \end{cases}$$
- The set of atomic propositions of the product is $AP_\otimes = \{M, F\}$, representing marked and forbidden product states. These are the only state labels in the product, as the more complex restrictions based on the state labels of $\mathcal{M}$ are handled by the DBA before the synchronization.
- The state labeling function for the product is $\lambda_\otimes : \mathcal{S} \times Q \to AP_\otimes$ and it maps a labelled product state to either a marked or a forbidden atomic propositional label.
- The reward for the product is given by
$$\rho_\otimes(s,q) = \rho(s) + \begin{cases} \rho_M > 0 & \lambda_\otimes(s,q) = \{M\} \\ \rho_F < 0 & \lambda_\otimes(s,q) = \{F\} \\ 0 & \text{otherwise} \end{cases}$$
but it should be noted that the environment rewards $\rho(s)$ are often disregarded in practical purposes, and in this work it is only used in classical Q-learning and never with additional reward methods.

The product is well understood by the graphical example in [1]. The LTL formula $\varphi = \Diamond p \wedge \Box \neg q$, which is familiar from Section 3.5.1, can be realized by a DBA $\mathcal{B}_\varphi$, and computing the product between this DBA and a simple MDP $\mathcal{M}$ results in the product $\mathcal{M} \otimes \mathcal{B}_\varphi$.

The product system can be seen in Fig 4.1, along with $\mathcal{M}$ and $\mathcal{B}_\varphi$ individually. The MDP state labels are the atomic propositions p and q and the formula φ is formulated using those atomic propositions, but in the product the atomic propositions are just M and F (represented in the product by a double circle and a dashed

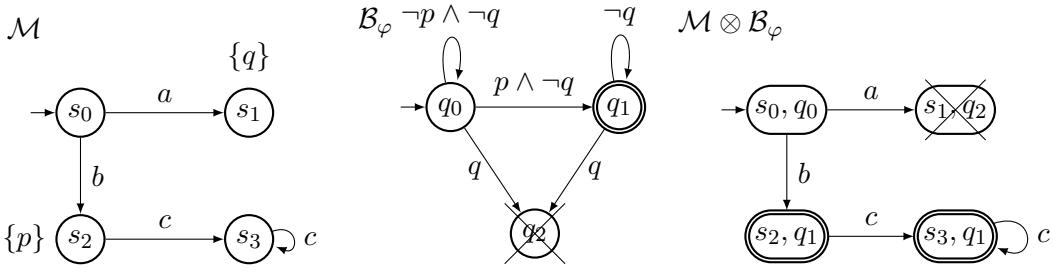


Figure 4.1: Visualized example from [1] of the product $\mathcal{M} \otimes \mathcal{B}_\varphi$ between an MDP $\mathcal{M}$ and a Büchi automaton $\mathcal{B}_\varphi$.

circle, respectively) depending on whether or not the DBA state is in a marked or forbidden state.

4.1.2 Online execution equivalence

Since the probability of transitioning in the product is either the probability of transitioning in the MDP or zero, and since the total reward is given as a sum of the MDP reward and the Büchi reward, the transition process of the MDP is independent on that of the Büchi. Transitioning in the product can be divided into first transitioning in the MDP, and then making a transition in the Büchi depending on the new state label in the MDP. Thus, computing transitions sequentially for the MDP and the Büchi is equivalent to transitioning in the product, and this is referred to as an *online execution* of the Büchi.

4.1.3 Algorithm presentation

Finally, an algorithm that describes the Q-learning procedure for the Büchi weighted product is presented. The underlying algorithm is the tabular Q-learning, explained in Section 2.4.1, but it is here extended by a dimension to also incorporate the automaton states. The algorithm is shown in Fig 4.2 and shows the full procedure of the reinforcement learning loop. The method proposed in [1] has been shown to guarantee *safety* and *liveness* for a model free Q-learning problem.

Result: Optimal policy $\pi^*(a|s)$ that satisfies the LTL formula φ.
Initialise
MDP $\mathcal{M}$,
Büchi automaton $\mathcal{B}_\varphi$,
Q table as in 2.4.1 but for the *product* state and actions,
episode number $= 0$,
iteration number $= 0$.
while *Q is not converged* **do**
  episode number++
  while $q \in \mathcal{B}_\varphi$ *is not forbidden and iteration number* $<$ *iteration threshold* **do**
    Choose MDP action $a \in \mathcal{A}(s)$ ϵ-greedily.
    Move to s' by a in the MDP.
    Read atomic proposition labels in s'.
    Move to q' in the Büchi according to the atomic propositions.
    Recieve the total reward $\rho_\otimes(s', q')$.
    $Q(s,q,a) \leftarrow Q(s,q,a) + \alpha[\rho_\otimes(s',q') + \gamma \max_{a' \in \mathcal{A}(s')} Q(s,q',a') - Q(s,q,a)]$
  end
  Update ϵ and *learning rate* α as function of *episode number* if desired.
end

Figure 4.2: Temporal logic constrained reinforcement learning.

4.2 Algorithm 2: LDBA Constrained Reinforcement Learning

This method, presented in [9] and [4], is similar to the previous algorithm but uses the limit deterministic Büchi automata class to assert that the LTL property holds. In the original paper, it is proven that the algorithm finds an optimal policy that maximises the probability of satisfying the LTL constraints if an optimal solution exists. If it is not possible to satisfy the given LTL formula, the control policy that is produced is still "reasonable".

4.2.1 Generalized Büchi automata application and ϵ-transitions

The environment is in this method, as expressed in [9], defined as an MDP tuple $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, P, s_0, AP, \lambda \rangle$, and note that there is no reward associated with $\mathcal{M}$. The goal is again to ensure that when using the derived policy, the sequence of states from s_0 and forward (often called a path or trace through $\mathcal{M}$) satisfies the LTL formula φ. This is done by reading the atomic propositions in the MDP state labels and evaluating the Büchi automata associated with the LTL formula.

As the Büchi automata $\mathcal{N} = \langle Q, \Sigma, \delta, Q_0, \mathcal{Q}_m \rangle$ is limit deterministic, the state set can be divided into two disjoint sets, Q_N and Q_D and the following definitions, all gathered from [9], of the sets are made:

- Q_D is the accepting set, and it is invariant which implies that the automaton is unable to escape from it when it is reached.
- Q_N is considered the initial set.
- Both Q_D and Q_N are deterministic.
- Between Q_D and Q_N there are non-deterministic so-called ϵ-transitions. These transitions are spontaneous, and do not require the reading of any atomic propositions.
- A non-accepting sink component is a subset $O \subset Q$ which induces a strongly connected directed graph and does not include all accepting sets in $\mathcal{Q}_m$. Furthermore, the sink component is such that there is no other set $O' \subset Q, O' \neq Q$ such that $O \subset O'$.

The ϵ-transition reflects a “guess” on reaching the invariant acceptance set. If a label cannot be read after such a transition, this means that the guess was wrong and the trace does not satisfy the LTL property.

4.2.2 Accepting frontier

The acceptance condition can be represented as several sets of accepting states. It may be noted that this acceptance condition can also be defined for a generalized

Büchi automaton, as the important feature is the sets of accepting states and not the non-determinism. In both cases, a way to make sure the agent visits all these sets infinitely often is needed. This method is defined in both [9] and [4] as the *accepting frontier function* $Acc : Q \times 2^Q \to 2^Q$, and reads

$$Acc(q, \mathbb{Q}) = \begin{cases} \mathbb{Q} \setminus Q_m^j & \text{if } q \in Q_m^j \wedge \mathbb{Q} \neq Q_m^j \\ \bigcup\limits_{k=1:f} Q_m^k \setminus Q_m^j & \text{if } q \in Q_m^j \wedge \mathbb{Q} = Q_m^j \\ \mathbb{Q} & \text{otherwise} \end{cases} \tag{4.2}$$

This function operates on the accepting frontier set $\mathbb{Q}$. This set is initialized as $\mathbb{Q} = \bigcup\limits_{k=1:n} Q_m^k$ where $Q_m^1, Q_m^2, \ldots, Q_m^k, \ldots, Q_m^n \in \mathcal{Q}_m$, the union of all sets of accepting states in the LDBA. When a new state is reached in the LDBA, the accepting frontier is updated using (4.2) as $\mathbb{Q} \leftarrow Acc(q', \mathbb{Q})$. This function evaluates if q' is in one of the sets $Q_m^k \in \mathcal{Q}_m$; if it is, this set is removed from the accepting frontier. Now, this set does not need to be visited again. This procedure continues until all sets $Q_m^k \in \mathcal{Q}_m$ but one have been visited and $\mathbb{Q} = Q_m^n$ where n denotes the index of the last set in $\mathbb{Q}$. After this, the accepting frontier is re-initialized to the union of all accepting state sets as before. In this way, the algorithm makes sure that all accepting conditions of the LTL formula are fulfilled.

To distinguish frontier sets in the automata of this work, the concept of coloured states from Section 3.4.1 is used. A specific frontier set is thus given a specific color, and this is also highlighted in figures, where all colors except white denote frontier sets.

4.2.3 Product MDP and reward function

As in the DBA constrained RL algorithm [1], a product between the MDP $\mathcal{M} = \{\mathcal{S}, \mathcal{A}, P, s_0, AP, \lambda\}$ and the LDBA $\mathcal{N} = \{Q, \Sigma, \delta, Q_0, \mathcal{Q}_m\}$ is computed in [4] and [9] as

$$\mathcal{M} \otimes \mathcal{N} = \langle \mathcal{S}_\otimes, \mathcal{A}, P_\otimes, s_{0,\otimes}, AP_\otimes, \lambda_\otimes \rangle \tag{4.3}$$

where $\mathcal{S}_\otimes = \mathcal{S} \times Q$, $s_{0,\otimes} = (s_0, q_0)$, $AP_\otimes = Q$, $\lambda_\otimes = \mathcal{S} \times Q \to 2^Q$ so that $\lambda_\otimes(s, q) = q$. The joint transition probabilities are given by $P_\otimes : \mathcal{S}_\otimes \times \mathcal{A} \times \mathcal{S}_\otimes \to [0, 1]$, implying that $P_\otimes((s, q), a, (s', q')) = P(s, a, s')$. The accepting condition is to visit all combined states in which the automaton state is in one of the marked state defined in $\mathcal{Q}_m$ of the LDBA $\mathcal{N}$.

In addition to this, there are two modifications that need to coexist with the above definition of the product MDP:

- The ϵ-transitions to LDBA state q are handled by an action ϵ_q that is added to $\mathcal{A}_\otimes = \mathcal{A} \cup \{\epsilon_q, q \in Q\}$. In the algorithm, both $\mathcal{A}$ and $\mathcal{A}_\otimes$ are used, so this formulation does not replace the defined actions of the product MDP.

- Probabilities to travel via the ϵ-transitions are given by
$$P_\otimes((s, q), \epsilon_q, (s', q')) = \begin{cases} 1 & \text{if } s' = s,\ q' \xrightarrow{\epsilon_q} q \\ 0 & \text{otherwise} \end{cases}$$

The reward function is furthermore defined using the accepting frontier function as

$$\rho(s^{\otimes}, a) = \begin{cases} r^{+} & \text{if } q' \in \mathbb{Q},\ s^{\otimes'} = (s', q') \\ 0 & \text{otherwise} \end{cases} \tag{4.4}$$

where r^{+} and 0 denote positive neutral rewards.

4.2.4 Algorithm presentation

By comparison to how the LDBA constrained reinforcement learning algorithm is presented in the original paper [9], the additional algorithm steps that calculate the probability of LTL property satisfaction have here been removed as they are not, in practice, central to this work. Furthermore, as the equivalence between direct computation of the product MDP and online execution of the Büchi holds in this case as well, the algorithm is formulated in terms of the product in order to keep the notations concise. The algorithm for LDBA constrained reinforcement learning is shown in Fig 4.3.

Next, a third algorithm that considers the concept of shielded reinforcement learning is studied.

Result: Optimal policy $\pi^*(a|s)$ that satisfies the LTL formula φ
Initialise
MDP $\mathcal{M}$,
LDBA $\mathcal{N}_{\varphi}$,
Q table $Q : \mathcal{S}_{\otimes} \times \mathcal{A}_{\otimes} \to \mathbb{R}_0^+$,
Accepting frontier set $\mathbb{Q}$,
episode number $= 0$.
iteration number $= 0$.
while *Q is not converged* **do**
  episode number ++.
  $s_{\otimes} = (s_0, q_0)$.
  while $q \notin$ *sink* $: s_{\otimes} = (s, q)$ *and iteration number* $<$ *iteration threshold* **do**
    Choose MDP action a as $\underset{a \in \mathcal{A}}{argmax}\ Q(s_{\otimes}, a)$, or ϵ-greedily.
    Receive the reward $\rho_{\otimes}(s_{\otimes}, a)$.
    Update the accepting frontier function as $\mathbb{Q} \leftarrow Acc(q', \mathbb{Q})$.
    $Q(s_{\otimes}, a) \leftarrow Q(s_{\otimes}, a) + \alpha[\rho_{\otimes}(s_{\otimes}, a) + \gamma \max_{a'} Q(s'_{\otimes}, a') - Q(s_{\otimes}, a)]$.
    $s_{\otimes} = s'_{\otimes}$.
  end
  Update ϵ and *learning rate* α as function of *episode number* if desired.
end

Figure 4.3: LDBA constrained reinforcement learning.

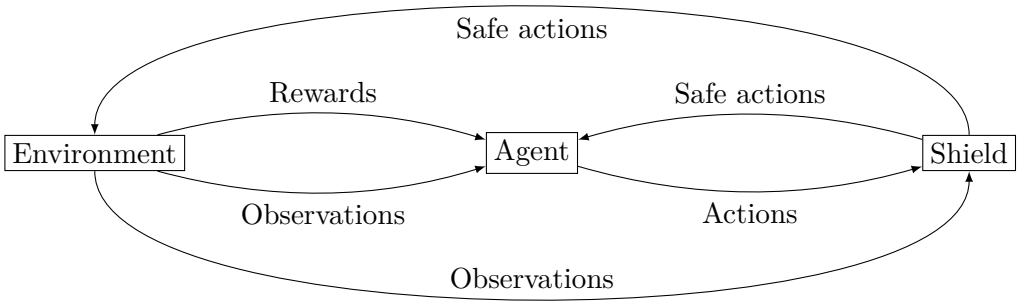


Figure 4.4: The shielding principle from [3].

4.3 Algorithm 3: Shielded Reinforcement Learning

The concept of a shield, proposed in [3], can easily be explained visually. A principle sketch is therefore provided in Fig 4.4. The idea is to have a second system, called a shield, that observes and analyses the actions selected by the reinforcement learning agent. When an action is considered unsafe, the shield suggests another action that is better.

4.3.1 Principles of shield synthesis

In [3], the shield is described as a *finite state reactive system*, which has an input and output alphabet, and is technically defined as a *Mealy machine*. However, the system operates in a similar way as the Büchi automata in the previous algorithms. The system satisfies a given LTL formula by accepting an ω-language, and the objective is that the LTL property shall be satisfied for infinite traces through the environment.

It is stated in [3] that the algorithm achieves safe reinforcement learning by considering the safety specification formulated in LTL and expressed as a "safety word automaton" $\varphi^s = \langle Q, \Sigma, \delta, q_0, Q_m \rangle$. The goal is achieved by solving a safety game constructed from φ^s and an abstraction of the environment. The environment behaves as an MDP $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, P, s_0, \rho \rangle$, and the abstraction $\varphi^{\mathcal{M}}$ can be described as the original MDP, with behaviour restricted by the safety word automaton φ^s. The game is played by the actual environment and the system, and in every step the environment and the system takes turn selecting input actions and outputs. If only safe states are visited, then the safety game is won.

An example of an MDP abstraction is discussed in [3], in which a water heater serves as the observed system, which behaves as an MDP. The energy consumed by the heater depends on the water level in the tank. The volume of the tank is known, and it is also known that there is an inflow and outflow of water, where the volume per time unit capacity is known for both the in- and outflow. With this knowledge only, a safety specification can be formulated for keeping the water level within the limits of running dry and overflowing; this is the abstraction of the water tank.

In [3], a shield is synthesized by transforming the safety specification φ^s and the abstraction $\varphi^{\mathcal{M}}$ into a safety game. After this, a winning region in which only safe states can be reached is computed, and then both the game and the winning region are translated into the reactive system that constitutes the shield.

4.3.2 Shielded reinforcement learning

While a more detailed description of the shield construction can be found in [3], the operation of the shield is quite straight forward. In each step of the reinforcement learning process, the action a_1 is selected by the agent. This action is forwarded to the shield, and if a_1 is unsafe according to the specification, it is substituted for the safe action $a \neq a_1$, and the environment executes a instead. It moves to the state s', and the agent receives the reward r' which is used to update the Q-function for the safe action.

There are two possible ways of using the obtained reward when an action a_1 is substituted for the safe action a. The first way is to punish the original selection of the unsafe state; this is done by assigning the punishment r' to a_1. The second way is to assign a reward r' to the same unsafe action a, which may seem counterproductive. However, while it does imply that unsafe actions are part of the final policy, the unsafe actions are in this way always mapped to a safe action. Hence, the behavior of mapping an unsafe action to a safe one is rewarded in the end.

The advantage of using any of the mentioned reward strategies is that the safety specification φ^s is never violated in practice, as unsafe actions are never executed in the environment but still punished in the Q-learning. The main downside is that the shield must in both cases still be active when the learning phase is done and the policy shall be executed.

A second subject that is discussed in [3] is the concept of action ranking. The purpose of this is for the system to be less restrictive to the learning algorithm. The idea is that in each step, the agent performs a ranking of some available actions, such that $rank = \{a_1, a_1 \dots a_j\}$. The action with rank one is prioritised according to some known information, such as a desired general direction through a maze. The ranking does not need to be performed on all available actions; the agent selects the highest prioritised action in the ranked set, and only chooses actions outside of the ranked set if all of the ranked actions are unsafe according to the specification.

With the three main algorithms described, two additional methods are discussed below. They are not considered to be separate algorithms, as they target very specific problems in general reinforcement learning. Rather, they are treated as helper functions that may offer solutions to certain problems that occur in specific learning situations.

4.4 Reward Shaping

Reward shaping, as described in [11], is an additional method affecting the behavior of the agent in a reinforcement learning problem by giving out additional rewards in a strategic manner to meet a criterion. In [11], the method is mainly motivated by the problem of teaching an agent to perform a sequence of tasks, for example to

visit certain states of an MDP in a specific order. It is shown that this can be done by the use of a potential function.

4.4.1 Potential based reward shaping function

According to the definition in [11], the reward shaping function has the structure $F : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ for an MDP. This is also valid for the labeled MDP type $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, P, s_0, AP, \lambda, \rho \rangle$ considered in this thesis. By selecting the potential reward function as

$$F(s, a, s') = \gamma\Phi(s') - \Phi(s) \tag{4.5}$$

the value of selecting an action in a particular state s and then transitioning to another state s' can be valued in terms of the next state and the current state. Here, γ is the discount factor of the reinforcement learning algorithm, and Φ can be seen as a "gravitational pull" (or any other type of physical potential, such as voltage).

In the experiments constructed on gridworld MDPs in [11], a *Manhattan distance* to a desired goal state is considered as a basis for the potential function. The Manhattan distance between two points s_1 and s_2 in a cartesian coordinate frame is given by

$$\text{Manhattan}(x_1, y_1, x_2, y_2) = |x_1 - x_2| + |y_1 - y_2| \tag{4.6}$$

In [11], a probabilistic MDP environment is considered. In each state, the probability of going in the intended direction upon selecting an action a is 80% while the probability of going in any other direction is 20%. Therefore, it is expected that the agent on average will perform 0.8 steps in the Manhattan direction per RL algorithm time step, so the weight 0.8 is assigned to the Manhattan function to accommodate for the probabilistic behavior of the MDP.

As a positive reward shall be handed out for a step in the direction towards the goal state s_{goal}, $\Phi(s)$ can be selected as $-\text{Manhattan}(s, s_{goal})/0.8$ and then

$$F(s, a, s') = \gamma\Phi(s') - \Phi(s) = -\gamma\frac{5}{4}\text{Manhattan}(s', s_{goal}) + \frac{5}{4}\text{Manhattan}(s, s_{goal}) \tag{4.7}$$

If s' is closer to s_{goal} than s is, it is easy to see that $F(s, a, s')$ will provide a positive reward.

4.4.2 Applications

As mentioned previously, this method is in [11] applied to problems where certain states in an MDP are to be visited in a sequence. This is just one of the possible uses of this function, and the function is in this work mainly used due to its independence from state space dimensionality.

As the potential is formulated as a distance between two points in a grid world MDP, it can be used in scenarios where the state space dimensions make exploration very time consuming. As it is often reasonable that some knowledge of the MDP strucure and where the goal regions are is available, this function can be implemented with limited but sufficient knowledge of where to navigate in an MDP, and particularly in a grid world example. Next, an additional method for handling large state spaces in RL problems is described.

4.5 Advice Based Exploration

Advice based exploration is in [10] motivated by the previously mentioned MDP state space dimensionality problem. It is noted that practical possible applications of reinforcement learning will most often involve problems with very large state spaces, and a method to handle these situations is therefore crucial.

A solution is presented in the form of advice based exploration, and advice is here described as a softer way of manipulating the development of an optimal policy in contrast to a harder constraint specification. In practice, the advice is in [10] implemented on the same basis as the three previously described TL constrained RL algorithms, as LTL specifications formulated as automata are used in the reinforcement learning loop to provide the advice for the agent. An MDP and an automata are used in an online execution similar to the one proposed in [1].

There are two important features that can be extracted from the method described in [10]. Firstly, the situation in which the agent enters a dangerous state is considered. In such a case, the automaton of the LTL formula may in certain cases reach a dead end, which is entirely possible during the learning phase even if the LTL formula forbids it. In practice, reaching this dead end implies that learning cannot proceed in a useful manner as the automaton which is used in the Q-function update is stuck. Therefore, in such situations the automaton is returned to its previous state in order to be able to continue the learning.

The second feature that is worth noting is the so-called background knowledge function. This is described as a function that can be defined manually, arbitrarily, and reasonably with respect to the available information about the MDP. As mentioned in an example in [10], a background knowledge function could be the number of steps required to reach a specific goal state from the current state.

Using the notation of automaton dead ends and the background knowledge function, the advice function central to the work in [10] is split into two properties.

- *Advice guidance* implies in a grid world scenario that actions taken in the direction of a goal state are recommended by the guiding function.

- *Advice warnings* imply that, by in each MDP state defining a subset of actions leading to the previously mentioned automaton dead end, undesirable actions are sorted out.

The key elements to take away from this is thus that it is often reasonable to assume that a distance based function, such as the one described in [10] or a potential function described in [11], can be obtained. Another interesting feature of the method in [10] is the advice warnings, which are given before transitioning to an undesired state. The use of this function suggests that some knowledge of the next state is available before actually making the transition, and that this information can be used to disregard dangerous transitions by one step ahead predictions.

4.6 Comparing and Combining Algorithm Features

In many practical situations, some of the elements in each algorithm described so far are very similar to parts in other algorithms. Therefore, the different algorithms are here viewed as variants of LTL constrained reinforcement learning methods, where each method has a key feature that differentiates it from the others. These key features are used to construct new proposed algorithms that utilise different types of available information.

It is very important to underline that the key features of each method may make the method hard to compare to the others, because it may involve having access to additional information that the others do not have. Therefore, it seems irrelevant to compare the methods in terms of "which one is better", but it is highly relevant to instead compare them in terms of "how does this particular additional information affect learning". In this setting, all comparisons of the methods illustrate how and why they work, and not which one is superior to the other. However, before the key features from each algorithm are described, features that apply to all of the proposed algorithms are covered.

4.6.1 General methods

Before describing and comparing the differences between the three composite algorithms that are experimented with in this work, some general features that apply to them all are considered.

Firstly, as described in Section 4.3, the concept of shielded reinforcement learning implies that actions selected by the agent are monitored by the shield and corrected if the shield determines that it will lead to a dangerous state. For this to be possible, some information about the next state and whether or not this is dangerous is to some degree necessary in the construction of the shield. Based on this, a reasonable assumption to make is that in a certain MDP structure such as a grid world, the next state can be crudely estimated one step ahead. For example, if the action N for north is selected, a reasonable prediction of what the next state will be is the state north of the current state.

Similar to the strategy for avoiding the dead end automaton states described in [10], a shield inspired assumption is made in all proposed algorithms. When an action that leads to a dangerous state is selected, the agent receives the punishment for that action selection. However, in practice, neither the MDP state nor the automaton state is in any algorithm updated to the corresponding dangerous and possible dead end states. This is to avoid getting stuck in bad states and halting the learning procedure. One way to look at this is to imagine that a "border" is set around the bad states, and the agent learns to avoid them using the same knowledge as in traditional reinforcement learning problems, and only actually transitioning to the dangerous states is avoided. Despite not being a very large change to the development of the policy, this procedure is not technically model free as a one step ahead prediction is made; this can be argued to also apply to both the shielding method and the advice learning technique.

4.6.2 Proposed algorithm 1: The LTL constrained RL algorithm

The algorithm described in Section 4.1 performs Q-learning for a product between the MDP and the Büchi automaton representation of the LTL specification, using the online execution. Therefore, the key feature of this algorithm is considered to be the usage of the deterministic Büchi automaton as opposed to the more complicated non-deterministic version. As such, the LTL specification is formulated as an automaton that can have both marked and forbidden states, regulating the reward supplied to the agent, and the transitions are formulated using the atomic propositions of the MDP.

This algorithm is in its simplicity considered to be the baseline reinforcement learning algorithm, where the behavior is specified by an LTL formula. The formula is realized by an automaton, and the type of automaton that is necessary depends on the specification at hand. The only modification to the algorithm proposed in [1] is that the total reward consists solely of the positive and negative rewards that are delivered when the automaton enters a marked or forbidden state in the online execution.

4.6.3 Proposed algorithm 2: LTL constrained RL algorithm with detached accepting frontier

The second reference algorithm, described in Section 4.2, is the algorithm that uses the online execution of the limit deterministic Büchi automaton to fulfil the LTL specification. In the practical cases considered here, the LDBA is found to be equivalent to the DBA. This makes it hard to argue for the difference between algorithms, but there is one more property that the original paper discusses, and that is the accepting frontier function.

The accepting frontier function in the LDBA constrained algorithm in [9] and [4] is defined for generalized Büchi automata, described in Section 3.6.4. The algorithm can handle language specifications requiring the visiting of several sets of marked states infinitely often, and the accepting frontier function is here intended to shift focus to another set of accepting states once an accepting state set has been visited in the automaton. Inspired by this mechanism, the idea is to detach the principle of an accepting frontier function from a specific automata type, and formulate it for simpler automata models.

Consider an automaton $\mathcal{B}_\varphi = \langle Q, \Sigma, \delta, q_0, Q_m, Q_x \rangle$ representing an LTL specification φ. Furthermore, extend the definition of the automaton to incorporate coloured states according to Section 3.4.1, so that the formal definition of the automaton becomes $\mathcal{B}_\varphi = \langle Q, \Sigma, \delta, q_0, Q_m, Q_x, C, \chi \rangle$. In this case, a subset of Q can be defined as the set of states in the automaton that are not necessarily marked, but crucial to visit in order to finally arrive in the marked state or states.

With this setup, an accepting frontier function is now defined similarly to Section 4.2.2. In the second proposed TL constrained RL algorithm of this work, the accepting frontier function is in each episode initialised as a set of coloured states called the frontier states, in the LTL specification automaton $\mathcal{B}_\varphi$. These need to be

visited in order to get to the final state. This frontier function can be considered as a detached observer function that has access to the coloured states and the current state of $\mathcal{B}_\varphi$, which is far from unreasonable as the specification is defined by the user.

The procedure in the algorithm is identical to that of the first proposed algorithm, except that an additional positive reward can be supplied to the agent if it enters a frontier state. When doing so, the frontier state is removed from the frontier set, implying that this type of frontier reward can only be handed out one time per frontier state until all frontier states are visited. Once the frontier state is emptied, the procedure can repeat itself, ensuring a balanced reward distribution to the frontier states.

As it is still necessary to provide both negative rewards when the agent enters forbidden states and positive rewards when it enters a marked state, the frontier reward is considered an additional feature that can be added to the first proposed algorithm in Section 4.6.2.

4.6.4 Proposed algorithm 3: LTL constrained RL algorithm with detached accepting frontier and potential based rewards

The third algorithm draws inspiration from the shielded reinforcement learning described in Section 4.3, the reward shaping potential function strategy in Section 4.4 and the advice based learning of Section 4.5.

Firstly, the shielded reinforcement learning algorithm [3] raises the important question of "do we treat all MDPs equally?". In many practical scenarios, conclusions or predictions can be made about the system that the MDP describes; from this, an abstraction can be made and analysed, even if the procedure after this is not necessarily shield synthesis.

For instance, having knowledge of that a system behaves in a predictive manner, such as a water tank that can be either filled or emptied as in [3], can allow the user to model an abstraction where an action, such as supplying water, in most cases leads to the state of the tank water volume being raised. Similarly, an abstraction for a navigation transition system can be formulated as a grid world, and knowledge of what will probably happen when selecting for instance the north direction. With this information, there are still differences between the abstraction and the actual environment, such as unknown transition probabilities and unknown locations of the forbidden states. With this said, not having reasonable knowledge of these underlying structures is a severe disadvantage compared to having accessed to this additional information.

Secondly, with an underlying MDP structure known, the reward shaping strategies of [25] can be used. In this third proposed algorithm, the potential based reward function from Section 4.4 is implemented as an MDP observer function that has access to the current state that the MDP is in, the next state that a selected action leads to, and a set of states from which a goal position can be formulated. To formulate the goal, two strategies are considered.

- The goal is initially a set of unvisited MDP states, and it is desired to go in the general direction of these. It is not desired to stay in one of these states only, so once reached, the state is removed from the set, much like the accepting frontier set of the LTL automaton, but for the MDP. The goal is now all of the remaining unvisited states, and the potential reward for a transition will be based on the average Manhattan distance to all remaining states in the set. This method is suitable when the exact position of a goal state is unknown, but a general region can be guessed.
- The goal is a list of states that must be visited in sequence. Once a state in the sequence is visited, it is removed from the list, and the next goal becomes the next state in the list. This version is suitable when an exact state sequence must be executed.

The usage of an additional function that provides a potential between the current state and a goal state is very similar to the approach of advice learning, where knowledge about going in the direction of the goal state is provided. The motivation behind implementing the potential based function is that the automaton strategies assume that the agent is able to find an accepting trace, or part of it, through the MDP and only after that receive a reward that reinforces this behavior. However, this assumption might not be fulfilled for certain combinations of MDPs and LTL specifications. In [25] the problem that is illustrated is the sequential state visits problem, and in [10] the problem of large state spaces is the main motivation behind these types of functions. Both these types of problems are investigated further in Section 5.1.

In the practical implementation of the third composite algorithm, the additional potential based reward is added to the method of LTL constrained reinforcement learning with detached accepting frontier function described in Section 4.6.3. As the forbidden states are not known, it is impossible for the potential function to steer the agent away from these.

In cases where the first form of potential is used, the accepting frontier function is also necessary, since upon arrival to a group of states included in the potential function, a specific order or state to visit is not specified in the potential function. For instance, if the specification says that the agent must observe a label p before going to the goal state, and the potential is set to explore a region in which that label is believed to be, only the accepting frontier function would provide incentive to find the exact location of the state in which p is.

4.7 Summary

The chapter describing the core algorithms in this work starts with the introduction of the first algorithm gathered from previous research in the field. It is described in [1] and is here referred to as the temporal logic constrained reinforcement learning algorithm. Within this section, the Büchi weighted product MDP is explained and exemplified, followed by a note on the equivalence between online execution of the product MDP and the Büchi and MDP pair. The first algorithm is then presented in terms of a pseudo-code.

The second algorithm from previous research, originally described in [9] and [4], is then described. It utilizes the concept of a limit deterministic and generalized Büchi automata, and also mentions a so-called accepting frontier function to organize the different acceptance sets in the automaton, both of which are described. The details behind the reward function is then touched upon, followed by a pseudo code description of the algorithm for clarity.

The third algorithm, described in [3] regards the concept of shielded reinforcement learning. Here, descriptions on how a shield is synthesised is provided, followed by a portrayal of the actual algorithm; however, a pseudo code is not necessary for this algorithm as it would not do the core concepts of the algorithm justice.

Next, reward shaping, specifically the Manhattan distance potential function, is described along with a note on its applications. After this, another additional method for handling large scale systems called advice based exploration is illustrated.

The section describing the composite algorithms contains the algorithms that are implemented in this work. They are called the proposed algorithms and take inspiration from the key elements of the algorithms from previous work in the field. The first of the three is the standard LTL constrained algorithm, while the second proposed algorithm is the LTL constrained RL algorithm with a detached accepting frontier function. The last algorithm builds upon the second and further uses the Manhattan potential function to perform guided learning.

5

Problems and Measurement Techniques

This chapter firstly discusses various problems in temporal logic constrained reinforcement learning, in order to evaluate the strengths and weaknesses of the algorithms and methods described in the previous section. Secondly, the fundamental methods for measuring the performance of the algorithms are described.

5.1 Evaluating LTL Constrained Reinforcement Learning

The goal of the first part of this project is to investigate LTL constrained reinforcement learning by answering the research questions in Section 1.4. The questions are focused on the potential problems or conflicts that may arise when trying to solve certain problems that are relevant to many practical reinforcement learning problems. Furthermore, the interesting problem of integrating additional information such as potential into the strategy is considered. Lastly, it is of interest to compare the performance of all three proposed algorithms; how do they compare to classic Q-learning and what are the reasons for the potential similarities or differences?

To answer all of the questions, a collection of scenarios must be formulated. Considering the potential conflicts of interest within the different objective functions of the algorithms, combinations of MDP structures and LTL specifications that can potentially put the algorithms in ambiguous situations are highly interesting. Furthermore, problems where it is expected that the algorithms will perform very differently are also of interest.

To answer the second research question, focus lies on ways of improving the convergence to the optimal solution of the reinforcement learning problem by the use of additional information, such as the potential function, and problems are formulated with this in mind. Lastly, as question three addresses the performance of all the proposed methods, there is a requirement on the type of data that must be collected during the experiments, which is why a thorough motivation for the selected measurements is needed.

In the next section, three types of problem categories are explained, and after that a description of the necessary performance measurements is provided.

5.2 Reinforcement Learning Problem Categories

For all reinforcement learning problem categories, the MDP is a grid world described by the tuple

$$\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, P, s_0, AP, \lambda, \rho \rangle \tag{5.1}$$

This MDP is always probabilistic to some degree. The set of actions is in all cases the four cardinal directions $\mathcal{A} = \{\mathrm{N}, \mathrm{E}, \mathrm{S}, \mathrm{W}\}$, and the initial state s_0 is for simplicity always the origin in the grid world Cartesian coordinate frame. Parameters that differ between the problems are the size of the state space $\mathcal{S}$, the set of atomic propositions AP, the state labeling function λ and the exact values of the probability to transition between states.

5.2.1 Safe navigation to destination

This category concerns the general MDP $\mathcal{M}$ where the states, actions, probabilities, and reward function are described above. The atomic propositions are $AP = \{p, q, r\}$, and the state labeling function maps the label p to the final destination or goal state, q to certain MDP states that are considered dangerous, and finally r to intermediate states that can be visited before the goal state but are not mandatory to visit.

The specification that is enforced has to steer the agent to the goal state, either via an intermediate state or directly towards the goal, and the LTL formula that expresses this is

$$\varphi = \Diamond\Big(p \vee (r \wedge \Diamond p)\Big) \wedge \Box\neg q \tag{5.2}$$

where eventually, either p or the alternative r and eventually p will hold, while q is never true. Due to the fact that the specification requires p to hold infinitely often once eventually reached, the language that satisfies this property is an ω-language and can only be realised by an automaton that can handle these languages, and similar examples can be found in [2].

In simulated cases it is not necessary to require infinite repetition. In the safe navigation to destination setting, the problem can be considered solved when the agent converges to a policy that ends in the destination state. However, this is only a practical distinction. In this work, it can be assumed that all specifications require Büchi automata realisations.

This setting is ideal for an initial experiment where classical Q-learning is compared to a constrained reinforcement learning problem. Since temporal logic constrained reinforcement learning, both with an accepting frontier function and with potential, should not behave differently in a setting where the standard Q-learning algorithm can find a correct policy, this situation is useful to compare the proposed algorithms performance in terms of execution time, convergence to the optimal policy and developed policy.

5.2.2 Sequential state visits

Here the MDP is again $\mathcal{M}$, but the atomic propositions are now $\{p, q, r, s \dots\}$. The state labeling function maps, as before, p to the goal state and q to dangerous states, but r, s and potential subsequent letters in the alphabet are mapped to states which are necessary to visit in a specified order before reaching the goal state. Upon reaching the goal state or a dangerous state, the agent must stay there forever. An LTL specification for this behavior, with an increasing number of required state visits, is

$$\begin{aligned} \varphi_1 &= \Diamond\Big(s \wedge \Diamond p\Big) \wedge \Box\neg q \\ \varphi_2 &= \Diamond\Big(r \wedge \Diamond(s \wedge \Diamond p)\Big) \wedge \Box\neg q \\ \varphi_3 &= \Diamond\Big(t \wedge \Diamond(r \wedge \Diamond(s \wedge \Diamond p))\Big) \wedge \Box\neg q \\ &\vdots \end{aligned} \tag{5.3}$$

This problem is interesting, because it describes a situation where the classic Q-learning cannot, except in very limited scenarios, be used to find the optimal policy. It cannot (in any practical scenario) receive a strong enough incentive to find an intricate sequence by purely ϵ-greedy exploration and environment rewards, compared to the TL constrained methods which supply the agent with rewards for selecting a specific path through the MDP.

In a sequential state visits problem, the accepting frontier may prove useful compared to the pure LTL constrained RL algorithm, which only receives rewards upon visiting forbidden or marked states. In the case of the third of the proposed algorithm, which uses potential, this problem may prove challenging if obstacle avoidance is to be performed beyond the sequential state visits. Here, a conflict of interest between the elements in the final method may arise.

5.2.3 Liveness and fairness

Other than safety specifications, there are specifications that enforce the repetition of "good" behaviors. An example of such a specification is the liveness property. This property, formally described in [2], is naturally an LTL property that cannot be realized by finite words, or traces, in a corresponding automaton. Intuitively, the property is in [2] defined as the ability for any finite prefix to be extended into an infinite trace that satisfies the liveness property. An example of a liveness specification that is used in this work is to demand that the agent travels back and forth between two states, indefinitely.

Fairness in the context of liveness is in [2] exemplified as a problem where n processes require service. The liveness specification may state that service of all processes shall be repeated infinetly often, expressed as several infinite acceptance conditions. If one of the processes constantly requests service, and also gets service upon its request, this corresponding acceptance condition is fulfilled more often than the others. This behavior might be valid according to the liveness LTL specification, but it is intuitively considered to be an unfair strategy.

A liveness LTL specification that is considered here is given by

$$\varphi = \square\lozenge p \wedge \square\lozenge r \tag{5.4}$$

Automaton realizations of this property can be found both in [4] and [2]. While the latter uses a generalized Büchi automaton with two acceptance conditions that must be fulfilled infinitely often, the former uses a standard Büchi formulation. Therefore, it is interesting to see if there is a practical difference between these formulations, and this problem thus investigates how important the selection of automaton is in practical situations, and how each of the proposed algorithms handle a scenario where there is no goal state to end up in.

5.3 Measuring Algorithms

To evaluate the algorithms, data on relevant properties must be collected when running experiments in the different settings that were described previously. Five different ways to measure algorithms are considered, but not all are used when performing experiments.

5.3.1 Time and memory complexity

Time and memory complexity is a common way of determining what is expected of an algorithm in terms of time and memory requirements [26], independent on hardware. Complexity is expressed using the mathematical concept of function order to compare the rate of growth of an algorithm. For two functions $f(n)$ and $g(n)$, a brief definition of order is formulated according to [27] as

- $f(n) = \mathcal{O}(g(n))$ if $\exists\ c > 0$ such that for large enough n, $f(n) \leq cg(n)$.
- $f(n) = \Omega(g(n))$ if $\exists\ c > 0$ such that for large enough n, $f(n) \geq cg(n)$.
- $f(n) = \Theta(g(n))$ if $\exists\ c, c' > 0$ such that for large enough n, $cg(n) \leq f(n) \leq c'g(n)$.

Then, the growth rate of $f(n)$ can be expressed in terms of $g(n)$. In this context, $f(n)$ is an algorithm while $g(n)$ is a function in n, which can be viewed as "number of elemental operations". For example, a triple nested for loop where each loop can execute n iterations has the time complexity of $f(n) = \mathcal{O}(n^3)$. Some common rates that can be found in [26] are the constant $g(n) = c$, logarithmic $\log(n)$, linear $g(n) = n$, linear logarithmic $g(n) = n\log(n)$, quadratic $g(n) = n^2$ and exponential rate $g(n) = 2^n$.

This method is applicable to both time and memory requirements of algorithms, but this way of measuring algorithms is not always necessary, and it does not always provide a relevant statistic that is practically usable. In supervised learning, it is interesting to study both computational complexity and sample complexity (how large must the training set be to achieve learning), but in the more realistic reinforcement learning setting, this is harder for a few reasons, as discussed in [28].

- The environment with which the agent interacts does not provide labeled training and testing sets, so there is no "ground truth" to compare to, making it hard to know the true number of interactions needed for learning.
- The information received by the agent is only "partly labeled", since the agent must maximise a long term reward while only seeing the current reward at a given time instance.
- There is no clear segregation between training and testing; the time the agent spends until convergence to the optimal policy occurs is dependent on the interaction with the environment and other parameters such as the trade off between exploration and exploitation.

In short, interacting with an environment makes it hard to measure performance in terms of complexity. A more common way of measuring the performance is therefore to measure directly how many interactions with the environment are necessary before the reinforcement learning algorithm converges to an optimal policy. Therefore, the most important parameter to measure is the number of episodes needed for convergence.

5.3.2 Convergence of entries in Q-table

Firstly, the term convergence must be explained. In the scope and context of this thesis, the term implies the specific development of a value as a function of discrete time, such that the value approaches a region where the fluctuations of that value are practically bounded around some arbitrary constant with an arbitrary tolerance.

In a tabular Q-learning problem, the values are updated as the agent collects rewards and moves around in the environment. If a problem is solved correctly, the agent develops a path through the environment, stops exploring additional states and thereby ceases to collect more rewards. This practically means that most elements in the Q-table are not updated anymore.

To see when an algorithm converges in a tabular Q-learning problem, one way to do so is therefore to look at the average value of all entries in the Q-table versus episode number. In this way, the development of the whole Q-table can be summarised in one value per episode, and that value will stabilise when the elements in the table cease to be updated for a sufficient number of episodes. Thus, the number of episodes needed for convergence to the optimal policy is easily accessible.

The reason for using this measurement instead of going by total collected rewards is that in a TL constrained RL setting, rewards are fairly sparse; on one episode, the agent might only collect either $+1$ or -1, making the reward per episode measurement very noisy. Therefore, Q-element value convergence is the most stable measurement available in this context.

5.3.3 Rewards

Since the rewards that the agent receives are few and may come from either an automaton, the environment or a potential function, it is interesting to see how

much reward the agent receives per episode and how it is distributed across all different types of rewards. Using this metric, it can be seen approximately how many intermediate states, goal states, dangerous states and automaton goal states are visited during an episode.

5.3.4 Time

In this work, all experiments are conducted on the same hardware. Therefore, it is in this case relevant to measure the time it takes for the machine to perform an episode of any of the proposed algorithms. Since each algorithm performs different sub-processes that the other algorithms do not, it is interesting to see if any slightly superior performance is worth it in terms of execution time.

5.3.5 Relevant parameter development

In reinforcement learning, it is common to change parameters between episodes during training. This is often done to manipulate the trade off between exploration and exploitation, but may also be done to regulate how much neighboring Q-table values shall influence the current Q-function value. Therefore, to see if there is a significant relation between convergence and parameter value development, all parameters that change with episode number are measured in the experiments of this project.

5.3.6 Policy visualisation

In these types of problems, the Q-table may converge while still not following the desired LTL specification. For example, if the LTL formula states that a sequential state visit shall be performed in a large MDP state space, it may not be possible to find all sequential states if they are very far apart. Thus, showing the resulting policy is crucial to determine if the policy is correct or not.

This can be done by running a greedy action selection algorithm on the given environment and potential automata, and the procedure is illustrated in the algorithm of Fig 5.1.

Result: Optimal policy from converged Q-table
Initialise
MDP $\mathcal{M}$ (and *automaton* $\mathcal{B}$ if present).
while *MDP (or product MDP) state is not the goal state/ accepting state* **do**
  Choose optimal action $a^* = \underset{a}{max}\ Q(s_\otimes, a)$.
  Record a^* and MDP state $s_\otimes \in \mathcal{M}$.
  Perform a^* and go to $s'_\otimes$ from $s_\otimes$.
  Set current state $s_\otimes$ to $s'_\otimes$.
end
Return the optimal action sequence and the optimal state sequence

Figure 5.1: Extracting the optimal policy from a converged Q-table.

The state sequence that this pseudo code snippet produces can then be visualized in the grid world so that its correctness can be determined visually.

5.3.7 Averaging over several experiment runs

To provide a stable measurement, and to examine the repeatability of the experiments, several runs of each algorithm can be made. Then the average result in terms of time, reward and mean Q-table element value can be derived. The optimal policies of each run, which hopefully are the same, can be visualised to see if all of them follow a single path through the MDP.

5.4 Summary

After having described the proposed composite algorithms central to this work in the previous chapter, the function of this chapter is to discuss the situations in which they are tested.

The chapter starts out with a motivation touching on how and why the algorithms are tested. Next, three problem categories are formulated in a grid world context; safe navigation to destination, sequential state visits and finally liveness and fairness.

After this, a formal description of the concepts of time and memory complexity is given, but along with this comes a motivation for why these ways of measuring algorithms are not used in this work. This is followed up with a description of the measurements that are used to describe performance; these are convergence of entries in the Q-table, rewards, processor time, non-static parameters, ways to visualise the final policy and finally the concept of running several experiments and averaging their data.

6

Implementation of Research Platform

Here, the work behind creating the small research platform is described. If detailed information about the project is needed, the entire source code is available on Github at github.com/CJHeiker/master.

6.1 Environment

A necessary part of this project is the implementation of an environment. This section outlines the technical specifications of this MDP structure and the implementation of the platform.

6.1.1 Requirements

The grid world environment is selected because it is natural to interpret a transition system as physically travelling through a square world. The standard probabilistic grid world is represented by the MDP $\mathcal{M} = \{\mathcal{S}, \mathcal{A}, P, s_{init}, AP, \lambda, \rho\}$, and the implementation of it must in some way meet the following requirements:

- The state space is given by the set $\mathcal{S} = \langle s_0, s_1, \ldots, s_i \rangle$ where $i = 63$ or 255. Furthermore, the grid world is always a square 8×8 or 16×16 grid of numbered states.
- The action space is restricted to tower chess moves, such that $\mathcal{A} = \{N, E, S, W\}$, where it can be noted that it is not possible to stay in a state. Furthermore, the actions are always available in every state.
- The probability map P may either be probabilistic or deterministic, meaning the probability of transitioning is in the probabilistic case the transition probability obtained from $P(s, a, s')$ is between zero and one, and in the deterministic case $P(s, a, s')$ gives either zero or one.
- For the initial state, $s_{init} = s_0$ in all experiments, for simplicity.
- The atomic propositions are $AP = \{p, q, r, s, t, \ldots\}$.
- The state labeling function λ maps the atomic propositions to different MDP states. These are used by the automaton to evaluate the LTL specification.

- The environment reward function ρ maps transitions to a real number, and the design of this function can be selected as needed for the reinforcement learning problem at hand. However, this type of reward is only used in the standard Q-learning case, without an LTL specification.

To clarify, the grid world is "square" in the sense that the graph representation of the MDP states and transitions can be ordered in a Cartesian coordinate frame, where the coordinates are given as a function of the state index. Using this property a state index can be translated to a coordinate in the plane by

$$\begin{aligned} y &= \text{floor}\left(\frac{i}{d}\right) \\ x &= i - y \times d \end{aligned} \tag{6.1}$$

where x and y are the column and row indices of a 2D grid, i is the state index, and d is the dimension of the grid world ($d = 4$ for a 4×4 grid world). For example, the coordinates for state $i = 5$ in a 4×4 grid world are $y = \text{floor}(5/4) = \text{floor}(1.25) = 1$ and $x = 5 - 1 \times 4 = 1$, which holds if compared to the grid world in Fig 6.1.

Moreover, according to the transition probabilities it is possible to for example perform the action $a = W$ in a state which is graphically considered to be the western border of the grid. These types of transitions are interpreted as self loops, and the intended destination state is the same as the origin state.

6.1.2 OpenAI Gym and the Frozen Lake environment

With the basic features required for the environment explained, it is now time to describe the implementation of it in the context of the other parts of the small research platform that is implemented in the first part of this project.

In the platform, it must be possible to generate MDPs which can be interacted with and visualized, run all of the proposed algorithms, create tests from which data can be extracted, and finally visualise the statistics. The platform is implemented entirely in Python.

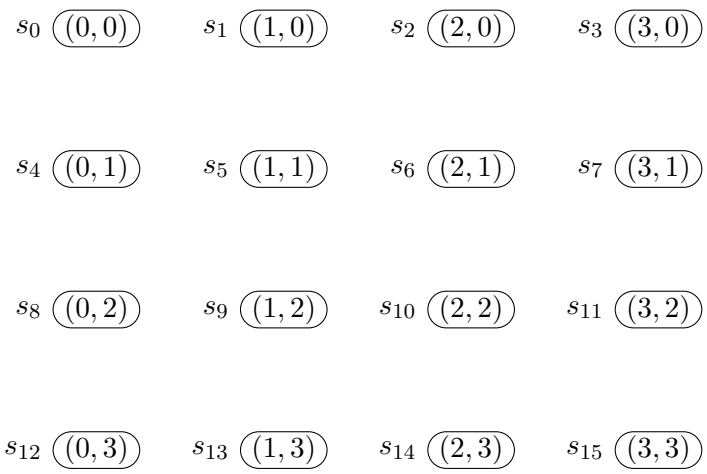


Figure 6.1: Example of indexed coordinate states.

The first attempts to create an MDP environment were made entirely from the ground up, defining the state space, actions and other units of the MDP using fundamental data structures in Python. However, some unforeseeable problems that appeared in later stages of use required extensive work to straighten out. In short, an entire structure based on a flawed foundation is very hard to change from the top, as both the errors and the required solutions propagate through the whole structure. A more stable starting point is therefore to use an existing MDP implementation as a basis, and customise it where necessary.

The OpenAI Gym [29] is a very popular Python toolkit designed entirely for reinforcement learning. Its purpose is to provide open source implementations of different MDP environments, often modeled on physical phenomenons such as inverted pendulums, balancing jointed bodies and classic Atari games.

The environment that is interesting for this work is the OpenAI Gym "Frozen Lake" environment. Placed within the "toy text" category of environments, Frozen Lake is a 4×4 probabilistic grid world MDP with one initial state and one state labeled goal. The remaining states can be either slippery ice or dangerous water, and the objective is to get from the initial state to the goal state without accidentally slipping into the water.

The Frozen Lake is a good basis that can be modified, so the source code of the entire OpenAI Gym is downloaded and installed locally as a new Python project, instead of using it as a "black box", which is how it is often used if no modifications are to be made to it.

6.1.3 Required modifications to Frozen Lake

The following properties of the standard Frozen Lake environment needs modification:

- Frozen Lake offers two predefined sizes of grid worlds, 4×4 and 8×8. This needs to be extended so that any sized square grid world can be used.
- Frozen Lake is probabilistic with unknown probabilities in its original description. There is a need to implement both deterministic environments and create a way to assign known probabilities for the non-deterministic versions.
- The only possible state labels are S (Start), F (Frozen), H (Hole) and G (Goal). For this application, the MDP must be extended so that arbitrary atomic propositions can be set as state labels in the Frozen Lake MDP.
- Frozen Lake offers a text based visual interface which becomes insufficient when more complicated state labels are added, and must therefore be exchanged for a new one.

The solutions to these problems are presented and discussed in the following sections.

6.1.4 State space extension

As Frozen Lake already describes a grid world, many parameters such as the number of actions and transitions follow a certain pattern. This is apparent in the standard

implementation, as there are several ways of changing the standard grid, and this implementation can be viewed on the Github page of OpenAI Gym Frozen Lake [30]. When creating the environment object, there is a choice between a standard 4×4 and an 8×8 grid world both with F and H states, and a randomly designed grid world of any square dimension can also be selected.

The two standard alternatives are represented as character arrays. The characters are the standard state labels S, F, H and G. The arrays are selected internally from a look-up table by the keywords "4x4" and "8x8" which the user provides as an argument to the environment constructor. As random environments are not interesting here, the way of implementing arbitrarily sized grid worlds is done in the style of character arrays. Any square character array is valid as an environment map, provided it has the correct characters in it. Therefore, the only modification done here is to instead provide the environment map from outside the constructor, and send it as an argument instead. In that way, the user has the ability to define the map freely.

6.1.5 Deterministic representation

The Frozen Lake environment does have a constructor argument that regulates if the grid world is deterministic or probabilistic [30]. In the deterministic case, only one transition is defined for a selected action and state. As this function is already implemented, this part of the standard implementation is left untouched.

6.1.6 Custom transition probability assignment

The environment constructor assigns probabilities automatically by considering that the agent either goes to an intended state, considered to be "forward" in the coordinate frame of the agent, or slipping to the left or right state of the current state. These probabilities are fixed to 0.33, meaning it is as likely to go to one of two incorrect next states as it is to go to the correct next state, and it cannot slip "backwards".

This is not a fitting probabilistic setup for this project. Instead, this method is redesigned to take a total probability p of going in any unintended direction, assign the probability of $1-p$ of going in the right direction, and distribute three (instead of two) probabilities that all sum to p among each incorrect next state corresponding to the left, right and backwards incorrect state. The pseudo code for the new method is shown in Fig 6.2.

A special case is defined when the agent moves to a dangerous state, in which the probability to remain there is set to one. Note that the agent may transition to a dangerous state in the same manner as to any other state, but once it gets there it cannot escape.

6.1.7 State labels and state labeling function

In the RL algorithm, interactions can be made with the environment via the *step* function. This function updates the state of the MDP, and returns a flag describing if a goal state or a forbidden state is reached. As this project requires an MDP state

Result: Transition probability mapping P
Input: Probability p of going in any unintended direction
Initialise $P(s, a, s') \rightarrow [0, 1]$ to zero
for *States* $s \in \mathcal{S}$ **do**
  for *Actions* $a \in \mathcal{A}$ **do**
    Assign a, a_1, a_2, a_3 and a_N as the intended, left, right, backwards and null actions
    if *s is a dangerous state* **then**
      Assign the stay in state probability $P(s, a_N, s) = 1$
    else
      Compute the resulting states s'_0, s'_1, s'_2, s'_3 deterministically
      Randomly select p_1, p_2 and p_3 such that $\sum_{i=1}^{3} p_i = p$ Select $p_0 = 1 - p$
      Assign the probabilities
      $P(s, a, s'_0) = p_0$
      $P(s, a, s'_1) = p_1$
      $P(s, a, s'_2) = p_2$
      $P(s, a, s'_3) = p_3$
    end
  end
end

Figure 6.2: The new transition probability assignment method.

labeling function, the ability to beforehand assign custom labels and also to access these when performing an MDP step is needed.

The state labels are assigned by sending a state label array similar to the MDP state construction character array as an argument to the constructor. A function that obtains the state label, taking a state as an argument, is also implemented in the MDP class.

6.1.8 Visualising the environment

The new grid world visualisation runs through the character array that defines where dangerous states, intermediate states and the goal state of the MDP are. It then creates a matrix representation of the grid world, where different state labels have different identification numbers. Then, a heat map with a custom colour associated with each identifier is plotted. The initial state is marked in green, the intermediate states are yellow, dangerous states are blue and the goal state is red. Furthermore, the states are indexed by their coordinates.

6.2 Automata

Intended to be used in parallel with the MDP, the automata class must have a similar type of step function as the MDP implementation. The automata class is implemented with the following properties in mind.

- There are different types of automata. In this project DBAs and GBAs are

the ones that require implementation. DBAs have one accepting state, and are designed to realise infinite accepting sequences, while GBAs have several accepting state sets where the acceptance condition is that all of these sets must be visited infinitely often.

- The step function takes a list of MDP state labels (atomic propositions) and performs a transition to another state if possible, returning the new state.
- As in the MDP implementation, a reset function is necessary to enforce the initial state when starting a new episode after having ended up in an automaton sink state.
- Marked and forbidden states must be described.
- The optional accepting frontier function is dependent on the existence of automaton colored states, which are not the same as marked and forbidden states, although they may coincide in some cases.

The most challenging implementation step of the automaton is the definition of actions and transitions. As it is known what labels can be expected from an MDP, each action can be implemented as a function that takes a list of boolean values representing the availability of a label in an MDP state. If the predefined configuration of label values corresponding to the transition condition in the automaton is true, the function returns the next state. Upon constructing the automaton, all actions can be defined accordingly, and then a list of source state and action index pair tuples will ultimately link the source state to the destination state via the action. The functions are implemented as anonymous lambda functions, and an example of this follows below.

Consider the transition between automaton state q_0 and state q_1. There are three labels that can be true or false, p, q and r. The transition condition between q_0 and q_1 is $p \wedge q \wedge \neg r$, and in this case the lambda function defining the transition condition is given by

$$\textit{lambda}\ \text{labels:}\ q_1\ \textit{if}\ \text{labels(0)}\ \textit{and}\ \text{labels(1)}\ \textit{and not}\ \text{labels(2)}\ \textit{else}\ \text{NaN} \quad (6.2)$$

If this is the action with index three in the list of all actions, and it can be performed from state q_0, then a transition tuple is defined by (q_0, a_3), effectively linking the state q_0 to state q_1 via action a_3 that can be performed depending on the truth values of the labels p, q and r.

6.2.1 Accepting frontier

The accepting frontier is dependent on the existence of coloured states, but the function itself is implemented outside of the automaton class. The frontier states are defined as a simple map just as the marked and forbidden states. The option to include frontier states is not necessary if the frontier function is not used. The frontier function is ultimately implemented exactly as Section 4.2.2 describes.

6.3 Statistics

The generated statistics are slightly different between each experiment, which makes it easier to have individual statistics subroutines for each experiment. In general, the plots that are generated show the development of the average Q-table value where the dimension describing automaton states is summed, making the trajectory comparable to the standard Q-table which does not have this dimension. The trajectory is normalised to the individual maximum for easy comparison. An example of this is seen in Fig 7.3, describing the statistics of the first experiment.

In the second plot of Fig 7.3, the average reward distribution over environment, automaton, accepting frontier and potential rewards for an algorithm is shown as a bar graph. This is not normalised, as it is important to compare the actual levels of the distribution. Below the reward distribution comes the exploration development, which describes the value of the exploration parameter as a function of episode number.

The episode time is also shown, below the exploration development in Fig 7.3, but it should be noted that the shown statistic is a moving average of the actual time, which is noisy. This is considered reasonable, as the average value over a number of subsequent episodes still provides enough information about the time it generally takes for one episode.

The state sequence derived by stepping through the converged Q-tables as described in Section 5.3.6 is both visualised as a trace through a coordinate frame, and as a curve describing the state identification number versus step index. Examples of this can be seen in Fig 7.4. In the coordinate frame plot, it can be observed how many times the agent enters each state in the policy, and in the chronological plot it can be seen when these steps were taken, as this cannot always be deciphered from the coordinate trace. It is very important to note here that when an experiment is performed, a list of MDP states is provided containing all of the interesting states that need to be contained in what is regarded as a successful typical state sequence derived from the Q-table. This list does not care about order, and therefore a visualised MDP trace might still need visual inspection. Furthermore, if several experiments are run, the algorithm can generate one or more example traces per experiment, and then visualise the first one that contains the desired states. If it is desired to see how many of the example traces that are successful, each of them can be inspected in the text file that is generated after each experiment. This file also contains other data of the experiment, such as:

- Date and time for the experiment.
- Number of MDP states.
- Probabilities for all transitions.
- Information about the automaton used.
- Experiment ID, and other information of the parameters used in the RL loop.
- All the policy candidates for each sub-experiment *a)*, *b)*, *c)* and *d)*.

6.4 Experiment Implementation

An experiment is defined as a separate Python file, and is always divided into four sub-experiments that are to be compared. In each experiment an MDP is defined, followed by an automaton, and configurations regarding accepting frontier and potential. All settings required for the experiment, and for the statistical plots are made in this file.

An important feature of the experiments is that they can be run several times to provide a more reliable measurement. The defined transition probabilities of the MDP are not changed between these runs. Thus, while the transition probabilities are identical between runs, the selected transitions will still change according to the outcome of the probabilistic scenarios, which can affect learning. If a very high probability to step in the wrong direction is selected, it is recommended to perform more runs than if the probability is very low or zero. When all runs are performed, the average Q-entry, reward distribution, and time per episode is averaged between episode number over all runs. The exploration development does not change between runs, and does not need to be averaged over the runs.

The way that the exploration factor ϵ, described in Section 2.4.2, evolves through episodes is called exploration development. This development is in all experiments exponentially decaying according to

$$\epsilon(k) = (\epsilon_{max} - \epsilon_{min})e^{-\beta k} + \epsilon_{min} \tag{6.3}$$

where k denotes the episode number, ϵ_{max} is the maximum exploration factor, ϵ_{min} is the exploration when k approaches infinity (and not the exploration factor value at the last episode) and β is the exponential decay rate coefficient. This decay model is selected because it is a bit more flexible than a linear model, but it can be changed easily by replacing the episode dependent lambda function that implements it in the main RL loop.

Furthermore, through ϵ_{min} it is possible to define an exploration that does not approach zero after all episodes have terminated. This possibility is included because personal experience has shown that it is sometimes not possible to find a policy with an exploration factor approaching zero.

6.5 Summary

The research platform is a cornerstone of the first part of this project, and this chapter describes its implementation. Starting with the demands on the environment implementation followed by a description of the OpenAI MDP starting point, details are given concerning the required modifications on the OpenAI Frozen Lake environment that are made to fit this application.

A fundamental part of the implementation is the automata structure, which is shown next, along with a brief description of the accepting frontier implementation. Lastly, some details on how statistics are generated are provided, followed by a description of the general structure of the experiment implementation. Here, a function for the exploration factor development is provided as well.

7
Experiments

In this chapter, all experiments conducted in Part I are described in detail in terms of environment, specification, setup, hypothesis, result and analysis. Results for each of the experiments are shown in two figures; a statistics plot and a graph of a typical state sequence derived from the converged Q-table. Before describing the experiments in detail, it is necessary to describe the outline of each experiment and motivate their design.

7.1 Experiment Structure

In each experiment, four different algorithms are used to solve a problem. In most of the experiments, these algorithms are standard Q-learning, basic temporal logic constrained reinforcement learning, the accepting frontier extended version, and the extended version that uses both the accepting frontier and the potential function. However, in a few of the experiments such as in the very last one, the test cases are a bit different. In the last one, for example, only the algorithm that uses both the accepting frontier and the potential function is used as this experiment mainly explores properties of the potential function.

The problems are expressed as combinations between an MDP and a specification, illustrating different scenarios from Section 5.2. In each experiment, both the MDP and the specification automaton are presented and explained. Before describing the experiments, the parameters, MDP types and statistics interpretation are covered.

7.1.1 Basic parameters

Each experiment is run ten times. The statistics shown in the first figure of each experiment is an average taken over all ten runs, producing a more representative and stable statistic than one experiment alone. The main reason for doing so is that all environments are probabilistic, with the probability of going in an unintended direction being 0.15. Additional to the ϵ-greedy Q-learning algorithm, the probabilistic environment will cause variations in the learning process to some degree, and it is therefore desired to show the average development between the runs.

All algorithms shown have the tabular Q-learning algorithm as a common denominator. For this, the discount factor γ and the learning rate α are kept at 0.9 and 0.1 respectively through all of the experiments. In this thesis, the fundamental Q-learning algorithm is not itself meant to be experimented upon, and therefore

these parameters are not tuned any further.

7.1.2 Two grid world MDP sizes

The environment MDPs used in the experiments of Part I in this project are different in detail, but there are only two categories in terms of state space dimension; 64 and 256. An environment with 64 states is a square grid world with dimensions 8×8, and a 256 state MDP thus represents a 16×16 grid.

The smaller environment MDPs are designed with the intent to eliminate problems related to dimensionality of the state space. Experiments conducted on this type of MDP include the safe navigation to destination problem, along with sequential state visits and finally liveness and fairness, all described in Section 5.2.

For the last two experiments of Part I, 16×16 grid world MDPs are used, but with the same actions as before and the affected attributes adjusted to accommodate the higher state dimensionality. Moreover, these grids do not contain any dangerous states. The experiments conducted on these MDPs address exactly what the experiments done on smaller grids avoid, and methods to solve state dimensionality problems are tested. Again, all environments are probabilistic with the risk of performing unintended transtions kept at 0.15.

7.1.3 Interpreting the results

The statistics plots firstly show the average Q-table element value. The reason for displaying this value instead of the collected reward per episode is that in the setting of TL constrained RL, this is a more stable measurement of the same property and a more detailed motivation can be found in Section 5.3.2. Since the algorithms can collect very different amounts of rewards during an episode, each corresponding Q-element value plot is normalised to its individual maximum, which makes it easier to (visually) compare the points of convergence of each algorithm.

Regarding the reward distribution plots, there are four sources that rewards can be collected from; the environment, the marked and forbidden states of an automaton, the detached accepting frontier function and the potential function. To show which of the sources each algorithm uses the most, a reward type distribution averaged over all 10 runs is provided. This distribution is not normalised in order to show the actual relationships between the reward sources and algorithms. In the plots, this is represented as a bar graph with the four categories on the x-axis and the average reward displayed as a colour-coded bar where each colour represents an algorithm.

Next, the development of the exploration factor ϵ is provided, one for each algorithm. The time measured between starting and finishing an episodes is also provided, and this measurement is also averaged over all runs. As it has quite a high variance, it is low pass filtered to display the overall trend over a few episodes. Finally, a chronological graph shows one of the states sequences derived from each of the converged Q-tables, with intermediate states and goal states in the MDP marked as black x:es where the state sequences visit these. These sequences are also shown on a representation of the grid world in the second figure of each experiment

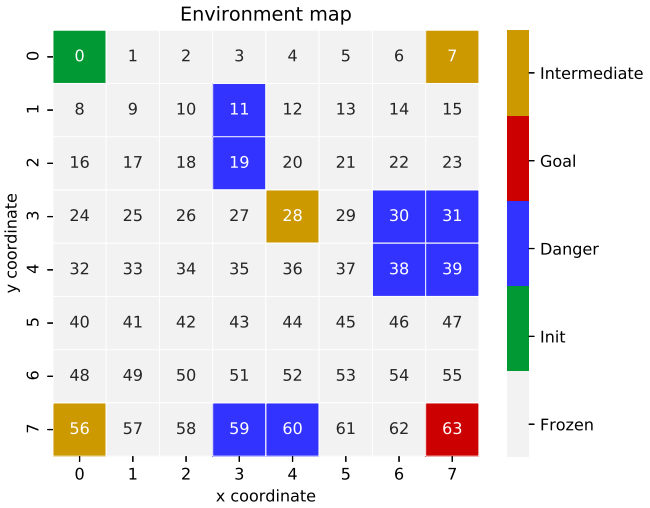


Figure 7.1: Grid world representation of the safe navigation to destination problem.

result section.

It is important to note that the state sequences, also shown in the second figure of each experiment, are derived by stepping through the product MDP and greedily selecting actions recommended by the Q-table; with the probabilistic setting, different paths may therefore be derived from the same Q-table, and they are not guaranteed to always satisfy the LTL specification. However, when a complicated LTL formula is satisfied, it is not by chance and the algorithms producing the results can then be assumed to produce a correct MDP trace in the majority of cases due to the relatively low probabilistic MDP behavior used in these experiments.

7.2 Safe Navigation to Destination

This section concerns the experiment in which the agent must navigate safely from the initial state to a goal state. The environment is depicted in Fig 7.1. From the initial green state, the agent must learn a policy that guides it to the red state 63, either by visiting one or several intermediate yellow states or by going directly to the final state. Dangerous states, marked in blue, must be avoided in the final policy. In each intermediate state is the label r, in the dangerous states lies q and in the goal state is the label p. The LTL formula that specifies this behaviour is

$$\varphi = \Diamond\Big(p \vee (r \wedge \Diamond p)\Big) \wedge \Box\neg q \tag{7.1}$$

and states that one possible behavior that satisfies this formula is observing p once, which implies observing it forever. Another alternative is to observe r, and then eventually observing p forever. At all times, q is never observed as this would lead to observing q forever. The automaton realisation of the specification in this experiment is shown in Fig 7.2. Here, the accepting frontier states are marked using colored states, and the blue color denotes that states q_1 and q_2 make up the frontier set.

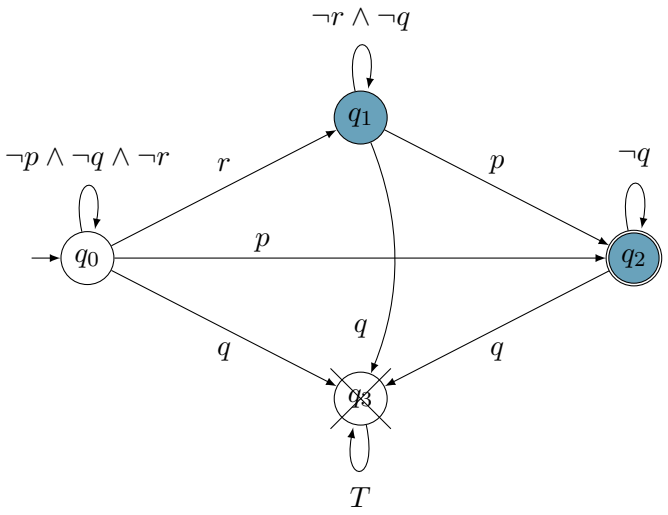


Figure 7.2: Colored deterministic Büchi automaton specifying safe navigation to a destination.

An important note is that the automaton is formulated under the assumption that at most one label p, r or q can be observed (or true) in a state. Therefore, it is valid to state that r alone causes a transition to q_1 from q_2, as q cannot be true at the same time. Moreover, if r is true, the loop condition $\neg p \wedge \neg q \wedge \neg r$ is false. Having at most one true label per state also motivates why it is enough to have only p or q as conditions for the outgoing transitions of state q_1.

Lastly, the specification assumes that once p is observed, the agent can either observe q and go to the forbidden state, or not observe q over and over. In the practical implementation, however, the learning is terminated as soon as the marked state is reached once.

7.2.1 Setup

The first algorithm to be examined is the ordinary Q-learning with ϵ-greedy exploration. The second is the temporal logic constrained RL algorithm from Section 4.6.2, which uses the automaton in Fig 7.2. The third algorithm used is the second proposed algorithm form Section 4.6.3 and it uses the accepting frontier function that regards both DBA state one and two as coloured frontier states. These states have the color blue in Fig 7.2. Lastly, the aforementioned temporal logic constrained RL algorithm is extended with the potential function, which constitutes the third proposed algorithm described in Section 4.6.4 and is thereby the fourth algorithm tried in the experiment.

The experiment described above is conducted using the parameters in Table 7.1. Note that ϵ_{min} is a parameter, denoting exploration in the limit. In practice, the minimum exploration in this experiment is around 0.5.

Table 7.1: Parameters for the safe navigation to destination experiment.

Exp	Algorithm	No. exp	ϵ_{max}	ϵ_{min}	β	Eps	s/e
a)	Standard Q	10	0.9	0.1	0.0001	10000	100
b)	TL-RL	10	0.9	0.1	0.0001	10000	100
c)	TL-RL w/ AF	10	0.9	0.1	0.0001	10000	100
d)	TL-RL w/ AF & Pot	10	0.9	0.1	0.0001	10000	100

7.2.2 Purpose

The purpose of this is to show that in certain simple settings, an RL problem can be solved without being constrained by temporal logic. With this being said, it is interesting to see if a temporal logic specification, accepting frontier and a potential function can make the algorithm converge faster to the optimal policy. The automaton in Fig 7.2 realizes the first LTL specification in Section 5.2.1, an ω-language, as the marked and forbidden language includes an infinite number of visits to the single marked and forbidden states.

However, even though this automaton is formulated in the experiment, the episode is terminated by the MDP when the goal state is reached, meaning the looping behavior will not be learned. This is not considered to be a problem, since an increasing number of rewards would be handed out upon arrival to this state if the MDP would not terminate, and there is no doubt that this would make the optimal policy stay in the final state once it is reached one time.

7.2.3 Hypothesis

For the Q-learning to work, there must be a sufficient initial exploration that decreases towards greedy as the Q-table converges. This algorithm only receives rewards from the environment, meaning the intermediate rewards must be significantly smaller than the final goal reward, and the previously mentioned exploration must be high enough for the agent to move past the intermediate rewards until it gets to the goal state. The hypothesis for the Q-learning algorithm is that it will approach the intermediate state that is not as close to the dangerous states as the others, and then go to goal.

When it comes to the standard temporal logic constrained reinforcement learning algorithm, rewards are received only for going to the goal state. Intermediate states are a possible detour in the specification, but they are not rewarded. In relation to the ordinary Q-learning algorithm that receives rewards for intermediate states, rewards are sparse in the LTL constrained RL solution. This may result in a slower convergence, but it does not run the risk of staying in an intermediate state as the Q-learning does due to possibly unbalanced reward weightings between intermediate and goal state transitions.

In the case of the accepting frontier extended algorithm, there is a slight chance that the algorithm goes to several intermediate states before goal, which might make

it converge to a policy that selects a longer state sequence. However, it is believed to solve the problem of sparse rewards discussed before, without risking to get stuck in intermediate states as frontier states are removed upon the first visit.

When adding potential, there must be a balance between aiming for a the intermediate and goal states and avoiding regions with dangerous states. In this case, there is a clear path to goal which makes it possible for the potential reward to be quite high. Among all algorithms, this one is believed to converge first and also converge to the optimal policy that selects the shortest path to goal.

7.2.4 Results

The results of the safe navigation to destination experiment are shown in Fig 7.3. A heatmap describing an example of a selected path derived using the converged

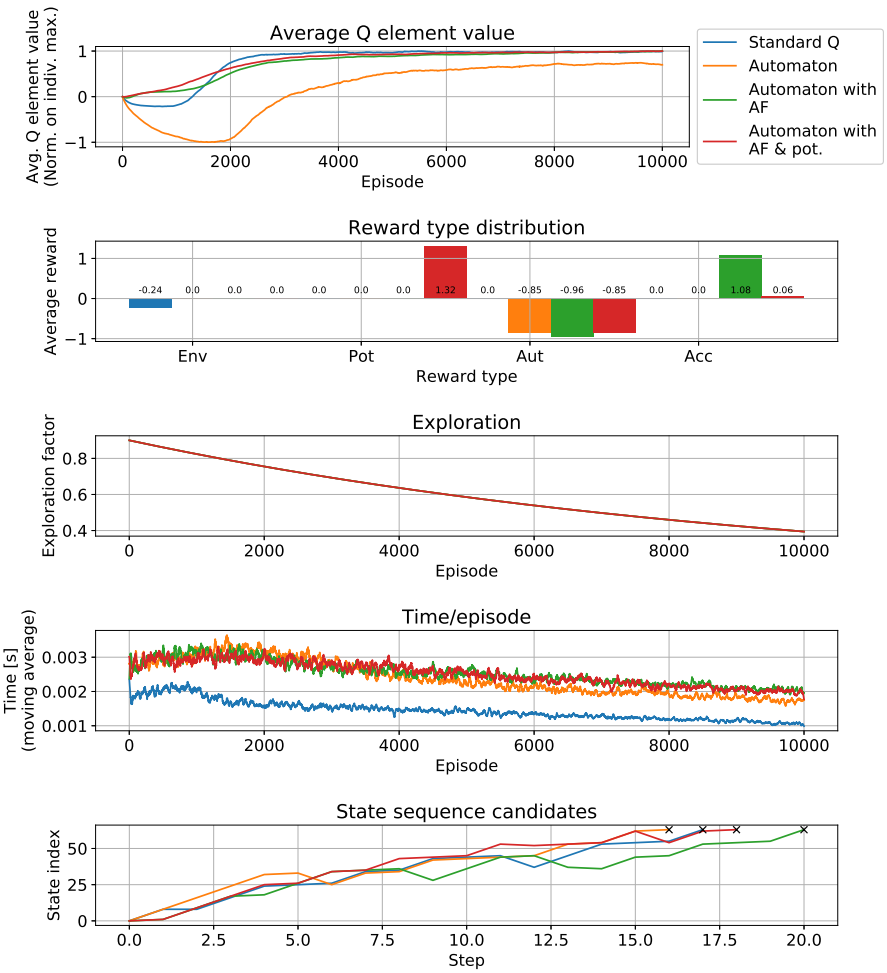


Figure 7.3: Statistics from the safe navigation to destination experiment.

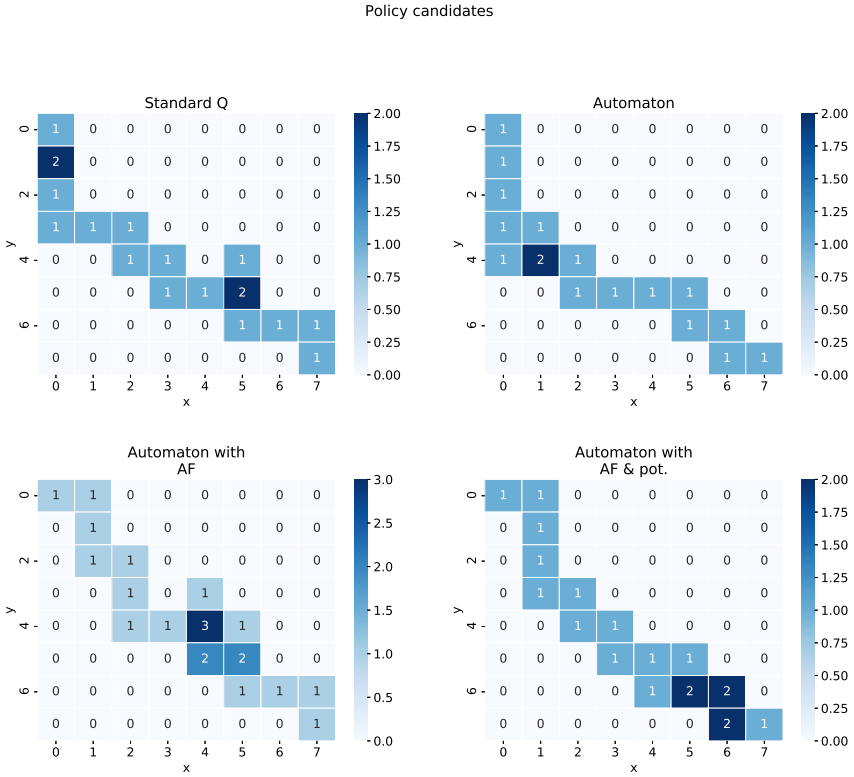


Figure 7.4: Representative state sequences derived from the converged Q-tables.

Q-table is shown in Fig 7.4.

7.2.5 Analysis

All algorithms converge, which agrees with the initial hypothesis. According to the hypothesis, the standard Q-learning algorithm would go to the goal state via an intermediate state. This is not represented in the sequences of Fig 7.4 which is somewhat surprising but may be due to the much larger reward received by going to goal directly.

With the standard TL constrained RL algorithm, intermediate states are disregarded; something that is entirely plausible due to that no rewards are collected for this. Using the accepting frontier, the hypothesis states that the agent would visit several intermediate states, but this also turns out to be incorrect for this particular state sequence. Furthermore, the prediction stating that the potential function converges to the shortest path policy due to the clear path to goal is incorrect, but it is actually almost as long as the standard Q-learning by inspection of the last plot of Fig 7.3.

By inspecting Fig 7.3, it can be seen that the standard Q-learning algorithm and the standard TL constrained RL algorithm receive many negative rewards in early episodes. This is because they find the optimal policy by visiting many dangerous states and eliminate them from the list of possible paths to take. The potential function algorithm increases steadily and in a smoother fashion than the accepting

frontier algorithm, which has one acceleration phase per newly found reward state.

An interesting side note is that the automaton constrained algorithms have two possible alternative behaviours that are enforced in the learning; going directly to goal or going to goal via one or more intermediate states. The accepting frontier rewards transitions to the intermediate states, and in the learning this is represented by positive acceleration phases when a new correct MDP trace is found. Despite this, actually stepping through the product MDP could naturally result in either of the allowed traces, as in Fig 7.4.

Regarding the reward distributions, the potential algorithm has the option to collect both rewards from the accepting frontier and the reward from the marked and forbidden automaton states, but it more or less disregards the accepting frontier rewards due to the greater reward accumulated from the potential. Getting a reward from the automaton is probably a bi-product from being drawn to the goal state in the MDP by the potential function. The automaton rewards are in all cases negative on average. In the case of the algorithm using potential, this property is crucial as the potential does not see dangerous states in the way of the optimal path.

The time it takes for the TL constrained RL, accepting frontier and the potential algorithms is in this case around 30% more than standard Q-learning. If implemented on larger problems, this may point towards a possible drawback.

7.3 Sequential State Visits Experiment 1

A visualisation of the environment is shown in Fig 7.5. In the first sequential experiment, the agent must learn a policy in which a sequence of states must be visited, while dangerous states are avoided.

According to the specification, the agent must first visit state 7, then state 56 and finally state 63 without visiting any dangerous states. State 7 has the label r, state 56 is labeled s, state 63 is labeled p and the dangerous states all have the label

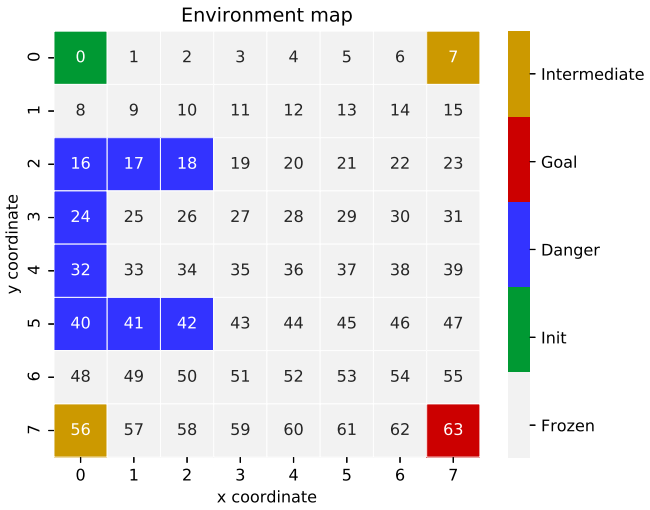


Figure 7.5: Grid world MDP visualisation for the first sequential experiment.

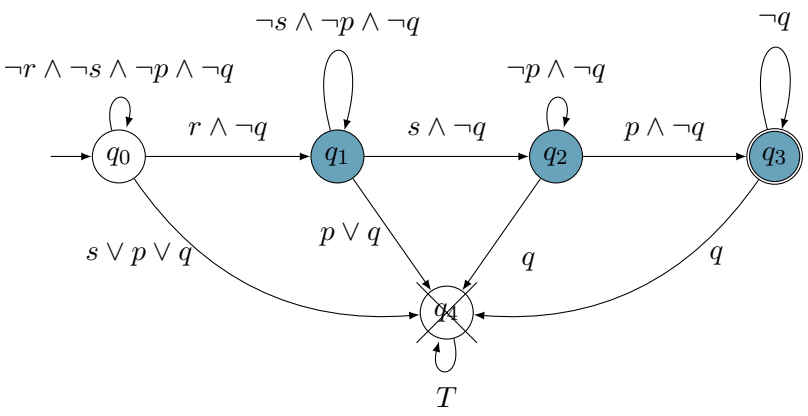


Figure 7.6: Deterministic Büchi specification for the first sequential state visits experiment.

q. The LTL formula for this behavior is thus

$$\varphi = \Diamond\Big(r \wedge \Diamond(s \wedge \Diamond p)\Big) \wedge \Box\neg q \tag{7.2}$$

This formula states that eventually, the sequence r, s, p is observed. Furthermore, q is never observed. The Büchi automaton that realises this is shown in Fig 7.6. In this automaton, frontier states are colored blue, the marked state is depicted with a double circle and the forbidden state is crossed out.

In this case, the transition to the forbidden state from the marked state by observing q is included, but in practice, the learning stops as soon as the marked state is reached. It is assumed that the agent has the same incentive as in the previous experiment to stay in the final state once it is reached, given that $\neg q$ is observed indefinitely in the end. Furthermore, the assumption that there is only one state label per MDP state makes it possible to formulate the transition between state q_0 and q_4 without regarding if r is true or false. If s or p or q is true there, r cannot be true. This principle holds for the rest of the transitions as well, which makes the automaton completely deterministic.

7.3.1 Setup

Classical Q-learning and the LTL constrained RL method are in this experiment compared with the detached accepting frontier extended version and the algorithm that uses both the accepting frontier and the potential function. Furthermore, this experiment is run using the parameters in Table 7.2.

7.3.2 Purpose

This problem may be solved by ordinary Q-learning, but only under special circumstances, for example if environment rewards for intermediate states and goal states are very carefully scaled to hopefully enforce a sequential policy. If the temporal logic specified version of the RL problem is a more convenient and more reliable way of enforcing the sequential behaviour, it is interesting to see if the accepting frontier or the potential function can improve convergence.

Table 7.2: Parameters for the first sequential state visits experiment.

Exp	Algorithm	No. exp	ϵ_{max}	ϵ_{min}	β	Eps	s/e
a)	Standard Q	10	0.9	0.7	0.0001	15000	100
b)	TL-RL	10	0.9	0.7	0.0001	15000	100
c)	TL-RL w/ AF	10	0.9	0.2	0.0001	15000	100
d)	TL-RL w/ AF & Pot	10	0.9	0.2	0.0001	15000	100

7.3.3 Hypothesis

The problem is considered small enough for all methods using an automaton to manage the problem. Classical Q-learning does not have any incentive to induce any specific seqential behavior and will in essence treat this problem as a safe navigation to destination problem. The TL constrianed RL algorithm does not get a reward until the whole sequence is finished, which is a disadvantage. This may require a higher amount of exploration throughout the learning process. If this is the case, the accepting frontier should remedy this and reward the agent for going to the intermediate states.

From Fig 7.2, it can also be seen that it is not possible for the agent to be rewarded for completing the sequence in the wrong order. The potential function may enforce the same behavior, but does not punish the agent for going to the dangerous states, which is why there is a risk of having a conflict of interest between reward sources around state 56 which is close to dangerous states. However, as the potential function aims to push the agent towards the intermediate states independently of exploration, it is believed that if the algorithm does produce a correct policy, it will also produce the shortest state sequence.

7.3.4 Results

The results of the first sequential experiment are shown in Fig 7.7. A heatmap depicting an example of a selected path through the MDP, where actions are selected using a converged Q-table is shown in Fig 7.8.

7.3.5 Analysis

All algorithms except standard Q-learning shall converge to the specified policy according to the hypothesis, but this is only partly a correct prediction. Q-learning indeed does not manage to find a solution that contains all the necessary sequential states, which is the reason that no state sequence is displayed in the corresponding top left sub figure of Fig 7.8.

Although a higher degree of exploration is used, the standard TL constrained RL algorithm using only an automaton can not collect a sufficient amount of positive rewards to induce the sequential behavior. However, according to Fig 7.7, the Q-table elements are updated to negative values which indicate that incorrect behaviors

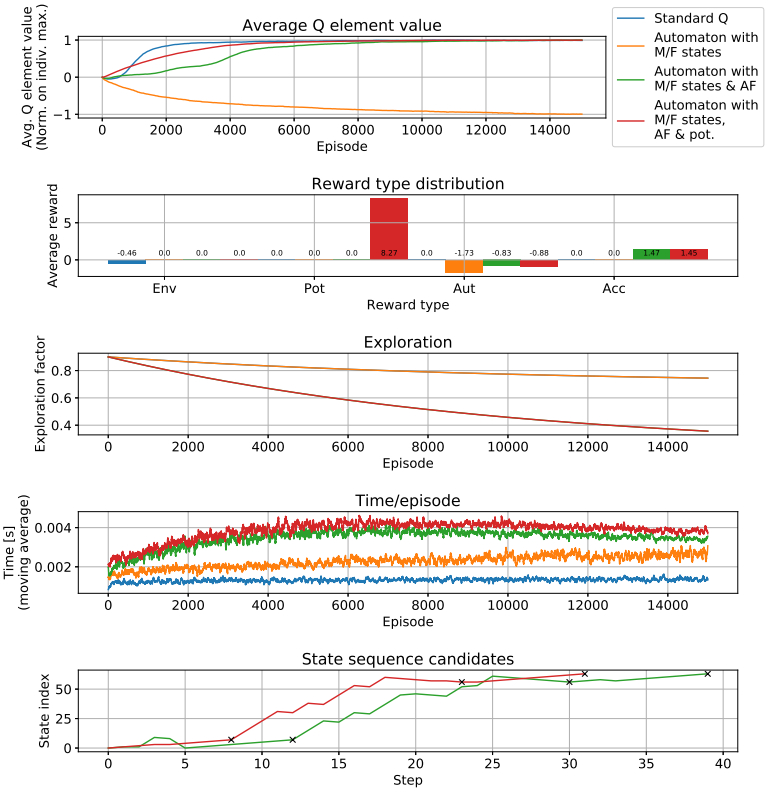


Figure 7.7: Statistics from the first sequential state visits experiment.

are at least punished. Therefore, even though the derived sequence does not contain the correct states, increasing the number of episodes and steps per episode may eventually lead to the agent finding the correct path. Even so, that is an inefficient strategy.

The accepting frontier does remedy the incorrect behavior of the standard TL constrained RL algorithm, and makes the agent complete the specified sequence, which is somewhat expected. For the fourth algorithm sequence seen in the bottom right sub figure in Fig 7.8, the accepting frontier and the potential do seem to agree with each other on where to go around the intermediate states, contradictory to the hypothesis that states that they would disagree when intermediate states are close to dangerous states. In fact, the sequence is in this case slightly more defined when using the potential; this may indicate that as long as there are no conflicts between the potential and the automaton, using potential will help produce a more stable policy.

In the statistics plot of Fig 7.7, the algorithm using potential has a smooth Q-element convergence curve on average, which seems to agree with the previous statement about potential producing a stable policy. The reward from the potential function makes up a large portion of the total reward, and in comparison to the algorithm that only uses the accepting frontier it is easy to see why the Q-learning curve

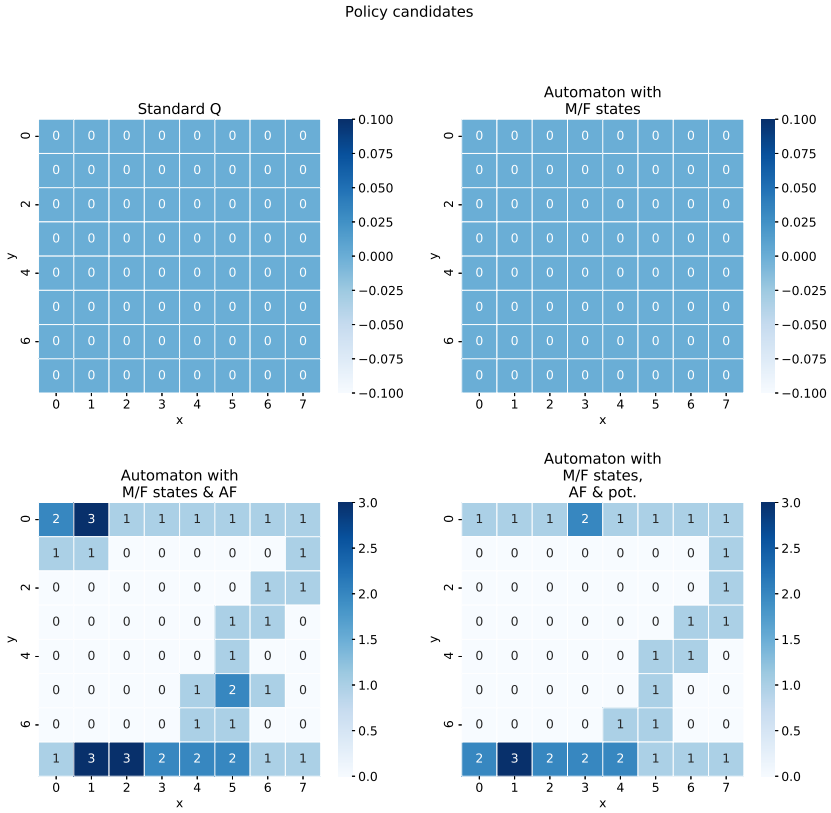


Figure 7.8: Representative state sequences derived from the converged Q-tables.

is so much smoother than the same curve corresponding to the accepting frontier algorithm. The latter reacts quickly once it gets a reward from a newly discovered correct sequence state, which can be seen in the bumps of the average Q-element curve. The standard Q-learning algorithm converges very quickly and collects positive rewards, but to a policy that is incorrect according to the specifications. In theory, there is a chance that the Q-learning algorithm can find a sequence either by chance or by strategic placement of rewards in intermediate states, but in this case it fails to do so.

As can be seen both in Table 7.2 and in the third sub figure of Fig 7.7, a very high exploration is given to the algorithms that are believed to not converge to the correct policy easily, but this attempt does not bear fruit. Time-wise, the algorithms seem to take up more time as they find all the states, and the time it takes for an episode converges to a somewhat constant trend of around three to four milliseconds. Lastly, of the two working algorithms, the algorithm using potential does indeed produce the shortest state sequence, which is predicted in the hypothesis.

7.4 Sequential State Visits Experiment 2

In the second sequential state visits experiment, the idea is to offer the agent two alternative paths from start to goal. This MDP has no dangerous states, but four

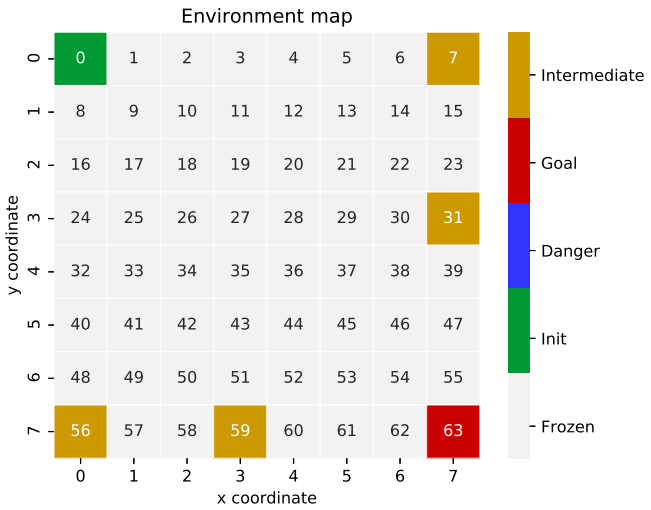


Figure 7.9: Grid world MDP for the second sequential problem.

different intermediate states (7, 31, 56 and 59) and a goal state (63), which are depicted in Fig 7.9.

Here, the dangerous states are eliminated for simplicity, and the intermediate states 7, 31, 56 and 59 are marked in yellow. They are assigned the labels r, u, t and s, respectively. The final state 63 is marked red as before, and is labeled p. In this MDP, there are no dangerous states labeled q.

A specification in which two different sequences can be selected is defined, such that the agent must travel either from state 7 to state 59 and then to 63, or it must select the alternative sequence 56, 31, 63. The LTL formula describing this is

$$\varphi = \Diamond\Big(r \wedge \Diamond(s \wedge \Diamond(p))\Big) \vee \Diamond\Big(t \wedge \Diamond(u \wedge \Diamond p)\Big) \wedge \Box\neg q \tag{7.3}$$

which states that either the sequence r, s, p or t, u, p is observed while never observing q. The colored Büchi automaton that realizes this specification is seen in Fig 7.10, where blue and yellow denote two different accepting frontier sets. The marked state is coloured red, since it has to be part of both accepting frontier. In practice, another state color corresponding to the chosen sequence is applied to the marked state.

Fig 7.10 shows the two alternative sequences that leads to the marked state. As in previous specifications, this automaton is deterministic due to the fact that only one label can be observed in the MDP at a time. For example, if r is observed in q_0, it is assured that no other labels are true. Therefore, it is not possible to travel to q_6, q_0 or q_4 since the conditions for these transitions are not true if r is true. This principle holds for all transitions in the automaton.

Although there are no dangerous states in this experiment, the specification still accommodates for this since there are many ways of ending up in the forbidden specification state even without dangerous states. Should there exist dangerous states, which is actually something that is not known in a "real world" scenario, these must be punished as well as incorrect sequential behavior, even if the sequence is completed. As in earlier experiments, the learning is terminated once the marked

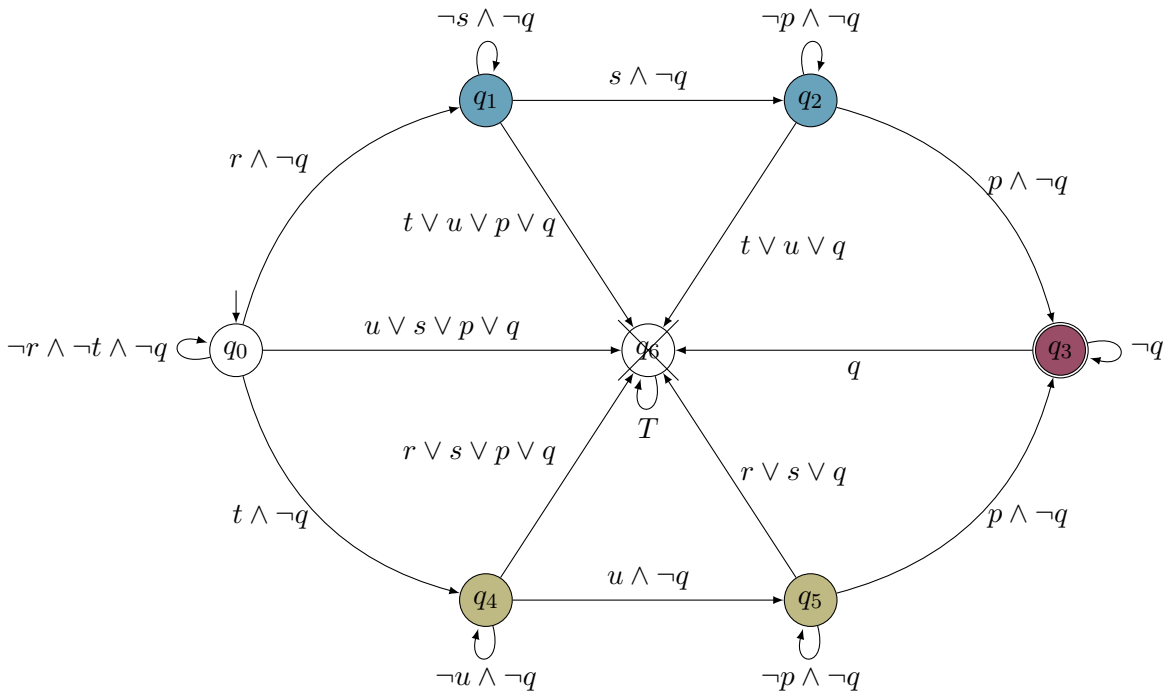


Figure 7.10: Deterministic Büchi automata with one set of colored states per specified sequence.

state is reached.

7.4.1 Setup

This experiment compares classical Q-learning with standard TL constrained RL, again with the accepting frontier and potential extensions applied. The parameters used are displayed in Table 7.3.

7.4.2 Purpose

This specified behavior is very difficult to solve with a standard Q-learning algorithm, as it has no incentive to disregard some intermediate states or choosing to visit others. This must be shown in the results, which is why standard Q-learning is still used as the first algorithm. Furthermore, it cannot in one state receive two

Table 7.3: Parameters for the second sequential experiment.

Exp	Algorithm	No. exp	ϵ_{max}	ϵ_{min}	β	Eps	s/e
a)	Standard Q	10	0.9	0.7	0.0001	30000	100
b)	TL-RL	10	0.9	0.7	0.0001	30000	100
c)	TL-RL w/ AF	10	0.9	0.1	0.0001	30000	100
d)	TL-RL w/ AF & Pot	10	0.5	0.1	0.0001	30000	100

different recommendations to go to different states in a sequence depending on what previous sequence states have been visited. Thus, LTL constrained Q-learning is very much required in this scenario, and this experiment underlines that statement.

Since there are two alternative paths, once a sequence has been chosen, no incentive to visit states of the other sequence are allowed. This means for example that there are different ways of implementing the accepting frontier; in one alternative, all states $q_{1...5}$ have the same color. In this case, either sequence is enforced, as there is only one way to the accepting state in the Büchi. Or, there are two colors, making it possible to switch between a desired sequence. In this case, however, the states cannot be implemented simultaneously because the color mapping is one to one by definition, and as the final state is part of both sequences, the option to select which sequence shall be colored is implemented.

When it comes to the potential function, it is not possible at all to include all intermediate states in the list of sequence states, simply due to the fact that it is only desired to lead the agent through one of the sequences. Although a powerful function if the position of the sequence states are known, potential is naturally prone to produce a faulty policy if this information is incorrect.

7.4.3 Hypothesis

This problem is different to the first sequential problem in a not so apparent way. A problem with TL constrained Q-learning is that it is only awarded for complete sequences, and going to another sequence state in the wrong order implies a transition to a forbidden, and punished, Büchi state. With an alternative sequence as well, the agent is further punished if it visits states in the alternative sequence if it first selected the first sequence. This implies extensive punishment, from which a correct path may emerge. Therefore, the basic TL constrained RL method may have a better chance of performing well for this problem.

Furthermore, it is expected that the accepting frontier performs well, probably better than the standard algorithm, and the same goes for the potential function. Due to the lack of dangerous states, the potential function may have an even greater chance of success than in the previous experiment. Lastly, the chance that standard Q-learning converges to a correct policy is slim, to say the least.

7.4.4 Results

The results of the second sequence experiment are shown in Fig 7.11. An example of a path through the MDP where actions are selected using a converged Q-table is shown in Fig 7.12.

7.4.5 Analysis

The fact that there are more alternative sequences implies the existence of more paths that lead to a forbidden state in the Büchi automaton, since it is only allowed to visit states in one of the sequences. This means that it is easier for the standard TL constrained RL method to find the optimal policy, by excluding all forbidden

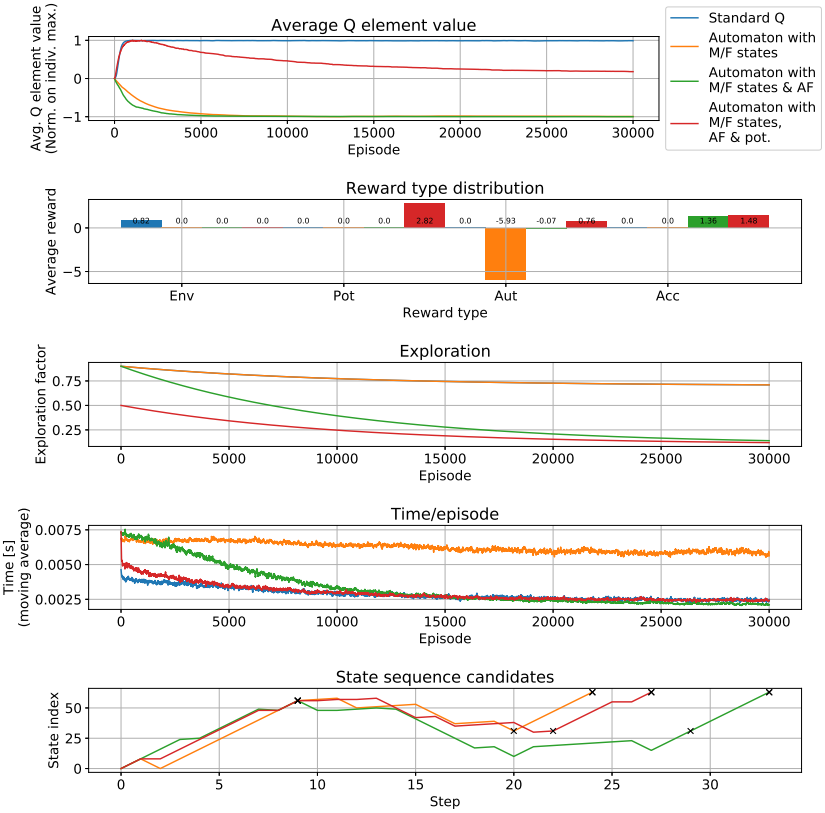


Figure 7.11: Statistics related to the second sequential state visits experiment.

paths. This part of the hypothesis seems to be fulfilled, as seen in the upper right sub figure of Fig 7.12.

When using the accepting frontier, one sequence is selected beforehand, which works. Lastly, using both the accepting frontier and the potential function is also successful, but some lingering behavior around the intermediate states occurs here. However, as in the previous experiment, using potential seemed to introduce a bit more stability to the state sequence; how much of this that is by pure chance is hard to determine. Furthermore, this raises the question of whether or not it is wise to use the potential function for small problems, as it may do more harm than good.

From Fig 7.11, it becomes apparent that both the standard TL constrained RL algorithm and the one that uses the accepting frontier mainly receive negative rewards as the mean value of all Q-elements is negative. The accepting frontier algorithm and the potential algorithm converge slightly quicker than the standard algorithm, which is expected, but they are not as superior as what is initially predicted.

Looking at the reward distributions of Fig 7.11, the role of the potential in the fourth algorithm is played down. This is to ensure that incorrect behaviors are punished; although the potential does not directly make the agent go to incorrect states in the sequence, it may very well disregard going around them. Therefore,

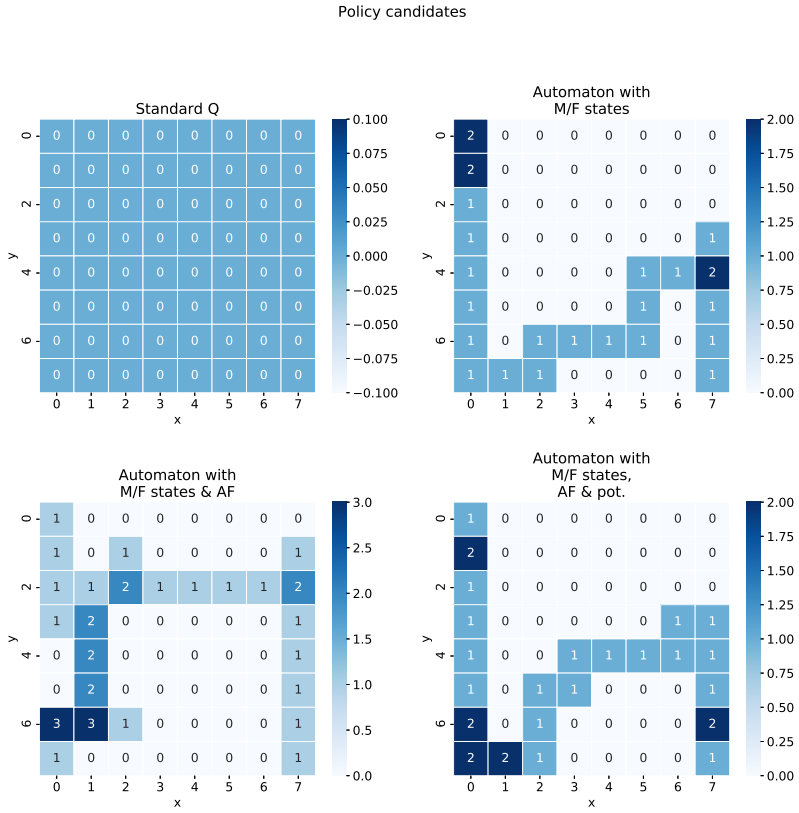


Figure 7.12: Representative state sequences for the second sequential state visits experiment.

the automaton is needed to override the potential function when threading carefully around other sequence states is desired. In the case of the standard TL constrained RL algorithm, the automaton rewards are mainly very negative which is underlines the explanation that extensive wrongdoing can lead to finding the correct path. Lastly, the standard Q-learning algorithm can be seen to converge quickly, albeit to an incorrect policy.

When inspecting the development of time per episode of each algorithm, a correlation with exploration can be observed. This is not unexpected, as unnecessary exploration leads to visiting more states before arriving to the goal state and terminating the episodes. Therefore, it becomes apparent that the exploration should be minimised as much as possible to reduce the demand on time for any algorithm. Lastly, the sequential state sequence plot shows that the standard TL constrained RL algorithm actually produces the shortest state sequence, which is unexpected. Out of the two remaining TL constrained RL algorithms, the one that uses potential produces the next shortest state sequence, a result that was also seen in the previous experiment.

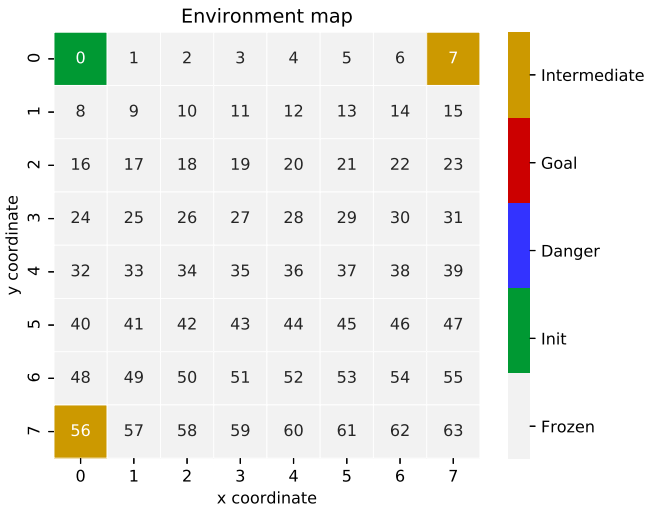


Figure 7.13: Grid world MDP for the first liveness and fairness problem.

7.5 Liveness and Fairness Experiment 1

According to the definition of liveness and fairness in Section 5.2.3, the constraint says that two states in the MDP shall eventually be visited infinitely often. The fairness part of the constraint requires an equal distribution between visits to the first and second marked state. In the first experiment on liveness and fairness, the MDP has the initial state 0 and two intermediate states 7 and 56. In Fig 7.13, these are coloured green and yellow, respectively, and the two intermediate states 7 and 56 have the labels r and p. In this experiment, there are no dangerous states and no goal state either.

Two automata are considered, and these realise the specification

$$\varphi = \square\lozenge p \wedge \square\lozenge r \tag{7.4}$$

which states that both p and r are always eventually observed.

The first automaton that realises the liveness and fairness specification, described further in [4], is a DBA and does have an infinite marked language but clearly enforces a sequential behavior. Although it may realise the LTL-specification, this automaton does not provide a real choice in what order the states are visited, and the agent must observe p before r over and over.

The second, taken from the chapter on liveness and fairness in [2], is a generalized Büchi automaton with an accepting condition requiring infinitely often visiting two different states. In this specification, the order and distribution between state visits are not enforced properties. As usual, the accepting frontier states are blue, and in this case they coincide with the set of marked states denoted by a double circle. Both automata can be seen in Fig 7.14, where the colored (frontier) states are blue and the marked states have double circles. In the GBA to the right, tautologies T are used to immediately transition back to the initial state once a marked state has been visited.

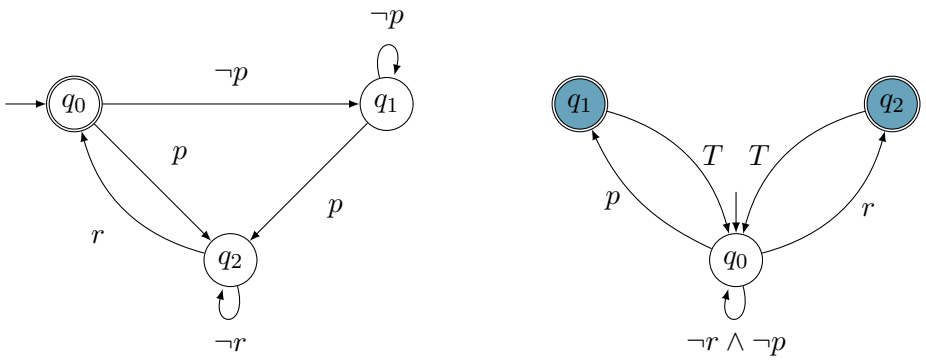


Figure 7.14: DBA for liveness and fairness suggested in [4] with generalized deterministic Büchi automaton proposed in [2].

Moreover, these automata are deterministic under the assumption that if p is true, r cannot be true and vice versa, since an MDP state has at most one label per state. Finally, both automata are formulated for an MDP that has no dangerous states marked q.

7.5.1 Setup

Firstly, this experiment runs the standard Q-learning algorithm followed by the standard LTL constrained RL algorithm, constrained by the DBA seen to the left in Fig 7.14. Secondly, the LTL constrained RL with the accepting frontier function is performed, but where the constraint is realised by the GBA to the right in Fig 7.14. Lastly, the algorithm using both the accepting frontier and the potential is run, also here with the constraint implemented by the GBA in Fig 7.14.

When conducting the liveness and fairness experiment, the parameters seen in Table 7.4 are used.

7.5.2 Purpose

The goal is to investigate the difference in behavior between the two different specification automata realizations. Moreover, while classic Q-learning can simulate a

Table 7.4: Parameters for the first liveness and fairness experiment.

Exp	Algorithm	No. exp	ϵ_{max}	ϵ_{min}	β	Eps	s/e
a)	Standard Q	10	0.9	0.1	0.0001	5000	100
b)	TL-RL	10	0.9	0.1	0.0001	5000	100
c)	TL-RL w/ Büchi 1 & AF	10	0.9	0.1	0.0001	5000	100
d)	TL-RL w/ Büchi 1, AF & Pot	10	0.9	0.1	0.0001	5000	100

sequential behavior from one MDP state to the other if increasing environment rewards are placed sequentially, this strategy becomes problematic when the "bread crum" traces cross paths. In a liveness problem, which would require placing increasing rewards in a circular pattern in the grid world, the agent would once having come full circle be forced to select a reward much smaller than the one before. This would ultimately make the agent stay in the one state giving the largest reward.

In essence, the natural solution to these problems is introducing time or event dependent rewards, removed after a cycle is completed. This is naturally not possible in classical Q-learning, but is effectively what happens in TL constrained RL methods. Considering the complex and sensitive nature of strategically placing rewards in classical Q-learning to find even a partly correct liveness policy, this experiment may showcase both the power and simplicity of LTL constrained reinforcement learning, in relation to classical Q-learning.

7.5.3 Hypothesis

In theory, there is no reason to expect different behaviors from the different specification realisations. However, there is a clear difference between the automata, in that the GBA automatically returns to the initial state after having visited a marked state. While the LTL formula is realized, this may imply difficulties in practice, as the Q-learning problem is still tabular, with the difference being that it has an extra dimension for automaton states. Therefore, for the same MDP state it should not be possible to get two different recommended transitions depending on the automaton state, as this will always be the initial state.

The hypothesis is that the automaton proposed in [4] works in practice, while the automaton in [2] is a more correct liveness and fairness specification. It is possible that the standard Q-learning algorithm can in fact find a policy that when used covers both states, but a cyclic behavior is to expect too much as there is no built in mechanism to handle this; rather, if the standard Q-learning method does find both states, it likely stays in the last state forever.

7.5.4 Results

In the first liveness and fairness experiment, the main results are presented in Fig 7.15. Examples of paths generated by the agent choosing actions from converged Q-tables are shown in Fig 7.16.

7.5.5 Analysis

The hypothesis from Section 7.5 states that the first of the two automata would work in practice because once one of the states in the liveness problem had been visited, the next reward would be mapped to a different automaton state in the Q-table, thus making it possible to recommend going in several different directions for each MDP state. By inspection of the state sequences shown in Fig 7.16, this seems to be a correct prediction, as only the upper right sub plot produces an acceptable behavior. There is a lingering behavior around both of the intermediate states, but it can in the last sub plot of Fig 7.15 be confirmed that the algorithm using the

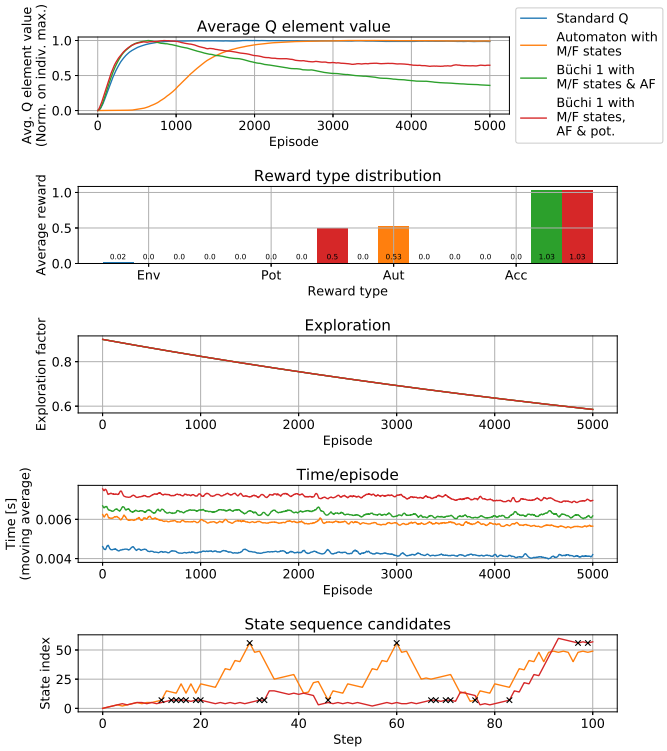


Figure 7.15: The statistics for the first liveness experiment.

first automaton has a cyclic behavior between the two extremities of the MDP state space.

The lower left sub figure of Fig 7.16, corresponding to using the accepting frontier function and getting rewards from marked and forbidden states from the second Büchi automaton, shows that no policy containing both states can be derived from the Q-table. When using the potential, as seen in the lower right sub figure, it is possible to visit both states, but the algorithm seems to first get stuck in the first state and then it breaks away and gets stuck in the other state instead. As both the accepting frontier and the potential focuses on one state at a time, the faulty Q-table mapping explanation seems to be the only reasonable one for this behavior.

In the statistics plots of Fig 7.15, all of the algorithms are evidently collecting positive rewards. Two of the three algorithms that produce unacceptable results have Q-table entries that decay in different degrees. This could illustrate that there are initial rewards being obtained, but after that they stop coming. The algorithm that uses the first automaton converges in a stable way, but is significantly delayed compared to the two faulty ones. The standard Q-learning algorithm is very quick to converge, but it must get stuck in the first intermediate state.

The distribution between rewards in the case of the second automaton and the potential function shows that most rewards come from the accepting frontier, while the potential reward is not as prominent. This is reasonable as the selected policy

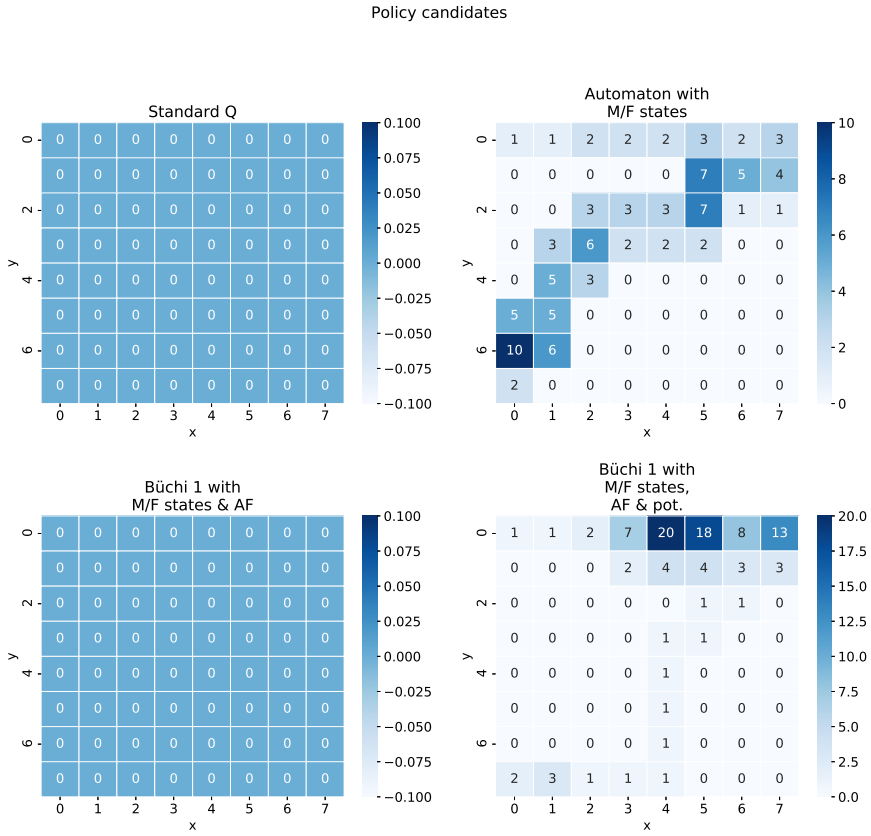


Figure 7.16: Representative state sequences for the liveness experiment.

in most cases does not approach the state that the source for the potential, as can be confirmed in Fig 7.16.

When it comes to average time per episode, the algorithm producing the correct trajectory is also the one with the lowest episode time, not including the standard Q-learning. As there is no goal state and no dangerous states that can terminate an episode, all episodes are 100 steps. Therefore, the difference in time is due to the specific differences in algorithms. However, the differences are not very significant, which can serve as a hint towards that the different procedures in the internal workings of each algorithm themselves take approximately the same amount of time.

Lastly, the chronological plot of the derived state sequences entails that the algorithm using the first automaton successfully alternates between the two states close to six times while the other, poorly working, algorithm alternates once and revisits intermediate states frequently. In essence, it seems as if the core hypothesis is verified; the first automaton works in practice while the second is not so successful.

7.6 Liveness and Fairness Experiment 2

In the second liveness and fairness experiment, the setting is the same as in the previous problem in terms of MDP and automaton, except for one detail. In the

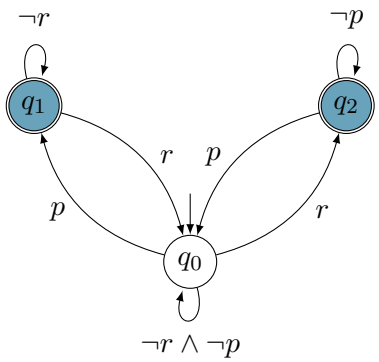


Figure 7.17: A modified version of the GBA from [2].

GBA specifying the LTL formula $\varphi = \square\lozenge p \wedge \square\lozenge r$, previously depicted in Fig 7.14, a small modification to the transitions is made.

In the new automaton, which can be seen in Fig 7.17, the immediate transition to the initial state is conditional on having observed both labels in a sequence. In the modified GBA, returns to the initial state after having been to a marked state are conditional on observing the second label. Transitions leading to the first state are mapped to q_0 in the Q-table while transitions performed afterwards are mapped to one of the other states q_1 or q_2 until the first state has been visited again.

As in the previous experiments, the automaton is completely deterministic due to the assumption that at most one label can be observed in the MDP. Thus, it is for example possible to transition between states q_0 and q_1 if p is true, because this assures that r is false, and vice versa. From the marked states, there are only transitions dependent on one variable, and therefore the other label is a don't care term. Although, if there were transitions from the marked states dependent on the other label, the two labels could not be true at the same time, anyway.

7.6.1 Setup

In this experiment, where the adjusted GBA is used, the standard Q-learning algorithm is compared to the more advanced methods. These consist of the temporal logic constrained reinforcement learning, first without any helper functions, and then with the accepting frontier and lastly with the accepting frontier and the potential function. The experiments are run with the parameters in Table 7.5.

7.6.2 Purpose

The experiment aims to investigate if a small modification to a GBA specification can solve the problem of that the second liveness and fairness specification in practice produces an unfair distribution between visits to the marked states. If so, an interesting limitation to the implementation is discovered.

Table 7.5: Parameters for the second liveness experiment.

Exp	Algorithm	No. exp	ϵ_{max}	ϵ_{min}	β	Eps	s/e
a)	Standard Q	10	0.9	0.1	0.0001	10000	100
b)	TL-RL	10	0.9	0.1	0.0001	10000	100
c)	TL-RL w/ Büchi 2 & AF	10	0.9	0.1	0.0001	10000	100
d)	TL-RL w/ Büchi 2, AF & Pot	10	0.9	0.1	0.0001	10000	100

7.6.3 Hypothesis

Upon first selecting one of the states, all algorithms now receive a reward for going to one of the marked states. After this, the agent must visit the other MDP state before being able to collect another reward. This creates a problem for the algorithm receiving both automaton rewards and accepting frontier; staying in a marked state must not generate unlimited rewards without going to the initial state in between. Therefore, the algorithm using the accepting frontier will only receive rewards from the accepting frontier. This holds for the algorithm implementing both the accepting frontier and the potential function as well.

Regardning the change made in the liveness automaton from [2], the hypothesised outcome can be explained by the use of an example. If the MDP state 7 is visited first, the GBA starts off in the initial state q_0, and Q-table entries at positions corresponding to the states q_0 and s_i are filled until $i = 7$ and the first destination in the liveness problem is reached. Without the modification, the GBA would immediately return to the initial state, and as the potential function and the accepting frontier would now focus on reaching MDP state 56, the Q-table would be filled at indices corresponding to the same GBA state q_0 and arbitrary states s_j of the MDP. Around the first destination, there is a chance that $s_i = s_j$, causing the table to overwrite previous entries that were going to lead the agent to the first destination. With the modification, it is ensured that after visiting the first destination, the GBA state is either q_1 or q_2, making it impossible to overwrite tables in the Q-table.

The hypothesis is now that this improvement helps the Q-learning to map the recommended actions correctly. After having been in the initial Büchi state until the first intermediate MDP state is visited, subsequent actions drawing the agent towards the other MDP state are mapped to another state in the Büchi. In that sense, the Q-table is able to recommend the agent to go in two directions at a specific MDP state, depending on where it came from, which is crucial in liveness type problems.

7.6.4 Results

The results of the second liveness experiment are shown in Fig 7.18. Example paths generated by selecting actions recommended by the different converged Q-tables are

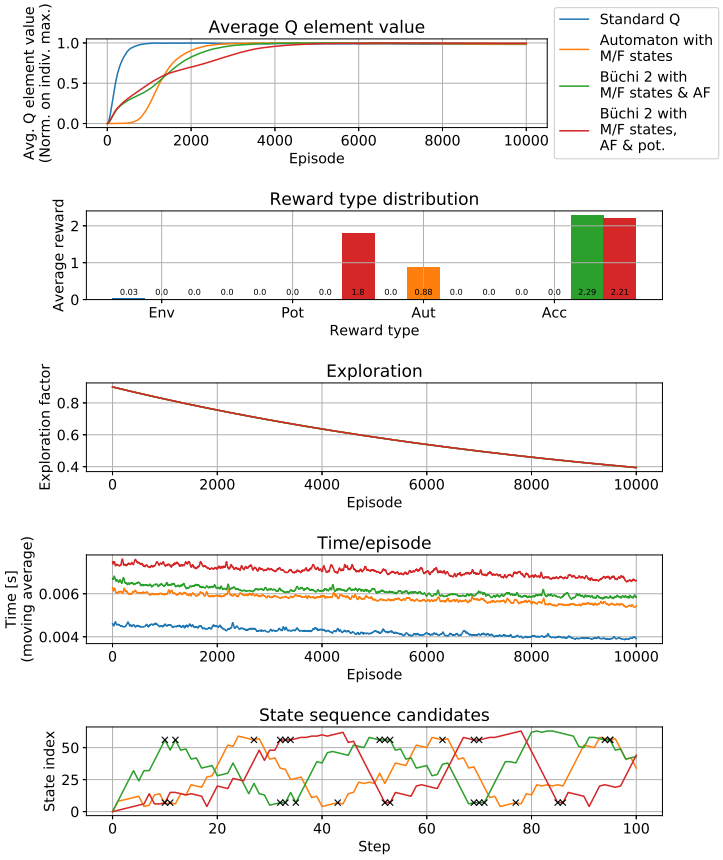


Figure 7.18: Statistics for the second liveness experiment.

displayed in Fig 7.19.

7.6.5 Analysis

The predicted outcome of this experiment is according to the hypothesis in Section 7.6.3 that the modification done to the GBA makes the Q-function map the rewards to different elements in the table. By observing the policies of Fig 7.19, this indeed seems to be the case. In all cases except the case of the standard Q-learning, the states 7 and 56 are visited five to seven times when using the GBA from [2] while the algorithm using the automaton from [4] makes only three visits to each state although having a more defined trajectory with a lower spread.

However, by inspecting the chronological plot in Fig 7.18, it becomes apparent that the algorithms actually produce as many cycles between the states, and the additional number of visits to states 7 and 56 are actually due to fluctuations back and forth to the state during the same cycle. In total, all algorithms produce around 5 alternations between the two states. Again, the standard Q-learning method does not produce a result that contains the two necessary states.

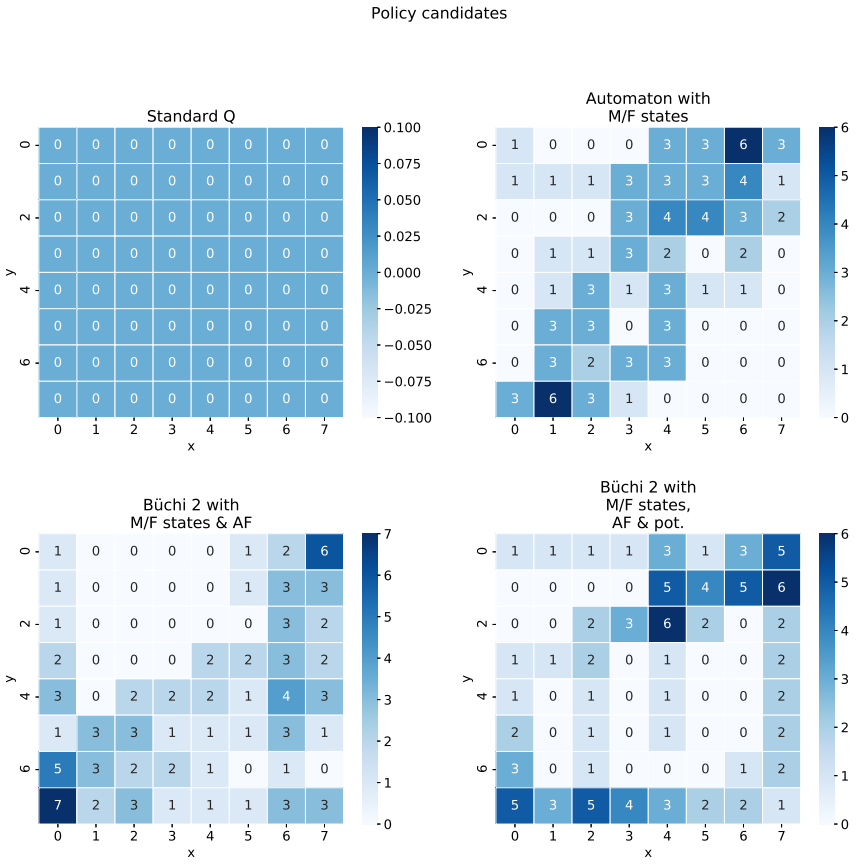


Figure 7.19: Representative state sequences derived from the converged Q-tables.

The convergence of the Q-elements looks quite different between the algorithms. In Fig 7.18, the algorithms using the accepting frontier and the potential function can be seen to start converging faster, but at a slower rate than the algorithm using only the first automaton and rewards for the marked state. It takes around 6000 episodes for all algorithms to converge, and that happens at about the same time. As before, the standard Q-learning algorithm converges to an incorrect solution very quickly, which can be used as an example on how much longer it may takes to find an LTL specified trace through an MDP.

The reward distribution between the algorithms is expected. The standard Q-learning algorithm receives very little in relation to the others, and the first algorithm using an automaton only receives rewards from that automaton. The first algorithm using the modified GBA has its rewards supplied only from the accepting frontier, and the second gets rewards from both the accepting frontier and the potential function. The exploration is the same over all algorithms, and the difference in time is not very significant, but in all algorithms it seems to partly correlate with the exploration as there is a slight decay in all time curves.

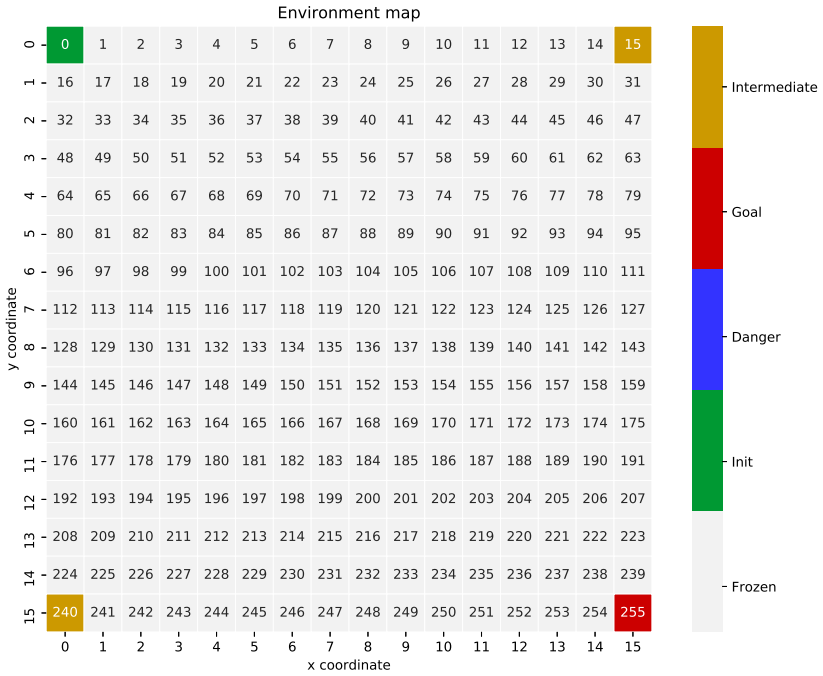


Figure 7.20: 16×16 grid world MDP designed for the large scale sequential problem.

7.7 Sequential State Visits Experiment 3

The third sequential state visits experiment is formulated as a sequential problem exactly as described in Section 7.3, but for a larger 16×16 grid world. In this experiment the agent must learn a policy in which the states 15, 240 and 255 are visited, in that order. The MDP depicted in Fig 7.20 is designed with two intermediate states in yellow, an initial state marked in green and a goal state marked red. State 15 has the label r, state 240 is labeled s and state 255 has the label p. While there are no dangerous states in this formulation, they would bear the label q.

The LTL formula for this specification is

$$\varphi = \Diamond\Big(r \wedge \Diamond(s \wedge \Diamond p)\Big) \wedge \Box\neg q \tag{7.5}$$

which states that eventually, the sequence r, s and p is observed while q is never seen. The specification is the same as (7.2) in Section 7.3 and is thus realised by the DBA in Fig 7.6, which is repeated here in Fig 7.21. Frontier states are coloured in blue, marked state depicted with a double circle and the forbidden state with a dashed circle. This automaton works on MDPs both with and without dangerous states.

Finally, the assumption stating that at most one label per MDP state can be observed holds in this experiment as well, and makes the automaton in Fig 7.21 completely deterministic for the same reasons as in the experiment conducted in Section 7.3.

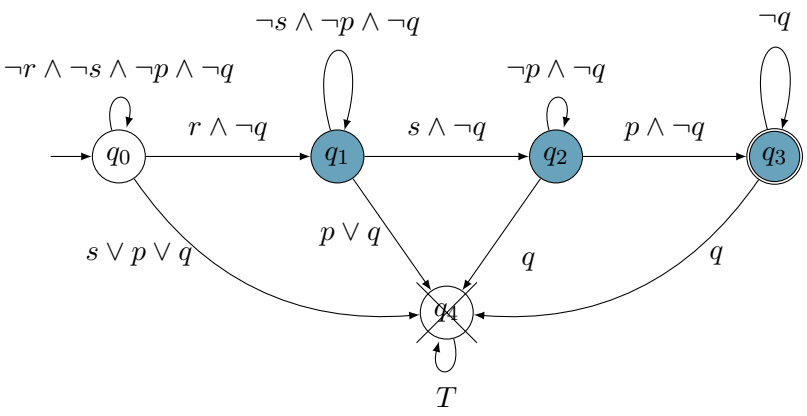


Figure 7.21: Deterministic Büchi realization of a sequential LTL specification.

7.7.1 Setup

The algorithms that are tested are firstly standard Q-learning, then the standard TL constrained RL algorithm, with the specification shown in Fig 7.21. Thirdly, the algorithm with the accepting frontier function working on the set of colored states, also visible in Fig 7.21, is investigated. Finally, the TL constrained RL algorithm with only the potential function extension constitutes the fourth algorithm in the experiment. The parameters that are used in this experiment are found in Table 7.6.

7.7.2 Purpose

The purpose is to find out how the algorithms handle larger state spaces. It is a very fundamental problem that must be solved if any of these methods are to be applied on real world applications which often feature systems with thousands of states.

7.7.3 Hypothesis

According to earlier hypotheses made for lower dimension sequence problems, it is improbable that the standard TL constrained algorithm observes its initial reward since it would have to complete the entire sequence on pure chance according to the

Table 7.6: Parameters for the first sequential experiment on a 16 × 16 grid world MDP.

Exp	Algorithm	No. exp	ϵ_{max}	ϵ_{min}	β	Eps	s/e
a)	Standard Q	10	0.9	0.7	0.0001	10000	100
b)	TL-RL	10	0.9	0.7	0.0001	10000	100
c)	TL-RL w/ AF	10	0.9	0.7	0.0001	10000	100
d)	TL-RL w/ AF & Pot	10	0.9	0.1	0.0001	10000	100

ϵ-greedy action selection. For conventional Q-learning problems it is usually said that an algorithm that visits all states will find the optimal policy, and a desired feature of an RL algorithm is usually for it to converge without having to do this exhaustive task. Here, even visiting all states is not enough, since some also have to be visited in a specific order. This is a daunting task if the state space is large as well.

The accepting frontier effectively rewards the agent for visiting intermediate states, increasing the probability of sequence completion. However, if at some limit the state space becomes large enough, the atomic propositions of the MDP are less frequently observed and therefore the problem must sooner or later, even with the frontier, experience similar reward sparsity problems as without the frontier. The question is if a 16×16 grid world is above or below this limit.

The only method that is invariant to the dimensionality issues is the potential function. The incentive to move towards a goal state is calculated before each step and rewarded, and it is unaffected on the actual distance to the target. Therefore, the hypothesis is that the algorithm that uses the potential function is guaranteed to work, and if the algorithm using the accepting frontier converges it is just a matter of the 16×16 specifically not being large enough to exploit the unavoidable. It is not believed that either the classical Q-learning function or the standard TL constrained RL converges to an acceptable policy for this problem, under reasonable number of episodes and exploration conditions.

7.7.4 Results

The results for the large scale sequential state visits experiment are found in Fig 7.22. Paths that are generated by using the converged Q-table of each tested algorithm are shown in Fig 7.23.

7.7.5 Analysis

According to the hypothesis, it is very improbable that either the standard Q-learning algorithm or the standard TL constrained RL algorithm would find the optimal policy, as the whole state sequence would have to be visited. While the accepting frontier is stated to have the ability of dividing the problem into subsequences that can be completed one after the other, the derived representative policies in Fig 7.23 imply that the distance is too far between the states for this to work. As expected, the only successful algorithm is the one that only used the potential function.

The statistics in Fig 7.22 indeed show that while the algorithm using the accepting frontier does not find an optimal policy, it does receive rewards for visiting intermediate states, just not enough of them to find the complete sequence. This algorithm may still find the optimal policy under different circumstances, such as other exploration settings or a greater number of episodes.

The algorithm using only potential converges smoothly while the standard algorithm actually converges to negative Q-table element values. As a reminder, the plot of the average Q element value is normalised to the individual maximum of

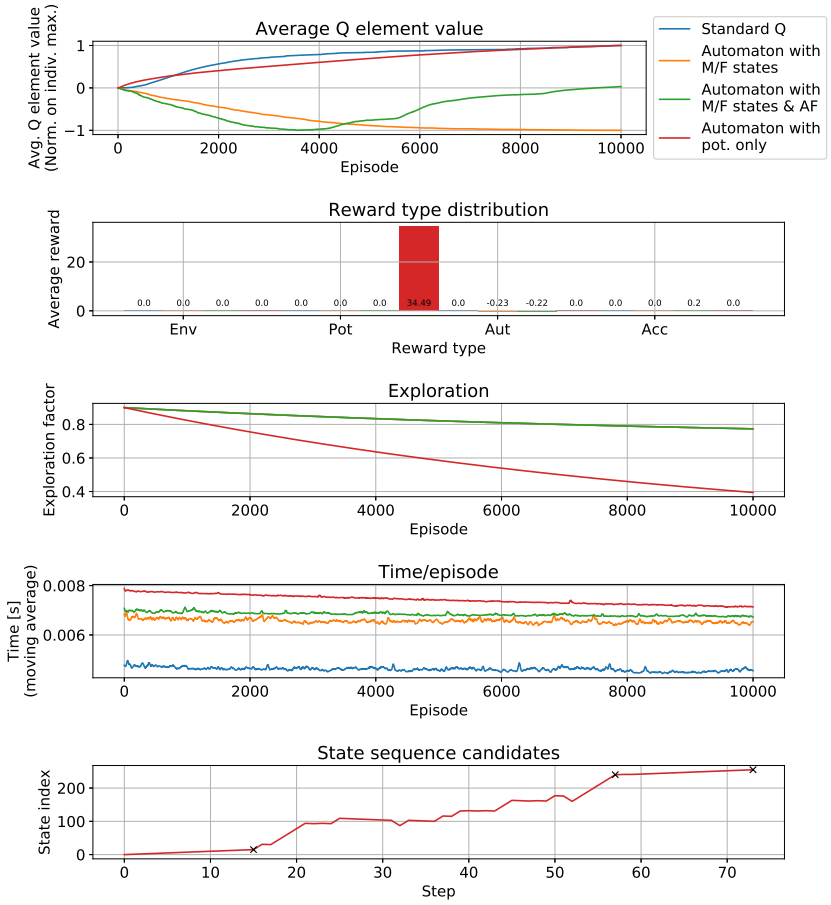


Figure 7.22: Statistics for the third sequential experiment.

the curve, which explains the relation between all the non zero curves and the very small to non-existent rewards in the reward distributions. The Q-element plot shows that the standard Q-learning algorithm converges to an incorrect policy including at least one of the intermediate states, and that the standard TL constrained RL algorithm makes mainly incorrect decisions without finding a correct policy in the end.

As in previous experiments, the difference in time is not very significant for this type of problem, and the problem needs to have much higher complexity before a practically noticeable difference in time is observed. Furthermore, despite the larger risk of endless exploring that takes too much time, significant ability to explore is given to the algorithms that does not use potential, but this is to no avail.

Lastly, the chronological plot confirms that the algorithm using potential indeed visits the states in the correct order, and the representative state sequence takes around 75 steps from start to finish. With this knowledge, the upper limit of 100 steps may be optimistic, and may be one reason for why the algorithm that uses the accepting frontier and the automaton rewards does not find the correct sequence.

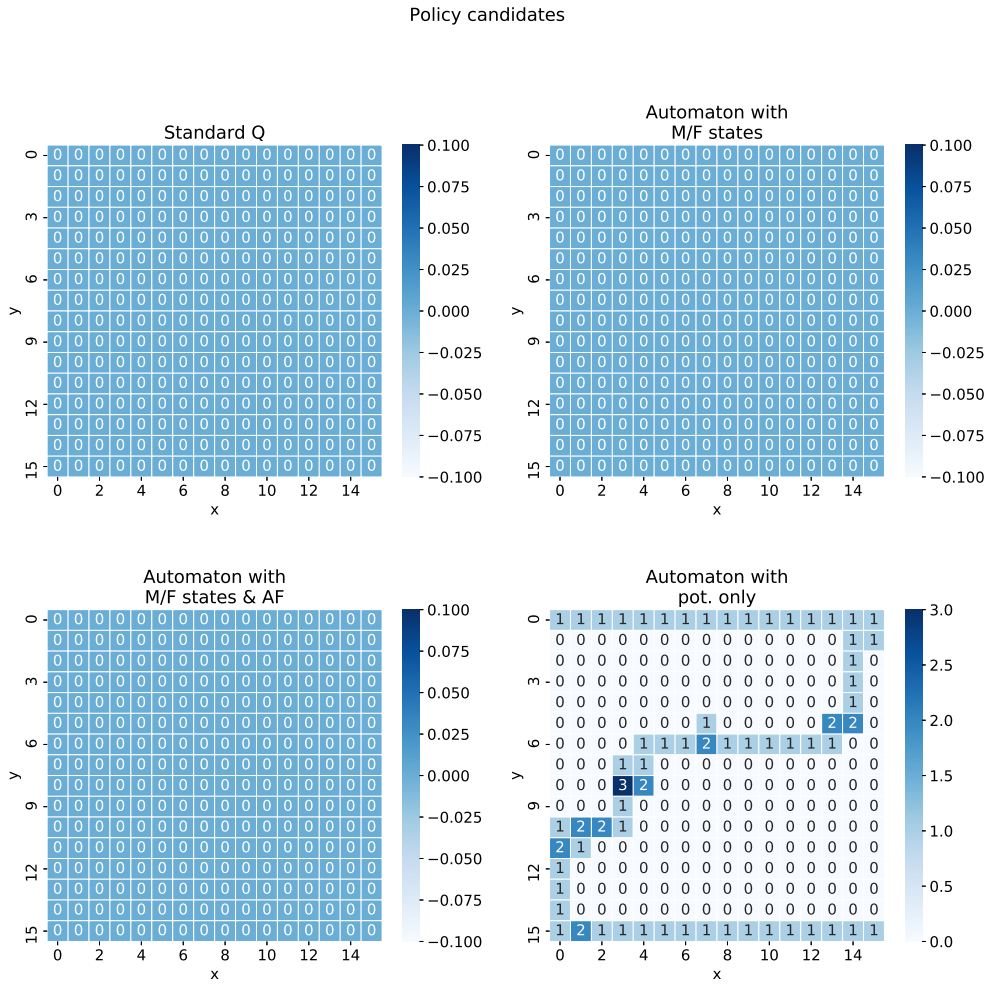


Figure 7.23: Representative state sequences derived from the converged Q-tables.

7.8 Potential for Initial Guiding Experiment

This experiment investigates the idea of using the potential function under a few episodes in the beginning, and then once a trajectory through the MDP has been found once, switch to fine tuning the algorithm by means of an accepting frontier with the standard TL constrained RL method.

This scenario can be motivated by a person who is training for an upcoming championship in the sport of *orienteering*, which is originally a Swedish military training exercise that consists of optimally navigating between checkpoints in a forest using only a compass and a paper map. Being an engineer, the person understands that this task is much better solved by using a GPS. However, the GPS is low on battery and the information calculated by it is therefore very expensive. The engineer can only use the GPS a few times to find an approximate route, but once the battery dies, only the surrounding environment can be used to perfect the route.

The problem is modelled in the exact same way as the sequential problem in

Table 7.7: Parameters for the potential guiding experiment.

Exp	Algorithm	No. exp	ϵ_{max}	ϵ_{min}	β	Eps	s/e
a)	Standard Q w/ AF & Pot, no limit	10	0.1	0.1	0.0001	20000	100
b)	TL-RL w/ AF & Pot, 8k limit	10	0.1	0.1	0.0001	20000	100
c)	TL-RL w/ AF & Pot, 4k limit	10	0.1	0.1	0.0001	20000	100
d)	TL-RL w/ AF & Pot, 25 limit	10	0.1	0.1	0.0001	20000	100

Section 7.7. The 16×16 grid world in Fig 7.20 is used here, and there are still no dangerous states, although the automaton used (visible in Fig 7.21) accommodates for the inclusion of these while still working as intended without dangerous states.

7.8.1 Setup

Four different settings are explored. All algorithms use identical combinations that include the rewards from the standard TL constrained RL method and the accepting frontier. The difference is that for the first configuration, the potential reward extension is present throughout all the episodes, while for the three other algorithms it is switched off after a number of episodes. When this happens, the other reward systems take over and are present until all episodes have passed.

The experiment uses the parameters shown in Table 7.7. With the exception of standard Q-learning, the same algorithm is used, but different episode limits are used on the potential function.

7.8.2 Purpose

The goal is to investigate for how long the potential function is needed, such that it can be used in situations where the information coming from the potential function is so expensive that it cannot be used through the whole learning process.

7.8.3 Hypothesis

It is not far fetched to assume that the policy may be found as soon as the complete sequence is produced. However, there is still a risk that the exploration factor is high enough to send the agent off from the correct trajectory before a sufficient number of Q-table elements that are a reasonable distance from the trajectory have enough non zero elements for the greedy part of the algorithm to work. Furthermore, the same stochastic issue can be induced by the fact that the MDP is probabilistic to start with. All things considered, there is a risk that the potential function must be present until a number of alternative but correct paths have been discovered in the

MDP before the potential can be disengaged.

7.8.4 Results

The results for the experiment concerning initial usage of the potential function are shown in Fig 7.24. In the different cases, the paths taken by using the recommended actions from the converged Q-tables are shown in Fig 7.25.

7.8.5 Analysis

The hypothesised behaviour that is formulated in Section 7.8.3 is that while it is reasonable to assume that the correct path needs to be found only once before the potential could be turned off, but this is not confirmed by the experiment outcome. This may be because of two factors, which are considered in the hypothesis.

Firstly, the ϵ-greedy exploration selects a random action in 10% of the cases, which provides a source for diverging from a previously found path. This is the

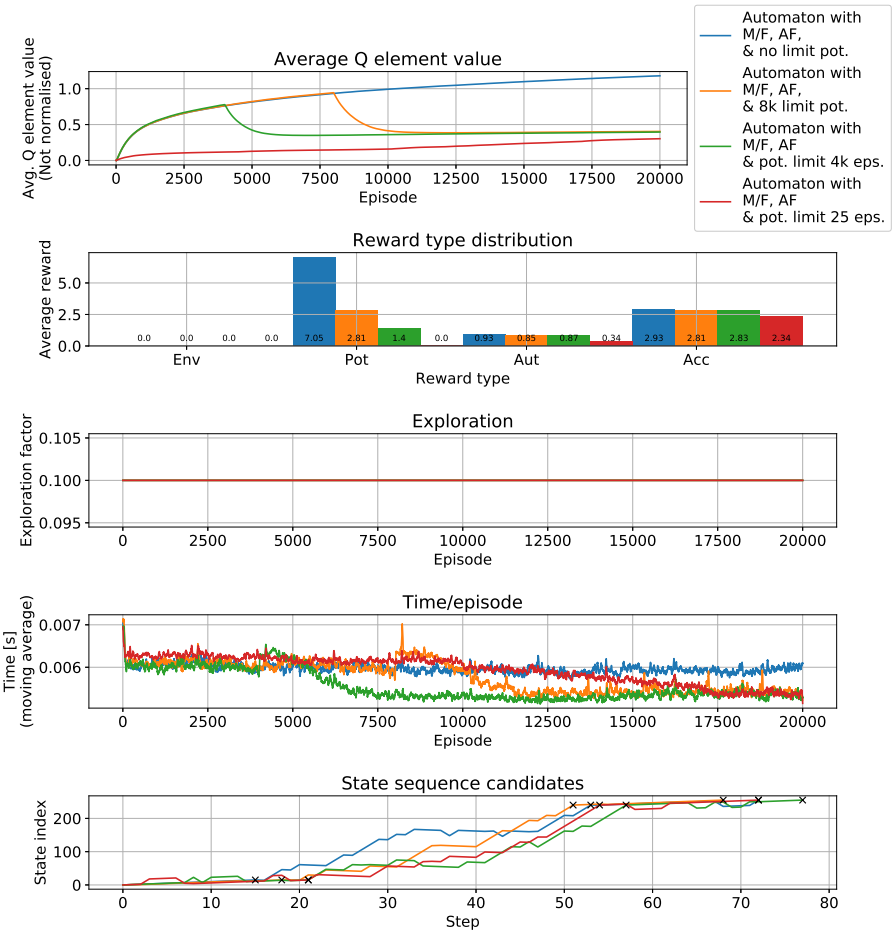


Figure 7.24: Statistics for the initial potential guiding experiment.

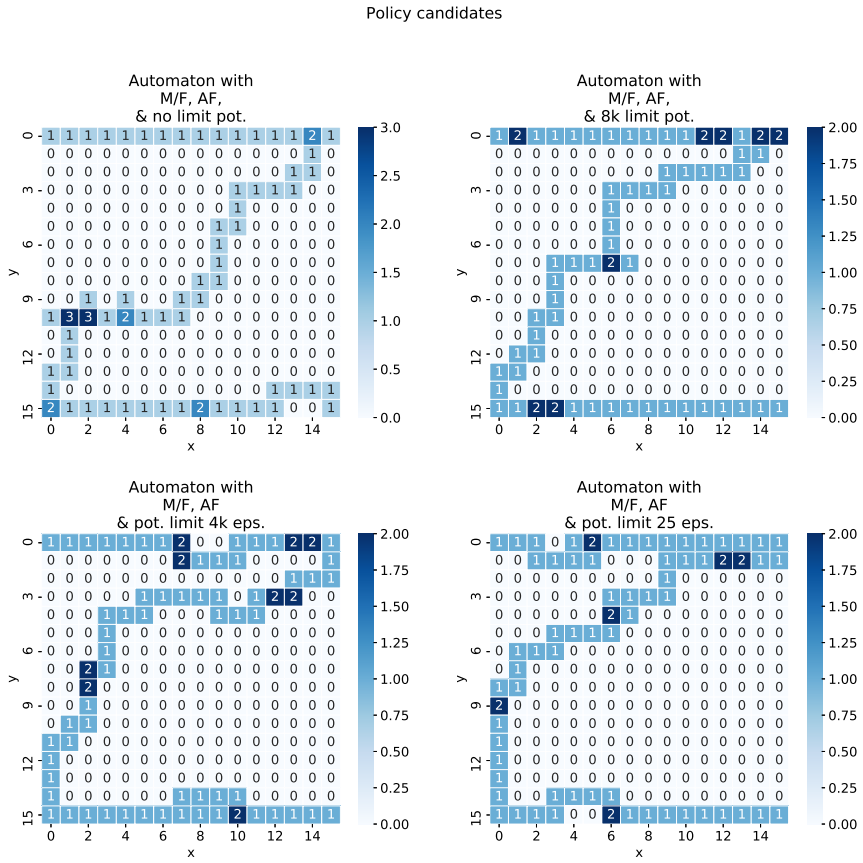


Figure 7.25: Representative state sequences derived using the Q-tables of the initial potential guiding experiment.

purpose of the exploration, as there may exist better state sequences that imply a greater reward, but this also requires that some alternative paths are derived by the use of potential before turning it off. If the algorithm diverges from one of the trajectories, it can regain foothold again and complete the sequence.

Secondly, the same problem arises in probabilistic MDPs, such as this one. It is not certain that the agent will follow one single path each time, even without exploration, as there is a 15% chance that the agent ends up in an unintended state. Therefore, the second part of the hypothesis seems to be correct, in that several correct paths needed to be discovered before turning off the potential.

By inspection of the representative state sequences in Fig 7.25 derived from the converged Q-tables, it can be seen that all algorithms found the correct path. The lowest stable limit that can be achieved is turning off potential after 25 episodes, which is very few in relation to that 20000 were needed for the Q-tables to converge. There is no difference in quality between the traces in Fig 7.25, which is very interesting as a trade off between little work and quality results is usually expected.

The average Q-function element plot in Fig 7.24 shows very clearly how the potential is turned off at the different limits of 25, 4000 and 8000 episodes. The curves first follow the trajectory of the algorithm that does not turn the potential

off, and then they converge to other equilibrium that are formed by receiving the sparse rewards from the accepting frontier and the automaton.

In the plot describing the reward distribution, it is clear that the first algorithm receives much more rewards from the potential, and some from the accepting frontier and the automaton. When the potential is turned off after 8000 episodes, the distribution is more equal between the potential and the accepting frontier while the rewards collected from the automaton are more or less equal. When using a limit of 4000 episodes, there is actually more reward collected from the accepting frontier than from the potential function.

The results show that all algorithms reach the goal state after having completed the correct sequence approximately the same number of times. Furthermore, it can also be seen that the reward coming from the potential is extremely low in the case where it is turned off after 25 of 20000 episodes. Without the potential, the algorithm would not converge (as demonstrated in the previous experiment), but not much is needed to get the agent on the right path. In Fig 7.24, the reward from the potential is not even high enough to display a column, on average.

Moreover, as the potential is independent on exploration, a very low amount of exploration is needed. This exploration is needed only when the other reward mechanisms take over, as there is still a chance that the potential only guides the algorithm to approximately the right region. Regarding time, it still follows the exploration, which is constant, but it becomes apparent that there are other factors that weigh in on this as well. One of these are when the potential is switched off, and it appears that having no potential takes longer on average, but turning it off after a while makes the episode faster compared to always having it on. Finally, in the chronological plot it can be seen how all derived policies find the correct sequence states in order, and they take approximately 75 to 80 steps from start to finish.

7.9 Summary

This chapter covers the experiments conducted in the first part of this project, starting with a description of the basic structure and the fundamental parameters that are consistent throughout all experiments. Furthermore, a note on the grid world sizes is provided, along with a small guide on how to interpret the visualisation of the results.

After this, seven experiments are described using the same structure. Firstly, an MDP and an LTL specification automaton are provided to set the scene. These figures are followed by setup, purpose, hypothesis, result and analysis sections. In each result section, two figures are provided; statistics and representative state sequence. The seven experiments are Safe Navigation to Destination, Sequential State Visits Experiment 1 and 2, Liveness and Fairness Experiment 1 and 2 and finally the two experiments conducted on 16×16 grids, namely Sequential State Visits Experiment 3 and Potential for Guiding Experiment.

8

Conclusions of Part I

In the final chapter of Part I, an evaluation of the first half of the project is conducted in terms of the chosen methods, the results and the overall procedure. Some ideas for future research are provided, along with a set of final remarks regarding the algorithms for temporal logic constrained reinforcement learning that have been evaluated.

8.1 Research Platform Evaluation

The first part of this project includes the construction of a platform on which temporal logic expressions can be implemented in the form of automata. Furthermore, Markov decision process models of so-called grid worlds can be created, with different options regarding state space size, goal state, intermediate states and transition probabilities. There is also the option to create a completely deterministic grid world to evaluate algorithms on if necessary. Moreover, standardised ways of conducting temporal logic constrained tabular Q-learning are implemented, in which it is possible to run several experiments sequentially and derive statistics that are averaged over all runs. Methods for plotting and saving experiment data along with an MDP visualisation function are also implemented.

The decision to build the platform with the OpenAI environment as a base proves to be a successful concept, and is crucial in order to be able to focus on designing experiments. However, there are large quantities of the OpenAI framework that are not used. These include methods for creating other types of MDP environments of different complexity, and there are even functions designed to render three dimensional environments. It should be noted that initially, an attempt was made to create a minimal environment from the ground up, but this attempt was unsuccessful. Several problems related to the implementation arose in later stages, and the decision was made to move over to an implementation that was known to work. With that being said, by careful analysis of the currently used parts of the OpenAI framework, these parts can be lifted out to construct a minimal implementation which is still partly based on OpenAI. An effort can then be made to generalize the experimentation files even further, and an attempt could be made to make the environment structures even more effective.

The implementation of automata models focuses on creating a system that can update a state variable depending on a list of arguments (atomic propositions) that can be either true or false. This implementation is independent on the length of the list of variables to make it more versatile, but there may be ways to both create

a more user friendly way of defining the automata, and creating a more efficient implementation. For example, the current use of lambda statements to formulate the transition conditions is considered the most readable solution, but it is not the most efficient method, and it still requires that a Büchi automaton is already designed. The ideal case would be to prompt the user for the LTL formula, and design an automaton based on that, but that is considered an extensive task and also outside the scope of this project. Moreover, this would probably be even less efficient, which could be a problem. Lastly, implementing non-deterministic automaton is impossible in the current version of the platform. This is a drawback if maximum versatility is to be obtained, but for the most realistic scenarios non-deterministic automata are avoidable.

The way the experiments are conducted in Part I is by manually creating a file for each experiment, with individual plotting functions to accommodate differences between the information that is to be highlighted in the plots of every experiment. There is a need for a more coherent experiment structure in which the user can more easily define new experiments and change the necessary parameters without having to write a whole new Python file.

Over all, the current structure still fits the first half of the project well, and the ability to construct the necessary experiments is not hindered by the way the research platform is implemented. However, the efficiency of the implementation can be improved upon, both in the technical sense and in the way the experiments are implemented.

8.2 Conducted Experiments

There are three categories of experiments in Part I of this project; safe navigation to a destination, sequential state visits and finally liveness and fairness.

8.2.1 Safe navigation to destination

One experiment of this category is conducted, which is is due to the simple structure of the problem. Many of these problems do not require the use of temporal logic and the idea is to show that a simple problem that a standard Q-learning algorithm can solve with environment rewards could just as easily be solved with a temporal logic specification. In such a case, all rewards come from the automaton, or the accepting frontier and the potential function if they are used.

The first conducted experiment is a good starting point, as it establishes a type of ground truth for the rest of the experiments. It can be confirmed that the automata models work as expected and all the mechanisms, ranging from defining a model to deriving and visualizing an optimal state sequence, are working properly.

8.2.2 Sequential state visits

This category of problems is more interesting from a temporal logic perspective. While it can be said that some sequences can be solved with ordinary Q-learning and a strategic placement of environment rewards, the ability to do this quickly

becomes unrealistic for more complex sequences, such as sequential state visits. The sequential problem category is very well suited for showcasing the potential of temporal logic, as it often requires the agent to revisit a state and from there go in a completely new direction because some other state has been visited before.

Different problems from the sequential state visits category are solved in Part I of this project. When it comes to realising a single sequential behavior in an 8×8 grid world, either an accepting frontier function or the additional inclusion of a potential function is recommended. Even for such a small grid world, it is not recommended to use only an automaton if it is suspected that the total number of rewards will be sparsely handed out. However, when selecting between two different sequences, the rewards have been seen to be plentiful enough due to that the agent sees more punishment and is taught to develop a correct sequence by the principle of exclusion. In this case, the accepting frontier function is not necessarily needed, but it may still be implemented to ensure that convergence will occur. It is moreover advised to not use the potential function since it may disagree with the reward sources that have the ability to steer the agent away from bad states. In cases where the state space is large, however, the potential function is absolutely crucial.

The choice to include experiments of this nature is all in all a good one, and some interesting results can be observed, such as the ability of specifying a temporal logic specification that describes two alternative sequential paths to a goal state.

In theory, even more complex sequential problems could be formulated and solved, and the possibilities of this category of problems has definitively not been exhausted by the experiments conducted in this work.

8.2.3 Liveness and fairness

The liveness and fairness experiments serve two purposes. Firstly, they show an example in which two different kinds of automata can be used to solve the same problem in practice. Secondly, they point to an interesting practical problem that is visible only in the context of tabular Q-learning; some automata models may be correct in theory, but are not suited in the application that is TL constrained tabular Q-learning.

The experiments show that it is important that there are a sufficient number of states in the specification automaton, and that the transitions are designed in a correct manner to ensure that the Q-table mapping can be done to avoid overwriting elements. This is a problem that emerges in practical applications, which is why it is not enough to have a correct theory in terms of automata language and the fulfillment of the LTL property.

When it comes to recommending which of the proposed algorithms to use, the advice is similar to the one given for the sequential problems; if the state space is large, use the potential function. In cases where it is not needed, it may actually introduce lingering behavior that can be seen in the chronological state sequence plot in Fig 7.18, which is undesired.

The ability to conduct more complex experiments on liveness and fairness opens up for more experiments, such as liveness and fairness for large scale systems, with more than two states or overlapping paths between intermediate states. This cate-

gory of experiments also has many practical applications, which increase the value of deriving methods that solve this type of problems.

8.3 Answers to Research Questions

The different methods that are evaluated in the first part of this project are based on temporal logic constrained Q-learning, with additional support from a detached accepting frontier function and a potential based reward function. In light of the advantages and disadvantages of these methods, the main conclusions are formulated by answering the research questions. The detailed questions can be found in Section 1.4, and if possible, the answer to these are answered in the following section.

8.3.1 Conflicts and how they are handled

The first question concerns imposing temporal logic constraints on ordinary Q-learning problems, and what type of conflicts arise when doing so. Furthermore, the question asks what the solution to these dilemmas would be if they exist.

In the experiments, it is shown that the additional mechanisms such as the accepting frontier and the potential function must be designed to only focus on one state at a time in complex problems, as there is a risk of getting stuck in intermediate states. The behavior of the agent can in many situations be affected by placing environment rewards in certain states. For example, placing environment rewards in intermediate states would undoubtedly disrupt a sequential problem as the agent would get stuck. However, if negative environment rewards were to be placed in dangerous states, this would probably not induce a behavior that is unfavorable to the specification. The same goes for placing additional rewards in a goal state; the agent would probably be able to go there as long as the environment reward does not cancel the constraint reward.

These situations can be summarised by saying that in some cases, if a standard reinforcement learning problem coincides with the specified behavior, there are normally no problems except that an agent can get stuck in intermediate states. However, if a contradictory behavior is specified, the constraint can easily be violated.

The solution to problems in which temporal logic constraints are formulated on top of a classic reinforcement learning problem is simply to disregard the original problem in terms of environment rewards, and instead reformulate the specification to incorporate any desired behavior that was formulated before. An example of this is the sequential state visit experiments, where the classic RL path planning problem is to go from an initial state to a goal state, and the specified behavior concerns the desired state sequence. Temporal logic has the ability to formulate both safety specifications and the maximising of other value functions, and express them as one complete automaton. If it is impossible to formulate the total problem as one specification, it is very likely that there is no solution to the problem, and that may even become apparent before an attempt is made to solve it.

8.3.2 Methods to improve solutions

The question of what ways knowledge of a model can be incorporated into the solution is investigated in terms of the additional extensions that are made to the standard temporal logic constrained reinforcement learning algorithm.

For example, if it is known that a certain sequence shall be fulfilled, the first step is to implement a Büchi automaton that realises the corresponding LTL specification formula. It is then very straight forward to define a set of colored states for the frontier function, making it possible to realise complex behaviors more or less implicitly from a quite vague formulation. If it is known that a certain type of state shall be visited first, and then another before going to the goal, this information can now be taken into account through the specification. A detailed model or prediction of the MDP is not the only way that information about a system can be taken advantage of.

Furthermore, for certain systems, a basic structure of the MDP may be known, for example that it is a grid world. If a specific location, or approximate region of some states is known and it is desired to visit these states, the potential based function can be used to perform a type of guided learning until an approximate trajectory through the MDP has been found. Then, a temporal logic specification can be used to reinforce a detailed behavior and find an optimal policy that satisfies the specified behavior.

To conclude the answer to the second research question, the extensions to the basic temporal logic constrained reinforcement learning algorithm discussed in this work make it possible to incorporate different kinds of information to produce a solution to a reinforcement learning problem.

8.3.3 Performance of temporal logic constrained reinforcement learning

The final research question regards the performance of the proposed algorithms, and if there are certain situations in which one method is recommended over another. Furthermore, the computational efficiency is of interest, and specifically, the question asks if there are any undesirable trade offs between computational inefficiency and the quality of the solutions produced.

The experiments in Part I of this work do not point towards there being a case where the temporal logic methods take an unreasonable amount of time to converge, at least not to such a degree where the solution cannot be obtained realistically; the algorithms seem fast enough. Furthermore, if a specification is to be implemented in a reinforcement learning problem, the methods described in this work are quite minimal, even though some suggestions have been made that could improve the efficiency.

Regarding recommended methods for specific problems, there is much to be said. When it comes to problems concerning safe navigation to a destination, there are few cases in which a temporal logic solution is recommended. Many of these problems are simple enough to be solved with classical Q-learning, but it is certainly possible to instead formulate an automaton that specifies the behavior if another additional

behavior is needed that is hard or inconvenient to formulate by placing environment rewards strategically. For more complex problems than safe navigation to destination, standard Q-learning is never recommended, as indicated in the experiments. If possible, the effort to strategically place environment rewards manually far exceeds the work to implement an automaton that specifies the corresponding LTL formula.

Sequential problems are very much more convenient to solve with a temporal logic specification, and for most sequences it is recommended to define an accepting frontier to divide the problem into sub sequences, as sequential problems do not scale well at all in terms of the number of states in the sequence and chance of receiving a reward only from marked states.

Liveness and fairness problems may also be solved easily by defining an accepting frontier and formulating the specification as a GBA. However, as the experiments show, there are several ways of solving these problems and care should be taken to make sure that the automaton is compatible with the tabular structure of the Q-function, if such an underlying reinforcement learning algorithm is used.

Finally, the potential function is recommended only if the state space is very large. In some cases it can be shown to disrupt the procedure of the accepting frontier, and it is in many cases not entirely necessary, but crucial when the scale of the problem exceeds the abilities of the accepting frontier.

8.4 Additional Conclusions and Suggestions for Future Work

One of the most interesting conclusions that can be made in the first half of this project is actually the ability to perform guided learning up to an episode limit that is much lower than the limit at which an algorithm converges, and then let a temporal logic specification take over to find the final policy.

As the potential function can be formulated differently to incorporate both exact and approximate knowledge of the location of desired state, this method is very versatile and can thus increase the probability of finding the solution to a temporal logic specified reinforcement learning problem, in situations where the problem is considered to be too large or otherwise unfeasible. This result is determined to be quite general and useful for implementing temporal logic specifications in large scale systems, which is more or less a criterion for the method to gain recognition in industry. In short, potential is a good way to get the most out of the limited knowledge of a system.

Part I of this project can be summarised as exploring some different ways to incorporate temporal logic into reinforcement learning. Certain situations are investigated through experimentation, and the work highlights a few of the issues that may arise when trying to solve these types of problems. In some cases, suggestions and ideas that solve these issues are successfully implemented.

8.4.1 Future work

As mentioned before, the platform itself can be minimised and shaped more towards the type of problems that are of interest in this work. Following in the style of the experiments that are already conducted, more complex problems such as intricate sequential and liveness problems would be interesting to experiment with.

Additional to this, experimenting with multiple specifications would be interesting. This implies that the Q-table has multiple dimensions, one for each automaton that is used simultaneously. This might be an alternative solution to the sparsity problem, as multiple automata could in theory specify different parts of the total specification, and thereby give out rewards more frequently.

Another interesting area of experimentation would be to introduce timed automata as specifications, to formulate another way of ensuring that a specification is completed. This would introduce another way of incorporating information about the model into the algorithm. As an example, it might be known that the agent should reach a specific part of the specification in a maximum number of time steps. This method would then be a property of the automaton, and not the MDP.

The scope of this project is restricted to a basic model free tabular Q-learning setting, but there are many other reinforcement learning algorithms using for example function approximation and neural networks that would be interesting to integrate into the temporal logic specification framework. As the temporal logic functions used in this work are designed to be modular, they are somewhat agnostic towards the underlying reinforcement learning method, and trying different types of RL methods would open up a whole new dimension to the work conducted here.

Part II

Modular Analysis

9
Introduction to Part II

9.1 Background

In the first part of this project, focus lies on evaluating different methods of enforcing constraints formulated in linear temporal logic and realised as Büchi automata. While the technique is shown to be a good way of inducing specified behaviors in how a learning agent navigates through discrete time grid world Markov decision processes, there are situations where additional methods are necessary to ensure that the agent behaves according to the specification.

One of the issues investigated in Part I is formulating specifications for large state space Markov decision processes. More specifically, the problem lies in the agent not being rewarded at the frequency required for fast convergence to a policy. In certain situations, the processes are such that the occurrence of necessary state labels is so sparse that the agent has no incentive to proceed further in a large state space, and gets stuck in a state without finding a policy that meets the specification.

In the first part of this project, methods are formulated to help solve this problem. One of the most efficient methods is the detached accepting frontier, which allows for bread crumb style rewards to be handed out to the agent for fulfilling parts of the specifications. These are generally smaller than the reward of completing the entire specified behavior, and are designed to lead the agent to fulfill the specification rather than settle in an intermediate state.

Another method is the Manhattan potential function, where knowledge of a Markov decision process is utilised to implement a guiding algorithm that leads the agent to complete the specification. The strength of this method is that additional information is integrated into the temporal logic reinforcement learning algorithm, such that large scale problems can be solved efficiently.

Part II of the project addresses yet another way of making the temporal logic constrained reinforcement learning more efficient in certain scenarios, by introducing *modular analysis*.

9.2 Problem Formulation

The focus of the second project part is thus a new improvement method that also utilises additional process information. In [1], the concept is formulated by considering situations in which a Markov decision process can be seen as the synchronous composition of multiple local or modular processes.

In a process composed by multiple sub-processes, [1] suggests that safety can be

guaranteed by drawing conclusions concerning each of these individually, a strategy which is in this work referred to as modular analysis. In a Markov decision process, modular analysis therefore implies making decisions that affect the optimal policy of operation for the complete system by dividing it into smaller systems. As an example, large disadvantageous regions in a path planning problem can be disregarded early, if action sequences that will undoubtedly lead to these regions can be identified long before the region is entered. The problem that is investigated in Part II of this work is thus the control of modular Markov processes, but more specifically, an example of a modular process that consists of a grid world and a queueing process is investigated.

In [12], admission control is described as a type of queueing systems control, where the objective is to decide an optimal policy for admitting arriving customers into a queue. Once admitted, customers are serviced, after which they leave the queue. The queue can be seen as a Markov chain, and the admission control problem is thus a Markov decision process. In many situations, as pointed out in [1], admission control problems have *threshold type solutions*, and such is the case in the control problem discussed in [12]. Threshold type solutions in queueing systems are optimal control policies that switch between two values, depending on the length of the queue. The problem is then reduced to identifying the threshold, after which the optimal control policy can be applied.

The idea that the second part of this project focuses on is thus the temporal logic constrained reinforcement learning of a Markov decision process that can be divided into two different parts, one of which is a queueing system for which admission control should be implemented. The core issue with this problem is that the queueing subsystem has an infinitely large state space, which makes it hard to solve the complete MDP problem using temporal logic constrained reinforcement learning. First of all, this is because the exploration of an infinite state space might be endless. Secondly, the reinforcement learning algorithm considered in this work is a tabular method, and with an infinite state space, this table runs the practical risk of exceeding memory capacity.

The idea is then to find a way to analyse the queueing sub-process locally, and to motivate that it has a threshold type solution. If admissions are only allowed up to a specific queue length, this can be seen as making the state space of the infinite length queueing sub-process finite. By this, the state space of the total process is also reduced. So, by solving the admission control problem for a part of the modular process, the rest of the Markov decision process problem can be solved by using temporal logic constrained reinforcement learning.

Finally, the research questions that are investigated here are mainly the second and third questions already formulated in Section 1.4 of Part I. The second question focuses on how additional information can be used in the learning to make it more efficient, while the third research question deals with the efficiency of temporal logic constrained reinforcement learning. As the problem investigated in this part is motivated by performance improvement due to the usage of additional information, the selected questions are well suited for this part of the thesis.

9.3 Limitations

There are many ways to formulate a Markov decision process that consists of two sub-processes, but in Part II of this project, the focus lies on a process that is already explored in Part I. The main limitation of the second part of this work is therefore that the total Markov process that is considered is a combination of a slippery grid world environment such as the ones investigated in the first part, and a queueing system. The queueing system is a single server infinite capacity queue, known as an $M/M/1$ queue, the details of which are described in later sections.

Another limitation is that as focus lies on modular analysis of the queue, the specifications formulated for the joint process should be simple enough to expect that a behavior can be successfully enforced. However, given that the modular analysis is successful in reducing the state space to a similar size as the processes discussed in Part I, it is reasonable to assume that more complicated specifications could be enforced for the problem considered in Part II, too.

The third and last limitation is similar to the second, but regards additional methods. In the problem considered here, no additional methods other than the modular analysis are considered. This is to isolate modular analysis as the single method under evaluation.

9.4 Changes to Implementation

In Part I, experiments are conducted on a research platform that is constructed based on the OpenAI Frozen Lake MDP implementation in Python. Some of the suggestions for changes formulated in this part include scaling down the method with the hopes of producing a faster algorithm. A change in implementation can also be motivated with the fact that the purpose of the second part of the project is to evaluate a highly specific problem formulated as a grid world in combination with a queueing system.

Therefore, the experiments conducted in Part II of this work are implemented entirely in Matlab. Furthermore, as the goal of producing a very user friendly research platform has proven to be quite redundant, focus now lies on finding ways of representing the problem as effectively as possible. This includes formulating Markov processes using transition and probability lookup tables, and using basic numerical structures to represent the Büchi automata. In total, this should produce a new temporal logic constrained reinforcement learning algorithm that is more efficient than what is implemented in Python.

The final reason for switching to Matlab is that the syntax is not tailored to final product implementations, but to experimentation. Furthermore, producing efficient implementations are done efficiently by using the Matlab Profiler environment to analyse functions and expose bottlenecks.

10

Generalised Semi Markov Processes and the Poisson Distribution

The starting point of Part II of this thesis is to consider the fundamental properties that outline the foundation for the type of modelling that is used here. While discrete time Markov processes make up a cornerstone of Part I, the goal here is to understand how to use *continuous time* Markov processes as models in temporal logic constrained reinforcement learning. The ultimate purpose of this is to use modular analysis on these models to reduce the computational burden of the learning problem by finding solutions to the individual problems using other methods than reinforcement learning.

This chapter starts out by formulating the important Markov property, known from Part I, in continuous time. This property is then applied on stochastic sequences called renewal processes, that are generated by specific stochastically timed automata structers that are known as Generalised semi Markov processes (GSMPs). An important class of renewal processes, the Poisson process, is then formulated which leads to the definition of the Poisson distribution and some of its properties. The first chapter finishes with some practical applications of the Poisson distribution, and a description of what happens when multiple processes are superpositioned.

10.1 Stochastic Timed Automata and GSMP

In this work, focus lies on the modelling of subsystems that behave as Markovian queueing networks, and as with any other modelling framework, there is a need to understand the underlying mechanisms in order to apply it. Therefore, the next sections regard developing a way of modelling stochastic queueing processes.

10.1.1 The Markov property and renewal processes

The starting point is, as in [12], to define a stochastic process $\{X(t)\}$ as

$$X(t) = \{X(t_0), X(t_1), \dots X(t_n)\} \tag{10.1}$$

consisting of all the random variables for which a joint cumulative probability distribution can be formed. Finding a distribution that describes all random variables

in a stochastic sequence is generally difficult, but in the cases when it is possible, the stochastic process $\{X(t)\}$ is valuable for modelling purposes.

As initially mentioned in Section 2.2.1 of Part I but here described as in [12], the Markov property (also called the *memoryless property*) describes that if a stochastic sequence $\{X(t)\}$ is observed to take values from $\{x_0, x_1, \ldots, x_{k+1}\}$ through the history defined by time points $t_0 \leq t_1 \leq \cdots \leq t_{k+1}$, and

$$\begin{aligned} &Pr\{X(t_{k+1}) \leq x_{k+1} | X(t_k) = x_k, \ldots X(t_0) = x_0\} = \\ &= Pr\{X(t_{k+1}) \leq x_{k+1} | X(t_k) = x_k\} \end{aligned} \tag{10.2}$$

then $X(t)$ possesses the Markov or memoryless property and is said to be a Markov process. In this expression, Pr denotes general probability, just like in Section 2.2.1 of Part I.

A specific type of Markov process is the *renewal process*, defined according to [12] as

$$N(t_0 = 0) \leq N(t_1) \leq \cdots \leq N(t_k) \tag{10.3}$$

for the times $t_0 = 0 \leq t_1 \leq \cdots \leq t_k$. $N(t)$ can assume values $0, 1, \ldots$ and is thus often used to count occurrences of events on the time interval $(0, t] = 0 < t \leq t_k$.

To summarise as in [12], the Markov property has in both the continuous and discrete case two properties. Firstly, measurements of the state earlier than the last measurement are irrelevant as the stochastic history of the process is encapsulated in the previous state measurement. Secondly, the time that the process has been in a state is also irrelevant. If the process is in a certain state at time t_1, any predictions that can be made will not be different at time t_2 which is why the second property must hold.

10.1.2 Clock structures and timed automata

To describe the stochastic renewal or counting processes, the focus lies on finding a probability distribution that gives the probability to observe some number in the count. To do this, it might be important to describe when in time the process changes. Here, an intuitive interpretation of the *stochastic clock structure* formally defined in [12] is given as another fundamental element of this modelling framework.

A clock structure is a set $V = \{v_i : i \in E\}$ of clock sequences $v_i = \{v_{i,1}, v_{i,2} \ldots\}$ that maps an event i to time points in v_i. The clock structure can then describe when events occur in time. A *stochastic* clock structure also consists of clock sequences $\{V_{i,k}\} = \{V_{i,1}, V_{i,2} \ldots\}$ for each event i, but these clock sequences are determined from a set of probability distributions $G = (G_i : i \in E)$, where $G_i = Pr\{V_i \leq t\}$. So, although somewhat intricately described, an intuitive view of this is that events i occur at randomly selected points in time according to probability distributions G_i. As opposed to discrete events occurring at equidistant time indices, events that happen according to this procedure can be regarded as discrete events taking place in continuous time.

Using the concept of stochastic clock structures, the *stochastic timed automaton* can be defined. It is, as in [12], an automaton defined by the tuple

$$\langle \varepsilon, \mathcal{X}, \Gamma, P, p_0, G \rangle \tag{10.4}$$

This type of automaton is not used in the same sense as other automata classes described in this work, but rather as a stepping stone to motivate future modelling concepts and assumptions. In (10.4), ε is a set of events, $\mathcal{X}$ is a state space, and $\Gamma(x)$ is a set of feasible events in the state x. Note that this type of state notation is different from that of the MDP structure in Part I; this is to avoid confusion with the type of states considered in Markov processes.

Furthermore, $P(x, e, x')$ is a transition probability mapping, describing the probability of transitioning to state x' by the feasible event e from state x, and $p_0(x)$ describes the probability of the initial state being x. Finally, $G = \{G_i : i \in \varepsilon\}$ is the previously defined stochastic clock structure.

This automaton thus has a timing structure G that will trigger events at random points in time. This will naturally result in stochastic state sequences over time.

At this point, the Generalized Semi Markov Process (GSMP) can be defined as in [12] as a stochastic sequence $\{X(t)\}$, defined as in Section 10.1.1, generated from the stochastically timed automaton given by (10.4). The name of this process stems from that although the time sequences are random, the state probabilities in the GSMP obey the Markov property. Furthermore, in a semi-Markov process, the distributions G_i describing the clock structure are defined externally while the generalised form uses G directly to describe the time sequences.

Next, the relationship between certain stochastic processes and the clock structure of the underlying GSMPs are exemplified by the *Poisson counting process*.

10.2 Poisson Counting Process

As is described in Section 10.1.1, a renewal process can be used to count events. Here, this method is used to count the events of a very simple system, and the point is to expose some valuable results regarding certain types of systems. These results can then be used when modelling stochastic discrete event systems.

10.2.1 Derivation of the Poisson distribution

In this section, the Poisson distribution is derived from a basic definition of a Poisson counting process. Although the definition and derivation of a Poisson process is given in [31], the procedure is described in the style of the derivation given in [12]. This procedure is included because it gives an intuitive understanding of the detailed mechanisms that make up a Poisson process. The described way of thinking about counting processes can be used to determine if any stochastic counting process is possible to model using the Poisson distribution in order to be able to use its associated results.

Start with a discrete event system that has only one event, which can occur at times $0 \leq t_1 \leq \cdots \leq t_k$. The interval length between each time point is arbitrary, as long as the aforementioned order of magnitude holds for the time instances. Now let a renewal process $\{N(t)\}$, first mentioned in Section 10.1.1, describe the number of events that has occurred on the interval $(0, t]$. The count $N(t)$ can only be

incremented at times $\{t_k : 0 \leq t_1 \dots t_k \leq t\}$, such that

$$N(0) \leq N(1) \leq \cdots \leq N(t_k) \leq \cdots \leq N(t) \tag{10.5}$$

where $N(0) = 0$. Furthermore, the number of events that occur on the interval $(t_{k-1}, t_k]$ is denoted $N(t_{k-1}, t_k) = N(t_k) - N(t_{k-1})$.

As the occurrences of the events are stochastic, it follows that the counting of them, $N(t)$, is a stochastic variable, and the aim here is to derive the probability distribution $P_n(t) = Pr\{N(t) = n\}$ for $n = 0, 1, 2 \dots$. This notation describes the probability of having a count of exactly n occurrences on the interval $(0, t]$ in general probability terms, and is thus technically a *probability mass function*.

To obtain the probability distribution, the counting procedure needs to follow three rules, which are often the starting point when describing Poisson processes, such as in both [12] and [31]:

1. Multiple events cannot trigger simultaneously. At time t_k, either one or zero events occur.

2. The event counts during multiple non overlapping time intervals are mutually independent. Having observed some count in the past does not affect the possibility of observing any future count.

3. The probability $Pr\{N(t_{k-1}, t_k) = n\}$ can not depend on a specific time instance t_{k-1} or t_k, but may depend on the length of the time interval. In this sense, the probability is stationary.

The procedure from here is outlined in three steps. Firstly, $Pr\{N(t) = 0\}$ is determined. Secondly, the probability $Pr\{N(\Delta t) = 0\}$ where Δt is a small interval is determined. Then, $Pr\{N(\Delta t) = n\}$ is obtained for $n = 1, 2, \dots$, followed by the expression for $Pr\{N(t + \Delta t) = n\}$. Lastly, letting $\Delta t \to 0$ will reveal the equation $P_n(t) = Pr\{N(t) = n\}$, $n = 0, 1, 2 \dots$, which is solved in the seventh step to conclude the derivation.

Step 1: $\mathbf{Pr\{N(t) = 0\}}$

Continuing on with the derivation from [12], the third rule, which says that the distribution of $N(t_{k-1}, t_k)$ is only dependent on the interval length $s = t_k - t_{k-1}$ and not on specific time instances t_{k-1}, t_k, implies that the expression
$Pr\{N(t_{k-1}, t_k) = n\} = Pr\{N(s) = n\}$ can be rewritten as

$$Pr\{N(t, t + s) = n\} = Pr\{N(s) = n\} \tag{10.6}$$

For time points $0 \leq t \leq t + s$, the count being zero at $t + s$ implies that it was also zero at t, since there can only be positive increments. Therefore,

$$Pr\{N(t + s) = 0\} = Pr\{N(t) = 0 \wedge N(t, t + s) = 0\}$$

Since a rule is that $N(t)$ and $N(t + s)$ are independent,

$$Pr\{N(t + s) = 0\} = Pr\{N(t) = 0\} \cdot Pr\{N(t, t + s) = 0\}$$

Using the fact that $Pr\{N(t, t+s) = 0\} = Pr\{N(s) = 0\}$, the previous expression becomes

$$Pr\{N(t+s) = 0\} = Pr\{N(t) = 0\} \cdot Pr\{N(s) = 0\}$$

For the resulting equation, rewritten as

$$P_0(t+s) = P_0(t) \cdot P_0(s) \tag{10.7}$$

some notes can be made. Firstly, $P_0(0) = 1$ since it is known that the count starts at zero. Secondly, $P_0(t) \leq 1$ for all $t \geq 0$. Then, by assuming that $P_0(t+s)$ and $P_0(s)$ are differentiable, differentiating (10.7) with respect to s yields

$$\frac{d}{ds}P_0(t+s) = P_0(t) \cdot \frac{d}{ds}P_0(s)$$

By then setting $g'(s) = \frac{d}{ds}g(s)$ and $s = 0$, the previous equation becomes

$$\frac{d}{ds}P_0(t) = P_0(t) \cdot P_0'(0) \tag{10.8}$$

Since $P_0(0) = 1$, a candidate solution of the differential equation (10.8) is

$$P_0(t) = e^{P_0'(0)t} \tag{10.9}$$

Next, $P_0(t) \leq 1$ for all $t \geq 0$ is satisfied if the candidate solution (10.9) is updated to

$$P_0(t) = e^{-P_0'(0)t} \tag{10.10}$$

If $\lambda = P_0'(0)$, the final expression for $P_0(t)$ is

$$P_0(t) = Pr\{N(t) = 0\} = e^{-\lambda t} \tag{10.11}$$

where $\lambda > 0$. It can also be verified that $P_0(t+s) = P_0(t) \cdot P_0(s)$ as it holds that $e^{-\lambda(t+s)} = e^{-\lambda t} \cdot e^{-\lambda s}$. Again, what is derived here is an expression for the probability that no event occurs in the interval $(0, t]$.

Step 2: $\mathbf{Pr\{N(\Delta t) = 0\}}$

An interval Δt is defined so that (10.11) for the interval Δt has the Taylor series

$$Pr\{N(\Delta t) = 0\} = e^{-\lambda \Delta t} = 1 - \lambda \Delta t + \lambda^2 \frac{\Delta t^2}{2!} - \lambda^3 \frac{\Delta t^3}{3!} + \dots$$

As the higher order terms becomes negligible as $\Delta t \to 0$, they are collected in the function $o(\Delta t)$ which has the property $o(\Delta t)/\Delta t \to 0$ as $\Delta t \to 0$, and the Taylor expansion becomes

$$Pr\{N(\Delta t) = 0\} = 1 - \lambda \Delta t + o(\Delta t) \tag{10.12}$$

which concludes the second step.

Step 3: $\mathbf{Pr\{N(\Delta t) = n\}}$

The first rule of the Poisson counting process is that two events cannot occur at the same time instance, but what about a time interval? The key step is to let the Δt approach zero such that the probability of having two or more events occurring during Δt becomes so small that $Pr\{N(\Delta t) = n\} = o(\Delta t)$ for $n = 2, 3 \ldots$.

The probability of the count being equal to any non negative integer is always one, expressed mathematically as $\sum_{n=0}^{\infty} Pr\{N(\Delta t) = n\} = 1$, and this combined with the probability of having multiple events in Δt being negligible results in

$$\begin{aligned} Pr\{N(\Delta t) = 0\} &= 1 - \lambda \Delta t + o(\Delta t) \\ Pr\{N(\Delta t) = 1\} &= \lambda \Delta t + o(\Delta t) \end{aligned} \tag{10.13}$$

if combined with (10.12). Thus, the only nonzero probabilities of events occurring during Δt are those of $N(\Delta t) = 1$ and $N(\Delta t) = 0$.

Step 4: $\mathbf{Pr\{N(t + \Delta t) = n\}}$

For two intervals $(0, t]$ and $(t, t + \Delta t]$, consider the possible, mutually exclusive, situations leading up to $N(t + \Delta t) = n$, namely

$$\begin{aligned} &N(t) = n \text{ and } N(t, t + \Delta t) = 0 \\ &\quad \text{or} \\ &N(t) = n - 1 \text{ and } N(t, t + \Delta t) = 1 \\ &\quad \text{or} \\ &\quad \vdots \\ &N(t) = 0 \text{ and } N(t, t + \Delta t) = n \end{aligned}$$

The probability of n events occurring in Δt can then be expressed as

$$Pr\{N(t + \Delta t) = n\} = \sum_{j=0}^{n} Pr\{N(t, t + \Delta t) = n - j \text{ and } N(t) = j\} \tag{10.14}$$

The second rule of the Poisson counting process is that the counts of two non overlapping intervals are independent, so

$$Pr\{N(t, t + \Delta t) = n - j \text{ and } N(t) = j\} = Pr\{N(t, t + \Delta t) = n - j\} \cdot Pr\{N(t) = j\}$$

As $Pr\{N(t, t + \Delta t) = n - j\} = Pr\{N(\Delta t) = n - j\}$, (10.14) can be rewritten as

$$Pr\{N(t + \Delta t) = n\} = \sum_{j=0}^{n} Pr\{N(\Delta t) = n - j\} \cdot Pr\{N(t) = j\} \tag{10.15}$$

In this expression, events $N(\Delta t) = n - j$ have probability zero except in two cases, either $[N(\Delta t) = 0$ and $N(t) = n]$ or $[N(\Delta t) = n - 1$ and $N(t) = 0]$. Hence, (10.15) reduces to

$$\begin{aligned} Pr\{N(t + \Delta t) = n\} =& Pr\{N(\Delta t) = 0\} \cdot Pr\{N(t) = n\} + \\ &+ Pr\{N(\Delta t) = 1\} \cdot Pr\{N(t) = n - 1\} + o(\Delta t) \end{aligned}$$

Using the derived probabilities in (10.13) with the definition $P_n(t) = Pr\{N(t) = n\}$ in this expression

$$P_n(t + \Delta t) = (1 - \lambda\Delta t)P_n(t) + \lambda\Delta t P_{n-1}(t) + o(\Delta t) \tag{10.16}$$

is obtained for integers $n > 0$.

Step 5: Derive equation for $\mathbf{P_n(t) = Pr\{N(t) = n\}}$

Subtracting $P_n(t)$ from both sides of (10.16) and then dividing by Δt where $\Delta t \to 0$ while remembering that $o(\Delta t)/\Delta t \to 0$ as $\Delta t \to 0$ results in the difference-differential equation

$$\begin{aligned} \lim_{\Delta t \to 0} \frac{P_n(t + \Delta t) - P_n(t)}{\Delta t} &= -\lambda P_n(t) + \lambda P_{n-1}(t) + \frac{o(\delta t)}{\Delta t} \\ \iff \frac{dP_n(t)}{dt} &= -\lambda P_n(t) + \lambda P_{n-1}(t) \end{aligned} \tag{10.17}$$

for integers $n > 0$.

Step 6: Solve the equation for $\mathbf{P[N(t) = n]}$

Using the already derived result $P_0(t) = e^{-\lambda t}$ in (10.17) for $n = 1, 2 \ldots$ results in

$$\begin{aligned} \frac{P_1(t)}{dt} &= -\lambda P_1(t) + \lambda e^{-\lambda t} \implies P_1(t) = (\lambda t)e^{-\lambda t} \\ \frac{P_2(t)}{dt} &= -\lambda P_2(t) + \lambda e^{-\lambda t} P_1(t) \implies P_2(t) = \frac{(\lambda t)^2}{2} e^{-\lambda t} \\ &\vdots \end{aligned}$$

which leads to the expression

$$P_n(t) = \frac{(\lambda t)^n}{n!} e^{-\lambda t} \tag{10.18}$$

This is the expression for the Poisson distribution, which gives the stationary probability that a Poisson counting process has reached n during the time interval of length t.

10.3 The Poisson Distribution

As is shown in Section 10.2.1, the Poisson counting process describes the counting of discrete events during a period of time denoted by the interval $(0, t]$. Given that the count denoted $N(t_k)$ is a random variable incremented at times $\{t_k | 0 < t_k \leq t\}$, it turns out that the probability of observing the count when it is exactly n is given by

$$P_n(t) = \frac{(\lambda t)^n}{n!} e^{-\lambda t} \tag{10.19}$$

Processes that behave as Poisson counting processes are commonly referred to as Poisson processes, and the probability of observing a count is determined by the discrete Poisson probability distribution written as (10.19).

It is important to emphasise that t denotes the number of *unit intervals* of time, and not a specific time instant. The unit time interval can describe any time period, for example c minutes or c hours where c is a positive, not necessarily integer, scalar. In this setting, t can denote multiples, but also fractions, of the given unit interval.

The rate parameter λ denotes the average count per unit time interval, and can as in (10.21) be viewed as the expected count during one unit time interval. Note here that as the Poisson distribution characterises a random variable, the meaning of λ being the expected count is that observing the count $n = \lambda$ has the highest probability, just as the expected value of other probability distributions. In one unit time interval the probabilities to observe higher or lower exact counts than λ are still significant, albeit decreasingly so.

In many cases, only one unit time interval is interesting, and (10.19) is evaluated at $t = 1$. The probability of the count $N(t)$ being exactly equal to n, given that $N(t) = \lambda$ is expected after one unit time interval, (10.19) writes as

$$P_n = Pr\{N(t) = n\} = \frac{(\lambda)^n}{n!}e^{-\lambda} \tag{10.20}$$

which is how the Poisson distribution is often described in literature.

10.3.1 Properties of the Poisson distribution

Two properties of the Poisson distribution that are also obtained from [12] are the expressions for the expected value and the variance of the random variable $N(t)$, given as

$$\begin{aligned} \mathbb{E}[N(t)] &= \sum_{n=0}^{\infty} nP_n(t) = \cdots = \lambda t \\ Var[N(t)] &= \sum_{n=0}^{\infty} n^2 P_n(t) = \cdots = \mathbb{E}[N(t)] = \lambda t \end{aligned} \tag{10.21}$$

The complete, although short, derivations of the expressions in (10.21) are omitted here but can be found in [12].

Another important feature of the Poisson distribution is the question of how the *inter-arrival times*, the time between each count increment, are distributed. Since it is known how the probabilities of the count variable $N(t)$ in a Poisson process are distributed, the time intervals between arrivals are also random variables for which a probability distribution can be derived.

This derivation, also formulated in its entirety in [12] but omitted here, results in that the probability of the interarrival time variable v taking a value in relation to the time interval length t is described as

$$G(t) = Pr\{v \leq t\} = 1 - e^{-\lambda t} \tag{10.22}$$

for $t \geq 0$. This is the cumulative density function (CDF) of the *exponential* probability distribution and its probability density function (PDF) is obtained by differentiating the CDF with respect to time, which gives

$$g(t) = \lambda e^{-\lambda t} \tag{10.23}$$

Lastly, if a counting process is a Poisson process, it is implied that the interarrival times are exponentially distributed, and vice versa.

10.3.2 Practical uses of the Poisson distribution

To get an intuitive view of how the Poisson distribution can be used practically, an example is in order. Assume that during the unit time interval $t = 1$ day at a hospital emergency room, patients arrive according to a Poisson process with rate parameter $\lambda = 20$. In simplified terms, the patients are expected to arrive one by one, and during one day the most probable exact total patient count is 20. What are then the probabilities that exactly 17, 20 and 23 patients have arrived? According to (10.20),

$$Pr\{\text{No. patients} = 17\} = \frac{20^{17}}{17!}e^{-20} = 0.076$$
$$Pr\{\text{No. patients} = 20\} = \frac{20^{20}}{20!}e^{-20} = 0.088$$
$$Pr\{\text{No. patients} = 23\} = \frac{20^{23}}{23!}e^{-20} = 0.067$$

As can be seen, the probabilities for exact patient counts are quite low by their own; while the probability to find λ patients is the highest, it is only at around nine percent.

What, then, is the probability that *at most* 20 patients have arrived? This probability is obtained by summing over the probabilities of individual exact counts up to 20, which gives

$$Pr\{\text{No. patients} \leq 20\} = \sum_{n=0}^{20} \frac{(20)^n}{n!}e^{-20} = 0.560$$

This total probability is larger. Intuitively, the probability of observing *any* number of patients is the probability of observing a exactly one, plus that of exactly two, and so on up to the infinite integer limit, giving the total probability one.

Lastly, what is the probability to observe at most 30 patients arriving during the course of *two* days? And what is the probability that between four and eight patients arrive after lunch, a time period corresponding to half a day? To answer the first question, (10.19) is used with two unit time intervals and the same value for λ as before to obtain

$$Pr\{\text{No. patients} \leq 30 | t = 2 \text{ days}\} = \sum_{n=0}^{30} \frac{(20 \times 2)^n}{n!}e^{-20\times 2} = 0.062$$

During two time periods, 40 patients are expected. Given that the computation above will result in a total probability of 0.542 if 40 is set as the upper patient

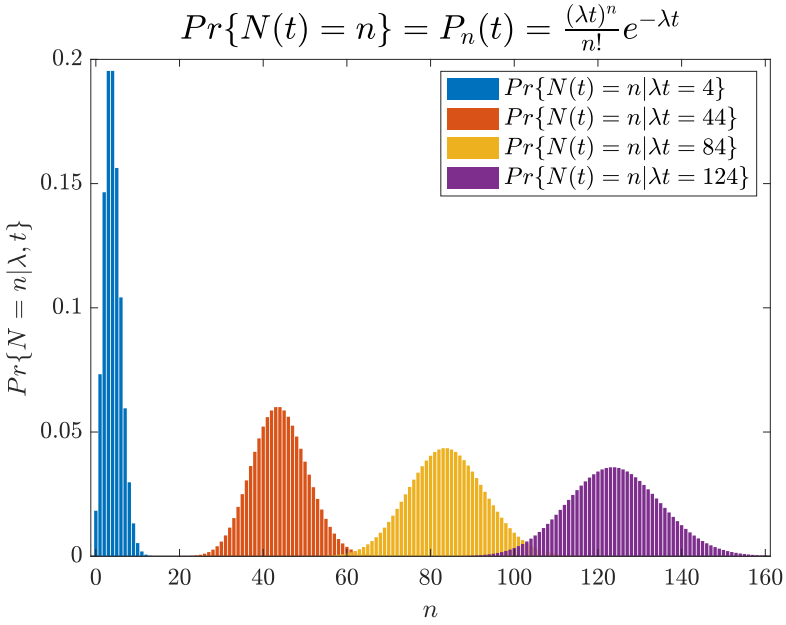


Figure 10.1: The Poisson distribution for different λ and t.

limit, it is implied that changing the parameters may yield seemingly unintuitive results, which further underlines the importance of technically evaluating stochastical models instead of guessing.

The second question is answered in the same manner, but the time interval is now halved. The Poisson distribution is stationary, which means that it is independent of specific time instances, and only depends on the length of the unit time interval. Thus, it does not matter if the interval is before or after lunch, and the probability that between four and eight patients are waiting is given by

$$Pr\{4 \leq \text{ No. patients waiting} \leq 8 | t = 0.5 \text{ days}\} = \sum_{n=4}^{8} \frac{(20 \times 0.5)^n}{n!} e^{-20\times 0.5} = 0.323$$

The Poisson distribution can be visualised for different parameters to help gain an understanding of how they affect the distribution. In Fig 10.1, it is shown how the Poisson probability distribution changes when different values for λ and t are selected.

10.4 Superposition of Multiple Poisson Processes

In the fundamental definition of a Poisson process, recall that the inter-arrival (sometimes called intervenent) times between events are by definition exponentially distributed. But it might also be interesting to see what happens if multiple Poisson processes are observed together.

Now, as explained in [12], consider m mutually independent Poisson processes. These can be seen as renewal or counting processes with rates $\lambda_1 \dots \lambda_i \dots \lambda_m$, each with a random variable $N_i(t)$ denoting the count of each independent process. The intervenent times are then stochastic variables $Y_1 \dots Y_m$.

With all m processes acting together at some point in time (the superposition of the m processes), the time until any next event is the smallest intervenent time being Y^*. Each of the intervenent times is exponentially distributed as

$$Y_i \sim 1 - e^{\lambda_i t} \tag{10.24}$$

Then, Y^* is distributed as

$$Pr\{Y^* \leq t\} = 1 - Pr\{Y^* > t\} = 1 - [\min_i\{Y_i\} > t] \tag{10.25}$$

where $Pr\{\cdot\}$ denotes probability in general terms. The minimisation term is an event, equivalent to the event $Y_1 > t \wedge Y_2 > t \wedge \cdots \wedge Y_m > t$, according to [12]. These processes are independent, and according to the fundamental definition of stochastical independence, m processes $Y_1 \ldots Y_m$ are independent iff

$$\begin{aligned} &Pr\{Y_1 = y_1 \wedge Y_2 = y_2 \wedge \cdots \wedge Y_m = y_m\} = \\ &= Pr\{Y_1 = y_1\} \cdot Pr\{Y_2 = y_2\} \ldots Pr\{Y_m = y_m\} \end{aligned} \tag{10.26}$$

Then, (10.25) can be expressed as

$$\begin{aligned} Pr\{Y^* \leq t\} &= 1 - \prod_{i=1}^{m} Pr\{Y_i > t\} \\ &= 1 - \prod_{i=1}^{m} e^{\lambda_i t} \end{aligned} \tag{10.27}$$

where the last step can be made by using (10.24).

If the sum of all Poisson parameters is expressed as $\Lambda = \sum_{i=1}^{m} \lambda_i$, the exponential term in the product sum can be expressed in terms of Λ, and then the distribution of the intervenent times of all the m Poisson processes together making up the joint system is expressed as

$$Pr\{Y^* \leq t\} = 1 - e^{-\Lambda t} \tag{10.28}$$

So, a process that is a superposition of m independent Poisson processes has exponentially distributed intervenent times with parameter Λ. This implies that the superimposed m Poisson processes is also a Poisson process, with parameter Λ.

Lastly, the above derivation of the distribution of the intervenent times of the superposition is based on that events from all m Poisson processes are occuring at any time. Thus, the count of all the independent events, which summed together form the count of the superimposed Poisson processes is

$$N(t) = N_1(t) + N_2(t) + \cdots + N_m(t) \tag{10.29}$$

The conclusion is that the intervenent times are exponentially distributed with parameter Λ, which implies that the superimposed Poisson process is also a Poisson process with rate Λ. Therefore, another way of viewing the superposition is that the sum of all the counts from m independent Poisson processes is distributed according to the Poisson distribution

$$Pr\{N(t) = N_1(t) + \cdots + N_m(t) = n\} = \frac{(\Lambda t)^n}{(n!)} e^{-\Lambda t} \tag{10.30}$$

where $\Lambda = \sum_{i=1}^{m} \lambda_i$.

10.5 Summary

The theory behind the models used for modular analysis in Part II of this work starts with the definition of the Markov, or memoryless, property in continuous time. This property is used to describe how discrete events can occur at any point in continuous time, and a stochastically timed automaton class called the GSMP is used to generate such events.

Next, an example of how events generated from the GSMP can be counted is the Poisson counting, or renewal, process. Based on three fundamental assumptions, the discrete probability mass function called the Poisson distribution is derived. This function gives the probability of the count assuming a specific integer value, given that the rate of event occurrences per unit time interval is known.

For a variable distributed according to the Poisson distribution, the two basic stochastical measurements of expected value and variance are mentioned. After this, some practical examples of how the Poisson distribution can be used are given in the form of patients arriving to a doctors office.

Lastly, superposition of mutually independent Poisson processes is shown to also be a Poisson process with a rate parameter that is the sum of all the individual process rates.

11

Modelling with Discrete and Continuous Markov Chains

The key take-away from the previous discussions of GSMP and Poisson processes is that a Poisson process, such as the one derived in Section 10.2.1 and the one illustrated in the waiting room example of Section 10.3.2, can be viewed as a Generalised Semi-Markov Process. This process is called generalised since the probability distributions with which event occurrences are described can be chosen. In this chapter, focus lies on how the relationship between the Poisson process and the GSMP with exponentially distributed intervenent times makes it possible to describe continuous time Markov chains.

The goal is to find how a Poisson process can be modelled as a continuous time Markov chain by determining a set of states, transition rates and initial state probability distribution. As an illustrative example, if the states in the Poisson counting process described in Section 10.2.1 are defined as the count, and the count increases by a Poisson process with rate λ, an equivalent continuous time Markov chain model exists given that the probability distribution for the initial count is known. In practice, this count is often deterministically set to zero.

As is known from Part I of this thesis, specifically in Section 2.2.1, the Markov property is a concept that says that although the state of a Markov process is conditioned on the history of the process, this history is always encapsulated in the probability of being in the previous state. This is not only a property of discrete systems but, as described in Section 10.1.1, the Markov (or memoryless) property can also be defined for random variables that changes in continuous time. Therefore, there are both discrete and continuous Markov chains.

To connect the continuous time Markov property to GSMPs with Poisson clock structures and the Poisson counting process, the natural next step is to describe the continuous Markov chain, and this is the starting point of this chapter. Properties of this type of chain such as rates, homogeneity and state probabilities in transient and stationary state are then discussed.

After this, the discrete time Markov chain is described in light of the differences to the newly formulated continuous Markov chain. The same properties that are described for the continuous process are described for the discrete one. With this knowledge, a procedure called uniformisation designed for constructing a discrete time equivalent Markov process from a continuous one is described.

The chapter ends with a proposed way of describing the joining of two independent continuous Markov processes, and how to uniformise the joint process.

11.1 Continuous Time Markov Chains

Markov decision processes are used in the first part of this project, and are first described in Section 2.2. MDPs are, broadly speaking, discrete transition systems that describe controlled but probabilistic transitions between states which obey the Markov property in discrete time.

The continuous time Markov chain can intuitively be seen as a more general form of MDPs. In its basic form, the continuous time uncontrolled Markov chain also describes probabilistic, discrete transitions between states. However, the transitions do not occur at discrete time steps as in the MDP, but at times points in continuous time, and the transitions are in the basic definition uncontrolled. In this section, the continuous time Markov chain is described in detail.

To define a continuous time Markov chain, the GSMP concept is used. Consider a GSMP that is defined to have one discrete event that triggers in continuous time such that if it is counted, the count increases as a Poisson process with rate λ. This count is defined as the state of the Markov chain, and it can be incremented by the single event. The state space is then

$$\mathcal{S} = \{s_0 = 0, s_1 = 1, s_2 = 2, \dots\} \tag{11.1}$$

In this example, there is only one process that changes the state. However, by the principle of Poisson process superposition from Section 10.4, states in a continuous time Markov chain can also be defined as the sum of multiple Poisson event count processes. This is equivalent to saying that the state of a Markov chain can be changed by multiple events. This is discussed in more detail further on, when some important properties of the continuous time Markov chain are described.

A very important property of the continuous Markov chain class is, according to [12], that the transitions between states has exponentially distributed intervenent times *by definition*, since the event count is a Poisson process. If there is only one event, the time interval between two events is then distributed according to the exponential distribution, such that

$$Pr\{v \leq t\} = 1 - e^{-\lambda t} \tag{11.2}$$

A way of interpreting this expression is that it gives the probability of an event occurring during the time interval $(0, t]$ given that the intervenent time v is exponentially distributed with rate parameter λ. This comes from the fundamental mechanism of how the events occur according to the Poisson clock structure, which is described in Section 10.2.1 but described in detail in [12]. This also implies that the time that the Markov process spends in each state, called state holding times, are equivalent to the intervenent times in this particular one event system.

Another defining feature of the continuous time Markov chain is the memoryless property in continuous time, first described in (10.2). If the event count is the state in a Markov chain, the continuous time Markov property is defined for those states as

$$Pr\{s(t_{k+1})|s(t_k), \dots s(t_0)\} = Pr\{s(t_{k+1})|s(t_k)\}$$

where $s(t)$ is the state of the continuous time Markov process at time t and assumes values from the state set $\mathcal{S}$.

As stated in [12], the first difference between the discrete and continuous time versions of Markov chains is that in the continuous case, one step probabilities which in the discrete case are collected in the probability matrix P, cannot be used. Now, the transition probabilities depend on a time interval, and since there are infinitely many time values, describing the probability matrix in the continuous case is not convenient. In the discrete case, the underlying clock structure is synchronised in discrete steps between the different transitions, making the one step probabilities possible to describe in a simple way.

The solution to this problem is to instead consider a matrix $Q(t)$ of transition rates, which is in [12] a bi product of defining the Chapman-Kolmogorov equation.

11.1.1 The Chapman-Kolmogorov equation and rate matrices

In the discrete time case, the Chapman-Kolmogorov equation is described in [12] as an expression that describes the probability that the state of a discrete time Markov process is j at discrete time $k+n$ given that the state is i at time k. Specifically, it describes the total probability of this with respect to an intermediate time u where the process has the probability to be in r different states. If there is a probability function P such that

$$P_{ij}(k, k+n) = Pr\{s(k+n) = s_j | s(k) = s_i\} \tag{11.3}$$

then the Chapman-Kolmogorov equation says that

$$P_{ij}(k, k+n) = \sum_{\text{all } r} P_{ir}(k, u) P_{rj}(u, k+n) \tag{11.4}$$

where $k < u \leq k+n$. Furthermore, $P_{ij}(k, k+n)$ can be defined for all states s_i, s_j and collected in the matrix $H(k, k+n)$.

In the continuous time case, the probability to transition between two states is now dependent on time, such that

$$P_{ij} = Pr\{s(t) = s_j | s(v) = s_i]$$

for two consequtive time points $v \leq t$. To be clear, here v is used to represent a point in time in the derivation of the Chapman-Kolmogorov equation, and has nothing to do with interarrival time, also denoted v in other sections. As is done in detail in [12], the matrix form of the Chapman-Kolmogorov equation thus also changes to

$$H(v, t) = [P_{ij}(v, t)] \tag{11.5}$$

for all $s_i, s_j = 0, 1, 2 \ldots$. From this, the relationship

$$H(v, t) = H(v, u) H(u, t) \tag{11.6}$$

can be obtained for times $v \leq u \leq t$. The rate of H can according to [12] be defined by introducing a time interval Δt and letting it approach zero. What is obtained is then

$$Q(t) = \lim_{\Delta t \to 0} \frac{H(t, t+\Delta t) - I}{\Delta t} \tag{11.7}$$

where I is the identity matrix. $Q(t)$ thus describes the rate of change in the transition probabilities between each state of the continuous Markov Chain with respect to time. Using $Q(t)$, the forward Chapman-Kolmogorov in continuous time can be defined as

$$\frac{\partial H(v,t)}{\partial v} = -Q(v)H(v,t) \tag{11.8}$$

However, in this work, the most important entity in the continuous Markov chain definition is the rate matrix $Q(t)$. Next, some key concepts of the continuous Markov chain are described.

11.1.2 Homogeneity, holding times and probabilities

When analysing Markov processes, there are some common and useful concepts that are necessary to explain.

Homogeneity

With the details omitted here, a homogenic continuous Markov chain is defined by that the transition probabilities are not dependent on absolute time, but only on time intervals. Due to the changes this implies for the Chapman-Kolmogorov matrix, this results in a constant transition rate matrix $Q(t) = Q$ and a τ dependent transition probability matrix $P(\tau)$, while the forward Chapman-Kolmogorov equation (11.8) changes into

$$\frac{dP(\tau)}{d\tau} = P(\tau)Q \tag{11.9}$$

In this work, all Markov processes that are considered for modelling purposes are homogeneous.

State holding times

The time that a discrete Markov chain spends in each state is called the state holding times in [12]. The state holding time $h(s)$ for a state s in the continuous Markov chain case is according to [12] necessarily exponentially distributed by definition. Again, the probability of a state holding time being less than or equal to t reads as

$$Pr\{h(s) \leq t\} = 1 - e^{-\Lambda(s)t} \tag{11.10}$$

for $t \geq 0$, which is said to be equivalent to the probability that an event takes place on the time interval of length t. $\Lambda(s)$ denotes the state holding time parameter, and the fact that state holding times are distributed using $\Lambda(s)$ and not λ as before is connected to the distribution of the intervenent times of *multiple* Poisson processes merged through superposition, described in Section 10.4.

Therefore, this definition of state holding times is done in the general case where superpositioned Poisson processes with different rates are seen as one Poisson process. An intuitive view of this is that from one state s, multiple events can result in the same transition, at different rates. This can be related to what is said in Section 10.4, and then it might be intuitive to view the transitions to different states in a continuous Markov chain as increments affected by multiple superpositioned Poisson processes.

State probabilities

Next, the question of state probabilities in the continuous case is addressed. This is an important feature as it describes the probability of the continuous time Markov process being in a specific state at a certain time. For a homogeneous Markov chain, given that a transition probability matrix $P(\tau)$ is defined along with a transition rate matrix Q, the relationship $P(\tau) = e^{Q\tau}$ holds. Furthermore, if a state probability is defined as $\pi_j = Pr\{s(t) = s_j\}$, and a state probability vector is defined as $\pi(t) = [\pi_0(t), \pi_1(t), \ldots, \pi_n(t)]$ where n denotes the size of the continuous time Markov chain state space $\mathcal{S}$, two relationships can be obtained.

Firstly, according to [12],

$$\pi(t) = \pi(0)e^{Qt} \tag{11.11}$$

which denotes the probability to be in all states at t, given that the probabilities are known at $t_0 = 0$.

Secondly,

$$\frac{d\pi(t)}{dt} = \pi(t)Q \tag{11.12}$$

which is a matrix that can be used to express the *probability flow* into a state at a certain time. This relationship describes the homogeneous Markov process in its transient state. According to [12], although (11.11) can be found if Q and $\pi(0)$ are provided, the explicit expressions for individual state probabilities are difficult even for small chains, as a system of differential equations generated by (11.12) must be solved. Instead, what is often most interesting is how the state probabilities look after a long time, when the system has had a chance to reach its *stationary* or *steady* state.

Steady state

The stationary or steady state is potentially obtained by investigating the probabilities at the limit defined by $t \to \infty$, but certain types of chains cannot reach a stationary state. Steady state is defined when (11.12) is set to zero, as

$$\frac{d\pi(t)}{dt} = \pi(t)Q = \pi Q = 0 \tag{11.13}$$

defines the steady state. When the probability flow between states is zero, the steady state is reached.

Finally, [12] states that one way to completely define a continuous Markov process is to specify a finite state space $\mathcal{S}$, a transition rate matrix Q and an initial state probability distribution $\pi(0)$. In the case of a Poisson process, the state space is the count $s(t) = N(t)$, and since the rate λ with which the Poisson process increments the count, the constant transition rate matrix describes that the rate between each state in the process is λ. This is known as a *pure birth chain*, and it is depicted in Fig 11.1.

The continuous Markov chain is thus a new addition to the theory in this work, as it is not used in Part I. To highlight the differences that this new way of modelling processes implies, the previously used discrete time Markov processes are shown in the new light of what characterises the continuous time version.

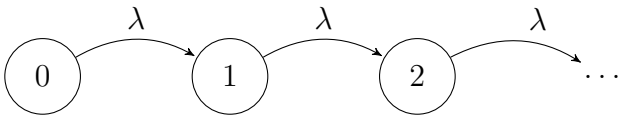


Figure 11.1: Continuous Markov chain describing a pure birth process.

11.2 Discrete Time Markov Chains

In the discrete time case, the discrete Markov property holds, and that is a fundamental property used in Part I of this work. This property is quite similar to that of the continuous case, but several other properties change when discrete time is replaced with continuous time.

11.2.1 The Chapman-Kolmogorov equation and probability matrices

The Markov property for discrete chains are defined using the equidistant time instances $0, 1, \ldots k$ and reads

$$Pr\{s(k+1)|s(k), s(k-1), \ldots, s(0)\} = Pr\{s(k+1)|s(k)\} \tag{11.14}$$

For discrete time Markov chains, [12] states that the starting point is again to consider the underlying GSMP defined by $\{\varepsilon, \mathcal{X}, \Gamma, p, p_0, G\}$, and focus on the probability $Pr\{x'|x\}$ that a transition between states x and x'. In discrete Markov chains, events cannot occur at any point in continuous time. The fact that they can only occur at discrete, equidistant points can be seen as a *constraint* compared to the free movement allowed in continuous systems. Between distinguishable time points, it is therefore possible to determine the stochastic behavior of the system in simple probability terms.

Using the Markov property, transition probabilities for both one step ahead and n steps ahead between Markov process states s_i and s_j in this setting can be denoted

$$P_{ij}(k) = Pr\{s(k+1) = s_j|s(k) = s_i\}$$
$$P_{ij}(k, k+n) = Pr\{s(k+n) = s_j|s(k) = s_i\}$$

Given the familiar rule of total probability, $\sum_{\text{all } j} P_{ij}(k) = 1$, the event $s(k+n) = s_j$ can be conditioned on an intermediate event $s(u) = s_r$ for $k < u \leq k+n$, as

$$\begin{aligned} P_{ij}(k, k+n) = \sum_{\text{all } r} & Pr\{s(k+n) = s_j|s(u) = s_r, s(k) = s_i\} \cdot \\ & \cdot Pr\{s(u) = s_r|s(k) = s_i] \end{aligned} \tag{11.15}$$

Using the Markov property,

$$Pr\{s(k+n) = s_j|s(u) = s_r, s(k) = s_i\} = Pr\{s(k+n) = s_j|s(u) = s_r\} = P_{rj}(u, k+n)$$

and by setting $Pr\{s(u) = s_r|s(k) = s_i] = P_{ir}(k, u)$ in (11.15),

$$P_{ij}(k, k+n) = \sum_{\text{all } r} P_{ir}(k, u) P_{rj}(u, k+n) \tag{11.16}$$

for $k < u \leq k + n$. The expression in (11.16) is recognised as the Chapman-Kolmogorov equation in discrete time, described in (11.3).

11.2.2 Homogeneity, holding times and probabilities

The same concepts that are investigated for continuous time Markov chains in Section 11.1.2 are here described for the discrete time Markov chains to highlight the differences between the two.

Homogeneity

Starting with the definition of a homogenous discrete time Markov chain, [12] states that if the elements of the transition probability matrix function $P_{ij}(k)$ are independent of the time instance k for all states i, j, then the chain is homogenous. In such a case, the Chapman-Kolmogorov equation in (11.16) and its matrix form are changed since the n-step probabilities of (11.16) are only dependent on n, and not on k.

To clarify, the probability that the Markov chain assumes a value after n consecutive iterations is a product of one step probabilities, and homogeneity simply states that the fundamental "one step" probabilities do not change as k evolves. As stated for continuous Markov processes, the homogeneous version of Markov-style models are assumed also for discrete processes in this work, but it should be kept in mind that this assumption can be quite a simplification in many practical cases.

Instead of time interval dependent probability matrices which forces the uses of rate matrices, in discrete time the homogeneous Markov chain can be defined as $P = [P_{ij}]$ for all states s_i, s_j with indices $i, j = 0, 1, 2, \ldots$. This matrix is of the form $\mathcal{S} \times \mathcal{S} \rightarrow [0, 1]$ where $\mathcal{S}$ is a state space and P maps any two states to a transition probability. Note that the rule of total probability must hold, so P must express that the probability to transfer from one state to any next state is one in total.

State holding times

In the continuous time Markov chain, events are said to occur at any time, and the intervenent times are therefore exponentially distributed, which is why the state holding times are also exponentially distributed. When the events are scheduled to occur at equidistant discrete time points, this property understandably changes. In [12], the distribution of the state holding times $h(i)$ is shown to be geometric instead of exponential, such that $Pr\{h(s) = n\} = (1 - P_{ii}) \cdot (P_{ii}^{n-1})$ where P_{ii} is the expression for the one step transition probability of a self loop in a state with index i, obtained from the function P defined for the discrete chain. Since only discrete time instances are considered, the state holding time is expressed as the chain of probabilities of staying in a state for n consecutive discrete steps.

State probabilities

The state probabilities in a discrete chain are determined by a different method than that of the continuous Markov chain. As in [12], the state probability is denoted

$$\pi_j(k) = Pr\{s(k) = s_j] \tag{11.17}$$

Furthermore, it is worth noting that if the state space $\mathcal{S}$, the state probability matrix P and probability to be in the initial state is given as $\pi(0)$, the discrete time Markov chain is completely specified, which is different to the continuous case where the rate matrix Q is needed for the definition.

As in the continuous case, both transient and stationary state exists for discrete time Markov processes. By starting at a point k where k can take values $0, 1, 2, \ldots$ the state probabilities at $k + 1$ can be expressed by

$$\pi(k + 1) = P\pi(k) \tag{11.18}$$

where the one step probabilities between all states are gathered in P. The transient phase can now be intuitively defined to begin at the initial time point $k = 0$, and then go on until $\pi(k + 1) \approx \pi(k)$.

11.2.3 Steady state

The steady state analysis of a discrete Markov chain stands in contrast to the transient state analysis, just as in the continuous case. The transient phase is in [12] described by considering a vector of initial state probabilities $\pi(0) = [\pi_0(0), \pi_1(0), \ldots \pi_n(0)]$ where n denotes the number of states in the Markov chain. The one-step probabilities will, as discussed earlier in this section, not change in a homogeneous discrete Markov chain, but the probability to be in a state for k consecutive time steps will change during the transient phase.

The steady state is defined for the limit value $\pi_j = \lim_{k\to\infty} \pi_j(k)$. By the previous definition of the transient state in (11.18), the steady state j is defined when $\pi_j(k+1) \approx \pi_j(k)$ for increasing values of k. Note also that this limit might not exist for all states, which is why the notation here describes only one state.

As an example partly inspired by one in [12], consider the state space $\mathcal{S} = \{0, 1, 2\}$, with the initial state probability vector $\pi = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}^T$ and two different one step probability matrices

$$\begin{aligned} P_1 &= \begin{bmatrix} P_{00} & P_{01} & P_{02} \\ P_{10} & P_{11} & P_{12} \\ P_{20} & P_{21} & P_{22} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 1 \\ 0.5 & 0 & 0 \\ 0.5 & 0 & 0 \end{bmatrix} \\ P_2 = (\cdot) &= \begin{bmatrix} 0 & 0.25 & 0.25 \\ 0.5 & 0.25 & 0.5 \\ 0.5 & 0.5 & 0.25 \end{bmatrix} \end{aligned} \tag{11.19}$$

Both of these are legitimate one step probabilities as their columns sum to one, but if P_1 is used, the probabilities will not reach a constant steady state as k increases. In Fig 11.2, the development of the state probabilities are shown. They do not

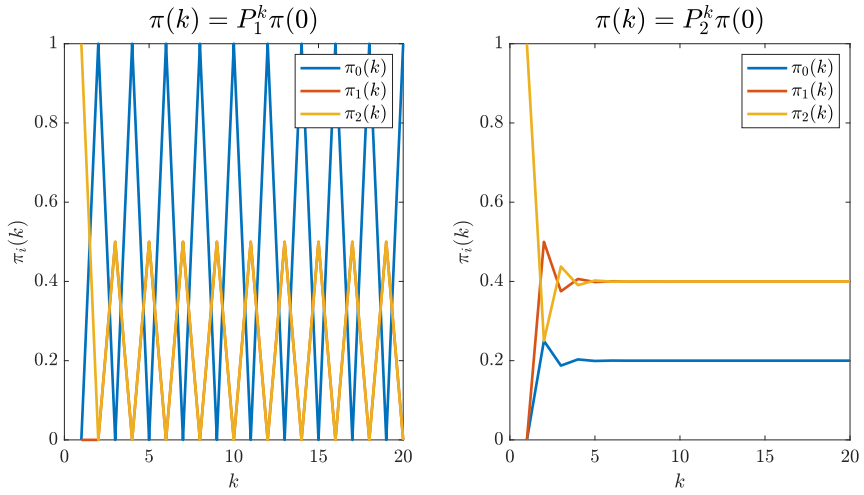


Figure 11.2: Oscillatory versus stationary state probability development.

reach a constant steady state if P_1 is used, and the state probabilities oscillate for increasing k. However, if P_2 is used, the steady state is reached for all three state probabilities.

11.2.4 Continuous or discrete Markov processes?

The conclusion that can be drawn from the two latest sections is that it seems much easier to handle discrete Markov chains, which are restricted in such a way that events can only occur in a synchronised manner at discrete points in time. However, many problems are better formulated by using continuous Markov chains. Recalling Part I of this thesis, the temporal logic constrained reinforcement learning requires the use of discrete Markov processes. On the other hand, in modular analysis, the idea is to use all possible knowledge of a system. As is shown in later chapters of this thesis, there are continuous Markov chains that have interesting and highly useable properties when it comes to finding optimal ways of solving problems related to them.

Fortunately, there is a way to convert continuous Markov chains to discrete ones while conserving all of the important probabilistic features of the continuous version. This method is called uniformisation, and it is covered next.

11.3 Uniformisation

Uniformisation is an important strategy, as it is in [12] described as a way to convert a continuous Markov chain to a discrete one, without losing any vital information that the continuous chain contained.

The starting point in uniformisation is to focus on the transition rate matrix Q. In this matrix, it is common that the diagonal elements are negative, so that the total flow out of a state sums to zero. This is in [12] said to represent the total event rate characterising each state. The process of uniformisation is to select a new rate that all existing rates can be normalised with, in such a way that the largest possible

rate is converted to a probability that does not exceed one, and the smallest rate is converted to a relatively low probability.

The uniformisation parameter is denoted γ and is selected as

$$\gamma \geq \max_i \{-Q_{ii}\} \tag{11.20}$$

Using this, the continuous time transition rates are converted to transition probabilities, so that the one step transition probability function P can be formed for the uniformised process. The elements of P are

$$P_{ij} = \begin{cases} \frac{Q_{ij}}{\gamma} & \text{if } i \neq j \\ 1 + \frac{Q_{ii}}{\gamma} & \text{if } i = j \end{cases} \tag{11.21}$$

Only the maximum rate across the entire continuous chain has the chance of being translated to a probability of one. Fictitious self loops are also introduced, and this is necessary to assure that if the process is in a state i, then the total probability of transitioning anywhere should sum to one. By themselves, the transition rates out of a state cannot be assured to sum to one if normalised by gamma, and the probability to stay in the state must be manually defined.

11.3.1 Example of uniformisation

In this section, a uniformised version of a continuous Markov chain model of a population that can only be reduced is made.

Pure death Markov chain

In contrast to the similar pure birth chain that has been covered before, a model of a so-called pure death process is considered here. In the model, a finite population of size N is reduced in continuous time by two Poisson processes until it is zero. The death rates of the two processes are μ_1 and μ_2, and the second process has double the efficiency of the first process. This implies that this process decrements the population by two each time its events occur. The population count is the state space $\mathcal{S} = \{N, N-1, N-2, \ldots, 0\}$, and it can be noted that since the states represent a count, the notation used to describe them is simply the integer value. Based on this, the transition rate matrix can be determined as

$$Q = \begin{bmatrix} -(\mu_1+\mu_2) & \mu_1 & \mu_2 & 0 & 0 & 0 & \ldots \\ 0 & -(\mu_1+\mu_2) & \mu_1 & \mu_2 & 0 & 0 & \ldots \\ 0 & 0 & -(\mu_1+\mu_2) & \mu_1 & \mu_2 & 0 & \ldots \\ \vdots & \vdots & \ddots & \ddots & \ddots & \ddots & \end{bmatrix} \tag{11.22}$$

In Q, the diagonal elements are selected so that the total probability flow from a state is zero. To uniformise, the uniformisation rate is selected according to (11.21) to be grater or equal to the largest negated diagonal element, for example $\gamma = 2(\mu_1 + \mu_2)$. The probabilities are then set to

$$P_{ij} = \begin{cases} \frac{Q_{ij}}{2(\mu_1+\mu_2)} & \text{if } j = i-1 \\ 1 - \frac{(\mu_1+\mu_2)}{2(\mu_1+\mu_2)} & \text{if } i = j \end{cases} \tag{11.23}$$

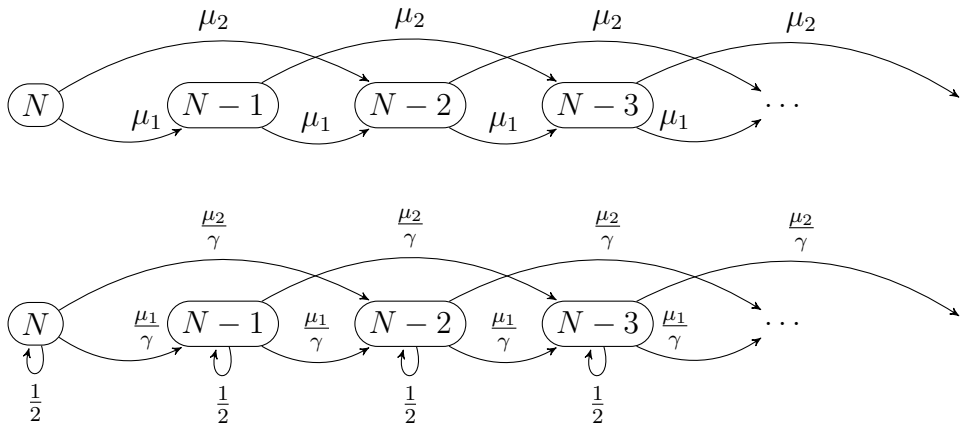


Figure 11.3: State transition rate and state probability diagrams for the continuous time pure death Markov chain and its uniformised discrete counterpart.

Note that the probability of a self loop changes depending on how large the uniformisation rate is, which is set according to the largest diagonal rate matrix element. If the smallest possible γ is selected, the fictitious self loop probability is zero. However, as the selected γ is greater than the maximum negated diagonal element of Q, the self loop probability will be above zero this time. The stochastical properties of the continuous Markov chain are preserved in both cases, according to [12], no matter what uniformisation rate is selected. The continuous pure death Markov chain can be observed along with the uniformised version in Fig 11.3.

Finally, note that in each state, the probability of a self loop is 1/2. The two other possible transitions have the total probability

$$\frac{\mu_1}{2(\mu_1+\mu_2)} + \frac{\mu_2}{2(\mu_1+\mu_2)} = \frac{1}{2}$$

meaning that in each state, the total probability of transitioning anywhere sums to one.

In the next section, a for this work central model definition is developed. The section describes a so-called joint Markov chain that can be seen as both a single Markov process, and as two independent ones.

11.4 Joint Markov Chains

In the form of modular analysis that is considered in the context of the temporal logic constrained reinforcement learning described in this work, the idea is that certain parts of a system can be modelled and evaluated to reduce the computational burden of the reinforcement learning algorithm. It is therefore important to determine how and in what way a total system can be divided into subsystems.

11.4.1 Joint behaviour of discrete and continuous models

Consider two separate Markov processes $\mathcal{M}_1$ and $\mathcal{M}_2$. Both processes $\mathcal{M}_1$ and $\mathcal{M}_2$ have one set each of possible states, and transitions between these states that

occur at some rates. With the previous discussions on continuous and discrete time Markov chains in mind, there are a few possible scenarios that can be imagined.

Discrete $\mathcal{M}_1$ and continuous $\mathcal{M}_2$

In the first scenario, $\mathcal{M}_1$ is considered a discrete time system while $\mathcal{M}_2$ is a continuous system, or vice versa. In this setting, the discrete system can transition between events at discrete, equidistant, points in time while the continuous system can transition at any time. This implies that if the discrete time Markov process is observed at discrete time points t_k and t_{k+1}, it can be expected that it has only transitioned at most one time in between observations at t_k and t_{k+1}. This cannot generally be said about the continuous process if it is observed during the same interval; even with very low transition rates it can never be guaranteed that this system has made at most one transition in between observations. This is natural because the discrete process does not carry the information of how long the time interval between discrete observation time points is. Even if the time interval was known, it would need to approach zero to guarantee that the continuous process does not transition more than once during that time interval.

This is a problem if it is necessary to synchronise the two processes, which first of all means scheduling their transitions on a common timeline. For this to be possible, one of the processes cannot be discrete if the other is continuous.

Continuous $\mathcal{M}_1$ and $\mathcal{M}_2$

If both $\mathcal{M}_1$ and $\mathcal{M}_2$ are continuous, there are no discrete time points to consider, and both systems can be described using transition rate matrices. If it is indeed possible to isolate $\mathcal{M}_1$ and $\mathcal{M}_2$ as two subsystems of a joint process, this scenario allows an observer to view the transitions of both systems at any time when they occur, and it can be guaranteed that no transitions are missed.

Discrete $\mathcal{M}_1$ and $\mathcal{M}_2$

In the final case, both systems $\mathcal{M}_1$ and $\mathcal{M}_2$ are considered discrete. In this way, both systems have transitions defined by probabilities, and not rates. From a discrete observers viewpoint, it can be guaranteed that all transitions are observed, as opposed to the case where one process is discrete and the other is continuous.

What can be taken away from this section is that if two Markov processes are to be observed, they must either both be modelled as continuous or discrete processes; not one of each. If both systems can be described individually in both the continuous and discrete case, it is now interesting to imagine what happens if the states and transitions of both systems are viewed as transition between joint system states. To find out what happens in the continuous time case, a good starting point is to consider what happens if multiple Poisson processes are connected.

11.4.2 Joint continuous Markov processes

A continuous Markov process is in [12] said to be fully defined if a state space $\mathcal{S}$, a transition rate matrix Q, and an initial state probability distribution is provided. Consider the state spaces of two systems that can be modelled as two *independent* continuous time Markov processes $\mathcal{M}_1$ and $\mathcal{M}_2$. For simplicity, each finite state space may be denoted directly by positive natural numbers as

$$\begin{aligned} \mathcal{S}_1 &= \{0, 1, \ldots, c_1\} \\ \mathcal{S}_2 &= \{0, 1, \ldots, c_2\} \end{aligned} \tag{11.24}$$

Note that the state space description is not limited to positive natural numbers, but can be set to whatever fits the problem that the state space describes. For notation simplicity, and because it agrees with the problems studied in this work, positive natural numbers are selected as state identifiers.

These state spaces can be considered as two separate continuous time Markov processes, that are each generated by a GSMP with exponentially distributed intervenent times. Each GSMP has a set of events, and these events are also independent from each other, meaning that an event in the first GSMP cannot affect transitions in the other. In that sense, the event set of the joint system can be divided in two, and the events in one subset causes transitions to a unique state in the joint state space that cannot be reached by transitions caused by the events in the other event subset.

In other words, what is considered here are two completely independent continuous Markov processes. An example of this related to the patient waiting room scenario of Section 10.3.2 is that these processes represent an emergency room and a neighboring dentists waiting room. Patients in the emergency room and patients in the separate dentists office arrive according to two separate Poisson processes with rates λ_1 and λ_2, and they are counted separately which forms the two separate and independent state spaces $\mathcal{S}_1$ and $\mathcal{S}_2$.

To be clear, this is not a case of superpositioned Poisson arrival processes from Section 10.4, as the sum of the event counts do not form the state space. Rather, these are two separate state spaces that are affected independently by two isolated processes.

Even though the processes are completely separated, it is completely valid to describe them in relation to each other. For every state in one process, the other process can be in any of its states independently of the first. This relationship is similar to a form of synchronous composition, similar to that used in [1] but for continuous Markov chains and not discrete Markov decision processes. The joint state space is given by the product

$$\mathcal{S}_{12} = \mathcal{S}_1 \times \mathcal{S}_2 = \{(i_1, i_2) : i_1, i_2 \in \mathbb{N}_0\} \tag{11.25}$$

where $\mathcal{S}_1$ and $\mathcal{S}_2$ can be finite or infinite sets.

Each system model has independent transitions with equally independent rates that can be executed at any point in continuous time. In Fig 11.4, two separate continuous Markov chains each with two events and two states in their respective state space are considered. The joint behaviour of these are then described as the

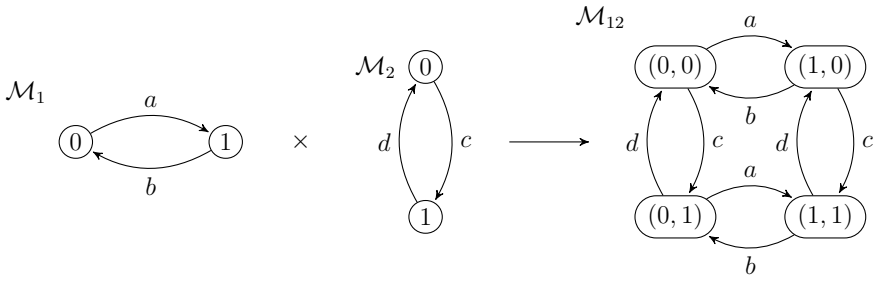


Figure 11.4: Two independent continuous Markov processes that form one joint system.

four state system where the states are denoted by tuples describing the states of each chain.

Examples of joint continuous Markov processes can also be found in [12], but the focus here lies on the ability to express the joint behavior has two separate but synchronised behaviors.

Lastly, consider not two, but n processes that are mutually independent in the same way as above. By denoting the state space of the joint system as a set of n-tuples, the principle of the joint process can be extended by adding a new dimension to the joint process for every independent process. In each state of the joint process, all possible transitions of the individual process states are available with their respective individual rates.

11.4.3 Uniformisation of joint continuous Markov Processes

From previous sections, it is well known that individual continuous Markov processes can be uniformised to form corresponding discrete time Markov chains. It is also shown that n separate processes can be described either by themselves or as a joint system under the assumption that both processes are independent in terms of state spaces and transitions. In the joint systems, the laws of transitional and total probabilities must hold as in the individual systems.

It it thus interesting to see how the uniformisation of the individual processes compare to that of the joint process. For n mutually independent continuous Markov processes generated by n mutually independent GSMPs with exponentially distributed intervenent times, consider the n individual transition matrices Q^k where $1 \leq k \leq n$. In each of these, there is at least one state that has a maximum flow of rates out of it. The flow out of such a state is given by

$$\gamma_k = \max_i \{-Q^k_{ii}\} \tag{11.26}$$

where Q^k_{ii} is a diagonal element of Q^k, i denotes the state in the process with index k that has the maximum rate flow out of it. It has already been shown that for the individual and independent process k, the *minimum* uniformisation rate is given by (11.26).

Now, for an n process joint system, the state space is the cross product of each individual state spaces, such that

$$\mathcal{S}_\times = \mathcal{S}_1 \times \mathcal{S}_2 \times \cdots \times \mathcal{S}_k \times \cdots \times \mathcal{S}_n \tag{11.27}$$

As a general rule, each transition of each individual system has a respective transition in the joint state, with the same rate as in the individual system. Thus, the uniformisation principle described in (11.21) does not change, although all source and target states for the joint transitions are described in terms of cross product states. A cross product state is denoted by the tuple of indices

$$I = (i_1, i_2, \ldots, i_k, \ldots, i_n) \tag{11.28}$$

representing the individual states of each process where $i_1, \ldots, i_n$ are non negative integers. For example, if process 5 is in its state indexed by 2 then $i_5 = 2$ and so on. This of course implies that each process has non negative integer indexed states. Similarly, when a transition is considered between two joint system states, the source state is denoted by the above formulation I while the target state is denoted by

$$J = (j_1, j_2, \ldots, j_k, \ldots, j_n) \tag{11.29}$$

In the individual processes, a self loop is defined for states i and j when $i = j$ for every diagonal element Q^k_{ii} in the rate matrix Q^k. In a joint state, the transitions of each individual process are represented, and this includes self loops. In the joint system, multiple self loops can occur in one state, but the easiest way to think of these are as one self loop where the rate is a sum of the individual self loop rates.

The probabilities resulting from joint uniformisation by rate are described in the one step transition probability function P in total defined by

$$P_{IJ} = \begin{cases} \frac{Q^\ell_{ij}}{\gamma} & \text{if } I \neq J \text{ for transition between } i \text{ and } j \text{ in subprocess } \ell \\ 1 + \sum_{k=1}^{n} \frac{Q^k_{ii}}{\gamma} & \text{if } I = J \end{cases} \tag{11.30}$$

As a transition probability between two different joint states is always defined by an individual process, the transition probability between two joint states is denoted $P_{I,J}$ but specified with the index ℓ to highlight the individual process $1 \leq \ell \leq n$ that caused the joint state transition.

Now comes the time to select the minimal uniformisation rate γ. In Section 11.3, it is shown that the minimal uniformisation rate is selected as the largest negated diagonal element in the rate matrix. This is so that the state with the largest flow of rates should be the only one without a probability to self loop. If sampled, this state has the highest possible frequency of transitions out of it compared to other states, reflecting the higher rate.

In each state of a joint process, the available transitions are those of each individual system. These are considered to be executable in *parallel*, meaning that a transition between any two states in the joint system corresponds to any one of the possible individual transitions; not two or more in series. As all possible permutations of individual process states are included in the joint system, the joint state with the maximum total flow is naturally the state that combines all of the individual states that have the maximum flow in each system, respectively. The flow of such a joint state is thus

$$\Lambda = \sum_{k=1}^{n} \gamma_k \tag{11.31}$$

To select the lowest possible rate, it is reasonable to assume that for such a rate, the maximum rate flow state will not have any self loops. From (11.30), set the probability of self loops to zero and obtain

$$1 + \frac{1}{\gamma} \sum_{k=1}^{n} \min_i (Q_{ii})^k = 0 \iff -\sum_{k=1}^{n} \min_i (Q_{ii})^k = \gamma \tag{11.32}$$

The term in the sum can now be replaced by the maximum outgoing rates Λ_k of each process k, defined as the maximum negated element of each rate matrix Q^k since the expression is evaluated in the maximum flow state, and an expression for the minimal joint uniformisation rate γ is obtained as

$$\gamma = \Lambda \tag{11.33}$$

This can be verified by setting $\gamma = \Lambda$ in the expression for the $I \neq J$ case in (11.30). The rate of an individual transition ℓ causing a transition between I and J can never be larger than Λ which denotes the maximum possible sum of the largest negated individual rate flows.

Similarly, in the maximum flow state, there is no probability of a self loop. Therefore, all transitions are to other joint states, with individual max flows. In that state,

$$\sum_I P_{IJ} = -\frac{1}{\gamma} \sum_{k=1}^{n} \min_i (Q_{ii})^k = \frac{\Lambda}{\gamma} = 1 \tag{11.34}$$

where $\sum_I P_{IJ}$ denotes the sum of uniformised probabilities to transition between I and all other possible joint states J.

In all states but the maximum flow state, at least one term in the sum in (11.34) will be lower than the individual maximum sum of outgoing rates in that process. If the state is not the maximum flow state, the term $\sum_{k=1}^{n} \min_i (Q_{ii})^k$ has at least one element that is not an individual maximum flow, which implies that this sum is below one, and the difference between this expression the sum in (11.34) will be as large as that single element that is not a maximum flow. In that way, the fictitious self loop ensures that each state has a total probability of one, also in the joint case.

Example of joint uniformisation

To clarify the joint uniformisation procedure, a small scale example is given. For the two individual and independent continuous Markov processes in Fig 11.4, the two rate matrices are

$$Q^1 = \begin{bmatrix} -a & a \\ b & -b \end{bmatrix}, Q^2 = \begin{bmatrix} -c & c \\ d & -d \end{bmatrix} \tag{11.35}$$

It is assumed that $a > b$ and $d > c$. Furthermore, the relationship between the elements of Q^1 and those of Q^2 is unknown. There are only one transition out of each state in both processes, and the states with maximum transition flow is thereby state 0 in the first process and state 1 in the second process. Individually, the minimal uniformisation rates are therefore $\gamma_1 = a$ and $\gamma_2 = d$, but in the joint uniformisation the minimal rate is $\gamma = \gamma_1 + \gamma_2 = a + d$.

The cross product between the two state spaces is

$$\mathcal{S}_1 \times \mathcal{S}_2 = \{0,1\} \times \{0,1\} = \{(0,0),(0,1),(1,0),(1,1)\} \tag{11.36}$$

In Fig 11.4 it can be seen how for every joint state $I = (i_1, i_2)$, at most one transition T in either individual system defines the transition to the new joint state $J = (j_1, j_2)$. For example, between the joint state $I = (0,1)$, an individual transition with rate a in the first process defines the transition to the joint state $J = (1,1)$.

It becomes apparent from Fig 11.4 that in the joint process, since the relationship between the largest individual rates a and d is unknown, the only way to assure a sufficient minimal uniformisation rate is the sum of the largest rates. Not that this holds under the assumption that the rates are positive. This principle can also be verified if the rate matrix for the joint system is evaluated, which yields

$$Q = \begin{bmatrix} -(a+c) & c & a & 0 \\ d & -(a+d) & 0 & a \\ b & 0 & -(b+c) & c \\ 0 & b & d & -(b+d) \end{bmatrix} \tag{11.37}$$

The rows and columns of Q regard the joint states $(0,0)$, $(0,1)$, $1,0$ and $(1,1)$. The largest negated diagonal element of Q is $(a+d)$, the sum of the largest negated diagonal elements of both individual processes. Thereby, if this is the selected joint uniformisation rate then this state will have no self loops as all transitions out of the joint state together have a probability of one. The joint uniformisation formula then yields the discrete model depicted in Fig 11.5.

It can be verified that in any joint state of the model in Fig 11.5, the sum of transition probabilities is one.

11.5 Summary

Firstly, this chapter introduces a new topic in the form of continuous time Markov chains. One of the notable differences between these and the discrete counterpart

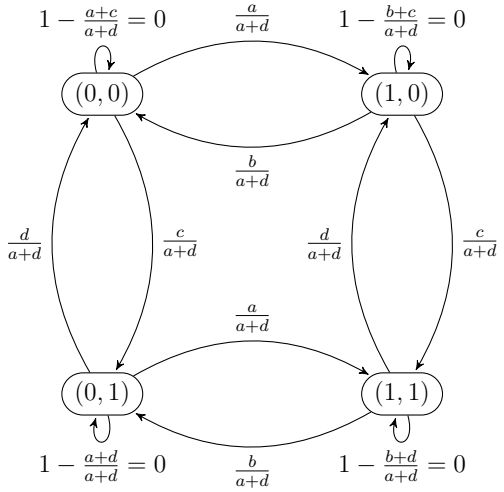


Figure 11.5: Example of a uniformised joint system.

is that events can now take place in continuous time in contrast to the discrete equidistant time points of the discrete Markov process. Other memorable properties of the continuous time Markov chain is that each event occurs according to a Poisson process with a specific rate parameter λ; this implies that the time interval between two events of the same type is an exponentially distributed random variable with the same rate parameter λ.

The rates of the continuous time Markov chain are collected in the transition rate matrix $Q(t)$, and for homogeneous Markov chains, this matrix is constant with respect to time. Other important features of the continuous Markov process are that expressions for the probability that the process is in a specific state can be derived at any time, but it is much easier to derive these in stationary state. As opposed to the transient state, the stationary state is defined as the point where the evolving state probabilities reach constant values after some time.

The benefits of discrete Markov chains are visible in contrast to the properties of the continuous time Markov chain. Among these are the possibility to easily describe the probability for the system to transition to another state one step ahead in the future. However, the properties of homogeneity and stationary state probabilities exist in the discrete version, too.

An important topic that is introduced in this chapter is uniformisation, which normalises the transition rates in a continuous Markov process by a uniformisation rate that is at least as large as the maximum total rate flow out of a state in the process. Uniformisation thus creates a discrete Markov chain out of a continuous one without disrupting any of the important stochastic properties. Therefore, modelling that is done on the continuous model, is still valid for the discretised model, which can be used in for example reinforcement learning.

Lastly, the basic concepts of joint continuous Markov processes are suggested from the basic assumption that two or more processes can be measured together while still being completely independent. For the joint processes, the joint uniformisation procedure is then formulated, which normalises the joint system by a minimal rate consisting of the sum of the maximal rate flows of each individual process. This procedure is illustrated using a small scale example.

12
Queuing Theory

The more in depth view of Markov chains that is described in the previous chapter is here put into practice in the specific process of modelling queueing systems. With knowledge of Poisson processes and how continuous and discrete variations of a Markov chain behave, it is easier to find, experiment with and verify queueing system models.

The concept of waiting in line is a very broad application of probability theory. Given that industrial applications, computer systems and daily life tasks always suffer from operational limitations which result in waiting for service, the fundamental concepts are easy to relate to.

Continuing with the hospital emergency room analogy from Section 10.3.2, patients who have arrived in the waiting room require *service* from a doctor. However, there are many patients, and they need to get in line as they arrive to the emergency room. Different configurations of these arrival and service processes together form the basic concepts of queueing systems.

In Fig 12.1, an example of a queueing system is depicted. In this basic setup, customers arrive to the queue from the population with rate parameter λ, after which they are served by a single server with rate parameter μ.

This chapter starts with some basic notations and concepts commonly used in queueing theory, such as Kendall notation and how arrivals and services are modelled. After this, the class of Markovian queueing systems are introduced, and this is the only type of queues that are investigated in this thesis. Specifically, the unlimited $M/M/1$ queue and the capacity limited $M/M/1/K$ are studied, along with the uniformised versions of these. The next step is to introduce cost functions and the concept of control into continuous Markov chains, which allows for the analytical formulation of optimal control policies in the controlled Markov process. Furthermore, a specific type of control policy called a threshold type solution is described, and a five step guide on how to identify such problems is shown.

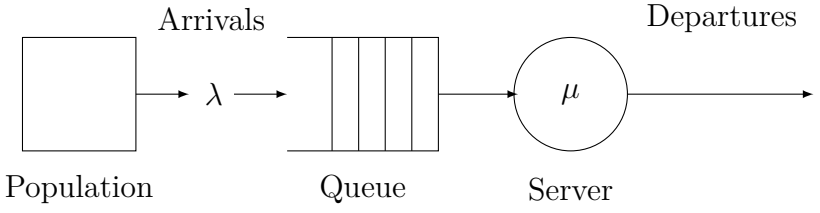


Figure 12.1: A basic single server queueing system.

12.1 Concepts and Notation in Queueing Theory

Before going into specific models, some general comments about the notation frequently used in queueing theory are needed. Here, the most important concepts and practices in queueing theory, gathered from [31] unless otherwise stated, are covered.

12.1.1 Population

Starting with the leftmost entity in Fig 12.1, the population describes a source of customers that can be either finite or infinite. Although a finite source is in many cases more realistic, infinite sources are often preferred due to the much simpler mathematical modeling properties of infinite source systems. If a queueing system has a finite source, the arrival rate will approach zero when the source is emptied. In that case, customers are either waiting or being served.

12.1.2 Rates, structure and discipline

While it is already touched upon in Sections 10.2.1 and 10.3.1, λ and μ describe the average rates of customers per unit time for the customer arrival and service processes, respectively. In queueing theory, the models used for the arrival and service are usually stochastic processes. While arrival and service rates are important parameters for the behavior of the queue, the performance of a queueing system is also dependent on both the structure and discipline of the queueing system. Common structural parameters are for example the number of servers and queues active in the network. The queue or service discipline is the order in which customers enter and exit the queue, such as FCFS, FIFO or LCFS, meaning "first come, first served", "first in, first out" and "last come, first served", respectively.

12.1.3 Interarrival and service times

Related to the arrival rate is the stochastic interarrival time, here denoted z, describing the time between the arrival of customer $k-1$ and customer k. Similarly, v denotes the stochastic service time describing the required time for a customer k to be served. The probability distributions describing interarrival times and service times are defined as

$$\begin{aligned} A(t) &= Pr\{z \leq t\} \\ B(t) &= Pr\{v \leq t\} \end{aligned} \tag{12.1}$$

The v and z for each customer are independent stochastic variables, and it is assumed that the interarrival and service times are described by the same distributions for each customer.

The mean of the interarrival time distribution $A(t)$ is the expected value of the interarrival time, while the mean of the service time distribution $B(t)$ is the expected service time value. These expected values are related to the average arrival and

service rates in that

$$\begin{aligned} \mathbb{E}[A(t)] &= \frac{1}{\lambda} \\ \mathbb{E}[B(t)] &= \frac{1}{\mu} \end{aligned} \tag{12.2}$$

respectively. Furthermore, at this point both the interarrival and service times can be distributed according to any probability distribution. However, in this work, only the exponential distribution is of interest in both cases.

12.1.4 Kendall notation

To describe a queueing model, it is customary to use the *Kendall notation*. In [31], this notation is of the form $A/B/c/K/m/Z$, where

- A denotes the distribution of the stochastic interarrival times.
- B denotes the distribution of the stochastic service times.
- c denotes the number of servers.
- K denotes the queue capacity.
- m denotes the number in the population.
- Z denotes the queue discipline.

There are various distributions that A and B can assume, but one of the most common ones are the exponential distribution, denoted M for "Markovian". As described in Chapter 11, it is implied that the queue can be seen as a GSMP generated stochastic sequence, which has a corresponding Markov chain description since the intervenent times are specified as being exponentially distributed. As discussed in Section 10.3.1, this also implies that the arrival and service processes are Poisson processes.

The population m is usually infinite, and the discipline Z is often "first come, first serve". When this is the case, m and Z are often omitted in the Kendall notation. Two examples of common queues that are central to the work done in this thesis have the notations $M/M/1$ and $M/M/1/K$. In both cases, the interarrival and service times are exponentially distributed, and there is only one server present, while the populations are infinite and the disciplines are FCFS. For the $M/M/1$ system the queue capacity is omitted in the description and therefore assumed infinite, while the capacity is K elements in the case of $M/M/1/K$. The properties of both the $M/M/1$ and the $M/M/1/K$ systems are described in later sections.

12.1.5 Performance measurements

In queueing systems, both arriving and serviced customers are countable and by assuming the notation where a count corresponds to a time dependent state, $s(t) = N_a(t) - N_d(t)$ denotes the stochastic difference between the counted arrivals and the

counted departures. The random variable $s(t)$ is central to many of the different measurements that can be investigated in queueing systems, and many of them can be found both in [12] and [31]. As not all measurements are needed in this work, only the most commonly used ones are listed below.

- Stationary (steady state) queue length probability $\pi_n = Pr\{s(t) = n\}$, $n = 0, 1, 2, \ldots$, where $s(t)$ denotes the stochastic queue length variable, which can also be regarded as a time dependent state.
- Intensity with one server $\rho = \lambda/\mu$ and intensity with m servers with equal service rates $\rho = \lambda/(m\mu)$.
- Average queue length $\mathbb{E}[s(t)]$.

When queues are simulated in later sections, the most commonly used measurements are the arrivals, departures and queue state trajectories with respect to time. When it comes to capacity limited queues, the rejected arrivals are also of interest, and in all cases, the stationary state probability distributions π are of high interest.

12.1.6 Arrival and service models

As described in Section 12.1, both the interarrival time and the service time take values according to a probability distribution. In this work, the interarrival and service times are both modeled according to the exponential distribution, described in (10.22) and (10.23) of Section 10.3.1.

Having exponentially distributed arrival times implies that the arrival process is a Poisson process, where arriving customers arrive to the queue according to the three rules described in the derivation of the Poisson distribution of Section 10.2. An important parameter of the arrival process is thus λ, the average arrival rate per unit time interval.

Furthermore, as the service times are also modeled using the exponential distribution, the departing customers can also be counted as a Poisson counting process. For this process, the important parameter is the departure rate parameter μ, but as departure times are dependent on when the arrivals, it is easier to view μ as the parameter regulating the exponentially distributed service time.

12.2 Markovian Queueing Systems

This work is limited to only handle Markovian queueing systems. Depending on the application, these can be either continuous or discrete, and have different structures and rate parameter configurations. Here, two basic queueing systems are discussed to give an idea of how information can be obtained analytically.

12.2.1 M/M/1 system

To summarise the theory behind the $M/M/1$ Markovian queueing system, the Poisson arrival process of this system is such that the probability of n arrivals during a

unit time interval of length t is given by

$$Pr\{A(t) = n\} = P_n^a(t) = \frac{(\lambda t)^n}{n!} e^{-\mu t}$$

Each arrival takes time to process by the single server. The service time is exponentially distributed with rate μ, such that

$$G(t) = Pr\{v \leq t\} = 1 - e^{-\mu t}$$

However, one of the most useful properties is the probability distribution that describes how likely the system is to be in a certain state. This state probability distribution is evaluated in stationary state, a concept that is introduced in Section 11.2.2 for discrete Markov chains and described in Section 11.1.2 for continuous chains. As in [12] and [31], a state transition rate diagram for the total system can be formulated.

From the usage of transition rates, it is implied that the $M/M/1$ queue is a continuous time Markov chain model. Therefore, the usage of this underlying model is limited to queueing systems with arrival and departure processes that can occur in continuous time.

What is interesting to find in this model are the state probabilities. As can be seen in Fig 12.2, the $M/M/1$ queue can be seen as a continuous birth-death chain with homogeneous rate parameters. Thus, the rate matrix is given as

$$Q = \begin{bmatrix} -\lambda & \lambda & 0 & 0 & 0 & 0 & \dots \\ \mu & -(\lambda+\mu) & \lambda & 0 & 0 & 0 & \dots \\ 0 & \mu & (-\lambda+\mu) & \lambda & 0 & 0 & \dots \\ 0 & 0 & \ddots & \ddots & \ddots & \ddots & \ddots \\ \vdots & \vdots & & & & & \end{bmatrix} \tag{12.3}$$

From this, the relationship in (11.12), which reads as

$$\frac{d\pi(t)}{dt} = \pi(t)Q \tag{12.4}$$

can be used to form the state probability differential equations for each step in the process. Now note that if this was a homogeneous pure birth chain, the differential equations would be

$$\frac{d\pi_j(t)}{dt} = -\lambda\pi_j(t) + \lambda\pi_{j-1}(t)$$

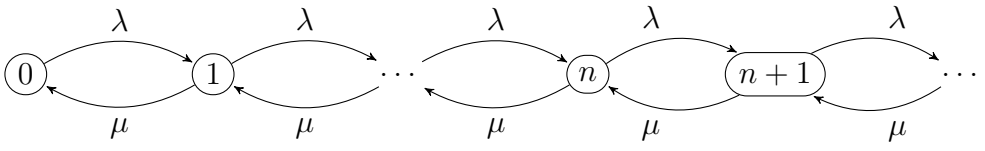


Figure 12.2: State transition rate diagram for the $M/M/1$ queueing system.

for any j. The solution form to these type of differential equations are

$$\pi_j(t) = \frac{(\lambda t)^j}{j!} e^{-\lambda t}$$

which is the Poisson distribution. While (12.3) still describes a homogeneous chain, there are service rates present. The differential equations then take a different form, which can be seen in [12]. These equations are much more difficult to solve, and it is much easier to use the steady state solution first described in (11.13), which implies setting the derivative in (12.4) to zero. Then, for the homogeneous chain where all transition rates are λ and μ, the derived expressions are

$$\begin{aligned} \bar{\pi}_1 &= \frac{\lambda}{\mu}\bar{\pi}_0 \\ \bar{\pi}_2 &= \frac{\lambda}{\mu}\bar{\pi}_1 = \frac{\lambda^2}{\mu^2}\bar{\pi}_0 \\ &\vdots \\ \bar{\pi}_j &= \left(\frac{\lambda}{\mu}\right)^j \bar{\pi}_0 \\ &\vdots \end{aligned} \tag{12.5}$$

In a state, all transition probabilities must sum to one. Therefore, the last term in (12.5) can be used to express the sum over all j terms, summing to one, as

$$\bar{\pi}_0 + \bar{\pi}_0 \sum_{j=0}^{\infty} \left(\frac{\lambda}{\mu}\right)^j = 1 \tag{12.6}$$

Expression (12.5) and (12.6) are used to form

$$\begin{aligned} \bar{\pi}_0 &= \frac{1}{1 + \sum_{j=1}^{\infty} \left(\frac{\lambda}{\mu}\right)^j} \\ \bar{\pi}_j &= \left(\frac{\lambda}{\mu}\right)^j \bar{\pi}_0 \end{aligned} \tag{12.7}$$

The sum will converge if μ is strictly greater than λ, and then the two expressions can be combined to obtain the steady state probability distribution of the $M/M/1$ system, which is

$$\bar{\pi}_n = (1 - \frac{\lambda}{\mu})\left(\frac{\lambda}{\mu}\right)^n \tag{12.8}$$

where $n = 0, 1, 2, \ldots$. In the style of [31], the expected queue length in steady state can also be obtained as

$$\mathbb{E}[s(t)] = \sum_{n=0}^{\infty} n\bar{\pi}_n = \cdots = \frac{\lambda/\mu}{1 - \lambda/\mu} \tag{12.9}$$

With the continuous $M/M/1$ queue defined it is time to investigate the uniformised discrete time version of this system.

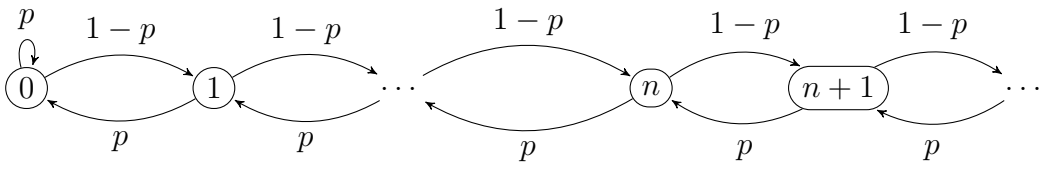


Figure 12.3: State transition rate diagram for the $M/M/1$ queueing system.

Uniformisation of the M/M/1 system

An important feature of the uniformised homogeneous $M/M/1$ queue is that its steady state probability mass function is stochastically equivalent to the original non-uniformised $M/M/1$ queue. This is shown in [12], and is summarised here.

Remembering that a non-uniformised $M/M/1$ queue with arrival rate λ and service rate μ where $\lambda < \mu$ has the stationary state probabilities of (12.8), a minimal uniformisation rate $\gamma = \lambda + \mu$ is selected for the $M/M/1$ queue. Uniformisation is done according to the strategy in Section 11.3, and since the minimal uniformisation rate is selected, a death gets the probability $p = \mu/\gamma$ while a birth has the probability $1 - p$ in each state. There are no self loops in any state, except for in the first one where the probability to stay in state is p. Since $1 - p = 1 - \mu/\gamma$ and $\gamma = \lambda + \mu$, $p - 1 = \lambda/(\lambda + \gamma) = \lambda/\gamma$, so the structure of Fig 12.3 can easily be verified as a diagram of the correct uniformisation.

In this discrete time process, the stationary state probabilities meet the condition $\pi(k+1) = P\pi(k)$. For the discrete time $M/M/1$ queue, also known as a birth/death sequence, the probability to be in a specific state n is given by

$$\pi_n = (1 - (1 - p)/p)((1 - p)/p)^n \tag{12.10}$$

described exactly as in [12] where p is the probability of a death occurring. In the uniformised case, the probability of a death, or decrease, is $p = \mu/\gamma$. Substituting this into the expression above yields

$$\pi_n = (1 - \frac{\lambda}{\mu})(\frac{\lambda}{\mu})^n \tag{12.11}$$

which is the stationary state probability distribution of the $M/M/1$ queue in (12.8). Thus, the stationary state probabilities of the uniformised $M/M/1$ queue is the same as for the original continuous time $M/M/1$ queue.

12.2.2 M/M/1/K system

In an $M/M/1/K$ system, the arrival and departure processes are still Markovian and thereby exponentially distributed. However, the queue can now only be K elements long. The continuous time Markov chain corresponding to this system has the transition rate diagram depicted in Fig 12.4.

In [31], this system is described as a queue in which the arrival process is turned off when the state, or length, of the queue reaches the capacity K. In [12], it is furthermore pointed out that turning off the Poisson arrival process is possible due

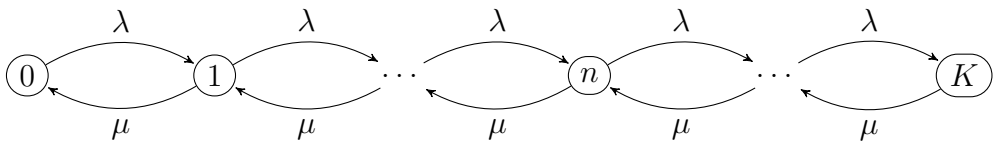


Figure 12.4: State transition diagram for the uniformised $M/M/1/K$ system.

to the fact that the distribution of the interarrival times will not change because the process is shut off - this is an effect of the memoryless property.

Since this type of queue is only different to the $M/M/1$ queue in that it has a limit, focus here lies on the property that is most important for the experiments conducted in this work, which is the steady state probabilities.

The steady state probability distribution is derived in a similar manner as for the $M/M/1$ queue, in that the steady state relationship $d\pi(t)/dt = \pi(t)Q = 0$ is used. In the homogeneous $M/M/1$ queue, the probability of the first state can be found using the first equation of (12.7). In the $M/M/1/K$ system, the infinite upper limit of the sum is exchanged for the capacity K, which implies that the now finite geometric series can be evaluated as a constant, albeit a different constant than in the infinite case. The entire expression for the probability of being in state zero in (12.7) is then

$$\bar{\pi}_0 = \frac{1 - \lambda/\mu}{1 - (\lambda/\mu)^{K+1}} \tag{12.12}$$

This is then used in the second expression of (12.7), and the steady state probability distribution for the $M/M/1/K$ system is obtained as

$$\bar{\pi}_n = \begin{cases} \frac{1-(\lambda/\mu)}{1-(\lambda/\mu)^{K+1}}(\lambda/\mu)^n & \text{if } 0 \leq n \leq K \\ 0 & \text{if } n > K \end{cases} \tag{12.13}$$

Uniformisation of the M/M/1/K system

In [12], where the concept of uniformisation is obtained, no proof is given on that the uniformised $M/M/1/K$ queue has the same stationary state distribution as the continuous time version, the stationary state probabilities of which is given in (12.13). However, the method that is used in the proof of the same property regarding $M/M/1$ queues can be adapted to fit $M/M/1/K$ queues.

Using the same uniformisation factor as in the previous uniformisation of the $M/M/1$ queue, the minimal rate $\gamma = \lambda + \mu$ is selected, since this represents the maximal flow of rates out of a state in the $M/M/1/K$ queue, which can be observed in Fig 12.4. Then, there are as in the $M/M/1$ case no self loops in any state, except for in the first and last state. If $p = \mu/\gamma$ represents the uniformised probability of a death, then $1 - p$ is the probability of a birth, and the probability of a self loop in the first and last states are p and $p-1$, respectively. The diagram of the uniformised $M/M/1/K$ queue is depicted in Fig 12.5.

From the stationary state relationship in a discrete Markov chain, $\pi = P\pi$ holds. From this and the probabilities shown in the state transition diagram of Fig 12.5,

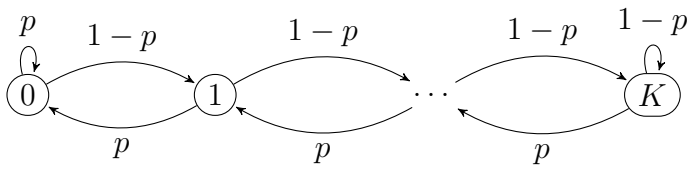


Figure 12.5: State transition diagram for the uniformised $M/M/1/K$ system.

the equations

$$\begin{aligned}
\pi_0 &= \pi_0 p + \pi_1 p \\
\pi_1 &= \pi_0(1-p) + \pi_1 p \\
&\vdots \\
\pi_n &= \pi_{n-1}(1-p) + \pi_{n+1} p \\
&\vdots \\
\pi_K &= \pi_{K-1}(1-p) + \pi_K(1-p)
\end{aligned} \tag{12.14}$$

are formed. By recursively expressing π_1 as a function of π_0, π_2 as a function of π_1 and so on, the general expression is found to be

$$\pi_n = \Big(\frac{1-p}{p}\Big)^n \pi_0 \tag{12.15}$$

From this expression, the probability of the next to last state can be expressed as

$$\pi_{K-1} = \Big(\frac{1-p}{p}\Big)^{K-1} \pi_0 \tag{12.16}$$

This is now inserted in the last equation of (12.14), such that

$$\pi_K = \Big(\frac{1-p}{p}\Big)^{K-1} \pi_0 (1-p) + \pi_K(1-p) \tag{12.17}$$

By reordering this, the probability of the last state reads as

$$\pi_K = \Big(\frac{1-p}{p}\Big)^{K} \pi_0 \tag{12.18}$$

This follows the structure of (12.15), so in general

$$\pi_n = \Big(\frac{1-p}{p}\Big)^n \pi_0 \qquad \text{for } 1 < n \leq K \tag{12.19}$$

The law of total probability now states that the sum of state probabilities should be one. The sum of the probability of state 0, the probabilities of states 1 through $K-1$ expressed in (12.15) and the probability of the last state K obtained from (12.18) is thus

$$\pi_0 + \sum_{n=1}^{K-1} \Big(\frac{1-p}{p}\Big)^n \pi_0 + \Big(\frac{1-p}{p}\Big)^K \pi_0 = 1 \tag{12.20}$$

Now, $c = \frac{1-p}{p}$ is set, such that

$$\pi_0 + \sum_{n=1}^{K-1} c^n \pi_0 + c^K \pi_0 = 1 \tag{12.21}$$

As $c < 1$, a well known result is that

$$\sum_{n=1}^{K-1} c^n = \frac{1 - c^K}{1 - c} - 1 \tag{12.22}$$

which is substituted into (12.21), which with some manipulation now reads as

$$\pi_0 \Big(1 + \frac{1 - c^K}{1 - c} - 1 + c^K\Big) = 1 \tag{12.23}$$

From this, π_0 is obtained from

$$\pi_0 \Big(\frac{1 - c^K + c^K(1 - c)}{1 - c}\Big) = 1 \implies \pi_0 = \frac{1 - c}{1 - c^{K+1}} \tag{12.24}$$

By considering the substitution $c = \frac{1-p}{p}$ in (12.19), $\pi_n = c^n \pi_0$ which can be noted to hold not only for $1 < n \leq K$, but also for $n = 0$ as $\pi_0 = c^0 \pi_0 = \pi_0$. Therefore, by considering the newly obtained expression for π_0,

$$\pi_n = \frac{1 - c}{1 - c^{K+1}} c^n \qquad \text{for } 1 \leq n \leq K \tag{12.25}$$

Since $p = \mu/\gamma = \mu/(\lambda + \mu)$

$$c = \frac{1 - p}{p} = \frac{1 - \mu/(\lambda + \mu)}{\mu/(\lambda + \mu)} = \frac{\lambda}{\mu} \tag{12.26}$$

If this is inserted into 12.25, then

$$\pi_n = \frac{1 - \lambda/\mu}{1 - (\lambda/\mu)^{K+1}} (\lambda/\mu)^n \qquad \text{for } 0 \leq n \leq K \tag{12.27}$$

which is exactly the expression for the stationary state probabilities for the continuous $M/M/1/K$ queue, seen in (12.13). Thus, this concludes the proof that the uniformised version of the $M/M/1/K$ queue has the same stationary state distribution as the original, continuous time $M/M/1/K$ queue.

With two Markovian queueing systems defined, it is time to start considering how to develop control policies for them.

12.3 Markov Decision Processes and Analytical Solutions to the Optimality Equation

Section 2.2 of Part I is where Markov decision processes (MDPs) are first introduced in this work. In short, Markov decision processes are Markov chains in which transition probabilities can be changed by control actions.

While the early formulation is ideal for the discrete reinforcement learning setting, and particularly Q-learning, this section formulates the decision problem for continuous Markov chains. Q-learning is one way of using a strategy related to dynamic programming to find the optimal policy for the MDP, but one of the key concepts of Part II of this work is to investigate situations where a continuous time MDP problem can be divided into two sub-problems. The idea is then that for one of these, the optimal policy can obtained analytically, and then this knowledge can be used to simplify the combined problem.

While specific control problems are presented in the next chapter, cost functions in continuous and discrete time are presented here, followed by a description on how the principle of dynamic programming solution can be used to find solutions to MDP problems. Lastly, a specific form of solution to MDP problems is described.

12.3.1 Cost functions in continuous time

Recall the discrete time expected cost over a finite horizon expressed in (2.10) of Section 2.3.2. Here, in the style of [12], the value function is instead defined for a continuous Markov process.

Let $C[s(t), a(t)]$ be the bounded cost of the process being in state $s(t)$ and selecting the control action $a(t)$ at time t. Then, it is reasonable that the total cost during a time interval $[0, T]$ in the continuous case is an integration that reads

$$V_\pi = E_\pi\Big[\int_0^T C[s(t), a(t)]dt\Big] \tag{12.28}$$

where V_π now denotes the total cost according to a policy π from the given state $s(0)$ at time $t = 0$. This function can also be expressed for an infinite horizon by taking the limit $T \to \infty$ in (12.28).

It can from a practical stand point be reasonable to use a discount factor, similar to the one used in Q-learning in Section 2.3.3. This reflects that expected costs that are closer in time affects the cost function more. With a cost factor β, the infinite time cost function is

$$V_\pi = E_\pi\Big[\int_0^\infty e^{-\beta t} C[s(t), a(t)]dt\Big] \tag{12.29}$$

Finally, the expected average cost can also be formulated by dividing the function in (12.28) by T and then letting $t \to \infty$.

12.3.2 Cost functions of uniformised continuous models

Strategies for finding the optimal solution to a continuous time MDP are often much easier to find in discrete time, since Bellman's principle of optimality can be used through the concept of dynamic programming to find an optimal policy, as is described in Section 2.3.2. Therefore, it is of interest to find a way to use these methods also when finding optimal policies for continuous time problems, and uniformisarion is the method of choice. In the uniformised model, transition rates

are replaced with transition probabilities through

$$\begin{aligned} \gamma &\geq \max_i\{-Q_{ii}\} \\ P_{ij} &= \begin{cases} \frac{Q_{ij}}{\gamma} & \text{if } i \neq j \\ 1 + \frac{Q_{ii}}{\gamma} & \text{if } i = j \end{cases} \end{aligned} \tag{12.30}$$

where Q_{ii} and Q_{ij} come from the transition rate matrix Q. In [12], is is shown that for a discount factor β and a unifomisation rate γ

$$V_\pi = E_\pi\Big[\int_0^\infty e^{-\beta t} C[s(t), a(t)]dt\Big] = \frac{1}{\beta+\gamma} E_\pi\Big[\sum_{k=0}^{\infty}\Big(\frac{\gamma}{\beta+\gamma}\Big)^k C(s(k), a(k))\Big] \tag{12.31}$$

where $s(k)$ and $a(k)$ are the state and control actions at discrete time index k. Thus, the problem of minimising the cost function of a continuous Markov decision process through a policy can be converted to a corresponding discrete cost minimisation problem for the uniformised model. Using this conversion, the principles of dynamic programming can be used also for continuous models.

12.3.3 Optimal solution using the DP algorithm

Now, the most important results regarding the expected cost, obtained from [12], may be presented. These form the foundation for determining analytical optimal solutions to specific Markov decision processes.

First, by setting $\alpha = \gamma/(\beta+\gamma)$ and letting the factor $1/(\beta+\gamma)$ be part of the cost $C(s(k), a(k))$, the cost of a discrete Markov decision process can be set to

$$V_\pi = E_\pi\Big[\sum_{k=0}^{\infty} \alpha^k C(s(k), a(k))\Big] \tag{12.32}$$

If the problem is already discrete, this expression would be obtained from the start, formulated without the uniformisation factor γ. The cost is determined by the control policy π, and the goal is to find the optimal policy that minimises the total expected cost.

For a discrete problem, [12] shows that the expected cost of being in a specific state s can be divided into two parts; the immediate cost $C(s, a)$ and the potential cost V at the next step. A key insight is now that the value of a state can be determined by the values of all the consequent states, multiplied by the probability to transition to those states. Consider a time horizon of length N, with k being the number of steps left to get to time N. Then, the expected value of some state s with $k+1$ steps left to the time horizon is given in terms of all the possible states that the process can be in when there are k steps left to the time horizon as

$$\mathbb{E}[V_{k+1}(s(k+1) = s)] = \sum_{\text{all } s'} P(s, a, s')V_k(s') \tag{12.33}$$

where $P(s, a, s')$ is an element in the transition probability function P corresponding to the probability of transitioning from state s to state s' by the control action a. Now, as described already in Section 2.3.2, under the condition that $V_k(s')$ denotes

the *optimal* value of a state and control pair, the optimal control action that can be selected in a state must be the one that minimises the sum of the immediate cost at the current state, $C(s, a)$, and the expected future costs. This is expressed in [12] as

$$V_{k+1}(s) = \min_a \left[C(s,a) + \alpha \sum_{\text{all } s'} P(s,a,s') V_k(s') \right] \tag{12.34}$$

for $k = 0, 1 \ldots N$. This minimisation problem is a formulation of the dynamic programming algorithm and thus follows the principle of optimality; if a sequence of optimal actions is extended by another optimal action, the new sequence is also optimal. As such, this expression can be compared to the other expression for the dynamic programming recursive expression in Section 2.3.2.

The cost when there are zero steps left to the time horizon is zero, such that $V_0(s) = 0$ for any state s. Similarly, the optimal total cost for the finite horizon problem is given by $V_N(s)$ for any state s.

The expression (12.34) is the optimality equation for the finite horizon case. However, the optimal control policy π^* that minimises the total cost (12.32) is what is interesting. One intuitive way of finding π^* is to let the time horizon N approach infinity. However, [12] states that this procedure does not guarantee a solution unless certain conditions hold for the available actions in state s. If the set of available control actions $\mathcal{A}(s)$ in state s is finite and the immediate cost is always such that $C(s, a) \geq 0$, then

$$\lim_{N \to \infty} V_N(s) = V^*(s) \tag{12.35}$$

for some initial state s_0.

12.3.4 Threshold type problems

By evaluating the optimality equation given by

$$V(s) = \min_a \left[C(s,a) + \alpha \sum_{\text{all } s'} P(s,a,s') V(s') \right] \tag{12.36}$$

on an infinite horizon, it can be shown that certain systems have very specific optimal solutions. One of these solutions is called a *threshold* type solution in [12]. These problems are formulated for systems with states indexed by
$s_0 = 0, s_1 = 1 \ldots s_i = i, \ldots$, and they can be shown to have an optimal policy for which one control value is assumed if the state index $i \leq i^*$ and another is taken if $i > i^*$. Note here that these types of problems are highly specific to certain cost function structures, and it can not be generally stated that certain problems have threshold type solutions unless explicitly investigating the structure of the cost function.

If problems can be shown to be of threshold type, obtaining the optimal policy boils down to obtaining the threshold, which compared to many other methods of optimization is much less demanding in terms of computation. In this work, the procedure outlined in [12] is used to analytically investigate if a problem is of threshold type. It consists of five steps.

Step 0: Uniformise the continuous Markov decision process

This is only necessary if the problem is continuous originally. For a discrete process, this can be skipped.

Step 1: Determine the optimality equation

For any state with index i, an expression for the optimal control action must be determined. Already, some restrictions on the Markov decision process can be seen; a constraint on the set of available control actions $\mathcal{A}(s)$ is already that it needs to be finite, but to keep complexity down it is naturally easier the smaller the set is. Often, the problem can be difficult enough to solve if there are only two different control actions to take for each state.

Furthermore, this procedure assumes that a general expression can be obtained for any state in the process. If there are different regions in the Markov process that need different functions V, the advice is to divide this problem into one smaller problem per region.

Step 2: Determine one step optimal control actions

In each step, it must be possible to minimise $V(i)$ by the use of a control action $a^*(i)$. In this way, identify all cases where each of the possible control signals minimise $V(i)$ in terms of elements in the optimality equation described in the previous step.

Step 3: Obtain a closed form expression for the optimal policy

Using the cases in which each possible control signal minimises the value function, determine an expression in terms of the optimal cost function development that decides when each possible control action should be used. Thus, the goal is to decide which control signal a to use, depending on the change in cost function between two states. This change is denoted $\Delta V(i) = V(i) - V(i-1)$.

Step 4: Show that $\Delta V(i)$ is monotonically increasing

Since the appropriate control signal is selected based on the value of $\Delta V(i)$, the goal is to find the value for which $\Delta V(i) = V(i) - V(i-1)$ enters the region where the optimal control signal changes. The i for which this happens is then i^*. However, for this to hold, the function ΔV must me nondecreasing for all i. On the positive side, once this property is proven, the remaining issue is finding the threshold i^*.

Step 5: Obtain the threshold i^*

Given that there exists a state index where ΔV crosses over to the region where the minimising control action changes, the threshold is found. Effectively, based on the difference in V between consecutive states, regions are identified where each possible control action is the minimising action. Again, it is possible to have more possible control values than two, but for each of these, a region must be defined in terms of

ΔV, which can be difficult. Furthermore, determining the optimal threshold i^* thus requires explicit knowledge of $V(i)$.

There are now multiple ways of determining this threshold value. The first idea is that since the nature of the control signal is determined up to the threshold i^*, the problem reduces to a Markov chain where the control is already known for states up to i^*. For many systems, such as queues, there are well defined methods to express stationary probability distributions that determine the probability for the system to be in a certain state. Since the value function is based on the expected state value, the stationary probability distributions can be used to evaluate the expected state.

If there is not a well defined stationary state probability distribution, the optimal threshold value can be estimated through simulation. If the problem is simulated for an appropriate number of times over a horizon, different threshold values can be experimented with, and an approximate region in which a minimising threshold may exist can be found. The strength of this method is that it does not rely on accurate expressions for state probabilities. The weaknesses is that it is computationally heavy and does not produce an optimal solution due to it being a simulation based method.

12.4 Summary

The queueing theory chapter starts with the concepts that are interesting to investigate in queueing systems. These continuous time processes can often be seen as customers arriving from a population that is in most cases infinite, and they wait for service from one or several servers that handle customers according to a service principle. In this work, two types of queueing systems are investigated, namely the $M/M/1$ and the $M/M/1/K$ queue. These are Markovian queues, which means that their interarrival and service times are exponentially distributed with rates λ and μ, respectively. The notation that these queues are described in is called Kendall notation, where the M stands for Markovian and the K denotes a queue that has a limited capacity and rejects arrivals when the length is K.

Different ways of measuring the performance of queues are discussed, and the most important property of the models are, in both the case of the $M/M/1$ and $M/M/1/K$ queues, the stationary state probability mass functions.

The concept of uniformisation is described for $M/M/1$ and $M/M/1/K$ queues, and the important property which says that the stationary state distributions are the same for continuous and uniformised versions of these queues is proven for the minimal uniformisation rate in both cases.

In the later part of this chapter, the concept of Markov decision processes are described for continuous Markov processes in general. In a decision problem, the objective is to minimise a cost function, and these can be formulated in both continuous and discrete time. An important property is that for a continuous cost function, an equivalent cost function can be formulated for a uniformised Markov decision problem.

Lastly, a procedure to identify threshold type control policies for specific cost functions is given in five steps. In the next chapter, this procedure is used to show that a specific admission control problem for an $M/M/1$ queue has a threshold type

solution. In that case, the control problem reduces to finding the threshold i^*, which can be done analytically or through simulation.

13
Control Problems

In this chapter, the problems that are investigated in Part II of this thesis are presented in detail. Firstly, the admission control problem for $M/M/1$ queues is formulated. This problem is motivated by the need to limit the number of customers admitted to a system. For such a problem, a cost function can be formulated. This function has two parts, as both keeping a long line of customers and rejecting customers comes at a cost. The goal is then to find the optimal policy at which customers are rejected.

In the second and final control problem in this work, a joint continuous Markov process is studied. The first of the two decision processes that make up the joint process is a path planning process for a slippery grid world formulated in continuous time. Although the concept is familiar from Part I of this thesis, the difference is now that the problem is described using a continuous time Markov process. The second decision process is the new admission control problem, and based on the discussions on how to join two independent Markov processes, the total model has to be expressed in continuous time since the admission control problem is formulated for a continuous time $M/M/1$ queue.

Lastly, modular analysis is presented in the form of analytically determining an optimal admission control policy for the $M/M/1$ queue part of the joint process. This can be used to reduce the computational burden that solving the joint MDP problem entirely using temporal logic constrained reinforcement learning would imply.

13.1 Admission Control for M/M/1 Queues

This problem, described in [12], is defined for a $M/M/1$ queueing system. As is shown in the previous chapter, this is a continuous, Markovian, single server queueing system with arrival rate λ and service rate μ.

13.1.1 Problem formulation

The problem consists of deciding if an arrival shall be admitted into the queue or not. States are in this problem formulation denoted by $s(t) = i$, which means that there are i elements in the queue. Thus, the possible control values are dependent on how many elements there are in the queue and they are denoted

$$a(i) = \begin{cases} 1 \text{ if arrival is rejected} \\ 0 \text{ if arrival is admitted} \end{cases} \tag{13.1}$$

when the state is $s(t) = i$ elements in the queue.

Rejecting a customer costs R, but it also costs to have admitted customers in the queue. The cost of this is B per customer per unit time. In continuous time, the discounted total expected cost is

$$V_\pi = \mathbb{E}_\pi\Big[\int_0^\infty e^{-\beta t} \cdot B \cdot s(t)dt + \int_0^\infty e^{-\beta t} R \cdot A(t) \cdot a(s(t^-))dt\Big] \tag{13.2}$$

where $A(t) = 1$ if there is an arrival at t and $A(t) = 0$ if there is not. Here, it is assumed that the initial queue state is zero elements, such that the generally formulated initial state at time $t = 0$ is $s(0) = 0$. The first integral is a measure of all admission costs over time, and the second is a measure of all rejections over time. The notation t^- implies that the rejection is decided just before it occurs.

Recall the transition rate matrix for the $M/M/1$ queue from Section 12.2.1 as

$$Q = \begin{bmatrix} -\lambda & \lambda & 0 & 0 & 0 & 0 & \dots \\ \mu & -(\lambda+\mu) & \lambda & 0 & 0 & 0 & \dots \\ 0 & \mu & (-\lambda+\mu) & \lambda & 0 & 0 & \dots \\ 0 & 0 & \ddots & \ddots & \ddots & \ddots & \ddots \\ \vdots & \vdots & & & & & \end{bmatrix} \tag{13.3}$$

The first step of finding an analytical solution is uniformisation, and the uniformisation rate is as usual the largest negated diagonal element in Q. Since this is a homogeneous Markov chain, the rate is selected as any of the negated diagonal elements as $\gamma = \lambda + \mu$. Following the uniformisation procedure of (12.31), the new cost parameters are

$$\begin{aligned} \alpha &= \gamma/(\beta+\gamma) \\ b &= B/(\beta+\gamma) \\ r &= R/(\beta+\gamma) \end{aligned} \tag{13.4}$$

For the discrete problem, a discrete time equivalent cost function is then

$$V_\pi = \mathbb{E}_\pi\Big[\sum_{k=0}^{\infty}\{\alpha^k bi + \alpha^k r A(k+1)a(i)\}\Big] \tag{13.5}$$

where the state notation $s(k) = i = 0, 1, 2, \dots$ is used directly.

Now, $A(k+1)$ denotes that an arrival occurs at step $k+1$. The discounted cost of having i elements is always present, but the cost r is only collected if an arrival occurs at the next step and rejection is selected at the step before, which means that the action when the state is i is $a(i) = 1$. Moreover, note that the next step implies one step further from the horizon.

Since transitions to an incremented state can be avoided by the use of a control value, the state transition probabilities are dependent on the control value. If admission is selected, the one step probability of transitioning to the incremented state is denoted by the constant $p = \lambda/\gamma$, but if rejection is selected, $p = \lambda/\gamma$ describes the probability of staying in the state. In both cases, $q = \mu/\gamma$, which is the discrete time probability to decrease the number of elements in the queue. A special case is

when $i = 0$; in this case it is not possible for departure to occur and the probabilities are changed thereafter.

In total, the probabilities found in the action dependent transition probability function P of the controlled process are

$$P(i,a,j) = \begin{cases} p[1-a] & \text{if } j = i+1 \\ q & \text{if } j = i-1 \\ pa & \text{if } j = i \\ 0 & \text{otherwise} \end{cases} \tag{13.6}$$

if $i > 0$. Here, a is the action selected at step k when the state is $s(k) = i$ and the next state is $s(k+1) = j$. For $i = 0$,

$$P(0,a,j) = \begin{cases} p[1-a] & \text{if } j = 1 \\ 1-p+pa & \text{if } j = 0 \\ 0 & \text{otherwise} \end{cases} \tag{13.7}$$

where the action is a is selected at step k when the state is $i = 0$ and the next state is $j = 1$.

13.1.2 Verifying the threshold type solution

Now, the five step solution to the admission control problem is presented, as it is in [12]. The uniformised model was obtained before, which means that the first step is to express the optimality equation for the problem. The starting point is to consider how the one step ahead optimality equation of (12.34) can be minimised by choosing an optimal control action. Recall that (13.5) is the cost criterion used in the optimality equation. The optimality equation can then express the *momentary* optimal cost that, if it is applied at all steps of the horizon, will minimise (13.5). Using the obtained uniformised probabilities, the optimality equation for the admission control problem becomes

$$V(i) = \begin{cases} \min\limits_a \Big[bi + \alpha\Big(p[1-a]V(i+1) + pa[r+V(i)] + qV(i-1) \Big) \Big] & \text{if } i > 0 \\ \min\limits_a \Big[\alpha\Big(p[1-a]V(i+1) + pa[r+V(i)] + (1-p)V(i) \Big) \Big] & \text{if } i = 0 \end{cases} \tag{13.8}$$

Since this expression must hold for the optimal V, the optimality equation says that in each step, the control signal that minimises the immediate cost and the expected future costs must be selected. Either, admission is selected at state i, and the expected value is dependent on $V(i+1)$ and $V(i-1)$ as $i-1$ and $i+1$ are the states to which the system can transition after admission is selected. If rejection is selected, the only possible state transitions are to states $i-1$ and i. In both cases, costs r and/or b are collected depending on the control action selected.

The second step is to determine regions where $a = 0$ and $a = 1$. This is done in [12] by combining and simplifying (13.8) to form

$$V(i) = [bi + \alpha(pV(i+1) + qV([i-1]^+))] + \ \min(\alpha p(r + V(i) - V(i+1)), 0) \tag{13.9}$$

Here, $[i-1]^+$ is either 0 if $i = 0$ or $i-1$ if $i > 0$. Note that the minimisation problem is now expressed as the minimum of the two alternative results of $a = 1$ and $a = 0$. It can now be seen that

$$a^* = \begin{cases} 0 & \text{if } V(i+1) \leq r + V(i) \\ 1 & \text{otherwise} \end{cases} \tag{13.10}$$

The third step is to obtain a closed form of the optimal policy. This is done by setting

$$\Delta V(i+1) = V(i+1) - V(i) \tag{13.11}$$

for any i. Using this, the optimal control can be expressed as

$$a^* = \begin{cases} 0 & \text{if } \Delta V(i+1) \leq r \\ 1 & \text{otherwise} \end{cases} \tag{13.12}$$

Now, since the set of available actions $\mathcal{A} = \{0, 1\}$ is finite, (12.35) states that

$$\lim_{N\to\infty} V_N(i) = V^*(i)$$

The next step is to show that the function $\Delta V(i)$ is monotonically increasing. If $\Delta V(i)$ is nondecreasing, there will be a point where $\Delta V(i+1) > r$ which makes $i = i^*$. Although there is no expression for $V(i)$, $\Delta V(i)$ can fortunately still be shown to be monotonically increasing, by the process of induction.

For a finite version of the problem, the optimality equation can be expressed as in (12.34). Remember that k denotes the number of steps left to the horizon N, so $k = 0$ implies that the optimality equation is evaluated at the horizon. At the horizon, $V_k(i) = 0$. The finite horizon optimality equation at $k = 0$ and at any time index $k + 1$ is then

$$\begin{aligned} V_0(i) &= 0 \\ V_{k+1}(i) &= \Big[bi + \alpha\Big(pV_k(i+1) + qV_k([i-1]^+)\Big)\Big] + \min\Big(\alpha p(r - \Delta V_k(i+1)), 0\Big) \end{aligned}$$

where the $\Delta V(i+1)$ notation is used. At $k = 0$, $V_0(i) = 0$ for all i, since the problem is at the horizon. Therefore, $\Delta V_0(i) = V_0(i) - V_0([i-1]^+) = 0$, which is zero but still non-decreasing.

For any k, now assume that $\Delta V_k(i)$ is non decreasing for all i. This is reasonable since the cost of increasing state values are made up of strictly increasing immediate costs, added to the previous state values. Nowhere is there a negative cost development as the state increases, for a specific k. Thereby,

$$\Delta V_k(i+1) \geq V_k(i) \tag{13.13}$$

always holds. From this, $\Delta V_{k+1}(i)$ can now be shown to be monotonically increasing. If this holds, the principle will by induction also hold for increasing indices up to N when N approaches infinity, which was shown to converge to the original infinite horizon solution.

For $k+1$, $\Delta V_{k+1}(i)$ is increasing if $\Delta V_{k+1}(i+1) - \Delta V_{k+1}(i) \geq 0$. The first term is

$$\begin{aligned}
&\Delta V_{k+1}(i+1) = V_{k+1}(i+1) - V_{k+1}(i) = \\
&= [b(i+1) + \alpha(pV_k(i+2) + qV_k(i))] + \min[\overbrace{\alpha p(r - \Delta V_k(i+2))}^{c_1}, 0] - \\
&- [bi + \alpha(pV_k(i+1) + qV_k([i-1]^+))] - \min[\underbrace{\alpha p(r - \Delta V_k(i+1))}_{c_2}, 0]
\end{aligned}$$

As $\min(c_1, 0) - \min(c_2, 0) \geq \min(c_1 - c_2, 0)$ for any c_1 and c_2, the two minimisation terms can be combined into

$$\min(c_1, 0) - \min(c_2, 0) \geq \min[\alpha p(\Delta V_k(i+1) - \Delta V_k(i+2)), 0]$$

Since $\Delta V_k(i)$ is assumed to be non-decreasing, the first alternative of the minimisation term is smaller than zero. If this inequality is used in the expression for $\Delta V_{k+1}(i+1)$, the resulting expression is

$$\Delta V_{k+1}(i+1) \geq [b + \alpha(p\Delta V_k(i+2) + q\Delta V_k(i))] + \alpha p(\Delta V_k(i+1) - \Delta V_k(i+2)) \quad (13.14)$$

For $\Delta V_{k+1}(i)$, the expression is

$$\begin{aligned}
&\Delta V_{k+1}(i) = V_{k+1}(i) - V_{k+1}([i-1]^+) = \\
&= [bi + \alpha(pV_k(i+1) + qV_k([i-1]^+))] + \min[\overbrace{\alpha p(r - \Delta V_k(i+1))}^{d_1}, 0] - \\
&- [b(i-1) + \alpha(pV_k(i) + qV_k([i-2]^+))] - \min[\underbrace{\alpha p(r - \Delta V_k(i))}_{d_2}, 0]
\end{aligned}$$

As $\min(d_1, 0) - \min(d_2, 0) \leq -\min(d_2 - d_1, 0)$ for any d_1 and d_2, these two minimisation terms can be combined into

$$\min(d_1, 0) - \min(d_2, 0) \leq -\min[\alpha p(\Delta V_k(i+1) - \Delta V_k(i)), 0]$$

In this case, the minimal term is zero. Therefore,

$$\Delta V_{k+1}(i) \leq [b + \alpha(p\Delta V_k(i+1) + q\Delta V_k([i-1]^+))] - 0 \quad (13.15)$$

If (13.15) is negated, the inequality is flipped. Then, by summing both sides of this expression with both sides of (13.14), the expression

$$\begin{aligned}
\Delta V_{k+1}(i+1) - \Delta V_{k+1}(i) \geq\ & [b + \alpha(p\Delta V_k(i+2) + \\
& + q\Delta V_k(i))] + \alpha p(\Delta V_k(i+1) - \Delta V_k(i+2)) - \\
& - [b + \alpha(p\Delta V_k(i+1) + q\Delta V_k([i-1]^+))] \quad (13.16) \\
& \Longleftrightarrow \\
\Delta V_{k+1}(i+1) - \Delta V_{k+1}(i) \geq\ & \alpha p(\Delta V_k(i) - \Delta V_k([i-1]^+))
\end{aligned}$$

The right hand side of the final expression in 13.16 is by assumption always positive. Therefore, it has been proven that $\Delta V_{k+1}(i+1) - \Delta V_{k+1}(i) \geq 0$, and this extends to the infinite horizon problem as well. Thus, $\Delta V(i)$ is a non-decreasing function.

Now that the admission control problem has been shown to be a threshold type problem, the fifth and last step is to identify the optimal threshold i^* for which the optimal policy switches from admission to rejection. This can generally be done in two ways, according to [12]; either by using a well known state probability distribution for the Markov chain, or by simulation.

13.2 Two Methods of Finding the Optimal Threshold

While the threshold type problem structure is a powerful property, finding the actual threshold can be easier said than done. In this work, two methods for deriving the threshold are discussed.

To formulate an admission control problem, a cost function is needed. Given a specific cost function, the goal is then to find a suitable method that can be used to compute the actual value of the threshold. Before going further into the two methods that can be used to do so, an important property from [12] must be highlighted.

In Section 12.3.2, it is said that the cost function of a continuous Markov decision process has an equivalent description in discrete time. If a process is uniformised, its cost function can be uniformised, too. As a reminder, the relationship between the two functions is repeated here. The total expected discounted cost from an initial state $s(0)$ up to an infinite horizon is

$$V_\pi = \mathbb{E}_\pi\Big[\int_0^\infty e^{-\beta t} C[s(t), a(t)]dt\Big] \tag{13.17}$$

and uniformisation with rate γ leads to the relationship

$$V_\pi = \mathbb{E}_\pi\Big[\int_0^\infty e^{-\beta t} C[s(t), a(t)]dt\Big] = \frac{1}{\beta+\gamma}\mathbb{E}_\pi\Big[\sum_{k=0}^{\infty} \alpha^k C(s(k), a(k))\Big] \tag{13.18}$$

where $\alpha = \gamma/(\beta+\gamma)$. On the right hand side of (13.18) is the uniformised discrete time cost function, expressed in terms of the uniformised process. Furthermore, as $\beta = 0$ gives the undiscounted infinite horizon cost function, the equivalence also holds for the undiscounted case. The point here is then that one can equivalently and arbitrarily select between using either the continuous time cost function or the uniformised discrete time cost function. In the two methods of deriving i^* described in this work, the continuous time cost function is used.

13.2.1 Finding the optimal threshold using the M/M/1/K distribution and an undiscounted cost function

Firstly, it must be noted that this solution revolves around the well known statistical relationship

$$\mathbb{E}[s(t)] = \sum_{n=1}^{K} nPr\{s(t) = n\} \tag{13.19}$$

where $Pr\{s(t) = n\}$ is obtained as the stationary state probability distribution of the $M/M/1/K$ queue. This relationship says nothing about the absolute time that is necessary for expressing the discount in (13.17). Therefore, β is set to zero, and

$$V_\pi = \mathbb{E}_\pi\Big[\int_0^\infty C[s(t), a(t)]dt\Big] \tag{13.20}$$

In the previous section, it is shown that the admission control problem has a threshold type solution. Since the queue cannot grow larger than i^*, the controlled

queue can be seen as an $M/M/1/K$ queue, where $K = i^*$. The idea is now to evaluate the cost function of the $M/M/1/K$ queue for increasing K, and then select i^* as the K that produced the lowest cost function. To do so, the cost function must first be explicitly defined. In [12], the procedure of actually finding the optimal threshold for this type of specific admission control problem is not described, but using the $M/M/1/K$ stationary state distribution or using simulation are two suggested directions.

The starting point is to reformulate the continuous time admission control cost function from [12], previously expressed in (13.2), for a finite horizon of N unit time intervals, which reads

$$V_\pi = \mathbb{E}_\pi\Big[\int_0^N e^{-\beta t} \cdot B \cdot s(t)dt + \int_0^N e^{-\beta t} R \cdot A(t) \cdot a(s(t^-))dt\Big] \tag{13.21}$$

where the first integral describes the cost to keep the queue, and the second describes the cost of rejection. Since the problem is undiscounted, the next step is setting $\beta = 0$, such that

$$\begin{aligned} V_\pi &= \mathbb{E}_\pi\Big[\int_0^N B \cdot s(t)dt + \int_0^N R \cdot A(t) \cdot a(s(t^-))dt\Big] \\ &= \mathbb{E}_\pi\Big[\int_0^N B \cdot s(t)dt\Big] + \mathbb{E}_\pi\Big[\int_0^N R \cdot A(t) \cdot a(s(t^-))dt\Big] \end{aligned} \tag{13.22}$$

Focusing on the second integral, this expression describes the control action just before an arrival takes place, at the time denoted by t^-. Note that this is not to be interpreted as a non-causal system, but rather by the physical analogy that a sensor reading from a car bumper can warn the driver that a crash event will occur in the next second. Instantaneously, $A(t) \cdot a(s(t^-)) = 1$ if an arrival occurs, and is multiplied by R to form the cost of a rejection. Now, set this instantaneous unit value to be the *Dirac unit impulse*, defined by

$$\delta(t - t_a) = \begin{cases} +\infty & t - t_a = 0 \\ 0 & t - t_a \neq 0 \end{cases} \tag{13.23}$$

where $t \leq t_a$. For each rejection that occurs during the interval $[0, N]$, a Dirac unit impulse is defined at the time of the rejection, and as they do not overlap due to the definition of the continuous time Markov process, the integrated sum of these pulses multiplied by R makes up the total cost of the rejections during the interval defined by the horizon. Then, the second integral becomes

$$\mathbb{E}_\pi\Big[\int_0^N R \cdot A(t) \cdot a(s(t^-))dt\Big] = \mathbb{E}_\pi\Big[R\int_0^N \sum_{j=1}^{N_r} \delta(t - r_j)dt\Big] \tag{13.24}$$

where r_j are the rejection times and N_r is the number of rejections occurring on the interval.

The integral of a Dirac delta function is well known as

$$\int_{-\infty}^{\infty} \delta(t - t_a)dt = 1 \tag{13.25}$$

and as such, integrating all N_r impulses occurring on the interval and multiplying the result with R is just $R \cdot N_r$. However, the number of impulses, or the number of rejections, is a random variable. N_r is then the expected number of rejections during the time interval between 0 and the horizon N, which can be calculated.

If the horizon spans N unit time intervals, then $N \cdot \lambda$ arrivals are expected during the N unit time intervals by the definition of the Poisson arrival process with rate λ. If the queue is in the final blocking state, the arrival is rejected. In stationary state, the probability for the $M/M/1/K$ queue to be in the final state is

$$P_K = Pr\{s(t) = K\} = \frac{1 - (\lambda/\mu)}{1 - (\lambda/\mu)^{K+1}}(\lambda/\mu)^K \tag{13.26}$$

This means that P_K can represent the share of expected arrivals that will be rejected. Thus, up until the horizon N, the total expected rejection cost is

$$\begin{aligned}\mathbb{E}_\pi\Big[R\int_0^N \sum_{j=1}^{N_r}\delta(t - r_j)dt\Big] &= R \cdot \mathbb{E}_\pi\Big[\int_0^N \delta(t - r_1)dt + \cdots + \int_0^N \delta(t - r_{N_r})dt\Big] \\ &= R \cdot \mathbb{E}_\pi[N_r] \\ &= R \cdot P_K \cdot N \cdot \lambda\end{aligned} \tag{13.27}$$

For the first integral of (13.22), which is the expected total cost of keeping the queue in its states, the starting point is to note that $s(t)$ is a piece-wise constant signal. As the only other term in the integral, B, is constant, the value of the integral is just a sum of integrated constants, independent on the order of the constant values. Moving the expectation notation inside the integral yields

$$\mathbb{E}_\pi\Big[\int_0^N B \cdot s(t)dt\Big] = B\int_0^N \mathbb{E}_\pi[s(t)]dt \tag{13.28}$$

and this expression allows for the use of (13.19). Then,

$$B\int_0^N \mathbb{E}_\pi[s(t)]dt = B\int_0^N \sum_{n=1}^{K} nPr\{s(t) = n\}dt \tag{13.29}$$

is obtained. This expression can be evaluated as N integrals over a unit length time interval, in which the stationary state probability distribution of the $M/M/1/K$ queue can be used to find

$$Pr\{s(t) = n\} = P_n = \begin{cases} \frac{1-(\lambda/\mu)}{1-(\lambda/\mu)^{K+1}}(\lambda/\mu)^n & \text{if } 0 \le n \le K \\ 0 & \text{if } n > K \end{cases} \tag{13.30}$$

This means that the previous cost expression becomes

$$B\int_0^N \sum_{n=1}^{K} nPr\{s(t) = n\}dt = N \cdot B \cdot \sum_{n=1}^{K} P_n \cdot n \tag{13.31}$$

Finally, the analytically derived total expected undiscounted finite horizon cost of the admission controlled queue is given by

$$V_\pi(K) = N\Big(B \cdot \sum_{n=1}^{K} P_n \cdot n + R \cdot P_K \cdot \lambda\Big) \tag{13.32}$$

As hinted in the notation, this expression is evaluated for increasing thresholds K. The K that gives the lowest cost $V_\pi(K)$ is then the optimal threshold i^*.

13.2.2 Finding the optimal threshold using simulation and a discounted cost function

There are cases when the analytical method of finding i^* is not applicable. Such scenarios could for example be not having an explicit description of the stationary state probability distribution or the need to evaluate the system in transient states, but it could also be that a discounted cost must be used in the problem. As stated earlier, the analytical steady state probability distribution of the $M/M/1/K$ queue does not contain the necessary absolute time information to express the discounted version of the cost function.

In these cases, simulating the queueing system can be a way of obtaining a cost approximation. In principle, this procedure consists of simulating traces of the $M/M/1/K$ queue for each K, and computing exact discounted costs for each trace. Then, these costs are seen as samples of the actual cost, and an average over many traces can be obtained. To start off, consider the discounted cost function over the finite horizon of length N unit time intervals

$$V_\pi(K) = \mathbb{E}_\pi\Big[\int_0^N e^{-\beta t} \cdot B \cdot s(t)dt + \int_0^N e^{-\beta t} R \cdot A(t) \cdot a(s(t^-))dt\Big] \tag{13.33}$$

For a simulated state trajectory $s(t)$, the exact cost is denoted by simulation index S such that

$$V_\pi^S(K) = \int_0^N e^{-\beta t} \cdot B \cdot s(t)dt + \int_0^N e^{-\beta t} R \cdot A(t) \cdot a(s(t^-))dt \tag{13.34}$$

where the threshold K of the queue indirectly influences the cost, as it affects the number of rejections N_r. As in the analytical procedure, the rejections in the second integral is expressed using non-overlapping Dirac delta unit impulses. During a trace, N_r rejections occur, and

$$V_\pi^S(K) = \int_0^N e^{-\beta t} \cdot B \cdot s(t)dt + \int_0^N e^{-\beta t} R \sum_{j=1}^{N_r} \delta(t - r_j))dt \tag{13.35}$$

Now, as there is an exponential discount term, order is important, but $s(t)$ is still piece-wise constant. This means that the first integral can be expressed as

$$\begin{aligned}\int_0^N e^{-\beta t} \cdot B \cdot s(t)dt &= B\Big(s(t_0)\int_{t_0}^{t_1} e^{-\beta t}dt + s(t_1)\int_{t_1}^{t_2} e^{-\beta t}dt + \cdots + s(t_{N-1})\int_{t_{N-1}}^{t_N} e^{-\beta t}dt\Big) \\ &= \frac{B}{\beta}\sum_{i=0}^{N-1} s(t_i)\Big[e^{-\beta t_{i+1}} - e^{-\beta t_i}\Big]\end{aligned} \tag{13.36}$$

For the second integral of (13.35), a well known property of the Dirac delta function can be used. It reads

$$\int_{-\infty}^{\infty} f(t)\delta(t - t_a)dt = f(t_a) \tag{13.37}$$

For all N_r rejections described by non-overlapping Dirac delta functions in the simulation, the second integral then becomes

$$\begin{aligned}
&\int_0^N e^{-\beta t} R \sum_{j=1}^{N_r} \delta(t - r_j))dt = \\
&= R \int_0^N e^{-\beta t} \delta(t - r_1) + R \int_0^N e^{-\beta t} \delta(t - r_2) + \cdots + R \int_0^N e^{-\beta t} \delta(t - r_{N_r}) \\
&= R \sum_{j=1}^{N_r} e^{-\beta r_j}
\end{aligned} \tag{13.38}$$

since for all rejections, $0 \leq r_j \leq N$ holds. For one simulation with index S, the expression for the total cost is then

$$V^S(K) = \frac{B}{\beta} \sum_{i=0}^{N-1} s(t_i) \left[e^{-\beta t_{i+1}} - e^{-\beta t_i} \right] + R \sum_{j=1}^{N_r} e^{-\beta r_j} \tag{13.39}$$

To obtain an average cost over the finite horizon N, N_S simulations are made, where N_S is large enough for the sample to statistically represent the actual expected cost. Finally, an estimate of the discounted cost function for a specific K is then obtained as

$$\hat{V}(K) = \frac{1}{N_S} \sum_{S=1}^{N_S} V^S(K) \tag{13.40}$$

This simulation is performed for different values of K ranging from one to some limit, just as in the analytical derivation of the optimal threshold. The K that produces the lowest $\hat{V}(K)$ is then the optimal threshold i^* with respect to the discounted total expected cost over the finite horizon.

13.3 Path Planning and Admission Control in Continuous Markov Processes

So far, Part I handled temporal logic constrained path planning problems in discrete time. Now, the objective is to solve a joint continuous Markov decision process problem, where one of the subsystems is a grid world for which path planning actions should be selected, while the second part of the total system is an $M/M/1$ queue for which admission control should be implemented.

The main issue which makes this control problem difficult to solve using temporal logic constrained reinforcement learning directly is that the state space of the joint process is infinite, since the $M/M/1$ queue has no capacity limit. While a policy may be possible to find eventually given that there is sufficient memory to store a very large Q-table, the strategy is instead to use modular analysis of the queue sub-process to limit the joint system state space, while also finding the optimal admission control action selection policy.

In Section 13.1, the admission control problem for queueing systems is introduced. For the specific cost function, the solution is shown to be of the threshold type, which implies that the cost of the admission control problem only increases

above the threshold. As the threshold is a queue length, no lower cost for either the queue or the joint process is expected above the threshold queue length, and this is what motivates that solving the admission control problem separately reduces the number of states in the infinite joint state space that actually needs to be explored. After this is done, the path planning part of the process can be solved with temporal logic constrained reinforcement learning to follow a specification for the joint system.

The starting point is to consider a joint continuous Markov process consisting of a model of a path planning problem and a $M/M/1$ queue for which admission control is to be implemented. In addition to this, an LTL formula that specifies the desired behaviour of the learning agent needs to be expressed.

13.3.1 Problem formulation

Consider an example of a joint system, consisting of a robot that has the task to navigate through a grid world and operating a machine at the same time. Parts arrive to the machine, and they depart as they are serviced by it one at a time. The machine can be modeled as a single server queueing system of infinite capacity, and the world that the robot navigates through is modelled as a path planning problem similar to those in Part I of this work. The queue is to be regulated by the use of admission control, and for simplicity, it is assumed that the robot can regulate the machine remotely. If the queue is not full, the robot should navigate to a desired geographical location while avoiding obstacles, and regulating admissions to the machine at the same time. However, if the queue is full, then the machine might malfunction, and the robot must physically visit it to make sure it does not break.

The idea is now to come up with a way for the robot to operate the queueing system in an optimal way, a way that also satisfies an LTL specification formulated for the joint process. In contrast to the problems discussed in Part I, the assumption is now that the models are continuous time Markov processes.

The procedure to solve this problem using modular analysis is outlined in two steps. Firstly, from Section 13.1, a threshold type admission control solution is found for the queue sub-process, and the threshold i^* is explicitly formulated either through the analytical method of Section 13.2.1 or through simulation from Section 13.2.2. This solution is independent of the path planning part of the joint process and thereby also unrelated to any control policy that regulates navigation.

Secondly, the temporal logic constrained Q-learning algorithm operates on a discrete time representation of the problem. Therefore, the joint system must be uniformised to form an equivalent model in discrete time. This must moreover be done in the presence of control actions, as these regulate the different transitions (and by extension the total transition rates) in each joint state.

Lastly, when an optimal threshold for the queue part is found and a discretisation is made, a temporal logic constrained reinforcement learning problem can be formulated for the discretised and finite state space joint Markov decision process. This specification may regulate any part of the process, but not the control actions regulating the queue part, which are determined through the previously described

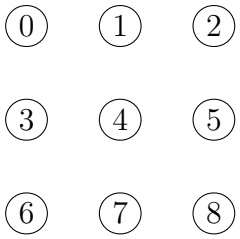


Figure 13.1: Example state space of the continuous time Markov grid world.

analytical solution.

To start off, the joint process needs to be described explicitly, and this is done by first recalling the gridworld state space structure.

13.3.2 Continuous time path planning

In Section 11.4.1, it is described how both of two Markov processes must either be discretely or continuously described to make sense of the joint process. If the joint process is continuous, a continuous time path planning problem must be considered.

The state space of this problem is an indexed finite set of natural numbers. Graphically, this system can be represented by a grid, shown in Fig 13.1.

GSMP, state space and transitions

In the underlying GSMP, there are eight possible navigation events; four actions representing north, east, south and west, and four slip events in the same direction. By counting the occurrences of each event, the counts can represent the number of steps taken in the four cardinal directions that the events represent. Therefore, this follows the same principle as how the queue state is really a counting function of arrival and service events, but in the grid world there are more actions and the state is a coordinate.

Since this is a continuous Markov process, the main directional events have rates, denoted λ_N, λ_E, λ_S and λ_W, respectively. Furthermore, the intervenent times of the events that trigger the directional transitions are assumed to be exponentially distributed with the same parameters; this implies that the process of only transitioning in one direction is a Poisson process with a rate parameter specified by one of the previously mentioned rates.

In addition, to reflect a slippery grid world such as the one used in Part I of this thesis, four extra events are considered in the underlying GSMP. For every main directional event, there is an associated slip event that has the exact same effect on the state space, but it occurs at a much lower rate than the main directional event. The rates of these are denoted ω_N, ω_E, ω_S and ω_W, and just as in the case of the main directional events, these events have exponentially distributed intervenent times with the previously mentioned ω-rates. Counting the occurrences of any of these events is therefore also a Poisson counting process, defined by the respective rate.

Just as in the queueing systems, considering all of the above events being possible in the same system is a way of superpositioning the eight different Poisson processes,

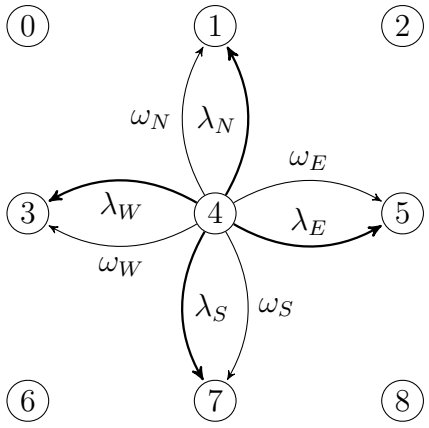


Figure 13.2: Intended direction transitions λ and slip transitions ω.

which is also a Poisson process according to the reasoning in Section 10.4.

For the example in Fig 13.1, the transitions available from the non-border state are depicted with their respective rates, shown in Fig 13.2. In any border state, the transitions that cannot be completed due to the lack of a northern, eastern, southern or western state are redirected back to the origin state.

Control

As in the admission control problem, and as [12] points out, an inherent property of the Poisson process is that it can be "turned off" without changing the probability distribution. This is because the probability is only dependent on time intervals and not on specific time instances; the probability to increase a Poisson process at one time is the same no matter what the absolute time is. By this, the same type of control used in admission control in which transitions are either made available or not, transitions between the states of the path planning problem can be controlled.

Consider a four element unit control vector $a(t)$, describing the control action selected in the joint state I at time t. The elements of this vector can only assume values of either one or zero, and only one of the elements can be one at any time. Next, denote a directional four element vector containing one of the direction rates by D and a slip transition rate vector by Ω. The non-zero element in $a(t)$ denotes which direction is selected, while the elements in D and Ω are selected to reflect that for a selected direction, the most frequent transition is to the intended state, but it is also possible to slip to any of the other states, at a lower rate.

For the non-border state depicted in Fig 13.2, the controlled transition rates are given as

$$\begin{aligned}
(D_N + \Omega_N)a(t) &= \Big(\begin{bmatrix} \lambda_N & 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & \omega_N & \omega_N & \omega_N \end{bmatrix} \Big) a(t) \\
(D_E + \Omega_E)a(t) &= \Big(\begin{bmatrix} 0 & \lambda_E & 0 & 0 \end{bmatrix} + \begin{bmatrix} \omega_E & 0 & \omega_E & \omega_E \end{bmatrix} \Big) a(t) \\
(D_S + \Omega_S)a(t) &= \Big(\begin{bmatrix} 0 & 0 & \lambda_S & 0 \end{bmatrix} + \begin{bmatrix} \omega_S & \omega_S & 0 & \omega_S \end{bmatrix} \Big) a(t) \\
(D_W + \Omega_W)a(t) &= \Big(\begin{bmatrix} 0 & 0 & 0 & \lambda_W \end{bmatrix} + \begin{bmatrix} \omega_W & \omega_W & \omega_W & 0 \end{bmatrix} \Big) a(t)
\end{aligned} \tag{13.41}$$

If for example the selected direction is east, then $a(t) = \begin{bmatrix} 0 & 1 & 0 & 0 \end{bmatrix}$ and the above

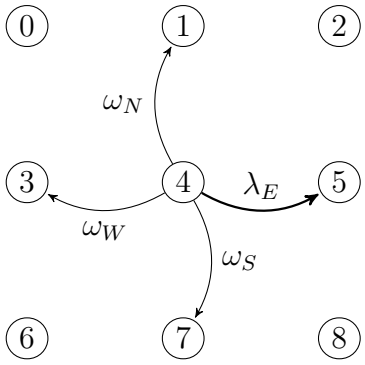


Figure 13.3: Possible transitions when $a(t) = \begin{bmatrix} 0 & 1 & 0 & 0 \end{bmatrix}^T$

expressions change to form the transitions depicted in Fig 13.3.

Uniformisation rate

Now that the continuous time Markov decision process that describes the path planning problem is described, the last key step is to identify the path planning state that has the largest transition flow out of it, which is to be used in uniformisation. Even though it is entirely possible to have completely unique rates for all events in the process, a simplification is that

$$\begin{aligned} \lambda_N = \lambda_E = \lambda_S = \lambda_W = \lambda_D \\ \omega_N = \omega_E = \omega_S = \omega_W = \omega_D \end{aligned} \tag{13.42}$$

Using this, it can be noted that for any control action vector $a(t)$ in the path planning problem, the maximum possible transition rate out of a state is

$$\gamma_{PP} = \lambda_D + 3\omega_D \tag{13.43}$$

since in all states, for all control actions, there is always one main directional transition with rate λ_D and three slip transitions with rates ω_D.

13.3.3 Jointly controlled queue and path planning in continuous time

Recall that a continuous time Markov decision process describing the admission control of an $M/M/1$ queue has a control signal that is either $a = 1$ for rejection or $a = 0$ for admission. When $a = 0$ in a state, there is a transition with rate λ to the incremented state, but when $a = 1$ this transition is redirected to be a self loop with rate λ. The maximum flow out of a state is therefore the sum of λ and μ. Therefore, even in the presence of control actions this process is uniformised with rate $\lambda_Q = \lambda + \mu$, which is also the case in the description in [12].

By now, the slippery grid world path planning and the $M/M/1$ admission control processes have been formulated with the inclusion of control actions. Before moving on to defining the jointly controlled MDP consisting of the admission controlled

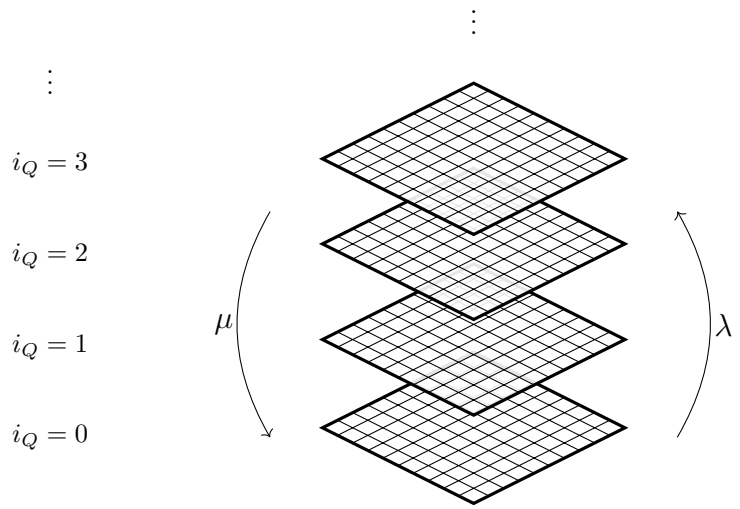


Figure 13.4: The joint process ordered as a tower of path planning state sets.

$M/M/1$ queue and the controllable path planning process, the important structure of the transitions in the joint continuous system must be described.

The most intuitive way to formulate the joint process is to first consider the state space of the path planning process, and an empty queue. It is possible for the path planning process to be in any of its states while an arrival occurs in the queue with rate λ. By considering the tuples state of the joint system, this implies the transition between the state $I = (i_{PP}, 0)$ and $J = (i_{PP}, 1)$ where i_{PP} denotes an integer valued state index in the path planning process. In short, from a state in the joint process, either queue transitions or path planning transitions can be made. The joint system can therefore be imagined as a three dimensional tower, where two of the dimensions correspond to the coordinates of the path planning process and the third dimension corresponds to the state of the queue. This is depicted in Fig 13.4.

The layout depicted in Fig 13.4 implies that a joint state, where the path planning state is a non-border state, is simply a path planning state specified by a queue value, and with two additional transitions that increase and decrease the queue part of the process. A typical joint state where the queue part is not empty and the path planning part is not in a border state is depicted in Fig 13.5.

In Fig 13.5, transitions made by the queue between joint states are regulated by admission control. When the admission control action is $a = 0$, a transition can be made to the joint state with an incremented queue index in the joint state tuple, while a self loop is induced if the admission control action is $a = 1$. The control actions for the path planning transitions are also present in the same manner as explained in Section 13.3.2.

To make the notation in the joint control problem simpler, the control vector $a(t)$ is augmented to fit the single element queue control signal. The joint control problem depicted in Fig 13.5 can then be expressed only by the vector $a(t)$ which contains both the path planning and admission control actions, and the transition notation in Fig 13.5 is then changed accordingly.

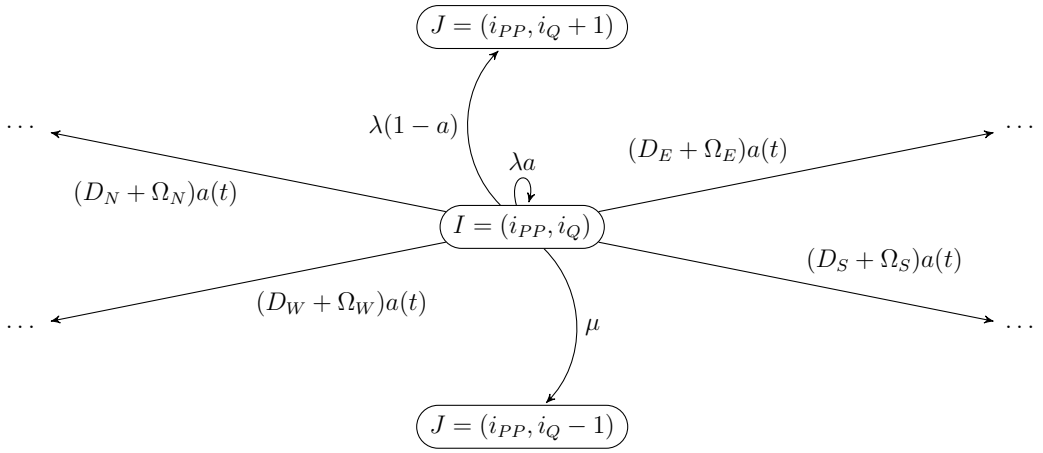


Figure 13.5: Joint state with admission control over the queue transitions and path planning controlled directional transitions with slip.

13.3.4 Uniformisation of the controlled joint Markov process

Now that the joint Markov decision process of the queue and the path planning is described in continuous time with both control over admissions to the queue and navigation in the path planning part, the goal is to find the path planning control policy using reinforcement learning. Since the temporal logic constrained Q-learning is the selected method, the problem must be formulated in discrete time in order to be solved.

In general, uniformisation is used to obtain a discrete version of any Markov process formulated in continuous time, as it can be recalled that this process does not change the important properties related to the probability distributions of the process according to [12]. Furthermore, as the control actions enter the Markov process linearly with respect to the transition rates, the uniformisation can be done before or after control is introduced.

The uniformisation is done according to the process described in Section 11.4.3, but it requires the identification of the joint state that has the maximum transition flow in order to find the uniformisation rate γ. From previous sections, it is known that this rate is the sum of the two individual minimum uniformisation rates, which are decided from the individual process states with maximum transition rate flow. In the queue, this is any state that has both the incrementing and decrementing transitions, which means that the minimal unformisation rate of the queue is $\gamma_Q = \lambda + \mu$. This is the case in all control cases, since the only thing that changes in admission control is the target state of the transition with rate λ. In the controlled path planning process, the minimal individual uniformisation rate is, according to the derivation and assumptions in Section 13.3.2, $\gamma_{PP} = \lambda_D + 3\omega_D$. Therefore, the uniformisation rate of the joint process is

$$\gamma = \lambda + \mu + \lambda_D + 3\omega_D \tag{13.44}$$

Next follows a detailed description of the uniformised transition probabilities of the controlled joint process of the path planning and the queue. The joint state

categories considered are edge states, corner states and any non-border state of the path planning problem, combined with the empty queue state and the non-empty queue state.

Uniformised probabilities

Formulating explicit expressions for uniformised transition probabilities is not fruitful as they are too many to keep track of. A more efficient way of depicting all scenarios is to formulate the probabilities in a standard situation and then describe the effect of different deviations from this case.

If the path planning is not in a corner or border state, and the queue is neither empty nor full, the controlled process is in the same situation as is depicted in Fig 13.5. Since the rates for the main directional transitions are equal, as are the rates of the slip transitions, the path planning control can be expressed arbitrarily such that it describes the desire to go in any of the four directions.

Furthermore, it is assumed that threshold type admission control is applied, such that the queue control action is to admit when the queue is not full. Therefore, the standard situation is that some arbitrary direction $d = 1$ is selected, and directions denoted 2, 3 or 4 symbolise any of the three slip directions. Additionaly, the queue can either increase or decrease. The control vector for this situation is denoted $a_1(t)$, and the uniformised probabilities in this situation are given as

$$P_{IJ}|a_1(t) = \begin{cases} (1 - \sum /\gamma) & \text{if } I = J \\ \lambda_D/\gamma & \text{if } I \neq J \text{ and dir. is 1} \\ \omega_D/\gamma & \text{if } I \neq J \text{ and dir. is 2} \\ \omega_D/\gamma & \text{if } I \neq J \text{ and dir. is 3} \\ \omega_D/\gamma & \text{if } I \neq J \text{ and dir. is 4} \\ \mu/\gamma & \text{if } I \neq J \text{ and } j_Q = i_Q - 1 \\ \lambda/\gamma & \text{if } I \neq J \text{ and } j_Q = i_Q + 1 \end{cases} \tag{13.45}$$

where $\sum$ denotes the sum of transition rates going out of state I. Note that the standard case is based in a maximum flow state so that $\sum = \gamma$, making the probability of a self loop zero.

Additional to this situation, there are eight different categories of special cases, all describing situations in which the queue is either empty, non empty or full, in combination with whether or not the path planning state is a corner or border state.

- Corner path planning state, empty queue. Two path planning transitions imply self loops, and a the queue cannot be decreased.
- Corner path planning state, non empty queue. Two path planning transitions imply self loops.
- Corner path planning state, full queue. Two path planning transitions imply self loops, and the queue cannot be increased.
- Border path planning state, empty queue. One path planning transition implies a self loop, and the queue cannot be decreased.

- Border path planning state, non empty queue. One path planning transition implies a self loop.
- Border path planning state, full queue. One path planning transition implies a self loop, and the queue cannot be increased.
- Middle path planning state, empty queue. The queue cannot be decreased.
- Middle path planning state, full queue. The queue cannot be increased.

These are all situations in which a self loop is either already defined in the path planning transitions, or will be induced by the uniformisation process since the state is not a maximum flow state because one queue related transition is unavailable. These situations are not explicitly defined, but they all boil down to making the self loop probability non zero by removing a transition to another joint state from the standard case. Obviously, since there are four different borders and four different corners in a square grid world, there are more than eight specific special cases.

13.3.5 LTL specification for the reduced joint process

Thus far, it is decided that part of the control policy is found analytically by analysing a known part of the joint Markov process. All joint states where the queue is above the optimal threshold i^* are disregarded, and the queue control actions are defined as admitting up to this limit, where they change to rejection. Thus, queue control tasks are not selected through reinforcement learning, but the path planning actions are.

The desired behaviour of the rest of the system is determined through reinforcement learning constrained by an LTL specification. This part of the solution has a certain flexibility, in contrast to any analytical solutions, since the policy is developed iteratively and not defined by a hard constraint from the start.

This context presents some intriguing scenarios where behaviours can be enforced for the admission control indirectly, by regulating the LTL specification of the joint system through the soft constraint of rewards. For example, while it is not valid to state that admission control actions are only available in certain parts of the joint system, it is valid to formulate an LTL specification that rewards the path planning process more for going to some specific path planning state when admissions are made. Furthermore, transitions that are caused by the queue sub-process are seen as uncontrollable from the perspective of the reinforcement learning algorithm, but will affect the learning since transitions between all joint states are considered in the Q-table update.

Lastly, any LTL formula for the joint process is defined as in Part I of this work, but for joint states. The LTL formula is realised as a Büchi automaton, where the transitions are dependent on that state labels can be observed in joint states. A specific LTL formula for a joint system is not formulated here; this is done in the final experiment of this thesis, found in the next chapter.

13.4 Summary

In this chapter, two control problems are formulated. The first is an admission control problem, which has a cost function that is shown to be of the threshold type. The key take away from this problem formulation is that for a specific cost function formulation, a threshold type policy exists.

The threshold type solution is not very efficient unless an explicit threshold value is found. For an $M/M/1$ queue, this is a specific queue length, and two methods of finding this value are discussed in this chapter. In both cases, the cost function is formulated for a finite horizon instead of an infinite one. The cost discount parameter regulates how much effect future expected state changes have on the cost function, which is evaluated at the initial state. If the discount parameter is zero, which yields an undiscounted cost, the optimal threshold can be found analytically by using the $M/M/1/K$ stationary state probabilities for different limits K. The limit that produces the lowest total expected cost is i^*.

If the discount parameter is nonzero, explicit time is necessary, which is not provided by a stationary state distribution. In that scenario, among others, it is better to simulate $M/M/1/K$ queue trajectories and compute the expected discounted cost as a sample average over all simulations. This is done for increasing K values, until the lowest cost estimate is obtained for $K = i^*$.

The formulation of the joint path planning and admission control problem starts with defining the path planning problem in continuous time. In this problem, it is possible to transition to a state either by intention or by slipping, and these transitions occur at different rates. Path planning control is implemented by the same principle as admission control, which is to consider control as a function that pauses some of the Poisson processes that describe the count of events associated with directional transitions.

Using the fact that part of the problem has a threshold type solution, the joint state space is reduced from an infinite state space to a finite one, limited at i^* in the queue dimension. This is followed by a formulation of all transition probabilities for the controlled joint process, expressed as deviations from a more general situation. This is done by joint uniformisation with a rate corresponding to the sum of the uniformisation rates of the individual path planning and queue process.

To finalise this chapter, LTL specifications concerning the joint process are said to be formulated in the same way as in Part I of this work, but for joint states. Using this specification, temporal logic constrained reinforcement learning can be used to obtain the path planning policy of the reduced state space joint problem. In this way, all actions in the joint process are selected and the solution is complete.

14
Experiments

In this chapter, the experiments conducted in part II of this project are presented. The purpose of these are to test the theory in the previous chapter, from the basic concepts of queues and their properties to the method used to solve the joint path planning and admission control problem. The chapter is divided into four sections that formulate and analyse experiments conducted with $M/M/1$ queues, $M/M/1/K$ queues, methods to find the optimal admission control threshold i^* and finally joint path planning and admission control.

The $M/M/1$ and $M/M/1/K$ queue experiments focus on simulating the two queueing processes in both continuous time and in discrete time for their uniformised versions. By doing so, the property that states that the uniformised and continuous queue descriptions have the same stationary state probabilities can be verified for the simulations. If the simulations produce the correct distributions, they can be used in computing accurate cost functions, such that the optimal admission control threshold can be found. The ultimate goal is to use the optimal threshold in the final joint path planning and admission control problem to reduce the state space of the joint process, and then use temporal logic constrained reinforcement learning to find the path planning actions in the total policy.

The section that is concerned with solving the joint path planning and admission control problem thus relies on the experiments in which a specific optimal threshold i^* is obtained. After having reduced the joint state space by using this threshold, the navigation actions are found using temporal logic constrained reinforcement learning.

14.1 M/M/1 Queues

Since this is the fundamental queueing system model in this work, the dynamics of the $M/M/1$ process need to be understood and demonstrated.

In Fig 14.1, the familiar rate diagram for the $M/M/1$ process is shown. In the experiments conducted in this section, the arrival rate λ is set to 50, and the service

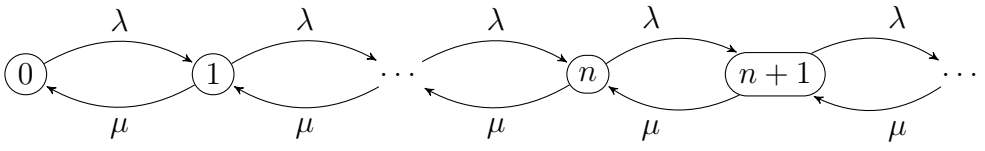


Figure 14.1: State transition rate diagram for the $M/M/1$ queueing system.

rate μ is set to 60. As has been described before, the stationary state probability mass function for this queue is

$$P_n = (1 - \frac{\lambda}{\mu})\Big(\frac{\lambda}{\mu}\Big)^n \tag{14.1}$$

The first experiment of Part II seeks to verify visually that the queueing system dynamics work as expected.

14.1.1 Continuous time M/M/1 queue simulation

The first experiment of the second part of this project is to simulate the $M/M/1$ queue from the basic definition given in Section 12.2.1.

Setup

Simulating a continuous Markov process is a somewhat more refined process than simulating a discrete time process. In many simulation settings, common practice is to divide a stretch of simulation time into smaller equidistant time steps in which one of several probabilistic outcomes are observed and logged. However, this technique cannot be used in the case of continuous Markov processes.

In a continuous Markov process, discrete events may occur at any point in continuous time, and because of this, no (realistic) sampling interval can be defined such that this is really the case. Furthermore, in the $M/M/1$ queue case, the interarrival and service times must be distributed according to the exponential distribution, with rates λ and μ respectively.

To achieve this, a method that ensures the fundamental property of the interarrival and service times is to use the exponential cumulative distribution function (CDF) to generate both interarrival and service times, and then schedule queue events based on these times. By using the CDF of the exponential function, the distributions that define the processes are assured from the start. An example of an exponential CDF is shown in Fig 14.2.

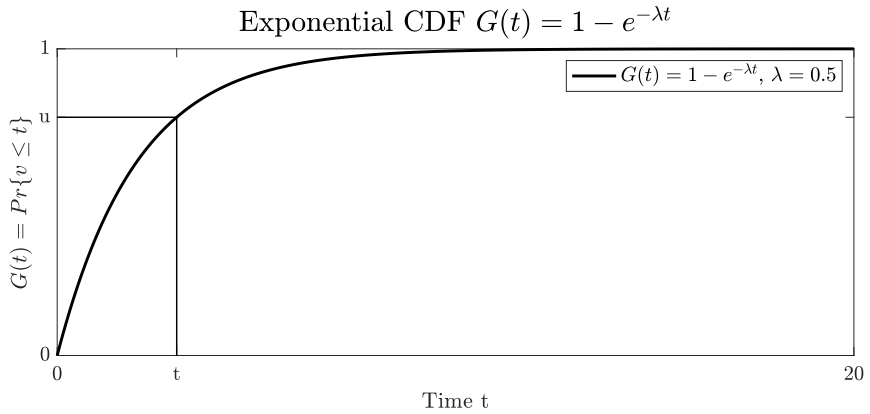


Figure 14.2: Relationship between probability and time interval through the exponential CDF.

The expression $Pr\{v \leq t\} = 1 - e^{-\lambda t}$ states the probability of the interarrival time v being t or less, or alternatively, the probability that an event occurs in the interval $(0, t]$. By simulating the outcome of $G(t)$ as a random number u between 0 and 1, which is a probability generated from a distribution where all outcomes are equally probable, the inverse of the exponential CDF $G(t)$ can be used to find the corresponding value t according to

$$\begin{aligned} u &= 1 - e^{-\lambda t} \\ &\Leftrightarrow \\ e^{-\lambda t} &= 1 - u \\ &\Leftrightarrow \\ t &= -\frac{\ln u}{\lambda} \end{aligned} \tag{14.2}$$

where the fact that if $u \sim U(0,1)$ then $u - 1 \sim U(0,1)$ is used. Here, t denotes the time taken by the random interarrival time variable v, and it represents the stretch of time between the event occurrence at $k-1$ and k.

To simulate the queueing process, a scheduling of queue arrivals and departures must be done. Firstly, arrivals are independent of services, and taking the cumulative sum of all generated interarrival times yields the absolute arrival times. For example, if the interarrival time between customer A and customer B is one $t_{AB} = 1$ second and customer A arrives at $t_A = 0$, then customer B arrives at $t_B = t_A + t_{AB}$.

Secondly, the departure of a customer is calculated by first determining when service starts. If the server is idle at the arrival time of a customer, service of the customer can start right away. If the server is not idle, the new customer must wait until the previous customer has left until service can begin. The departure time of a customer is thus dependent on when the previous customer is scheduled for departure.

For example, if the departure time of the previous customer A is scheduled for d_A, and the arrival time of the new customer B is t_B, then the departure of B is $d_B = t_B + s_B$ where s_B is the service time of customer B, if $d_A < t_B$. If, on the other hand, $t_B < d_A$ (arrival of customer B occurs before customer A has departed), then the departure time of customer B is $d_B = d_A + s_B$, i.e. B must wait for service until customer A has departed until service can begin.

A fixed number of customer arrivals and departures are simulated, and the trajectories of the arrival and departure processes are produced along with a queue trajectory. The simulation is done for a specified number of unit time intervals, and $t = 1$ always specifies one unit time interval. The simulation parameters are then the rates λ and μ along with the specified number of unit time intervals.

In this specific experiment, the queue is simulated for one unit time interval, and the time that is spent in each state is recorded in order to produce a state time distribution, which is compared to the stationary state probability mass function of the $M/M/1$ queue, found in (14.1).

Purpose

Modular analysis revolves around finding a useful model for a subdivision of a joint Markov process. A large portion of the work thus consists of verifying that the model is accurate, so that an accurate control policy for the subprocess can be found.

In this work, the subprocess under investigation has the structure of a queue, and the accurate simulation of this queue is necessary for finding the optimal threshold queue length i^* for when admission stops and rejection takes place. As the $M/M/1$ queue is the basic queueing system model, it is important that it can be verified to work correctly.

Besides simulating the optimal threshold, an accurately simulated $M/M/1$ queue can be used to verify the dynamics of uniformised versions of the queue, which are central in the solution to the admission control problem. If the dynamics behave as expected, the simulation of the $M/M/1$ queue can be used to draw conclusions about cost functions formulated for it, and the principle used to simulate the $M/M/1$ queue can be seen as a working basic principle when simulating other queues such as the $M/M/1/K$ queue.

Hypothesis

The simulated queue is expected to have around $\lambda = 50$ arrivals during the simulated unit time period. The service events are expected to occur after the arrivals, and the queue must increase at arrivals and decrease at departures. As λ/μ is close to but still below one, reaching stationary state is possible, but it will not necessarily happen in one unit time interval. Since the state time distribution should reflect the time spent in each state during both the transient and steady state of the queue, the similarity to the analytical stationary state PMF of (14.1) will not be apparent in the state time distribution if the queue is in its transient state for a long enough duration of the simulation.

Results

The results of the continuous time $M/M/1$ queue simulation are shown in Fig 14.3.

Analysis

In Fig 14.3, the arrival process behaves as expected by visual inspection, as no two events take place at the same time, and around $\lambda = 50$ arrivals occur during the course of one unit time interval. All departures happen after the arrivals, and the difference between the arrival curve and the departure curve at any height represents the time in the queue for the customer with the corresponding arrival and departure time. Long queueing times correspond well to increasing queue value at the same time.

The state time distribution in the lower part of Fig 14.3 shows that one unit time interval is not enough to reflect stationary state probabilities in the state time distribution, and the verification of the analytical state PMF is left to a later experiment. What is shown here is a visual demonstration of the continuous time $M/M/1$ queue, and the queueing system dynamics behave as expected.

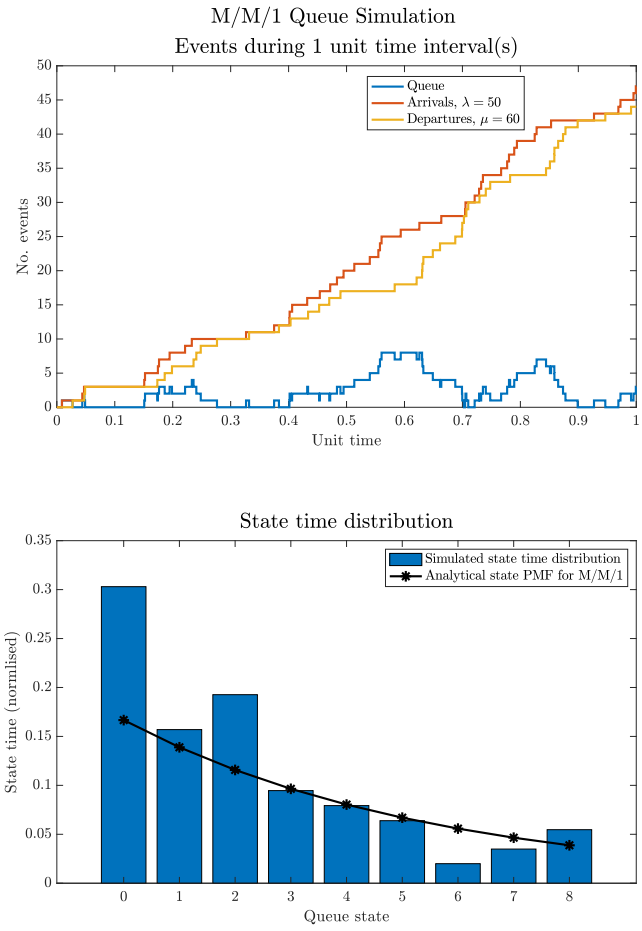


Figure 14.3: $M/M/1$ queueing system simulation with recorded state time distributions.

14.1.2 Uniformised M/M/1 queue simulation

In the second experiment, an $M/M/1$ queue is uniformised to form a discrete time queue representation. In Fig 14.4, the transition diagram of the uniformised queue can be seen with the computed one step probabilities.

Setup

In this experiment, the uniformisation rate is set to be exactly the largest total flow of rates out of a state in the continuous $M/M/1$ queue, which is when both an increasing and a decreasing transition is possible. Thus, $\gamma = \lambda + \mu$, and the probabilities of self loops are zero in all states but the first state. This is represented in the diagram of Fig 14.4.

Simulation of the uniformised discrete time queue is different, and less intricate than that of the original continuous time Markov process version. The queue starts at zero elements, and from this state the two possible outcomes in each simulation

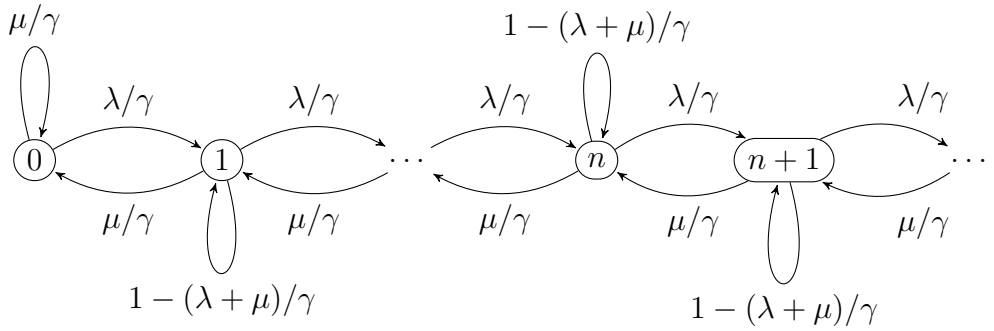


Figure 14.4: State transition diagram for the uniformised $M/M/1$ queueing system.

step is to either stay in the state or to increase the queue if an arrival occurs. For all other states, the two possibilities are either transitioning to an increased state or to a decreased state, since the minimal uniformisation rate is selected which means that self loops do not occur here.

In both the special case of the first state, and generally for all other states, an outcome in each discrete simulation step is evaluated by constructing a probability interval between zero and one. The interval is divided into sub-intervals, and each sub-interval has a length corresponding to its computed probability. There is one sub-interval for each possible transition, such as self loops, increases and decreases.

In each simulation step, a random number between zero and one is drawn from the uniform distribution, and the outcome transition is selected depending on what sub-division the number falls in on the probability interval. Over time, the distribution of executed actions will match the assigned transition probabilities. This method is also used in Part I when simulating discrete time MDPs.

As in the previous experiment, the selected rate parameter for arrivals is $\lambda = 50$, while services times has the rate μ. The uniformisation rate is $\gamma = \lambda + \mu = 110$, and the simulation is run for 100 discrete time steps.

Purpose

The goal of this experiment is to gain a visual understanding of the difference between continuous and discrete queue representations. Ultimately, the most interesting property of the uniformised queues is how their simulated state time (or state visits) distributions compare to the state time distributions of the original continuous time queue representations.

As uniformisation is used to transform a continuous time Markov chain to a form that can be used in reinforcement learning, simulating the uniformised queue behavior is necessary in order to show that modelled properties used in the solution of the joint path planning and queue control problem are reliable.

Hypothesis

A fundamental property of the uniformised queue is that the steady state probability mass function is the same as for the original non-uniformised queue, which is

described in Section 12.2.1. The belief is therefore that if the simulation is run for a sufficient number of iterations, the distribution of time spent in states (which in the discrete case can be abstracted to state visits of unit time length) will take the shape of the distribution for the continuous $M/M/1$ queue. The same uniformisation rate $\gamma = \lambda + \mu$ is selected exactly as in the proof of the equivalence between uniformised and continuous $M/M/1$ queues in Section 12.2.1, but as the purpose is firstly to illustrate the queue state dynamics, the state visits distribution is not expected to converge to the PMF of the stationary state $M/M/1$ state probabilities.

Results

The results of the uniformised $M/M/1$ queue simulation is shown in Fig 14.5.

Analysis

The upper graph of Fig 14.5 shows that events do take place at discrete points in time, instead of at any point in continuous time, visualised in the graph of Fig 14.3

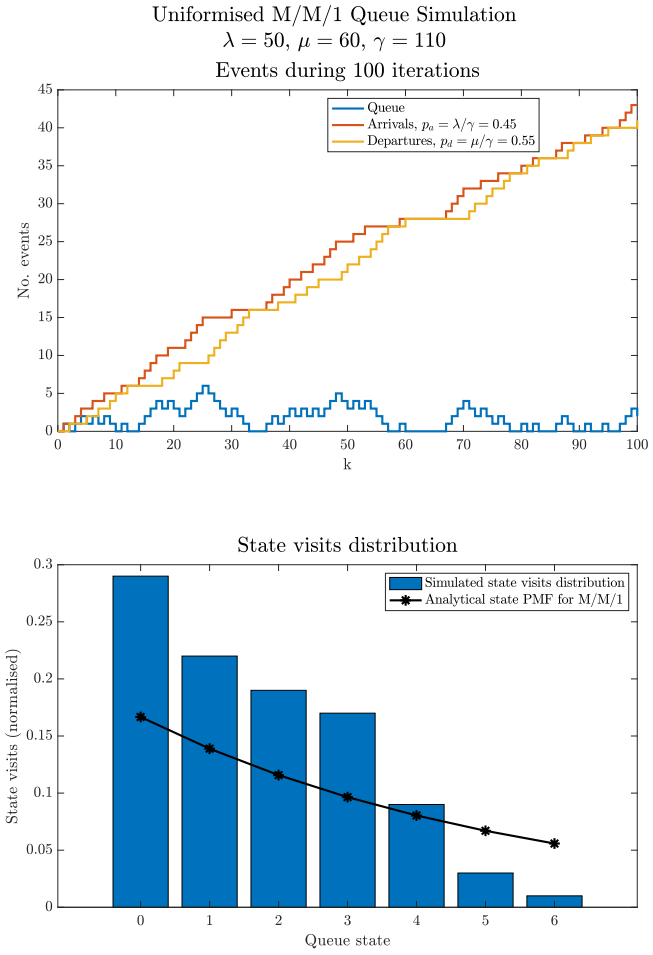


Figure 14.5: Results for the uniformised $M/M/1$ simulation experiment.

describing the previous experiment for the continuous time $M/M/1$ queue. The count of arriving customers is shown at steps k, along with the count of departing customers. At any height, the up-flank of the arrival count curve occurs at the arrival time index of a customer, and the up-flank of the departure count curve at the same height happens at the departure time index of the same customer. Therefore, the horizontal distance between the flanks of each customer arrival and departure gives the customers time in the system, and in periods where many consecutive customers have long system times, the queue state correlates well, as it increases.

Although the simulation is not run for many steps, such that the majority of state visits occur in stationary state, the state visits distribution in the lower graph of 14.5 is showing a trend similar to that of the analytical distribution. However, the next experiment is completely dedicated to verifying the state time distribution convergences of both the continuous time $M/M/1$ queue and its uniformised counterpart.

14.1.3 Simulated M/M/1 queue state probabilities

In this experiment, the proof of the property that the stationary state probability distribution of the continuous time $M/M/1$ queue is equivalent to that of the (with rate $\gamma = \lambda + \mu$) uniformised discrete time $M/M/1$, is verified by simulation.

Setup

The proof, found in Section 12.2.1, states that in stationary state the PMF describing the probability for the system is given by (14.1) for both the continuous time $M/M/1$ queue and the $\gamma = \lambda + \mu$ uniformised version.

To test this, an $M/M/1$ queue is run for 500 unit time intervals, with parameters $\lambda = 50$ and $\mu = 60$. The times spent in each state are recorded and normalised to a probability mass function that can be compared to the analytical PMF of (14.1). This is done using the exact same simulation implementation as the one done previously in Section 14.1.1, the only difference is that the simulation time is increased.

Similarly, a uniformised $M/M/1$ queue is simulated using $\lambda = 50$, $\mu = 60$ and $\gamma = \lambda + \mu = 110$. The implementation used to simulate this is the exact same as in Section 14.1.2, but here the number of simulated steps is 50000. The number of visits to each queue state is recorded, and a PMF is produced that can be compared to that of the continuous queue and (14.1).

Purpose

The purpose of this experiment is to test the theory that if the simulations of the $M/M/1$ queue and its uniformised version are run for a long time, the proportion of time spent in each state out of the total simulation time approaches the analytical PMF describing the stationary state probabilities of the continuous $M/M/1$ queue. This is under the assumption that the majority of the total simulation time is done when the system is in its stationary state.

If this is achieved, the simulated queue dynamics behave exactly as is analytically predicted, which verifies that the simulations behave correctly and are reliable. If so, it is motivated to rely on and draw statistical conclusions based on both the analytical expressions and the simulated behaviors. For example, the simulations can be used when calculating the expected costs of queueing system control policies in subsequent experiments.

Hypothesis

Firstly, both queueing systems are implemented in such a way that the underlying stochastical processes are clearly correct. For example, the simulated interarrival and service times are drawn directly from the exponential CDF in the continuous time case, while one step probabilities are formulated exactly according to the uniformisation process using rates in the discrete time case. Secondly, the rates are such that $\lambda < \mu$, which is necessary for stationary state to take place. The remaining factor is then only to run the simulations for a long enough time.

The hypothesis is finally that the recorded proportion of total simulation time spent in each state in both simulations should be given by (14.1), repeated here as

$$P_n = (1 - \frac{\lambda}{\mu})\Big(\frac{\lambda}{\mu}\Big)^n \tag{14.3}$$

Results

The results of the uniformised and non-uniformised $M/M/1$ state time distribution simulations are visualised in Fig 14.6.

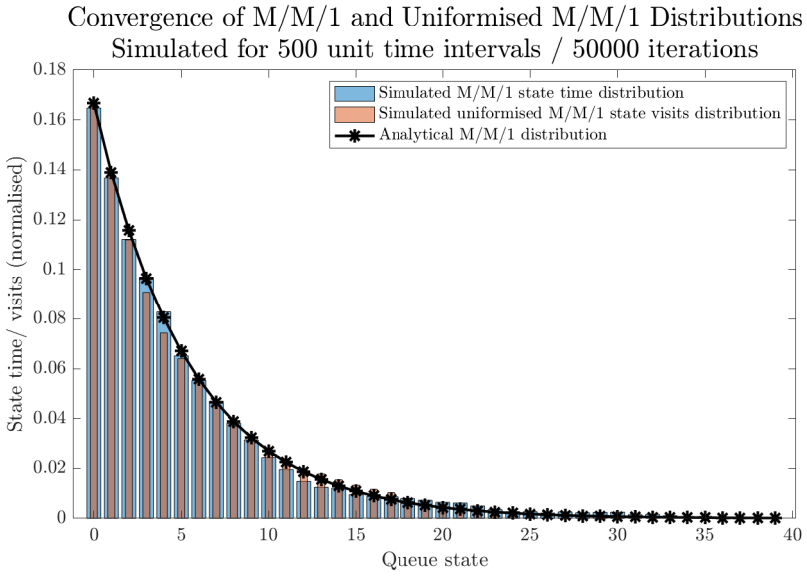


Figure 14.6: Stationary state time distributions for the $M/M/1$ and uniformised $M/M/1/K$ systems.

Analysis

The results of Fig 14.6 show a clear correlation between the simulated distributions and the analytical one described in (14.3). As the simulations are conducted with λ/μ being close to one, relatively long simulation times are necessary. Furthermore, since the simulations start when the queues are empty, both the transient and stationary state phases are included in the statistics. Therefore, unless the simulations are run for a very long time such that the time of the transient phase is completely insignificant in relation to the time in stationary state, there will be some differences between the analytical distribution and the simulated ones.

To conclude, λ/μ being close to one and including the transient phase in the simulated distributions are the most probable reasons for the small differences between the columns describing the uniformised and non-uniformised simulated distributions. However, this is small enough to be disregarded, and the distribution equivalence between $M/M/1$ queues and uniformised $M/M/1$ queues is verified, which implies that the simulations are correctly implemented and reliable.

14.2 M/M/1/K Queues

As is stated in Section 13.2, the optimal threshold of some admission control problems, especially the one considered in this work, can be found by treating the controlled queue as an $M/M/1/K$ process where K is the threshold. Therefore, correct simulations of $M/M/1/K$ queues, both uniformised and continuous, are very necessary in the solution of the admission control problem. A continuous time transition rate diagram of an $M/M/1/K$ queue is shown in Fig 14.7.

The stationary state PMF of the $M/M/1/K$ queue is repeated here as

$$\bar{\pi}_n = \begin{cases} \frac{1-(\lambda/\mu)}{1-(\lambda/\mu)^{K+1}}(\lambda/\mu)^n & \text{if } 0 \leq n \leq K \\ 0 & \text{if } n > K \end{cases} \tag{14.4}$$

The first experiment on $M/M/1/K$ queues regards visualising the process.

14.2.1 Continuous time M/M/1/K queue simulation

As for $M/M/1$ queue simulations, the starting point is to simulate the $M/M/1/K$ queue in the continuous case. However, the implementation of a capacity limited queueing system requires a specific scheduling process compared to the implementation of the unlimited $M/M/1$ queue.

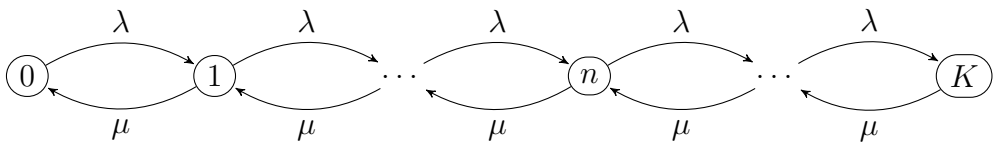


Figure 14.7: State transition rate diagram for the $M/M/1/K$ queueing system.

Setup

As in the implementation of the $M/M/1$ queue, described in Section 14.1.1, interarrival times and service times are generated using the exponential CDF with rates λ and μ, respectively. However, each requested arrival is now either accepted or rejected depending on the current queue state, in contrast to the $M/M/1$ case where all arrivals increased the queue state at all times. The difference between the $M/M/1$ queue and the $M/M/1/K$ queue implementations is thus the admission mechanism, which is outlined here.

At each customer arrival time, a check is made to see if the queue is full. This can be done as the number of elements in the queue is known in real time, and so is the time of the next scheduled departure from the system. If the queue is full at the arrival time, a rejection is logged and the queue is unchanged. If the queue is not full, then the departure time of the newly admitted customer is calculated in the same way as for the $M/M/1$ queue by considering if the customer needs to wait for service or not. Lastly, the arrival and departure are scheduled in the queue trajectory, such that this information can be used for the next arrival and admission request.

The arrival rate is in this experiment $\lambda = 50$ and the service time rate is $\mu = 60$, as in previous experiments. The queue is limited to $K = 3$ customers and the simulation is run for one unit time interval.

Purpose

Now, the purpose is to show what happens when the queue capacity is limited, visually. An intuitive understanding of this is necessary, and from visual inspection, it can be verified that the rejections, admissions and departures in the $M/M/1/K$ behave as expected and in a reasonable way. In later experiments, the stationary state PMF of the simulated $M/M/1/K$ queue is verified through simulations just as the $M/M/1$ queue, but this is not the purpose here.

Hypothesis

The simulation is expected to behave just as an $M/M/1$ queue, with the difference that the queue state never goes over $K = 3$ elements. Arrivals that are rejected are still logged, and if the queue is at full capacity, rejected arrivals and rejections are expected to be logged at the same time. The state distribution is not expected to approach the analytical one given by (14.4) as the simulation is only run for one unit time interval.

Results

The results of the continuous time $M/M/1/K$ simulation experiment are visualised in Fig 14.8.

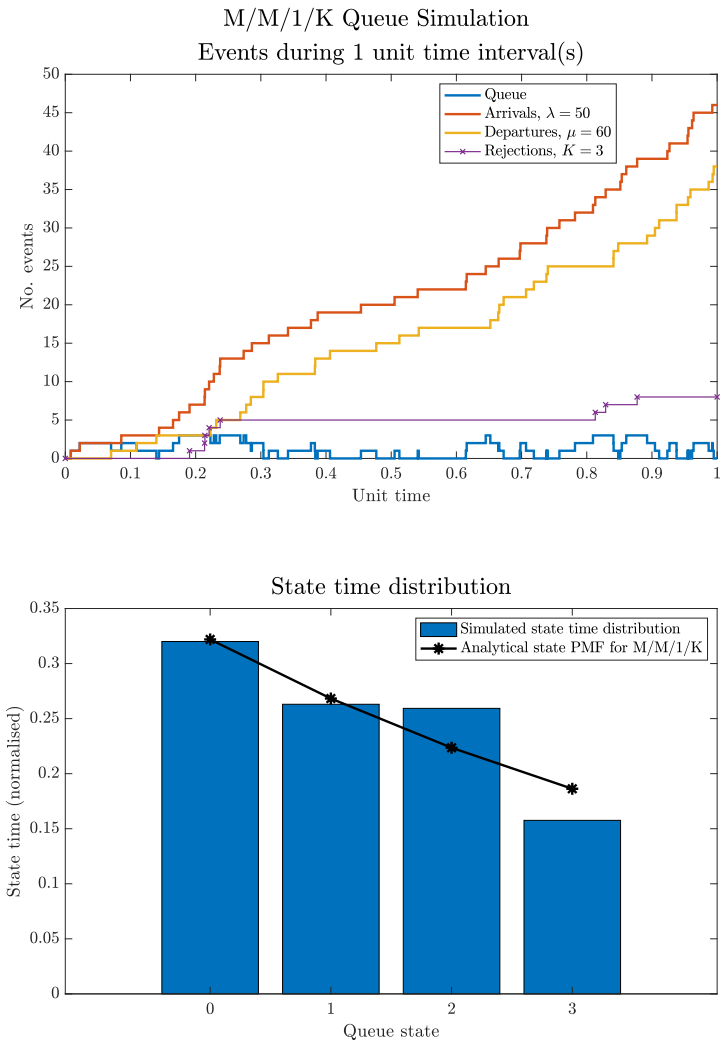


Figure 14.8: Results for the $M/M/1/K$ simulation experiment.

Analysis

In the upper graph of Fig 14.8, the count of arriving customers is visualised. The count is incremented as one unique customer arrives. The departure event of each customer occurs as before at the up-flank at the same height of the departure count curve for that unique customer. However, note now that if the queue is at full capacity during the time of an arrival, the rejection count is incremented at that arrival time. The queue is thus saturated at capacity $K = 3$, as expected.

Furthermore, note that the number of arrivals during this one unit time interval is around $\lambda = 50$. This can be compared to the $M/M/1$ simulation in Fig 14.3, which shows a similar figure. When it comes to the total number of departures, it is around 39 in Fig 14.8 and about 42 in the $M/M/1$ case depicted in Fig 14.3. This is reasonable, since around eight rejections are made in the $M/M/1/K$ queue. In

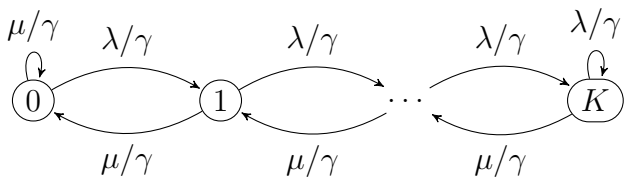


Figure 14.9: State transition diagram for the uniformised $M/M/1/K$ system.

the $M/M/1$ case, these would be admitted and would therefore eventually depart.

The state time distribution again shows a slight trend in the direction of the analytical PMF of (14.4). According to the hypothesis, full convergence is not expected when simulating only one unit time interval. This experiment is concluded to show that the simulated $M/M/1K$ queue dynamics are reasonable from visual inspection.

14.2.2 Uniformised M/M/1/K queue simulation

In the fourth experiment of the second part of this project, the limited $M/M/1/K$ queue is uniformised and simulated.

A uniformised $M/M/1/K$ queue is depicted in Fig 14.9. The rate of uniformisation is again selected as $\gamma = \lambda + \mu$, eliminating the self loops except in the first and last states.

Setup

The procedure to implement a limited queue in discrete time is again much simpler than doing so in continuous time. The difference between the implementations of the uniformised $M/M/1$ and $M/M/1/K$ simulations is just that when the queue state is at the limit K, the event with probability λ/γ now implies a self loop transition, and not a transition to an incremented state. Otherwise, the method is the same as described in Section 14.1.2 under the Setup heading.

The arrival rate is $\lambda = 50$ and the service rate is $\mu = 60$, while the uniformisation rate is as usual the minimally selected $\gamma = \lambda + \mu = 110$. The limit $K = 3$ and the simulation is run for 100 discrete time steps.

Purpose

The point of this experiment is to verify the uniformised $M/M/1/K$ queue visually in discrete time, and how the uniformised queue behaves in comparison to the continuous time representation. It is also interesting to find the number of simulation steps that are needed to simulate for example λ arrivals in the discrete time case. Lastly, the purpose of this experiment is as in the uniformised $M/M/1$ case not to verify the distributions, which probably has to be done by running the simulation for much longer than 100 steps.

Hypothesis

For this simulation, the queue is expected to realise the simple capacity limitation that implies rejecting arrivals at $K = 3$. It is moreover expected that the queue trajectory is saturated at this level, and that it is clearly visible that more admissions would be made if it were not for the limit. As in the continuous case, there should be at least some rejections due to the relatively high probability of arrivals. It is reasonable to assume that the number of arrivals, departures and rejections are similar to the continuous time case, as around 100 simulation steps are sufficient to ensure this in the uniformised $M/M/1$ simulation.

The state visits distribution is however not believed to be close to the $M/M/1/K$ PMF given in 14.4. However, as the simulated distributions have been showing a trend towards the analytical distributions in previous experiments, it is not unlikely that this is seen here as well.

Results

The results of the uniformised $M/M/1/K$ queue simulation are shown in Figure 14.10.

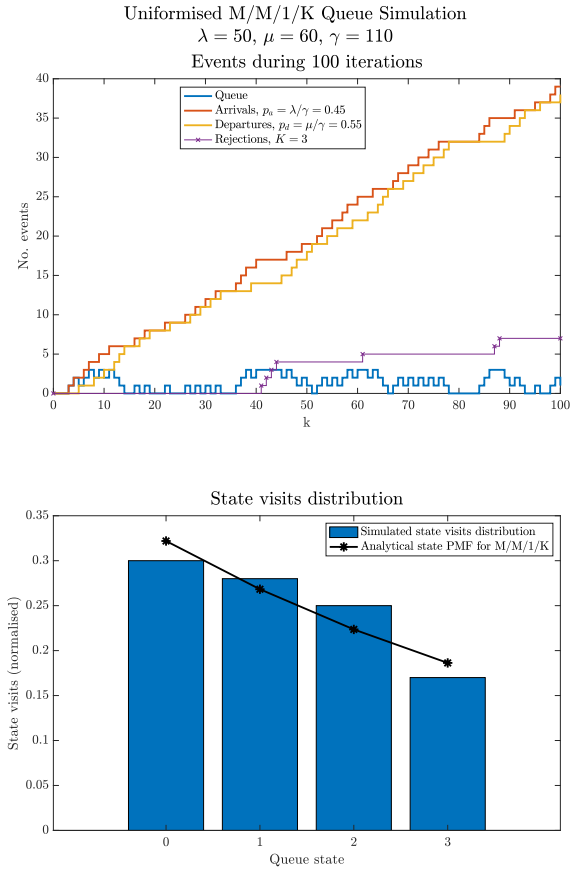


Figure 14.10: Results for the uniformised $M/M/1/K$ simulation.

Analysis

As expected, a similar number of arrivals and departures as in the previous uniformised queue simulation take place here. Five rejections are made in the uniformised case, which is a bit lower than in the continuous time case, but it is proportional to the number of arrivals.

The queue successfully rejects arrivals when it is in state $K = 3$, and while the simulated state visits distribution is not equal to the $M/M/1/K$ stationary state PMF of (14.4), the hypothesised trend is visible as less and less time is spent in states of increasing index.

This experiment verifies visually that the uniformised $M/M/1/K$ queue simulation behaves as expected.

14.2.3 Simulated M/M/1/K queue state probabilities

In this experiment, it is of interest to verify the property that states that the analytical stationary state probability mass function of both the continuous time $M/M/1/K$ and the version uniformised with rate $\gamma = \lambda + \mu$ is given by (14.4), through simulation.

Setup

Similarly to the experiment of Section 14.1.3, this is done by simulating the continuous time and the uniformised $M/M/1/K$ systems until the majority of the simulation is expected to be done in stationary state. The state times are recorded in both cases, and two distributions are made. These are then compared to the analytical expression of the $M/M/1/K$ queue. The simulations are just as in the experiment of Section 14.1.3 run for 500 unit time intervals and 50000 iterations, respectively.

Purpose

It is proven in Section 12.2.2 that if the minimal uniformisation rate is selected, the stationary state probability mass functions of the uniformised and continuous time $M/M/1/K$ queues are the same. If this is verified in the simulations, it can be assured that the implemented $M/M/1/K$ models used in the simulations are accurate, and that they can be trusted. Both the analytical PMF and the queue simulations can then be used when calculating the expected cost of $M/M/1/K$ queues, which is very central in finding the optimal admission control threshold queue length.

Hypothesis

Given that the visual inspection of the previous $M/M/1/K$ simulations found no issues, the simulated distributions are expected to converge to the analytical PMF, given by

$$\bar{\pi}_n = \begin{cases} \frac{1-(\lambda/\mu)}{1-(\lambda/\mu)^{K+1}}(\lambda/\mu)^n & \text{if } 0 \leq n \leq K \\ 0 & \text{if } n > K \end{cases} \tag{14.5}$$

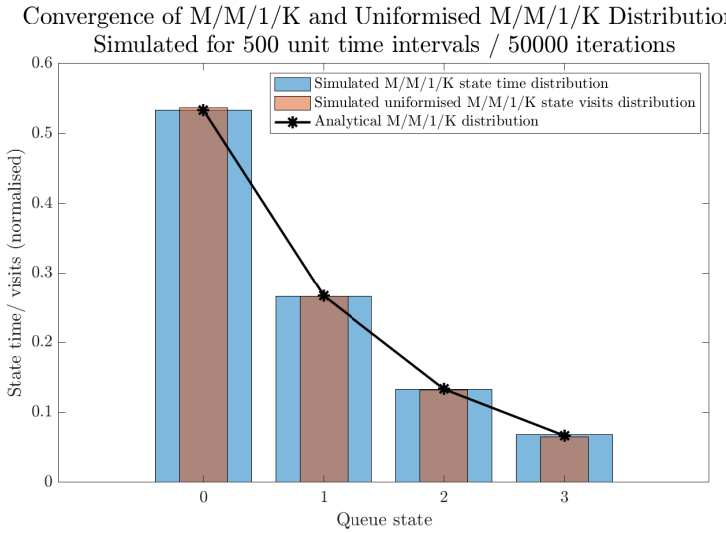


Figure 14.11: State time distributions for the $M/M/1/K$ and the uniformised $M/M/1/K$ systems.

The only difference between this experiment and the successful convergence simulation that is conducted for $M/M/1$ queues is the admission limit, which is shown to work as expected in previous sections.

Results

The results of the $M/M/1/K$ state time distribution convergence experiment are visualised in Fig 14.11.

Analysis

The results in Fig 14.11 show that the simulated state time distributions of the $M/M/1/K$ queue and the uniformised version are very close to the analytical PMF of the continuous $M/M/1/K$ queue given in the hypothesis section. The distributions in this experiment are actually closer to the analytical PMF than in the corresponding experiment regarding unlimited $M/M/1$ queues. The reason for this is believed to be that there are fewer alternatives in the limited queue, as the process can only be in four different states, and therefore more samples per states are generated which yields a higher accuracy.

The result of this experiment shows that the implemented queueing models for both continuous and uniformised discrete time $M/M/1/K$ queues are correctly implemented, and the simulation can be used as an accurate tool to produce other reliable results if necessary.

14.3 Optimal Admission Control Threshold

This is a core problem of the second part of this work, and it is divided into two parts that focus on one method each to find the optimal threshold i^*. The first

way of obtaining the threshold is the analytical method described in Section 13.2.1. It utilises the $M/M/1/K$ distribution, which is efficient, but only works for undiscounted cost functions. The second method is the simulation procedure described in Section 13.2.2, and in this way of obtaining the threshold the discounted cost is calculated for simulated queue trajectories, after which an average cost is derived for increasing threshold values.

14.3.1 Analytical derivation of i^*

This section covers the experiment in which an optimal admission control threshold is obtained based on an undiscounted cost queue function.

Setup

The final expression that is derived in Section 13.2.1 is

$$V_\pi(K) = N\Big(B \cdot \sum_{n=1}^{K} P_n \cdot n + R \cdot P_K \cdot \lambda\Big) \tag{14.6}$$

where

$$P_n = \frac{1-(\lambda/\mu)}{1-(\lambda/\mu)^{K+1}}(\lambda/\mu)^n \quad \text{if } 0 \leq n \leq K \tag{14.7}$$

In (14.6), N denotes the number of unit time intervals that make up the finite cost function horison. For example, if the horizon is evaluated for six unit time intervals after t_0, when x_0 occurs, then $N = 6$. B is the cost per customer per time, so that for example keeping three customers in the queue for 0.2 unit time intervals costs $B \cdot 3 \cdot 0.2$. The cost R denotes the cost per rejected value in the $M/M/1/K$ queue, and K is the limit of that queue.

In this experiment, the cost function (14.6) is evaluated for $1 \leq K \leq 30$. The rates in the underlying $M/M/1/K$ queue are set to $\lambda = 50$ and $\mu = 60$ as in all other experiments. The horison time is set to $T = 5$, so that the $N = 5$ unit intervals are evaluated in the cost function. Lastly, the cost constants B and R are set to 10.0 and 4.5 respectively.

Purpose

This is one of the most important experiments of the second part of this work. Finding a K for which a local minimum can be observed in the cost of the admission control implies finding the optimal threshold queue length i^*. If this value is found, it is possible to reduce the infinite queue dimension in the joint path planning and queue process, such that the rest of the problem can be solved with temporal logic constrained reinforcement learning.

Hypothesis

The cost function derived in Section 13.2.1 is based on the continuous time cost function described for admission control in [12]. In [12], this cost function is adapted for the uniformised problem, for which the admission control strategy is motivated,

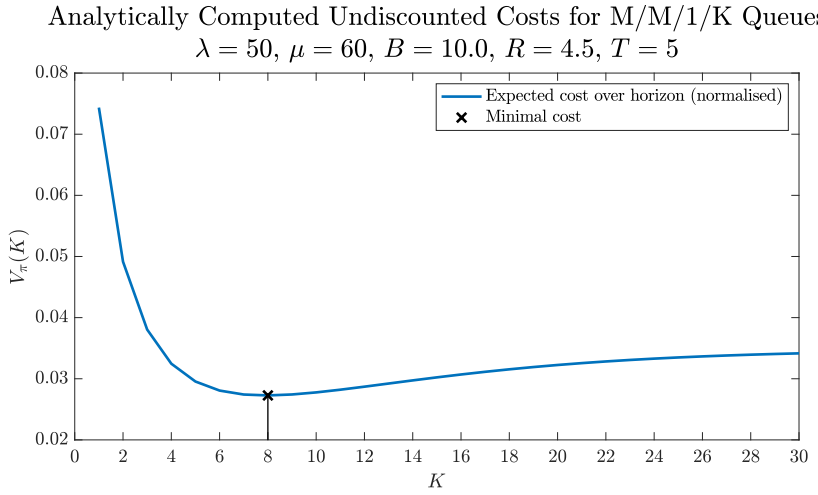


Figure 14.12: Expected total undiscounted cost of admission controlled queues with different thresholds.

but it is also said in [12] that the continuous time cost function is equivalent to the uniformised version. This is described further in Section 12.3.2 of this work, and this fact makes it possible to find an optimal threshold using the continuous time cost function and then apply it on the admission control strategy which is motivated using the discrete verison of the cost function.

Since $R < B$, it is expected that if the queue is allowed to be very large, i.e for high K values, the total cost will increase. On the other hand, if K is low, many rejections will be made while the queue state cost remains low. An equilibrium point in the total cost function is expected somewhere between these points for some K, at which the optimal threshold lies such that $K = i^*$.

Results

The result of the analytically computed undiscounted expected cost is presented in Fig 14.12.

Analysis

In Fig 14.12, a clear local minimum is present in $V(K)$ when $K = 8$. Therefore, with the selected cost parameters, $i^* = 8$. If the queue is allowed to be longer or shorter than this, the cost will only increase. Naturally, if any of the parameters were to change, such as arrival rate λ or service time rate μ, this numerical result would likely change.

A final note is an important one, and it regards the derivation of the threshold type solution in relation to the objective function V_π. In Section 13.1.2, the threshold type solution is based on the monotonic increase of $\Delta V(i) = V(i) - V(i-1)$. It is therefore important to highlight that it is not $V(i)$ that is shown in Fig 14.12, as $V(i)$ is the expression for queue- and rejection costs without having applied a control policy, evaluated from any initial state i. The function shown in Fig 14.12 is the expected total cost over a finite horizon, where the threshold type control policy

is applied for increasing thresholds K. Furthermore, this graph always describes the cost from the same $s_0 = 0$ to the finite horizon for each K. Thus, it is not at all a requirement for difference between two consecutive values the graph in Fig 14.12 to be monotonically increasing, but there is a need for a local minimum to establish the optimal threshold i^*.

14.3.2 Derivation of i^* through simulation

As is described in Section 13.2.2, there are times when a discounted cost is necessary. The discount is an exponential decay function dependent on time that states that events closer to the end of the horizon are not valued as high as events closer to the initial state. As described earlier, this makes it impossible to use the analytical solution from Section 13.2.1, since no information of absolute time required for the decay is available in the analytical stationary state distribution. In this case, simulation can be used, since absolute time measurements are available for each simulated value.

Setup

The discounted cost function for one specific simulated trajectory is derived as

$$V^S(K) = \frac{B}{\beta} \sum_{i=0}^{N-1} s(t_i)\left[e^{-\beta t_{i+1}} - e^{-\beta t_i}\right] + R\sum_{j=1}^{N_r} e^{-\beta r_j} \tag{14.8}$$

in Section 13.2.2. As in the undiscounted case, N denotes the number of unit time intervals that make up the horizon length, while B and R are cost constants for keeping the state and executing rejections, described in more detail under the Setup heading of Section 14.3.1.

In (14.8), N_r denotes the number of rejections and β is the exponential decay parameter for the discount. Furthermore, since S denotes a specific simulation run, $V^S(K)$ must be simulated up to N_S times for the estimated cost to be reliable, so

$$\hat{V}(K) = \frac{1}{N_S} \sum_{S=1}^{N_S} V^S(K) \tag{14.9}$$

This is the final estimated cost for a specific queue capacity limit K.

For this experiment, the exact same parameters are used as in the analytical cost experiment of Section 14.3.1. This means that $\lambda = 50$, $\mu = 60$, $B = 10.0$, $R = 4.5$ and the total simulation is $T = 5$ unit time intervals such that $N = T = 5$ in the cost function. Furthermore, the discount parameter is 0.2 and $N_s = 1000$ simulations are evaluated.

Purpose

The reason for conducting this experiment is to investigate if simulation returns a reasonable suggestion for an optimal queue length i^*. While there are many downsides with simulation methods, such that it requires significant time and energy, it

is quite more versatile than analytical methods which might not always be available. Furthermore, having a decay parameter specifically requires a method where absolute time is available.

Hypothesis

Due to the discount factor, it is expected that events that occur close to the horizon will not be valued as much as those events that are closer to t_0. For high K values, it will be more common that state values far from t_0 are high since the queue is allowed to grow. This might cause the expected cost to decrease for high K values.

For low K values, it is in previous $M/M/1/K$ simulations shown that the probability of being in the final queue state is similar to that of being in any of the other states. Therefore, rejection costs are expected to be discounted to the same extent as queue costs.

In total, given that enough simulations are made, it is expected that the cost function will be flatter towards higher K values. The impact of this might be that the optimal queue length i^* is found to be higher than the limit found with the undiscounted method.

Results

The results of the experiment in which the simulation method to find i^* is tested are shown in Fig 14.13.

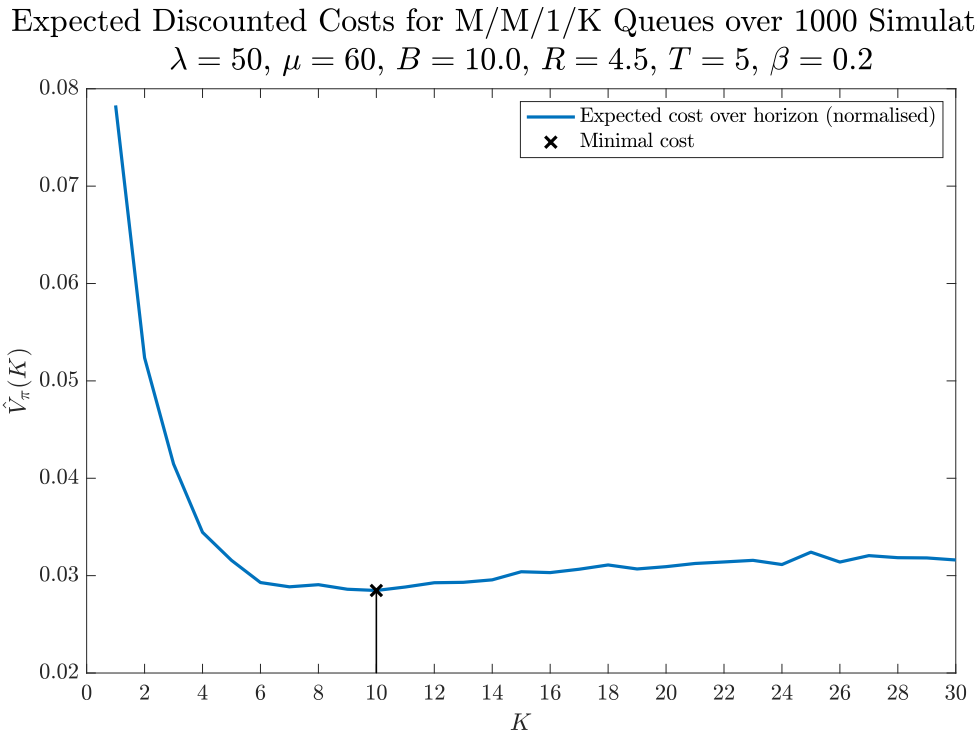


Figure 14.13: Simulated discounted cost of admission controlled queues for different limits K.

Analysis

Fig 14.13 shows that the simulated discounted cost to have a local minimum at $i^* = 10$, which is comparable to the undiscounted analytical cost method that gave the result $i^* = 8$. Although the simulation based cost trajectory shows visible fluctuations, specifically towards higher K values, there is still a clear local minimum.

The hypothesis of this experiment states that the optimal queue length i^* could be higher than the i^* derived by the use of the analytical method. This is the case here, but it is not certain that this is due to the discount factor, as fluctuations can occur between simulations. To ensure if this is the case, N_S could be increased, which should give a more steady cost trajectory.

Given that the same basic costs were used for two very different methods, the fact that the results are so similar is seen as positive, and both methods are deemed reliable.

14.4 Temporal Logic Constrained Q-learning of Joint Path Planning and Admission Control Process

In this section, the final experiment in Part II of this thesis is conducted. Here, the main problem formulated in detail in Section 13.3 is solved. The quite extensive problem is described from a practical standpoint, with the ultimate goal of solving it using temporal logic constrained reinforcement learning. Due to the extent and complexity of the problem, this experiment is presented in several stages.

The first stage concerns modular analysis. This means that a sub-process in the joint Markov process model is first identified and isolated, and a cost function is defined for it. An optimal control policy is then found for the sub-process using an offline method. The results of this are then used to reduce the joint Markov process, so that temporal logic constrained reinforcement learning can be used effectively to find a policy to select those remaining actions of the joint process that are not obtained through the offline solution.

As a reminder of the specific problem investigated in this work, the previously used robot analogy is recalled as a motivation.

14.4.1 Motivation

Consider a robot that performs different tasks on a slippery factory floor. One arbitrary task is performed at a specific corner location in the factory, which means that the robot has to travel as often as possible to this corner over the slippery floor. Furthermore, there are dangerous obstacles on the floor, and the robot must avoid these. Thus, this part of the process is a path planning problem.

In the factory, there is also a machine located in another corner of the factory. As parts arrive automatically to the machine, they can be admitted and processed, after which they leave the system. The robot must remotely control the admissions of parts to the machine, while at the same time performing its other task. Thus,

the robot can at any time choose to either reject or admit an incoming part in the machine. The machine can be modelled as an $M/M/1$ queue, for which an admission control policy can be formulated.

Moreover, to avoid issues with the machine, the robot must physically travel to the machine location and inspect it when it is about to perform a rejection.

The robot, or agent, can thus select path planning actions and admission control actions as it explores the environment. While navigating, slip transitions can be made to neighboring locations, and the arrival and departure of elements to the queue can only be regulated through admission control. Note here that the changes in the queue do not affect the geographical location of the robot, and vice versa; the processes are considered independent from each other.

14.4.2 Joint process model

As is described in Section 13.3, this problem consists of a joint continuous Markov process that can also be modeled as two independent sub-processes; a path planning process and an $M/M/1$ queueing process. Recall that these are considered independent, since the event set of the joint process can be divided into two subsets. In each of the event sets, the events cause transitions between unique states in the joint state space, and a few reflections can be made here.

Firstly, since both event sets and joint states are unique, each joint state can be described as having one part that changes due to the transitions caused by the first event set, and another part caused by the second. This is the tuple notation used in the description of Section 13.3.

Secondly, as the joint process is a continuous Markov process, transitions caused by each event in both sub-processes are Poisson processes. Specifically, this joint process is such that one sub-process can be modelled as a continuous time path planning grid world, and the other one can be modelled as an $M/M/1$ queue. The structure of this problem is best understood visually, as in Fig 14.14.

In Fig 14.14, the joint state space is infinitely large, since the queue sub-process is infinitely long. However, the state space of the path planning sub-process is finite.

When path planning control actions are applied in a joint state where the path planning process state has four unique neighbouring adjacent states and the queue is neither empty nor full, there are six possible transitions that can occur: queue arrivals, queue departures, navigation in the selected direction or slipping in any of the three remaining directions. As this is the maximum rate flow possible in a controlled joint state, joint uniformisation is performed using the rate parameter $\gamma = \lambda + \mu + \lambda_D + 3\omega_D$. The total and lengthy uniformisation process and the complete set of expressions for resulting state probabilities in all possible joint state types are found in Section 13.3.

The arrival and service rates of the queue sub-process are selected in the same way as in all previous queue experiments such that $\lambda < \mu$, which is necessary for stationary state to occur. The queue operates at a lower general rate than the events causing the main navigation transitions, while each navigational slip transition occurs at a rate much lower than that of the main directional transitions.

Through the uniformisation process, a discrete time representation of the con-

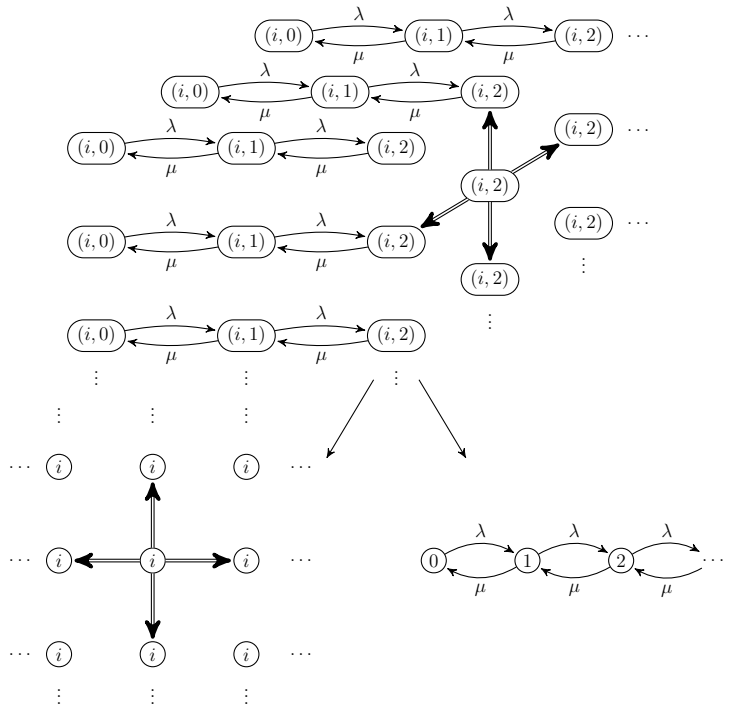


Figure 14.14: Principle sketch of the two ways of viewing a process.

trolled process is obtained, and one step ahead transition probabilities become available. In this way, the discretised problem can now be solved by temporal logic constrained reinforcement learning, while valid analysis can still be done on the continuous process. The numerical values of the rates in this experiment are shown in Table 14.1.

14.4.3 Joint process state labels

In this experiment, the path planning part of the state space is selected to be a square grid world with the dimensions 5×5. Therefore, there are 25 path planning states for every queue state. The queue is initially zero and goes to infinity. The joint initial state is when the path planning coordinates are $x = 0$, $y = 0$ (north western corner), and the queue is empty, $i_Q = 0$.

In the joint state space, there are three types of states; dangerous, goal and

Table 14.1: Rate parameters for the joint path planning and admission control experiment.

Name	Value	Explanation
λ	50	Queue arrival rate.
μ	60	Queue service rate.
λ_D	100	Main direction rate.
ω_D	10	Slip direction rate.
$\gamma = \lambda + \mu + \lambda_D + 3\omega_D$	240	Joint uniformisation rate.

regulator states. Dangerous joint states are labeled q, while goal states have the label p. The regulator states, where the robot needs to inspect the machine in the practical analogy, have the label r. Specifically, joint states where the path planning state has the coordinates $(x, y) = (3, 2)$ or $(x, y) = (4, 2)$ are always dangerous. A goal state is a joint state where the path planning coordinates are $(x, y) = (5, 5)$ (the south east corner), for all joint states except when the queue is such that it needs inspection. In that case, the joint state with coordinates $(x, y) = (5, 5)$ is now forbidden, while a regulator state is found at coordinates $(x, y) = (5, 0)$ (the north east corner). Note that this last part of the specification can be quite hard to meet unless the admission control policy is found to have a simple enough structure. In the experiment conducted here, the admission control policy is of threshold type, meaning that admission is selected until the queue state is i^*. Thus, in this specific case, the joint state $(5, 5, i^*)$ is marked forbidden and state $(5, 0, i^*)$ is marked as the regulator state.

With the joint process specified, the next step is to formulate the specification in terms of an LTL formula realised by a Büchi automaton.

14.4.4 LTL specification

Consider the LTL formula

$$\varphi = \square\lozenge p \wedge \square\lozenge r \wedge \square\neg q \tag{14.10}$$

This specification says that eventually, either a goal state label p or regulator state label r will be observed, while a dangerous state label q must never be seen. Observing r is of equal importance as observing p, and therefore the Büchi automaton representation must be of the general type, described in Section 3.6.4 of Part I of this work. This allows for the use of two marked states, such that the acceptance condition of the Büchi automaton is, with a level of abstraction, to observe both labels r and p an infinite number of times. The Büchi automaton that realises the formula in (14.10) is visualised in Fig 14.15.

In Fig 14.15, the acceptance condition is to visit states q_1 and q_2 an infinite number of times, while never visiting the forbidden state q_3. The specified behavior

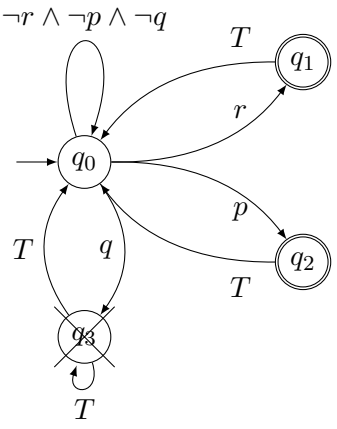


Figure 14.15: Generalised Büchi automaton for the final experiment.

is thus that the agent should go to the goal states as often as possible, but also to the regulator state when it is possible. Moreover, it is again assumed that at most one label can be true at the same time.

At this point, enough information is given to start the first part of the solution, which is the offline modular analysis step. Here, an optimal admission control policy is found for the $M/M/1$ sub-process.

14.4.5 Admission control policy through modular analysis

Part of the specification states that the queue shall be regulated by admission control, based on a separate cost function implying that the solution is of threshold type, defined as in [12]. As has been shown in Sections 13.2.1 and 13.2.2, the discounted cost function used in [12] can be solved analytically if the discount parameter is selected to be zero. Although it is also possible to find a solution using the discounted cost function through simulation, this experiment focuses on the analytical solution using the undiscounted cost function.

This infinite horizon cost function can be approximated by the finite horizon undiscounted cost function defined in Section 13.2.1, which reads as

$$V_\pi = \mathbb{E}_\pi\Big[\int_0^N e^{-\beta t} \cdot B \cdot s(t)dt + \int_0^N e^{-\beta t} R \cdot A(t) \cdot a(s(t^-))dt\Big] \tag{14.11}$$

As is also shown in Section 13.2.1, this reduces to

$$V_\pi(K) = N\Big(B \cdot \sum_{n=1}^{K} P_n \cdot n + R \cdot P_K \cdot \lambda\Big) \tag{14.12}$$

The threshold type policy that the cost minimising K produces is well known by now, and implies that there is a cost optimal queue length at which arrivals are rejected. Since the cost function is undiscounted, the procedure used to obtain the optimal threshold is chosen as the analytical method using the distribution for $M/M/1/K$ queues until $K = i^*$.

The selected cost parameters needed to solve this problem are presented in Table 14.2. These parameters are identical to those used in the analytical i^* derivation experiment of Section 14.3.1. In addition to this, the same arrival and service rates are used. Because of this, the solution to the admission control problem is already identified in Section 14.3.1 as $i^* = 8$. The queue cost development can thus be observed in Fig 14.12 of that same section.

Using this information, the exploration of the infinite joint state space is hereby limited to all joint states where the queue is between 0 and $i^* = 8$. This finite structure contains 225 states, and the queue process is operated by admitting all arrivals for $i_Q < i^*$.

Table 14.2: Cost parameters for admission control.

Name	Value	Explanation
B	10.0	Queue cost per state per time.
R	4.5	Cost per rejected arrival.
T	5	Horizon length.

14.4.6 Joint process path planning through temporal logic constrained reinforcement learning

Navigation-wise, the specified behavior is to go to a path planning goal state as often as possible, while avoiding dangerous states. Regulating the queue can be done remotely, but if a rejection might be made, the agent must move towards another geographical location which is the regulator state.

In its original form, this problem can in theory be solved by temporal logic constrained reinforcement learning directly, but the exploration is done on an infinite state space and runs the risk of going on forever. However, by limiting the joint state space at $i^* = 8$, the problem can be solved on the resulting finite state space.

Using the temporal logic constrained reinforcement learning algorithm on the joint process, where the queue actions are already selected, is thus the last step in solving the jointly controlled queue and path planning problem. This procedure is the main part of this experiment, considering that the admission control part is practically solved in the experiment conducted in Section 14.3.1.

With the specification formulated in (14.10) and shown in Fig 14.15, some final parameters regarding the temporal logic constrained reinforcement learning algorithm remain to be presented. These are shown in Table 14.3.

As can be seen in Table 14.3, the learning rate α is selected as 0.9 while the learning rate γ_l is set to 0.2. The ε-decay that decreases the exploration probability ε as a function of the episode number is initially maximally set to 1. Although the exponential decay factor d_ε seems very low at 0.0001, this causes the exploration probability ε to approach the order of 10^{-3} after around two thirds of the total number of episodes. The total number of episodes is set to 40000 while the number of transitions per episode is set to 100. However, if a dangerous state is reached, the episode is terminated. This stands in contrast to the procedure in Part I, where the decision was made to continue learning and not transitioning to the dangerous state. In this case, this precaution seems unnecessary, since the queue transitions can remove the agent from the dangerous state if it ends up there.

The reward for reaching the first marked state in the specification automaton is $\rho_1 = 10$ while the reward for reaching the second marked state is $\rho_2 = 1$. This is to enforce that the single regulator state has a large effect on the final policy. There

Table 14.3: Temporal logic constrained RL parameters.

Name	Value	Explanation
α	0.9	Q-learning learning rate.
γ_l	0.2	Learning rate.
ε_{max}	1	Initial maximum exploration.
d_ε	0.0001	Parameter for exponential decay of exploration ε.
No. episodes	40000	Number of RL episodes.
Max step	100	Maximum number of transitions in one episode.
ρ_1	10	Reward, Büchi state q_1.
ρ_2	1	Reward, Büchi state q_2.
ρ_3	-10	Punishment, Büchi state q_3.

is only one joint process state that has the necessary state label r to obtain this reward, while there are 8 joint state in which the goal state label p can be observed in order to collect the smaller reward. In total, the rewards that can be obtained from both states are considered to be balanced between the two.

Negative reward, or punishment, of $\rho_3 = -10$ is assigned to the observation of state label q. Again, this label is found on joint states where the coordinates of the path planning problem mark the two dangerous path planning states, but this label is also used to mark the path planning state that is usually the goal state when the queue process is at i^*. This is a good example of how the specification enforces a behavior based on the state of the complete process. While admission control actions are already selected, navigation can be specified as a function of the queue state, such as in this case where the agent is required to check up on another geographical location if the queue is at the limit.

Final policy hypothesis

The hypothesis in this experiment is formulated for how the reinforcement learning algorithm will recommend path planning actions in the default initial specification state q_0, with respect to the joint process state.

Given that the limited joint state space still has 225 states, a motivated question is if any helper mechanisms such as those presented in Part I of this thesis are needed. On the other hand, automatic exploration is enforced through the partly uncontrollable queue transitions, and in that sense the reinforcement learning algorithm only has to solve a 25 state problem albeit for nine different queue states.

What is interesting is the behavior in the joint states where the queue is at i^*. If the reinforcement learning is successful, the Q-table is expected to have values recommending the agent to approach the south eastern goal state corner of the path planning grid for any queue length, except when the queue is full. In that case, the recommended path should be to move towards the north eastern state. Using the robot analogy, this is where the machine is located on the factory floor, and since rejection might occur the robot needs to be close to the machine to make sure that there are no errors when the machine is at full capacity (a task which thankfully lies beyond the scope of this work).

Results

The results of the final joint path planning and admission control problem are visualised in Fig 14.16, which describes the Q-table development as an average of the values in all elements. An example of how the path planning policy is expressed is shown in Fig 14.17, where the recommended path planning action in each joint state is represented by an arrow pointing in the intended direction of travel.

Analysis

Fig 14.16 shows the average Q-table element value, taken over the MDP state, Büchi state and navigation action dimensions. While there might be a slight negative bump in the development during the very first few episodes, the rest of the table

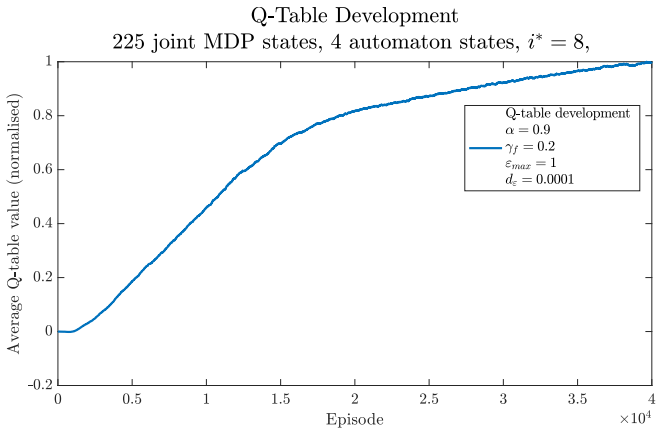


Figure 14.16: Q-table development.

development is positive. After around 20000 episodes, the curve starts to flatten out, but it never quite converges to a constant value. Provided that the agent does not terminate an episode by going to a forbidden state, this trend shows that the total reward per episode can always be increased.

The reason behind this could be that it is possible for the agent to simply stay very close to a goal or regulator state and collect reward after reward by going back and forth to that state. However, this behavior would sooner or later lead to the agent selecting a very specific sequence of actions through the process each time, and it would no longer be possible to achieve a larger total reward in a new episode compared to a previous episode. What could be the difference in this experiment is that there is a strong stochastic element in the joint process, namely the queue model. Since the queue changes state uncontrollably in the eyes of the path planning process, any new immediate reward is mapped to different Q-table elements, and if it is still possible for the Q-table to converge fully under these circumstances, it would probably require many more episodes of learning.

In Fig 14.17, the recommended directional actions that the Q-table gives in each path planning state, for each queue state, is shown for all joint states. For example, in the upper left graph, the arrows indicate which way to go at each coordinate if the queue state is zero. In this graph, the goal state is marked with a red circle at coordinate $(4, 4)$ while dangerous states are marked with black x:es at coordinates $(3, 2)$ and $(4, 2)$. In general, the arrows lead the way from the initial state $(0, 0)$ to the south eastern corner, just as intended. Even if an action is recommended out of a dangerous state, no arrows point towards the dangerous states, which is reassuring. A similar description could be given for the joint states when the queue state is one through seven, too.

However, something changes when the queue reaches the final state $i_Q = i^* = 8$, shown in the lower right graph. Here, the south eastern corner is marked as dangerous, while the previously unmarked north eastern corner state is marked with a green triangle. This is the so-called regulator state, in which the robot inspects the machine since risky rejections can be made here. Upon closer inspection, it is clear that the arrows now lead away from the south eastern corner and towards the north eastern regulator state. Still, the recommended path is to

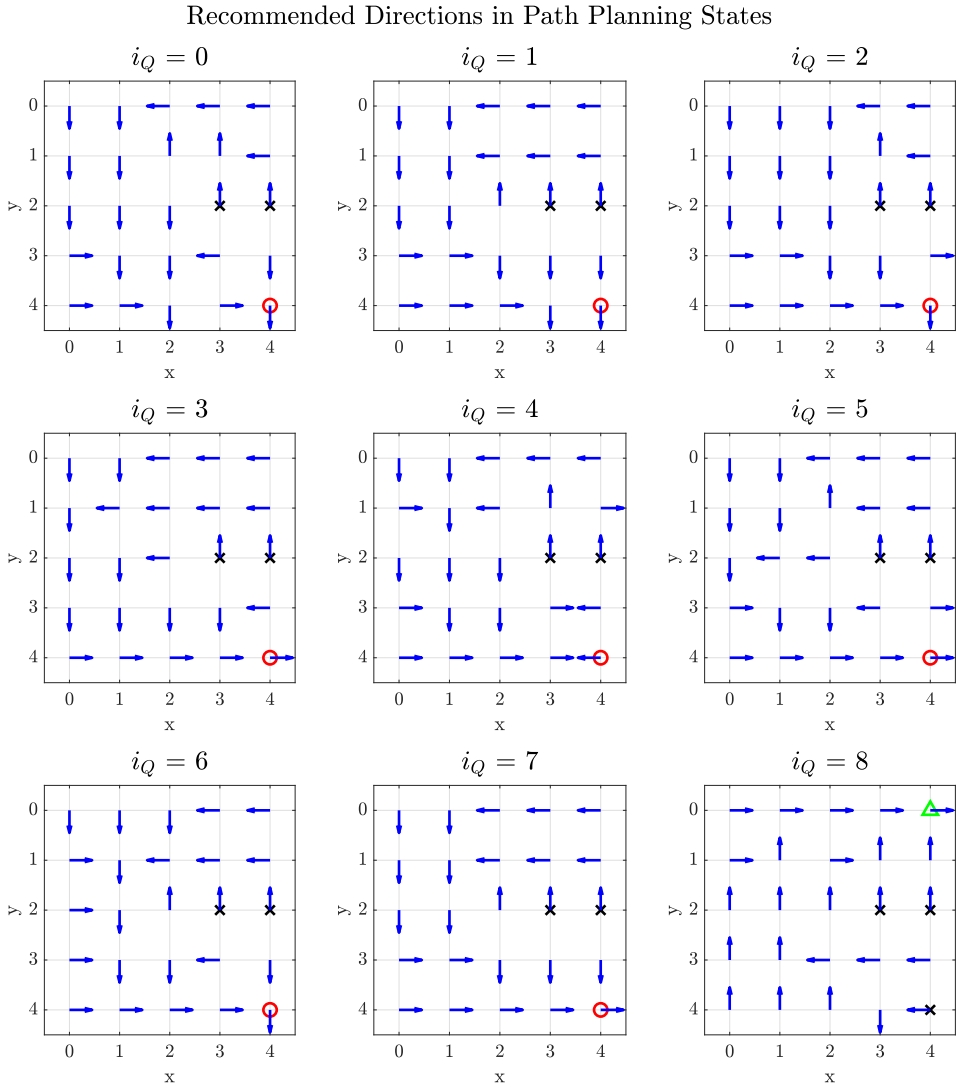


Figure 14.17: Recommended actions, denoted by arrows, for all joint states.

go around the two standard dangerous coordinates. In total, this very much agrees with the specification formulated for the joint problem in the beginning of this experiment.

To summarise the final experiment of this thesis, it can be noted that even though the Q-table does not show a strict convergence to a constant value, the result shows what could be described as asymptotic convergence towards a linear function. Whether or not that is desirable, it seems from the Q-table visualisation in Fig 14.17 that the specified behavior regarding the path planning is enforced. As a solution for the admission control problem is also provided and used, it can be concluded that an example of when control policies for joint Markov processes with infinitely large state spaces is solved by first reducing the problem using modular analysis, and then solving the rest of the problem using temporal logic constrained

reinforcement learning.

14.5 Summary

The experiment chapter of Part II of this thesis starts with experiments concerning $M/M/1$ queue simulations. The somewhat intricate simulation method of these continuous Markov processes is implemented by generating interarrival and service time from the exponential cumulative distribution function using different rate parameters.

The first $M/M/1$ queue experiment is conducted to get a visual understanding of when arrivals take place in relation to departures, and how the queue grows with time. The second experiment aims to do the same with the uniformised and thereby discrete time version of the $M/M/1$ queue, and the visual difference between the two is clear. In the last $M/M/1$ experiment, the simulated and normalised state time distributions for both the continuous time and the uniformised $M/M/1$ queues are shown to converge to the analytical stationary state probability mass function. This confirms that both simulations behave according to a previously formulated analytical proof of that the uniformised and continuous queue version shares the same stationary state probability distribution, and the simulations are verified as accurate.

Next, the same experiments are conducted for $M/M/1/K$ queues. This queue requires the implementation of a more intricate scheduling algorithm since arrivals are rejected based on the queue state, a type of information that is not needed when simulating unlimited queues. After having simulated the uniformised $M/M/1/K$ queue, an experiment is conducted to verify that the stationary state probability distribution of a uniformised $M/M/1/K$ queue approaches the analytically described distribution of the continuous time $M/M/1/K$ queue. The conclusion drawn from the results of these experiments is that the $M/M/1/K$ simulation is accurate, so that it can be used in finding the optimal threshold for the admission control problem.

Once the queueing experiments are done, the two methods for finding the optimal admission control threshold are tested in two separate experiments. The first experiment tests the analytical function, and finds the optimal threshold to be $i^* = 8$, while the second simulation based method finds $i^* = 10$.

In the final section, the extensive procedure of formulating a joint path planning and admission control problem is formulated based on a motivation concerning a robot that is tasked with navigating and regulating a machine at the same time. The complete procedure starts with the formulation of a Markov decision process, after which state labels are placed in the joint process states. Next, an LTL specification is formulated as a Büchi automaton specifying the learning agents desired navigation through the joint process. As the admission control problem is formulated with the same parameters as in a previous experiment, the threshold $i^* = 8$ is used to reduce the joint state space. Finally, the temporal logic constrained reinforcement learning algorithm is used to train the agent to navigate through the grid world conditioned on how the queue changes. The final experiment is considered successful, as the agent performs admission control and path planning as the LTL specification requires.

15
Conclusions of Part II

The conclusion of the second part of this thesis discusses the final project part from four perspectives. Firstly, the method that is used to conduct the project is evaluated in terms of both technique and technical implementation. Secondly, each experiment conducted in the second part is discussed, with the purpose of highlighting the results that it leads to and whether or not it is gainful in completing the bigger picture of the project.

The third section investigates if conducting the project has led to new insights in answering the research questions formulated in the beginning of this thesis. The fourth and last section of this thesis cements the final project conclusions, focusing on the overall project highlights and take-aways from both Part I and II of this project.

15.1 Evaluation of Method and Implementation

The second part of this project focuses on how modular analysis can be used to simplify large scale Markov decision processes by isolating sub-processes in the system and solving some of these analytically. Specifically, a process dubbed the joint path planning and admission control problem is investigated.

15.1.1 Choice of method

The method chosen to solve the problem of joint path planning and admission control is very much a divide and conquer strategy, and its steps are outlined here.

The specific joint process is in short a path planning MDP, well known from the first project path, combined with a continuous time $M/M/1$ queue that is to be controlled with admission control. The control policies for these sub-processes are found using two different methods, and this is the ultimate goal with the form of modular analysis that is described here.

While the idea is that path planning policies can be found using the same methods as in Part I, namely temporal logic constrained reinforcement learning, the initial strategy to find the admission control policy is different, and it requires the use of additional analytical methods.

First of all, modelling a sub-process as a queue implies to introduce a new type of Markov process, as queues are defined in continuous time. Due to this, the decision is taken to also formulate the path planning problem in continuous time, since both models are intended to operate at the same time.

Secondly, with the ability to accurately define and fuse the two models, the next step is to solve the admission control problem for the queue description, with the intent to find a threshold type solution that can effectively limit the joint state space at some queue length. This method not only finds an admission control policy, but also limits the state space such that the path planning actions can be solved with temporal logic constrained reinforcement learning. For this to be possible, a specification needs to be formulated for the joint process, which is the last theoretical step.

Over all, this method is determined to be a successful one, as the final joint problem is solved. The main strength is the way that the complete process is separated, without which the solution to find both path planning and control actions would not be possible to formulate as two separate stages.

The second strength of the method is the ability of finding a threshold type solution to the isolated admission control problem. This is the gateway to limiting the whole infinitely large joint state space, which makes it possible to use temporal logic constrained reinforcement learning. There are ways of formulating different cost functions that return different thresholds, and two of them are investigated in this method.

The method selected to solve the main problem does, however, have its weaknesses as well. Firstly, the way that a joint Markov process is formulated is a very convenient description of two systems working together. The main problem is that the model lacks the type of cross-correlation that would appear if the events of one sub-process could affect the states of the other. In this method, the models are separate while the control policies for path planning are selected as a function of the states of both processes, formed by the soft constraints of the reinforcement learning procedure. As such, this problem formulation is not as complex as the challenging scenarios that can be found in for example industrial applications.

Secondly, the success of the method is completely dependent on the existence of a threshold type solution to the admission control problem, without which the state space of the joint process could very well be infinitely large. As is shown, the threshold type solution exists for a specific, although quite useable, cost function. Furthermore, the relatively low threshold is defined for specific cost parameter, and if these were to be chosen arbitrarily, a practically applicable solution to the admission control problem might not exist.

However, in this specific setting, the selected method is shown to produce an acceptable solution that solves the initial problem.

15.1.2 Technical implementation

The previously described method requires several ways of simulating queueing systems, evaluating cost functions and running the temporal logic constrained reinforcement learning algorithm. The implimentation used in Part I suffers from being unnecessarily large and complicated. Formulating a new minimal model is thus a suggestion for further improvements in the first part.

In addition to increased efficiency of solving reinforcement learning problems, the implementation requires the possibility to simulate Markovian queues and produce

algorithms that finds optimal admission control thresholds. Lastly, a completely new reinforcement learning problem needs to be solved, which the previous implementation does not allow due to its structure.

Given the new type of highly specific technical demands of Part II, a new implementation is constructed in Matlab. The benefits of Matlab is that specific experiments can be formulated very quickly, and as the second part focuses on the solution of a very specific problem, this is determined to be the best solution.

Next, the main functions of the implementation are evaluated concisely.

Queue simulation

There are two different types of queue simulators, one for continuous queues and one for discrete time queues. In the continuous case, the input parameters are the arrival and service rate parameters λ and μ, along with parameters that decides the queue capacity and the simulation time. Thus, the same simulation can be used to for both $M/M/1$ and $M/M/1/K$ queues, which is highly efficient in experiment settings. For the discrete queues, the input parameters are transition rates and uniformisation rates. This allows the user to experiment with different uniformisation rates, and it is also possible to provide a queue limit and a simulation length parameter to this function as well.

In general, the queue simulation functions are highly useable and efficient. The only change that could be made is to combine them, so that one simulation function could simulate $M/M/1$ and $M/M/1/K$ queues in both discrete and continuous time.

Temporal logic constrained Q-learning on joint process

The new way of formulating the main problem of this thesis is modified to fit a joint state space. In this implementation, focus lies on efficiency, and not on user experience.

The first step to this is to base the implementation on two mappings. Firstly, a transition mapping for the joint Markov process is produced offline to store the connections between each source state, transition and target state. Secondly, as the path planning problem is uniformised, a transition probability mapping is also produced. These mappings allow for quick access to information that is requested in each step of the reinforcement learning loop.

The second step concerns the implementation of the Büchi automaton that is run in parallel to the MDP model. The few transitions that need to be described in this system are also collected in a table, and the transitions that are conditioned on MDP state labels are implemented using minimal boolean operators.

To find less obvious ways of making the implementation run faster, the Matlab tool called the Matlab Profiler is used to describe how efficient all subroutines in a function are. In this way, inefficient datastructures can be found and exchanged. This is considered a very successful method that provides insights into how inefficient some obscure details can be.

Over all, this way of implementing the temporal logic constrained reinforcement learning is much more efficient for very specific problems, but this also suffers from the drawback of not being able to formulate general Markov processes, which would

be a good idea if many different types of processes were to be investigated.

To summarise, this specific Matlab implementation is more efficient than the Python implementation, but the reason for this might be the narrow scope of the demands on the implementation of Part II.

15.2 Conducted Experiments

In this section, four categories of experiments are conducted with the goal of solving one specific problem. Here, each of these category of experiments are evaluated.

15.2.1 M/M/1 queue simulations

The $M/M/1$ queue simulation experiments start with visualising the process. This is not only a good starting point for demonstration purposes, but also a quite efficient way to see if the queue behaves as it should, and it is quite easy to verify that arrivals and departures occur at correct time points in relation to each other. This simulation also serves to show that the analytical distribution of the $M/M/1$ queue is not at all accurate in the transient phase, and gives a measurement on how long the simulation time needs to be, to reach the stationary state.

The second experiment visualises the uniformised queue, which is also necessary to evaluate visually in order to witness the difference to the continuous $M/M/1$ queue, and this experiment also verifies that stationary state does not happen directly in the discrete time case either.

The final experiment is valuable, as it verifies that both the uniformised and continuous queue approach the stationary state probability mass function. This is a technical verification of that the simulations work correctly, and the simulated distributions have also been used in the development of the simulations to test for errors. Furthermore, the $M/M/1$ simulation is the foundation for the $M/M/1/K$ simulation implementation, which is a central part of this project.

15.2.2 M/M/1/K queue simulations

When it comes to $M/M/1/K$ simulations, their success is of crucial importance as they are used to find the optimal threshold for the admission control part of the main problem.

Designed in the same way as the $M/M/1$ queue, the $M/M/1/K$ is a limited version, which is in the first two experiments verified to be working from visual inspection. In these experiments, the importance of reaching steady state is illustrated, and both the continuous and uniformised queue versions are verified to be working correctly. The last experiment that shows that both queues behave according to the stationary state distributions in the long run.

The fact that the $M/M/1$ queues can be correctly simulated is one of the most important results on the way to solving the main problem, as it allows simulations to be used with confidence in finding the optimal threshold of the admission control problem.

15.2.3 Optimal admission control threshold

These experiments concern the two methods of evaluating cost functions for limited queues in order to find the best possible limits. Naturally, the success of these experiments is a milestone for the project, as two thresholds that are very close to each other are found, using two completely different methods.

As the threshold that is found through simulation is one step higher than the one found through the analytical process, this agrees with the hypothesis that a discounted method would allow higher thresholds. This important experiment must however be described along with the successful queue simulation experiments, as they outline the fundamental process that makes the cost function experiment successful.

15.2.4 Joint path planning and admission control

This experiment relies directly upon the threshold found in the previous derivation. Although there are two methods to find the threshold, which comes up with two different thresholds, this does not make much difference from the perspective of how the value is used in the main problem. The analytically found threshold is eight while the simulation based threshold is ten, and choosing the higher threshold would only result in an additional level of joint states. All in all, the most practical solution is to set the threshold to one of the previously derived values, especially since the analytical method is deterministic and would return the same threshold if it the method were run again.

The state space is selected in such a way that it is not too large, and not too small. This decision is made based on the difficulties to solve large scale problems in Part I, as any helper functions would steal attention from the core values of the selected method, which is using modular analysis. Even with this in mind, the joint state space becomes quite large as it holds 225 joint states.

In Part II, the focus of the project is shifted from testing different LTL specifications in a reinforcement learning context, to using modular analysis as a way of integrating system information into the solution method. Because of this, the specification can be selected such that it is simple enough to work directly, while still illustrating the idea of that any specification can be applied. In theory, however, the assumption is that any of the specifications shown in Part I can be used in the final problem of Part II, but this is yet to be tested.

Overall, the final experiment is considered successful, but it highlights some weaknesses in the method, the first of which being the admission control threshold which must be shown to exist before this solution is attempted. This conclusion can be applied to all problems that aim to limit an infinite state space, and touches upon a common optimisation problem; how can it be made sure that a local optimum, if it exists, is also the global optimum? Of course, the answer to this question might be apparent in this specific admission control problem, but it can never be assumed.

The second issue with the final experiment regards the implementation, which now reduces the state space offline, before the reinforcement algorithm is run. A more general method would be to find a way of automatically detecting sub-processes and limiting the state space exploration online, as cost ineffective regions are ex-

plored. The idea of changing the method in this direction touches upon the second more philosophical issue with the solution method. Is it really more practical to isolate sub-process and solve control problems for them analytically if it takes away the automatic structure of pure reinforcement learning? The answer to this is not given here, but it can be related to the classic automation paradox where all automated processes seem to need manual supervision at some level, which raises the question of whether or not they are truly automatic.

15.3 Answers to Research Questions

Part II focuses mostly on two of the research questions formulated in Section 1.4 of Part I, and how the second part of this project helps to answer these is presented here.

15.3.1 Use of additional information

The second research question in Section 1.4 asks how additional information may be used to improve the solution of the temporal logic constrained reinforcement learning, or how to find the solution faster.

The form of modular analysis used in the second part of this project can be summarised as utilising the knowledge that there are specific sub-processes in the joint problem formulation, and that specific control policies can be formulated for them under certain circumstances.

Therefore, the conclusion regarding this research question is that there are indeed situations where reinforcement learning in general can be sped up by simply using a different method to find certain control policies through parts of a process. However, the question is if this can be considered as enhancing the reinforcement learning method, as the solution is to not use it.

In that sense, this part of the project presents situations where temporal logic constrained reinforcement learning is inefficient, and suggests a method to fuse an analytical solution procedure with reinforcement learning. This is not necessarily a negative result, and it shows that using several solutions in one problem is sometimes a possibility.

15.3.2 Performance

The third research question from Section 1.4 considers how temporal logic constrained reinforcement learning compares to other methods in terms of performance, and if there are other methods that are more effective in certain situations.

This question can be answered in the context of using modular analysis, since this is in part another method. In the particular situation where one part of a system is an $M/M/1$ queue that shall be controlled using admission control, the hypothesis is that it is easier to find the threshold type solution analytically, and then restrict the infinite state space, than it is to use reinforcement learning on the complete problem. However, this hypothesis remains untested, as it is not attempted to solve

the joint path planning and admission control problem by the use of temporal logic constrained reinforcement learning only.

Although the answer here is quite speculative, it is not considered completely impossible to solve the problem without modular analysis. This is because for every $M/M/1$ queue, an expected length can be computed by using the arrival and service parameters. The expression for this is found in Section 12.2.1, and could in theory be used as a way of obtaining the probable dimensions of the joint state space. Admission control policies cannot affect the arrival rates or the service rates, so even if all arrivals are admitted, the queue is still expected to be finite. In that sense, a policy obtained directly through pure temporal logic constrained reinforcement learning is not out of the question, and it would be interesting to see how this compares to the strategy used in this project.

In short, the performance question is hard to answer given that no alternative solutions are tested. However, it is probable that the modular analysis method is still quicker, since it is expected that more exploration is needed on an infinite state space, even if there exists a local optimum.

15.4 Final Project Conclusions and Suggestions for Future Work

This section draws some conclusions regarding both Parts I and II of this project.

The total project concerns the enforcement of temporal logic constraints in a reinforcement learning context, specifically for so-called path planning Markov decision processes. In the first part of the project, the focus firstly lies on evaluating different existing methods for formulating linear temporal logic specifications, and then to propose hybrid methods and test them on different learning scenarios.

In the first part, the most important conclusions regard the different additional methods that are needed to learn policies efficiently. In some scenarios, such as when dealing with large state spaces, certain specifications are practically impossible to follow as they require the agent to find a correct policy more or less by chance in the first iterations, which can be hard even in finite state spaces.

The first part is valuable as it highlights some quite limiting situations that concern specific experiments, but the results are still quite general. The most usable result is that it is entirely possible to formulate a specification that not only requires the learning agent to explore the whole state set, but also multiple states in sequence. The more intricate these requested sequences become, the longer it takes for the agent to find the specified policy. In many cases, there is a need to find a way of finding solutions that make the learning more efficient. With this in mind, the second part of the project addresses another way to reduce the burden for the reinforcement learning algorithm.

In the second part of the project, the method of modular analysis is formulated for the specific problem of a joint path planning and admission control problem. The strategy is to divide the problem into sub-problems that can be solved by other methods than reinforcement learning, and the lessons learned from this procedure are many.

Firstly, modular analysis as done in this work requires a lot from the problem formulation. The main difficulty is not that the problem is difficult to solve, as it simply relies on showing that certain parts of the state space can be disregarded due to the existence of a local optimum in parts of the joint problem. The difficulty lies in obtaining a motivation for why a model can be separated into two parts in the first place, as this is a somewhat convenient initial assumption. Secondly, even if the separation is motivated, the existence of a local optimum in a sub-process can never be assumed, and must be shown to exist analytically.

However, with these assumptions, this project still shows that if the situation presents itself, it is possible to reduce the exploration necessary for a learning agent by using modular analysis.

With this basic goal completed, some suggestions for future research are firstly to implement an online way of determining the limit at which exploration in an infinite state space should be stopped. This method could use for instance gradient descent strategies.

Secondly, an interesting research direction is to find a more realistic scenario, such as an industrial application, on which the algorithms tested in Part I and the modular analysis of Part II can be tested. This could shed light on unexpected difficulties and perhaps lead to finding situations where these highly theoretical methods can be used to solve practical problems.

Bibliography

[1] Bengt Lennartson and Qing-Shan Jia. Reinforcement learning with temporal logic constraints. Submitted for WODES2020.

[2] Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*. The MIT Press, 2008.

[3] Mohammed Alshiekh, Roderick Bloem, Rüdiger Ehlers, Bettina Könighofer, Scott Niekum, and Ufuk Topcu. Safe reinforcement learning via shielding. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[4] Mohammadhosein Hasanbeig, Yiannis Kantaros, Alessandro Abate, Daniel Kroening, George J Pappas, and Insup Lee. Reinforcement learning for temporal logic control synthesis with probabilistic satisfaction guarantees. *arXiv preprint arXiv:1909.05304*, 2019.

[5] The Institute for Ethical AI and Machine Learning. The responsible machine learning principles. ethical.institute/principles.html#commitment-8, 2020. Accessed: 2020-01-24.

[6] Swedish Energy Agency. Sveriges energi- och klimatmål. energimyndigheten.se/klimat–miljo/sveriges-energi–och-klimatmal, 2020. Accessed: 2020-01-24.

[7] Swedish Energy Agency. Yearly energy balance. energimyndigheten.se/statistik/den-officiella-statistiken/statistikprodukter/arlig-energibalans, 2018. Accessed: 2020-01-24.

[8] Technical University of Munich, OWL online demo. owl.model.in.tum.de/try. Accessed: 2020-05-11.

[9] Mohammadhosein Hasanbeig, Alessandro Abate, and Daniel Kroening. Logically-constrained reinforcement learning. *arXiv preprint arXiv:1801.08099*, 2018.

[10] Rodrigo Toro Icarte, Toryn Q Klassen, Richard Anthony Valenzano, and Sheila A McIlraith. Advice-based exploration in model-based reinforcement learning. In *Canadian Conference on Artificial Intelligence*, pages 72–83. Springer, 2018.

[11] Andrew Y Ng, Daishi Harada, and Stuart Russell. Policy invariance under reward transformations: theory and application to reward shaping. In *ICML*, volume 99, pages 278–287, 1999.

[12] Christos G Cassandras and Stephane Lafortune. *Introduction to Discrete Event Systems*. Springer Science & Business Media, 2009.

[13] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018.

[14] Lucian Busoniu, Robert Babuska, Bart De Schutter, and Damien Ernst. *Reinforcement Learning and Dynamic Programming using Function Approximators*, volume 39. CRC press, 2010.

[15] Richard Bellman. *Dynamic Programming*. Dover Publications, 1957.

[16] D Subbaram Naidu. *Optimal Control Systems*. CRC press, 2002.

[17] Christopher John Cornish Hellaby Watkins. *Learning from delayed rewards*. PhD thesis, King's College, Cambridge, 1989.

[18] David Silver. Lecture notes in the UCL course reinforcement learning. davidsilver.uk/teaching, 2015.
Accessed: 2020-03-30.

[19] Zachariah Levine. Lecture notes in the University of Waterloo course learning 2048 with deep reinforcement learning, January 2018.

[20] Arthur Gill. *Applied Algebra for the Computer Sciences*. Prentice-Hall Personal Computing Series. Prentice-Hall, 1976.

[21] Bengt Lennartson. Lecture notes in discrete event systems. Unpublished, used in the course SSY165 Discrete event systems, Department of Electrical Engineering, Chalmers University of Technology, September 2018.

[22] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Reading, MA: Addison-Wesley, 1979.

[23] Stoyan Mihov and Klaus U Schulz. *Finite-State Techniques*, volume 60. Cambridge University Press, 2019.

[24] Salomon Sickert, Javier Esparza, Stefan Jaax, and Jan Křetínský. Limit-deterministic Büchi automata for linear temporal logic. In Swarat Chaudhuri and Azadeh Farzan, editors, *Computer Aided Verification*, pages 312–332, Cham, 2016. Springer International Publishing.

[25] Marek Grześ and Daniel Kudenko. Online learning of shaping rewards in reinforcement learning. *Neural Networks*, 23(4):541–550, 2010.

[26] Mark Allen Weiss. *Data Structures and Algorithm Analysis in Java*. Pearson/Addison-Wesley, London, 2. ed. edition, 2006.

[27] Christos H. Papadimitriou and Kenneth Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Prentice-Hall, 1982.

[28] Sham Machandranath Kakade et al. *On the sample complexity of reinforcement learning*. PhD thesis, University of London London, England, 2003.

[29] Open AI. Safety gym. openai.com/blog/safety-gym, 2019. Accessed: 2020-01-30.

[30] OpenAI: Frozen Lake source code github. github.com/openai/gym/blob/master/gym/envs/toy_text/frozen_lake.py. Accessed: 2020-04-24.

[31] Arnold O. Allen. *Probability, Statistics, and Queueing Theory with Computer Science Applications*. Academic Press, Inc., Orlando, Florida, 1978.