University of Minho
School of Engineering

Gilberto Rui Nogueira Cunha

Quantum Bayesian Reinforcement Learning

november 2022

University of Minho
School of Engineering

Gilberto Rui Nogueira Cunha

Quantum Bayesian Reinforcement Learning

Masters Dissertation
Master's in Engineering Physics

Dissertation supervised by
Luís Barbosa
André Sequeira
Michael Oliveira

november 2022

Acknowledgements

This dissertation is a mark in my life, it represents the finish line of my Msc degree and possibly the end of my academic path. Even so, I do believe that these years have not been about reaching this point, as much as they were about the journey itself. These have truthfully been some of the best years of my life, and I would like to take a moment to thank the people that decided to spend them alongside me, in one way or another.

First and foremost, I would like to thank my family. Not all moments of these last years were sunshine and rainbows, and I always could count on you to help me get back up. I am especially grateful to my father. You have gone above and beyond to take good care of me. Thank you for looking after me.

To all of my friends, both the ones I kept from Fafe and the ones I made in Braga, thank you for all the fond memories. It has been the most fun of journeys and I know some of you will stick with me for life. You widened my horizons and made me realize there is more to life than school and work, and that balance is key in everything. Thank you for helping me live a happier and healthier life.

To my supervisor, professor Luís Barbosa, I thank you for giving me the liberty to change the direction of this thesis and aiding me in attending my first conferences. I am especially grateful to my co-supervisors, André Sequeira and Michael Oliveira. It has been wonderful both working and getting to know you better. I am extremely grateful for all of the amount of time you have dedicated to making this work more complete and refined, and also for helping me enjoy the process. Thank you for helping me reach my academic goals.

Lastly, I would like to thank my mother. You were the most wonderful person I have ever met, the one who has inspired me to reach greater heights and strive to be the best version of myself. Thank you for having been the best role model I could have ever asked for. I hope I have made you proud.

Statement of Integrity

University of Minho, Braga, november 2022

Gilberto Cunha

Gilberto Rui Nogueira Cunha

Abstract

Reinforcement learning has had many recent achievements and is becoming increasingly more relevant in the scientific community. As such, this work leverages quantum computing to find potential advantages over classical reinforcement learning algorithms, using Bayesian networks are used to model the considered decision making environments. For this purpose, this work makes use of quantum rejection sampling, a quantum approximate inference algorithm for Bayesian networks proposed by Low et al. [2014] with a quadratic speedup over its classical counterpart for sparse networks. It is shown that this algorithm can only provide quantum speedups for partially observable environments, and a quantum classical hybrid lookahead algorithm is presented to solve these kinds of problems. Moreover, this work also includes both a sample and computational complexity analysis of both this quantum lookahead algorithm and its classical alternative. While the sample complexity is shown to be identical for both algorithms, the quantum approach provides an up to quadratic speedup in computational complexity. Finally, the potential advantages of this new algorithm are experimentally tested across different small experiments. Results show that this speedup can be leveraged either to improve the rational decision making skills of agents or to reduce their decision making time due to the reduction in computational complexity.

Keywords reinforcement learning, Bayesian networks, quantum computing, quantum decision-making.

Resumo

A aprendizagem por reforço tem recentemente alcançado muito sucesso e a tornar-se cada vez mais relevante na comunidade científica. Este trabalho tira proveito da computação quântica para encontrar potenciais vantagens do seu uso comparativamente a algoritmos clássicos de aprendizagem de reforço. Nesta procura por vantagens, são utilizadas redes Bayesianas para modelar os ambientes de tomada de decisão considerados. Para este propósito, é utilizado o algoritmo de *quantum rejection sampling*, um algoritmo para inferência aproximada em redes Bayesianas proposto por Low et al. [2014] com um *speedup* quadrático comparativamente ao equivalente clássico para redes esparsas. É mostrado que este algoritmo quântico de inferência apenas tem vantagem na sua aplicação a ambientes parcialmente observáveis, e é apresentado híbrido clássico quântico um algoritmo de *lookahead* para resolver este tipo de problemas. Para além disto, é também incluída uma análise da complexidade de amostragem e complexidade computacional de ambos os algoritmos. Enquanto a complexidade de amostragem é idêntica para as duas abordagens, o algoritmo quântico apresenta um *speedup* na complexidade computacional que é quadrático no melhor dos casos. Por fim, as potenciais vantagens deste novo algoritmo são testadas com três experiências distintas de pequena dimensão. Os resultados mostram que este *speedup* pode ser utilizado tanto para melhorar a capacidade de tomada de decisão de agentes como para diminuir o tempo de tomada de decisão dos mesmos devido à redução da complexidade computacional.

Palavras-chave aprendizagem por reforço, redes Bayesianas, computação quântica, tomada de decisão quântica

Contents

I Introductory material **1**

1 Introduction **2**

1.1 Context 2

1.2 Motivation 4

1.3 Contributions 5

1.4 Outline 5

2 Bayesian Networks **8**

2.1 A brief introduction to Bayesian statistics 8

2.2 Bayesian networks 9

2.3 Inference in Bayesian networks 11

2.3.1 Exact inference 11

2.3.2 Approximate inference 12

2.4 Dynamic Bayesian networks 15

2.5 Dynamic decision networks 18

2.6 Summary 20

3 Reinforcement learning **21**

3.1 The reinforcement learning problem 21

3.2 Fully observable environments 23

3.2.1 Markov decision processes 23

3.2.2 Optimal behaviour in MDPs 27

3.3 Partially observable environments 29

3.4 POMDP lookahead 32

3.5 Summary 35

4 Quantum computing **36**

4.1 Context . . . 36

4.2 Quantum computing with quantum circuits . . . 38

4.2.1 Single qubit systems . . . 38

4.2.2 Multi-qubit systems . . . 40

4.3 Grover's algorithm and amplitude amplification . . . 42

4.4 Quantum Bayesian inference . . . 48

4.4.1 Quantum embeddings of Bayesian networks . . . 48

4.4.2 The evidence phase flip operator . . . 50

4.5 Summary . . . 52

II Core of the Dissertation **53**

5 Quantum sampling subroutines **54**

5.1 Quantum direct sampling . . . 54

5.2 Quantum belief update . . . 56

5.3 A classical quantum lookahead algorithm . . . 63

5.4 Summary . . . 63

6 Complexity analysis **65**

6.1 Sample complexity . . . 65

6.2 Computational complexity . . . 80

6.3 Summary . . . 84

7 Experimental results and discussion **85**

7.1 Comparison metrics . . . 85

7.2 Experiments . . . 87

7.2.1 Tiger problem . . . 87

7.2.2 Robot exploration problem . . . 88

7.3 Results . . . 89

7.3.1 Cumulative reward comparison . . . 90

7.3.2 Execution time comparison . . . 92

7.4 Discussion . . . 93

8 Conclusions and future work **95**

8.1 Conclusions 95

8.2 Prospects for future work 96

III Appendices **103**

A Quantum reward and observation sampling **104**

A.1 Quantum reward sampling 104

A.2 Quantum observation sampling 106

B Complexity analysis auxiliary results **110**

List of Figures

1 A BN over three binary random variables 10

2 A generic representation of a DBN 16

3 A DBN for speed monitoring in a car 16

4 A generic representation of how a DBN is stored in memory (CPTs not included). 17

5 The simplest DDN, with only one node representing the states, actions, observations and rewards for each time-step. 19

6 Generic representation of how a DDN is stored in memory (CPTs not included). 19

7 Interaction at time-step t between the *agent* and the *environment* 22

8 Gridworld environment representation, taken from Edwards et al. [2018]. 26

9 A two-step ($H = 2$) lookahead tree. Belief nodes are pentagon-ally shaped and observation nodes are circle shaped. For diagram simplicity, only one of the observation nodes and one of its belief nodes is expanded. 33

10 Representation of the qubit in Equation (4.7) in the Bloch Sphere. 40

11 Representation of the probability amplitudes of a quantum state during the application of the Grover Operator. a) represents the probability amplitudes of the initial state in a uniform superposition, b) represents the probability amplitudes of the initial state after the phase flip of the good state $|1\rangle$, and the red line represents the mean amplitude $\overline{\alpha} = 0.25$. Finally, c) represents the probability amplitudes after the inversion about the mean. 45

12 Representation of the rotational effect of the Grover operator G. 46

13 Quantum circuit encoding the BN of Figure 1. 49

14 Quantum circuit for the evidence phase flip operator $\mathcal{S}_e$ for the BN of Figure 1 with evidences $R = 1$ and $W = 0$. 51

15 Representation of a DDN for performing the belief update operation. 54

16 Quantum circuit for a belief update in the DDN of Figure 15 57

17 Representation of the tiger problem. 88

18 Diagram of the robot exploration problem 89

19 Comparison of the performance of the classical lookahead with horizon $H = 1$ over different numbers of classical samples for: a) the tiger problem, b) the robot exploration problem. The higher the cummulative reward, the better the performance. 90

20 Comparison of the performance of the classical lookahead with horizon $H = 1$ against the quantum lookahead over different numbers of classical samples for: a) the tiger problem, b) the robot exploration problem. Positive difference implies advantage to the quantum lookahead. 91

21 Cumulative reward difference of the quantum and classical lookahead with horizon $H = 2$ over different numbers of classical samples for: a) the tiger problem, b) the robot exploration problem. A positive difference implies quantum advantage. 91

22 Mean decision making execution time over time of the quantum and classical lookahead algorithms. The algorithms were executed with an horizon $H = 2$ and using 5 classical samples for the experiments: a) the tiger problem, b) the robot exploration problem. The lower the execution time, the better the performance. 93

23 a) A DDN for performing the reward sampling operation. Green nodes are encoded and red nodes are measured in the quantum circuit on the right. b) Quantum circuit for reward sampling using the DDN on the left. 104

24 a) A DDN for performing the observation sampling operation. Green nodes are encoded and red nodes are measured in the quantum circuit on the right. b) Quantum circuit for observation sampling using the DDN in the sub-figure on the left. 107

List of Tables

1 Comparison between classical and quantum complexity for rejection sampling inference in BNs, taken from Low et al. [2014]. 51

Part I

Introductory material

Chapter 1

Introduction

1.1 Context

In today's world, technology can be seen almost anywhere. It has become an instrumental part of how we communicate, perform transactions, work and entertain ourselves. All of this was made possible due to the invention of the computer, a digital device that performs arithmetic and logic as its basic operations. These operations are not special in the sense that only computers can do them, as we all perform simple arithmetic in our everyday lives. In fact, the term ”computer” used to refer to people that performed calculations by hand. One big difference between computers and humans when carrying out these calculations is that computers can do it much, much faster than any human ever could[1]. Of course, computers have greatly evolved over time, with one of the major turning points being the transistor: an electronic component used as a building block of computer designs, that allowed them to be made more compact, more efficient and more powerful. Being more and more refined and compactified since its conception, the number of transistors in a processor approximately doubled every two years, as according to the empirical Moore's law Schaller [1997]. However, it seems that Moore's law is slowing down, and possibly coming to a halt in the near future. As the refinement of the transistor seems to be reaching a point of diminishing returns, a better option for scaling the power of computers in the long term becomes increasingly more relevant.

During the 20th century, many extraordinary physicists, such as Albert Einstein, Werner Heisenberg, Erwin Schrödinger and Paul Dirac, developed a new theory of Physics: the theory of quantum mechanics. This theory proved time and time again to be one of the most precise scientific theories ever conceived, and paved the way for a number of scientific breakthroughs on the transistor, semiconductors and magnetic resonance imaging. As it became more and more apparent that quantum mechanics provided a better explanation of the universe, especially on the smallest of scales, the idea of simulating quantum systems

[1] Just to prove this point, most modern day laptops can perform around a thousand million of logics operations in a single second.

seemed very appealing, as this could lead to incredible developments in chemistry and medicine. One man in particular, Richard Feynman, which also played a big role in the development of quantum theory, found an interest in this idea of simulating quantum systems. In 1982, he published a study Feynman [1982] pointing out that simulating quantum systems by classical computers required an exponential overhead in both the size of the system and the simulation time, meaning that these computers were very inefficient at performing these kinds of simulations. In the same study, he also pointed out something that started a entire new school of computation: the possibility of efficiently simulating a quantum system using another quantum system. More specifically, simulating a quantum system on a quantum computer, a device which operated on quantum information according to the laws of quantum mechanics. Naturally, back then, the concept of a quantum computer was still a vague idea, but since then, they have been formalized, built, and are even theorized to be more efficient than classical computers for a variety of computational processes. For the computational processes where quantum computers seem to be advantageous, they could be the solution to the halting of Moore's law and the possible stagnation of classical computation, and hence have been getting a lot of attention in the scientific community.

One interesting application of computing devices that has been getting a lot of traction in the recent years is the field of artificial intelligence (AI). This field of computer science concerns itself with using computers to develop intelligent behaviour, such as automating the process of rational decision making. It leverages many mathematical fields as its foundations, among which is probability theory, a mathematical theory concerned with the formalization of probabilities. One of the earliest examples of the use of probability theory in AI is the use of Bayesian networks in expert systems Wiegerinck et al. [2010]. Bayesian networks are models that use Bayesian probabilities to represent and reason about probabilistic systems in an easily interpretable graphical manner, and are especially useful for dealing with uncertainty. This makes Bayesian networks a powerful tools, useful in a wide variety of domains, such as medical diagnosis Cruz-Ramirez et al. [2007], risk analysis Weber et al. [2012] or model maintenance systems Nannapaneni et al. [2016].

Recently, machine learning (ML), a data-driven approach for AI has been getting increasingly popular. Systems developed with ML principles learn patterns on a set of data and are later used to make predictions. This is a very different paradigm to anything previously seen, as the systems do not need to be programmed to follow specific rules to solve a problem, but instead learn by themselves from the provided data. One particular subfield of ML, called reinforcement learning (RL), uses this data-driven approach to learn rational decision making in order to reach a predefined goal. It has had a number of recent remarkable achievements: beating the best human players of Go Silver et al. [2016], solving the protein folding problem Jumper et al. [2021] and very recently finding efficient algorithms for matrix multiplication Fawzi et al. [2022].

This dissertation encompasses quantum computing, Bayesian networks and reinforcement learning and studies their intersection. More specifically, its main goal is to find if there are any advantages in using quantum computing for solving reinforcement learning problems using Bayesian networks.

1.2 Motivation

Learning and artificial intelligence have been among the most important developments of computer science of this century. There have been a number of advancements, allowing for models to generate images, stories and even programs from text, high accuracy text to speech applications, translation, recomender systems and the list goes on. These systems are somewhat intelligent, in the sense that they are able to accurately perform these tasks, sometimes even better than human can. Still, there has been a search for a higher form of intelligence, a general intelligence, that can not only learn patterns but understand concepts and apply them to solve a multitude of tasks without any supervision. This notion of intelligence seems to be far off into the future, and more research has to be made in order to improve current methods and create better ones, to improve the intelligent systems of the present day.

One of the possibilities for improving intelligent systems is quantum computing, as it is a very recent field of computation, with few works done on its intersection with artificial intelligence. Quantum computing is theorized to be more efficient than classical computing for certain problems, although it is known that it can not solve any problems that are unsolvable on classical computers. Naturally, it is very much possible, as some studies already show, that its advantages also apply to rationally solving decision making problems.

Bayesian networks are also very relevant tools for modeling uncertainty in a multitude of different systems, and the use of Bayesian statistics is of utmost importance to artificial intelligence. For intelligent decision making, it is crucial to be able to evaluate the likelihood of different outcomes, and to leverage that uncertainty in order to ponder across the different decisions that can be taken. Therefore, Bayesian networks present themselves as models that can be very useful for rational decision making applications.

In summary, Bayesian networks are merged with reinforcement learning for their easy interpretability and its capacity for dealing with uncertainty. Quantum computing is incorporated in this work in order to try to amplify the capabilities of the combination of reinforcement learning and Bayesian networks. More importantly, these topics are all extremely interesting from my own point of view, and the choice of this dissertation topic is deliberate in both its usefulness and in an attempt to fulfill my own curiosity for learning more about intelligent systems and Bayesian statistics.

1.3 Contributions

The first contribution of this work is the conception of a quantum classical hybrid lookahead algorithm for solving partially observable Markov decision processes. This entails the composition of a classical lookahead algorithm with quantum sampling subroutines. Mathematical proofs for the correct functioning of the quantum circuits responsible for the sampling subroutines are also provided.

In order to compare the classical and quantum classical versions of the lookahead algorithm, a detailed derivation and theoretical analysis of both the sample and computational complexity of both of these algorithms is provided. It is found in this analysis that the quantum classical variant does indeed provide an up to quadratic time complexity speedup over its classical counterpart.

Finally, the two lookahead algorithms are also compared in experimental terms based on two different studies. The first study compares the performance of the classical lookahead and the quantum classical lookahead with an increased number of samples, proportional to its speedup in complexity, across different experiments. The goal of this study is to verify whether the speedup in computational complexity of the quantum variant can also lead to better decision making agents. The second study compares both algorithm's execution times to verify the derived computational complexity speedup for the quantum algorithm.

Some of the contributions of this work were accepted as contributed talks at two conferences:

1. QIP22 (Quantum Information and Probability: from Foundations to Engineering), presented at Linnaeus University in Växjö, Sweden, June 15th of 2022.

2. WADT22 (26th International Workshop on Algebraic Development Techniques 2022), presented in Aveiro, June 28th of 2022.

1.4 Outline

This dissertation has the aim of studying Bayesian networks, reinforcement learning and quantum computing and their intersection. Bayesian networks' are presented in Chapter 2 as a powerful tool for representing probability distributions, mainly for two reasons: they are a memory compact way of representing these distributions (by making use of conditional probability tables) and are also easily interpretable, since

Bayesian networks are graphical models. As there is usually a tradeoff between space and time complexity, despite the memory compactness of Bayesian networks, they require the use of inference algorithms to extract probability distributions from them. Among these algorithms are exact algorithms and approximate ones, the approximate algorithms being less precise, but having a much more reasonable time complexity, hence being preferred in the context of this dissertation. Finally, to introduce both time dynamics and decision making into these structures, both dynamic Bayesian networks and dynamic decision networks, extensions of regular Bayesian networks, are introduced at the end of this Chapter.

Chapter 3 details how reinforcement learning is a relevant approach to rational decision making, formally introducing the notion of agency and environments. Moreover, Chapter 3 shows that reinforcement learning environments, represented as either fully or partially observable Markov decision processes, can be modeled using Bayesian networks, allowing the use of these graphical models to solve rational decision making problems and making the first connection between these two topics of study.

As an introduction to the field of quantum computation, Chapter 4 discusses the basics of the circuit model of quantum computing, from multi-qubit systems to the Grover and amplitude amplification algorithms. More importantly, a quantum version of an approximate inference method, quantum rejection sampling, is explained and shown to have a quadratic time complexity advantage over classical rejection sampling for some types of Bayesian networks. The possibility of performing inference, and more efficiently at that, using a quantum computer, connects Bayesian networks to quantum computing in the context of this dissertation. By further using these Bayesian networks as models of reinforcement learning environments, and applying this quantum inference algorithm over them, all of the three main topics of study are finally tied together.

Starting with the contributions of this dissertation, Chapter 5 provides a more extensive explanation of how quantum rejection sampling can be applied to reinforcement learning by using dynamic decision networks. It explains how the three sampling processes for performing the lookahead algorithm can be extracted with quantum circuits and provides mathematical proofs that these quantum circuits work as intended. The Chapter closes with a brief section explaining how a hybrid quantum classical lookahead algorithm can be used by leveraging these quantum sampling subroutines.

Chapter 6 provides a theoretical comparison between the classical and the quantum lookahead algorithms, described in Chapters 3 and 5, respectively. This comparison is made by mathematically deriving the computational complexity of each algorithm. At the start of the Chapter, the sample complexity for both algorithms is derived, providing constraints under which the lookahead algorithm can be considered probably approximately correct. The final section of this Chapter builds on the proofs of the sample complexity to derive the full computational complexity of these algorithms, showing that the quantum lookahead does have an advantage over its classical counterpart, one that depends on the particular problem it is applied to.

To experimentally analyze whether the time complexity advantage derived in Chapter 6 could be translated into better decision making agents, Chapter 7 experimentally compares the performance of the classical algorithm to the quantum algorithm with an increase in the number of samples, which is proportional to its time complexity advantage. Moreover, a comparison of the time complexity of both algorithms is also presented. These comparisons are performed over different experiments and considering varying configurations of the lookahead algorithm.

Chapter 2

Bayesian Networks

Performing intelligent decisions often depends on weighting different possible outcomes and choosing actions under uncertain situations. A rational decision making process is always performed in the context of some system, which can be dynamic and probabilistic in nature. One way of capturing these probabilistic relationships is through a *Bayesian Network* (BN).

This Chapter presents a description of BNs, a graphical probabilistic model that defines probabilistic relationships using conditional probabilities. It is shown how they can be used to model a dynamic system and how probabilistic information about that system can be extracted from the BN using *inference* methods.

2.1 A brief introduction to Bayesian statistics

When making decisions, it is not always the case that every relevant piece of information is available. To clarify, in this work, a distinction is made between two types of decisions, just as in Russell and Norvig [2009]:

- *Simple decisions*: a single decision made in a time-independent system. As time dynamics are not involved, how that action might condition future actions is not relevant for the decision process.
- *Complex decisions*: a sequence of decisions in a dynamic system that reacts to the actions taken. In this scenario, an intelligent agent[1] should not just consider the current action, but also how the actions it takes might affect its future options.

Most complex decisions are associated with some degree of uncertainty, since it is not always clear to the agent in what situation it finds itself in or even the possible outcomes of its actions. As intelligent

[1] An agent is an entity that takes actions in the context of a system in which it is inserted.

beings, people make these kinds of decisions everyday. In the same way, rational agents should be able to deal with these uncertainties and make good decisions according to their beliefs.

Beliefs, however, can be changed in light of new information. Consider that a person is outside and sees smoke in the hills. He knows that the probability of seeing smoke is $P(S) = 0.1$, that the probability of there being a fire is $P(F) = 0.01$ and that the probability of seeing smoke if there is a fire is $P(S|F) = 0.9$. He wants to answer the question: "what is the probability of there being a fire given that I am seeing smoke?". He knows fires are uncommon, but since he is seeing smoke, there being a fire seems a much more likely scenario.

This effect is very well known in probability theory and can be captured by *Bayes' theorem* Wasserman [2010]:

$$P(A|B) = \frac{P(B|A)\,P(A)}{P(B)} \tag{2.1}$$

Bayes' theorem relates the a *prior* probability distribution $P(A)$ to a *posterior* probability distribution $P(A|B)$. The prior probability is an initial belief, such as the probability of there being fire in the example above $P(F) = 0.01$, while the posterior distribution described how to effectively change our prior probability given some additional information. For the example above, knowing that there is smoke increases the chances of there being a fire, since $P(F|S) = \frac{0.9\times0.01}{0.1} = 0.09$.

As can be seen, this theorem helps to update beliefs in the presence of new information. Its usefulness is applicable to a very wide range of applications, from engineering to economics, and also exceptionally helpful for intelligent agents to make decisions under uncertain situations Ghavamzadeh et al. [2016]. Moreover, and most relevant in this Chapter, BNs can use conditional probabilities to model probabilistic environments.

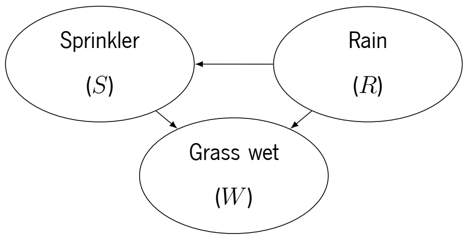
2.2 Bayesian networks

A BN is a probabilistic graphical model encoding knowledge over a set of variables X whose values are unknown (denoted as *random variables*). Random variables (RVs) can take *discrete* or *continuous* values, and are be connected by an edge if there is a probabilistic dependency between them. Moreover, there can not be any cyclic dependency between them. In this dissertation, RVs are denoted by a capital letter, while their values are denoted by a lower case one (e.g., RV X can take value x, or $X = x$).

A BN is a *directed acyclic graph* (DAG), where each node X_i contains a *conditional probability table* (CPT) of its values conditioned on its parent RV values (see fig. 1), mathematically written as $P(X_i|\mathit{parents}(X_i))$.

R	S	$P(S\|R)$
False	False	0.6
False	True	0.4
True	False	0.99
True	True	0.01

R	$P(R)$
False	0.9
True	0.1



R	S	W	$P(W\|S,R)$
False	False	False	0.6
False	False	True	0.4
False	True	False	0.1
False	True	True	0.9
True	False	False	0.01
True	False	True	0.99
True	True	False	0.01
True	True	True	0.99

Figure 1: A BN over three binary random variables

This representation of RVs and their dependencies allows to compactly represent *joint probability distributions* (JPDs) Russell and Norvig [2009], which can be written as follows:

$$P(x_1, x_2, \ldots, x_N) = \prod_{i=1}^{N} P(x_i | \textit{parents}(X_i)) \tag{2.2}$$

Furthermore, as discussed in the next Section, it is possible to infer any probability distribution $P(\mathcal{Q}|\mathcal{E} = e)$ from a BN, where $\mathcal{Q}$ is a set of *query variables*, $\mathcal{E}$ is a set of evidence variables and e is denoted as the *evidence*.

These characteristics make BN a very useful tool with applications in various scientific domains, ranging from performance evaluation Zhu and Deshmukh [2003], to health monitoring Kothamasu et al. [2006] or even aircraft engineering Li et al. [2017]. Even more relevant to this work, as inferring a conditional probability distribution from a BN can be viewed as an application of the aforementioned *Bayes' Theorem*, BNs are useful for modeling and solving decision making problems.

2.3 Inference in Bayesian networks

Once a BN has been constructed, some sort of algorithm is still needed to infer a *posterior* probability distribution $P(\mathcal{Q}|\mathcal{E}=e)$ from a given evidence e, since these probability distributions can only be directly extracted from the BN if they match with its CPT entries, which is often not the case. This extraction is referred in the literature as *inference*, and algorithms that perform it are called *inference algorithms*.

Inference algorithms can be grouped into two big categories: *exact inference* and *approximate inference* algorithms. The first kind can calculate the exact posterior probabilities at a high computational cost, while the second only obtains some approximation value with the advantage of being computationally more tractable Murphy [2002].

Despite the advantage of computing a true posterior distribution rather than an approximation, real-world applications often require the use of large BNs, which make it prohibitive to perform *exact inference* in a reasonable amount of time for most use cases, due to its high computational cost. For these applications, *approximate inference* methods are a necessity, where precision is traded for quicker computations.

As the next Chapter clarifies, BNs can be used to solve RL problems. In the process of searching for solutions, *inference* in the BN is required to extract the probabilistic dynamics of RL systems, which are called *environments*. This solution-finding process is computationally expensive, and inferring with exact methods further amplifies this effect. As solving RL problems is a main goal of this study, this Section focuses mostly on *approximate inference* methods, only presenting *exact inference* to demonstrate the exponential cost associated with these methods.

2.3.1 Exact inference

A simple example of exact inference is the *enumeration* algorithm. Consider $\mathcal{Q}$ a set of query variables, $\mathcal{E}$ a set of evidence variables that take the value e and $\mathcal{R}$ the remaining variables encoded in the BN (besides $\mathcal{Q}$ and $\mathcal{E}$). A posterior probability distribution can therefore be expressed by the following expression:

$$P(\mathcal{Q}|\mathcal{E}=e) \propto P(\mathcal{Q},\mathcal{E}=e) = \sum_{r} P(\mathcal{Q},\mathcal{E}=e,\mathcal{R}=r) \tag{2.3}$$

Notice that for each possible value of $\mathcal{Q}$, each term of the sum over r can be calculated by using Equation (2.2) describing the joint probability distribution of the BNs' RVs. By computing each term using that Equation, performing the sum in Equation (2.3) and normalizing the resulting distribution, one obtains the desired posterior probability distribution $P(\mathcal{Q}|\mathcal{E}=e)$.

The problem with this approach is that the number of terms to be summed over in Equation (2.3) scales exponentially with the size of $\mathcal{R}$. In fact, in the worst case, where most variables have to be summed over, this algorithm has a complexity of $\mathcal{O}\left(N2^N\right)$ Russell and Norvig [2009] for random binary variables, where N is the number of RVs in the BN. Exact inference algorithms are, in general, *NP-Hard* problems Ben-Gal [2008], which are intractable for most real-world applications.

2.3.2 Approximate inference

Given that exact inference algorithms are generally too computationally expensive, seeking approximate methods is essential for these networks to be applicable to a wider variety of applications. The following approximate inference algorithms rely on *randomly sampling* (also denoted *Monte Carlo* algorithms) from a known probability distribution modelled by the BN.

The first algorithm, named *direct sampling*, can be used to extract a sample from the JPD encoded by the BN. If applied repeatedly while collecting all samples, an approximation of this JPD can be obtained.

Getting a sample from the network involves iterating over its RVs in topological order [2]. For the root nodes, a value is sampled according to the probabilities of each values' occurrence. Those values are then propagated to their children, so that a value for those child nodes can be sampled according to the values previously sampled from their parents. This procedure is described in Algorithm 1.

Algorithm 1 Direct sampling algorithm. Returns a sample from the BN's joint probability distribution.

```
Require:
  $\mathcal{B}$, a BN representing the distribution $P\left(X_1, X_2, \ldots, X_N\right)$ with $N$ nodes in topological order.

  $i \leftarrow 0$                                   ▷ Counter to traverse all nodes
  Initialize $r$                                    ▷ Initialize a sample with $N$ empty values
  while $i \neq N$ do                               ▷ Traverse nodes in topological order
      $r[i] \leftarrow x_i \sim P\left(X_i \mid parents\left(X_i\right)\right)$       ▷ Randomly sample from $X_i$
      $i \leftarrow i + 1$                          ▷ Go to next node
  end while
  return $r$
```

For the case of the BN represented in Figure 1, the algorithm involves the following steps:

1. Sample from $P\left(\text{Rain}\right)$. Suppose it returns False

[2] A topologically ordering of a DAG is one where for any edge $\langle X_i, X_j\rangle$ from X_i to X_j in the graph, X_i comes first in the ordering. This sorting can be done using *Kahn's algorithm* Kahn [1962] and should be established only once after constructing the BN to prevent its usage every time inference is to be performed.

2. Sample from $P(\text{Sprinkler}|\text{Rain} = \text{False})$. Suppose it returns True

3. Sample from $P(\text{Grass Wet}|\text{Rain} = \text{False}, \text{Sprinkler} = \text{True})$. Suppose it returns True

4. The sample gathered is Rain=False, Sprinkler=True, Grass Wet=True

Direct sampling therefore allows the extraction of a random sample $x_1, x_2, \ldots, x_N \sim P(X_1, X_2, \ldots, X_N)$ represented in the BN. By repeating this algorithm and collecting a total of n_T samples, each occurring $n(x_1, x_2, \ldots, x_N)$ times, the joint probability distribution can be approximated by:

$$P(x_1, x_2, \ldots x_N) \approx \frac{n(x_1, x_2, \ldots, x_N)}{n_T} \tag{2.4}$$

As an example, if the sample Rain=False, Sprinkler=True, Grass Wet=True occurred 456 times out of a total of 1000 samples, its probability would be approximated by $\frac{456}{1000} = 0.456$.

A single sample can be extracted using *direct sampling* in $\mathcal{O}(NM)$ time Russell and Norvig [2009], where N is the total number of nodes in the BN and M is the maximum number of parents of any node in the network. The time complexity is linear with respect to both of these parameters, which is a large benefit when compared to the exponential cost of *exact inference* methods, since M is usually much smaller than 2^N (if the BN is not too densely connected). Nonetheless, this algorithm does not allow sampling from a posterior probabiity distribution $P(\mathcal{Q}|\mathcal{E} = e)$, as it can only sample from the joint distribution. To solve this issue, another approximate sampling algorithm, namely *rejection sampling*, can be used.

Rejection sampling follows naturally from *direct sampling*. To sample from a probability distribution $P(\mathcal{Q}|\mathcal{E} = e)$, first repeatedly sample from the joint distribution using *direct sampling* and only accepting samples with evidence $\mathcal{E} = e$, discarding any sample that does not satisfy that condition. Using all the accepted samples, a count for every possible value q of the query variables $\mathcal{Q}$ is stored and turned into an approximate probability as in Equation (2.4). This whole procedure is captured by Algorithm 2.

Due to the rejection of samples in this algorithm, it naturally follows that it is more costly to gather a single sample with *rejection sampling* rather than *direct sampling*. For *rejection sampling*, the time complexity to gather one sample is $\mathcal{O}(nmP(e)^{-1})$ Low et al. [2014], since the lower the probability $P(e)$ of the evidence taking value e, the more likely the sample is to be rejected.

Using this algorithm, it is possible to infer from the network any probability distribution involving the RVs of its nodes. This inference can be used to extract probabilistic information about any system, which is useful in decision making problems to let agents reason about their uncertainties and help them make decisions.

The algorithms presented in this Section are a small collection of the many existing inference algorithms for BNs, which were chosen due both to their simplicity and for the later introduction of *quantum*

Algorithm 2 Rejection sampling algorithm. Returns a conditional probability distribution.

Require:

$\mathcal{B}$, a BN representing the distribution $P(X_1, X_2, \ldots, X_N)$ with nodes topologically ordered;

$\mathcal{Q}$, the set of query variables;

$\mathcal{E}$, the set of evidence variables;

e, the values for the query variables;

n, the number of samples to generate;

```
$i \leftarrow 0$                                    ▷ Counter for the number of samples gathered
Initialize $r$                  ▷ Dictionary of counts over all possible values of $\mathcal{Q}$, initially zero
while $i \neq n$ do
    $x \leftarrow$ direct sampling $(\mathcal{B})$                  ▷ Get a sample from the BN's JPD
    if $x$ is compatible with $\mathcal{E} = e$ then
        $r[q] \leftarrow r[q] + 1$                ▷ Increase count. $q$ are the values of $\mathcal{Q}$ in $x$
        $i \leftarrow i + 1$
    end if
end while
$r \leftarrow$ normalize $(r)$                                ▷ Turn counts into probabilities
return $r$
```

rejection sampling, a quantum inference algorithm that has a speedup compared to its classical counterpart presented here. For a description of a larger range of inference algorithms in BNs, the reader is referred to Guo and Hsu [2002].

2.4 Dynamic Bayesian networks

BNs present a way of formulating a model involving multiple variables that are related to each other, but can only handle cases where the model is static. In some cases, information about past states of that object is needed to infer its current state. Consider the example of inferring the position of an object: to do this, information about the previous position and velocity are needed.

A DBN is then a collection of BNs, each corresponding to a different time-slice (represented by non-negative integers), whose RVs have temporal dependencies, meaning that any RV can depend on RVs at previous time slices. For simplicity, it is usually considered that the structure of the BN does not change between time-slices (it contains the same RV, edges and CPTs). Moreover, in this work, only *first order Markov models* are considered, meaning that the RVs of the BN can only depend on RVs of the previous time-slice Neapolitan and Jiang [2007]. First order Markov models are assumed because RL environments are generally defined in the same manner, as Chapter 3 clarifies.

Some clarification is due when referring to changing systems. A system can be dynamic by changing states over time according to its transition dynamics. Transition dynamics, however, can also evolve over time. In this work, a changing environment is one whose states change according to its transition dynamics, but whose transition dynamics are static (they do not evolve, the dynamics are the same for each time step). For a DBN to model time-dependent dynamics, its structure and/or CPT values should change between time-slices, but this work considers them fixed.

Since DBNs can model dynamic systems, they are very widely used for monitoring applications. In monitoring, a distinction between observable and unobservable RVs should be made. Physical information about a system is always obtained via a measurement device which outputs some sensory readings. These readings are therefore observable, while the true physical information of the system is unobservable. The sensor measurements do not necessarily have a one to one correspondence with the true physical information of the system. After all, sensors are susceptible to noise and have limited precision, but their measurement is still usually close to the true physical value. The relationship between observable and unobservable quantities is usually called the *sensor model* Russell and Norvig [2009].

Let S denote a set of unobservable RVs, or the unobservable *state* of a system, and O the observable

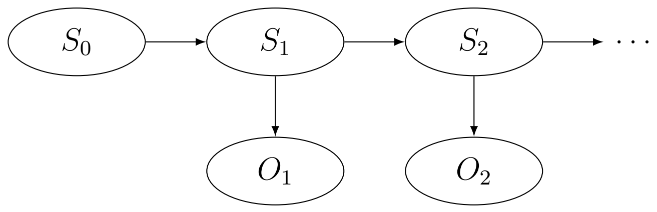


Figure 2: A generic representation of a DBN

RVs. A DBN, such as the one presented in figure 2, encompasses two models Russell and Norvig [2009]:

- A *transition model* $P(S_{t+1}|S_t)$, defining state transitions between time-steps, crucial to allow the system to evolve over time.
- A *sensor model* $P(O_t|S_t)$, defining the relation between the observable and unobservable quantities of the system, with the purpose of gathering measurement information.

Initializing a DBN requires defining both these models and a prior distribution over initial states $P(S_0)$. Figure 3 provides an example of a DBN for car speed monitoring, where the state RVs are acceleration$_t$ and velocity$_t$, while the observation RV is speedometer$_t$.

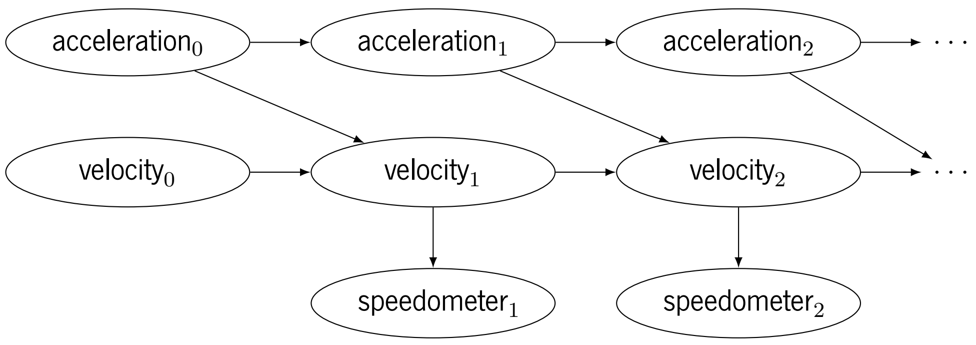


Figure 3: A DBN for speed monitoring in a car

Inference in a DBN can be treated the same way as a regular BN. It has not yet been explained how to store a structure like this computationally (it is a semi-infinite collection of BNs, which would require infinite memory to store), but for now assume it is just a very large BN and all the inference principles should be the same. The usefulness of inference in DBNs can be expressed in terms of four different tasks Russell and Norvig [2009]:

- *Monitoring*: consists of calculating a distribution over states given all observations so far $P(S_t|o_{1:t})$. This distribution is called the *belief state*.
- *Prediction*: similar to *monitoring*, but the distribution is over future states $P(S_{t+k}|o_{1:t})$.

- *Smoothing*: the task of calculating a distribution of past states given all observations so far $P\left(S_k|o_{1:t}\right),\ k < t$.
- *Most likely explanation*: finding the sequence of states that is most likely to have originated a sequence of observations. This involves computing the distribution $P\left(S_{1:t}|o_{1:t}\right)$ and choosing the state sequence $s_{1:t}$ with the highest probability.

In the context of this work, *monitoring* is the most relevant task. When an agent is in a system it can not directly observe, calculating the *belief state* is a way to quantify the probability of being in any state at any given point in time so that it can use this information to make good decisions. It is a way of quantifying its *uncertainty* about the system and using that uncertainty in a rational manner. However, inferring the *belief state* directly is a costly process, since at time-step t, inference over a DBN with t time-slices is needed. It can be easily seen that the computational cost of this inference grows with the time-step increase, which provides a big challenge for monitoring a system over a long period of time. Luckily, a *belief state* at time-step $t+1$ can be expressed in terms of the previous *belief state*, such that a recursive computation can be used to update beliefs Russell and Norvig [2009]:

$$P\left(S_{t+1}|o_{1:t+1}\right) \propto P\left(o_{t+1}|S_{t+1}\right)\sum_{s_t} P\left(S_{t+1}|s_t\right)P\left(s_t|o_{1:t}\right) \tag{2.5}$$

The update rule expressed in Equation (2.5) allows the computation of a *belief state* to have a constant time cost relative to the current time-step. Since the belief-state captures the information of every previous evidence so far, when used to replace the CPT of the current state, every previous time-slice of the network can be discarded (see Figure 4). This makes it possible, while performing computations, to only store two consecutive time-steps of the DBN [3], which also impedes memory usage to scale with the increase of the time-step in *belief state* inference.

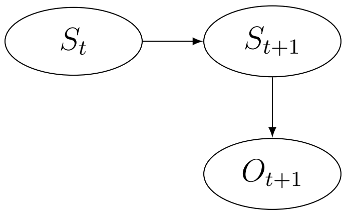


Figure 4: A generic representation of how a DBN is stored in memory (CPTs not included).

Since in this work only dynamic systems with time-independent dynamics are considered, the CPTs of the next state S_{t+1} and the next observation O_{t+1} need not be replaced when the time-step is increased,

[3] Only two time-steps have to be stored because only *first order Markov models* are considered. For a *Markov model* of order n, a total of $n+1$ time-steps need to be stored, since variables from time-step t depend on variables of the previous n time-steps.

as the dynamics of the DBN are time-independent. When the time-step increases, only the CPT of the current state S_t needs to be updated according to the belief update rule of Equation (2.5).

2.5 Dynamic decision networks

From what is shown in the previous section, DBNs are a collection of BNs representing time-slices whose state RVs are connected. DBNs can be used to represent dynamic systems, but are still unable to capture the notion of agency in their structure. This section presents a description of *dynamic decision networks* (DDNs), an extension of DBNs that adds agency (the ability to take actions) to dynamic systems.

To embed the notion of agency in the structure of DBNs, there are two aspects that need addressing:

- The notion of actions must be captured in the DDN structure. In order to do so, a new RV is introduced for each time-slice, an *action* node.
- In order to use some algorithm to find a good action to take, a notion for how good actions are is needed. This is done using another RV, the *reward*, a real number which quantifies how good a certain action is at each time-step[4].

The way these RVs depend on each other is not strict, as there are many possible ways to do so. The correct way to connect these new RVs to the existing ones (and vice-versa) in the DBN structure varies from problem to problem. Here, the following assumptions for a DDN structure (see Figure 5) are considered:

- Actions (denoted by RVs A_t) influence state transitions. As such, the *transition model* is now given by a probability distribution $P\left(S_{t+1}|S_t, A_t\right)$
- Actions influence the observations received. This changes the *sensor model* to be of the form $P\left(O_t|S_t, A_{t-1}\right)$
- Rewards (denoted by RVs R_{t+1}) depend on the state and action. This introduces a new model, the *reward model*, which is of the form $P\left(R_{t+1}|S_t, A_t\right)$.

A subtle but very important point of DDNs is that action nodes are always root nodes. This is done so that the decision making process remains unbiased Russell and Norvig [2009]. Also regarding action nodes, the way their CPTs are constructed can be different to other RVs. If an optimal action is found using some algorithm (which won't be covered until Section 3.4), a deterministic CPT with probability 1 of that action occurring and 0 otherwise is constructed and used as the CPT for that action node. Of course,

[4] A more detailed explanation of the reward is provided next Chapter. For now, the text will use this simplified notion of a reward.

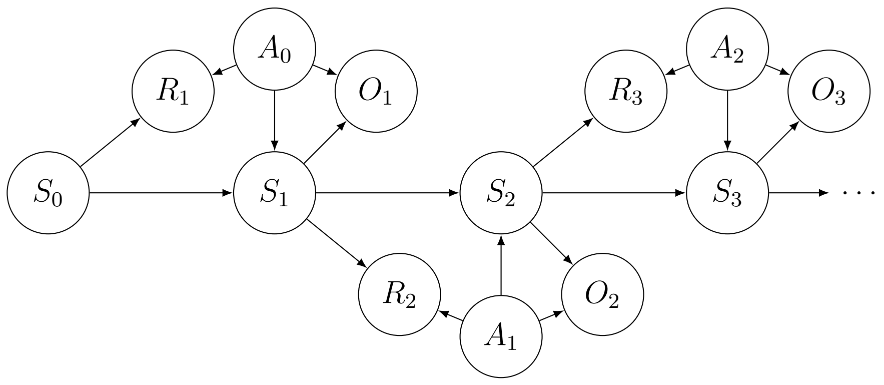


Figure 5: The simplest DDN, with only one node representing the states, actions, observations and rewards for each time-step.

if one wishes to encode the action node CPT with some other probability distribution $P(A_t)$ instead, that is also a possibility, but the former method is preferred in this work.

Naturally, the CPTs for the state and observation nodes now must change (relative to the CPTs of DBNs) in order to accommodate their new parent RVs. Another important point is that each state, action, observation and reward in a time-slice does not need to be represented by a single node as in figure 5 (the same holds true for states and observations in DBNs). Each of them can have multiple nodes interconnected in order to represent that state, action, reward or observation, as if they were BNs of their own, but connected to each other (although it is assumed from here on out, for simplicity, that these RVs are all represented by a single node).

Just as with DBNs, the belief update rule can be reformulated to also include actions as a source of information:

$$P(S_{t+1}|o_{1:t+1}, a_{1:t}) \propto P(o_{t+1}|S_{t+1}, a_t) \sum_{s_t} P(S_{t+1}|s_t, a_t) P(s_t|o_{1:t}, a_{1:t-1}) \qquad (2.6)$$

By performing this belief update, the CPT of the DDNs can also be updated, allowing only two time-steps of the DDN to be stored in memory when running these structures on a computer (see Figure 6).

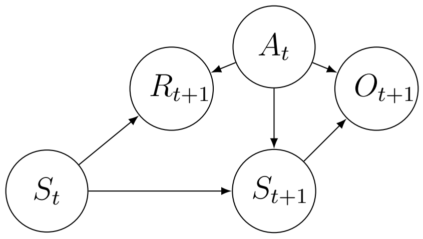


Figure 6: Generic representation of how a DDN is stored in memory (CPTs not included).

Even though this section does not cover any algorithms for actually performing rational decision making, which is left for later sections when RL is introduced, it lays the foundation of a very powerful model that can be used to make decisions in a dynamic system. Moreover, a DDN can actually model RL environments, as the next Chapter also clarifies.

2.6 Summary

Bayesian networks are graphical probabilistic models that can model systems by representing them with random variables and dependencies between them. They also present a compact and memory efficient way of representing joint probability distributions.

To extract probabilistic information from these models, there are two groups of *inference algorithms* to choose from. *Exact inference* is more suitable for situations where very precise calculations are needed and time is not a very relevant factor. On the other hand, when time is of the essence and calculations can be less precise, *approximate inference* methods offer a better solution.

Furthermore, it is important to note that Bayesian netorks can only model time-independent systems. When dynamic systems are involved, an extension of Bayesian networks, named dynamic Bayesian networks, can be used for their modelling. Dynamic Bayesian networks are also capable of modeling uncertainties that arise from possibly bad measurements via belief states and updating that uncertainty with new observations.

Finally, dynamic Bayesian networks can also be extended to include agency in their structure, forming dynamic decision networks. These networks are able to not only model dynamic systems, but also to add agency to those systems and make them react to the actions taken. Just as with dynamic Bayesian networks, they can be used to update beliefs based on new information, which can be leveraged in decision making processes to make informed decisions even with imperfect observations.

Chapter 3

Reinforcement learning

Intelligent decision making is a problem that concerns the field of artificial intelligence (AI). More specifically, one of the sub-fields of AI that tries to solve this problem is called *Reinforcement Learning* (RL).

In this Chapter, a formal description of RL is presented, defining agents capable of intelligent decision making in the context of an environment. Moreover, to link this formalism to the previous chapter, it is shown how DDNs can be used to both model an environment and solve decision making problems in the context of RL.

3.1 The reinforcement learning problem

RL is a sub-field of artificial intelligence concerned with the problem of rational decision making, achieving this goal by rewarding good decisions and punishing bad ones. There are four main components to RL D and Winder [2020]:

- *Actions*: actions are decisions. The main point of RL is to find the best actions in a sequential decision-making setting.
- *Environment*: Representation of a world, usually with many possible situations (or *states*) within which decisions have to be made.
- *Agent*: Represents a decision maker that lives in a world represented by the environment. It makes decisions in that environment order to reach a certain goal.
- *Rewards*: A signal that encodes the goal of the agent. Rewards are usually numeric and provide feedback on how good the agent's decisions are.

Some parallels with BNs can already be drawn. An environment is a dynamic system, but one which also has rewards and agency embedded in it, with an added interpretation that it represents the world an agent is inserted in. As clarified later on in this Chapter, these environments can be modeled using DDNs.

Each interaction between an agent and the environment involves the following steps (see Figure 7):

1. The *agent*, while in the current state "state$_t$" of the *environment*, takes an action "action$_t$"

2. As a reaction to action "action$_t$", the *environment* changes to state "state$_{t+1}$"

3. In the new state "state$_{t+1}$", the *environment* sends the *agent* an observation "observation$_{t+1}$" with information about state "state$_{t+1}$" and a *reward* "reward$_{t+1}$" that evaluates the decision made by the *agent* in step 1

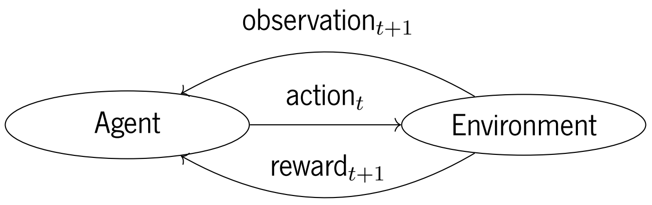


Figure 7: Interaction at time-step t between the *agent* and the *environment*

Given the observation sent to the *agent*, two types of *environments* can be distinguished Kaelbling et al. [1998]. If that observation completely defines the state of the *environment* (hence the *agent* always knows in which state it is), the *environment* is said to be *fully-observable*. However, if that observation only contains partial information about the state, meaning the *agent* can't be certain about its current state, then the *environment* is said to be *partially observable*. In the partially observable case, there is an added layer of uncertainty the *agent* must deal with, an therefore it is modelled in a different way from a *fully observable environment*, as the next sections clarify.

Solving a RL problem entails teaching the *agent* how to behave in its *environment* in order to maximize its chances of reaching a goal. It is a problem of *complex* decision-making, since it involves making good sequences of decisions in a possibly changing *environment*.

Nevertheless, how is this goal defined? And how is it possible to take actions in order to reach this goal? As stated above, a *reward* signal is associated with every interaction between the *agent* and the *environment*, evaluating the decision made in that interaction at that time. It would be tempting to assume that therefore, the *agent* should always take the action that gives him on average the highest *reward*, which isn't far off from the true answer, but is a *shortsighted* view. Doing so fails to consider the future rewards of the *agent*, which could possibly hinder the decisions it has to take later on[1]. A better way to achieve

[1] As an example, suppose you are choosing between working in companies A and B. After making that choice, you can definitely try to make the best out of your experience in the company you chose, but you can't take your choice back. If you end up choosing a bad company, this single decision will influence your decisions to come, and possibly affect how happy you feel with your career as well.

the agent's goal is to choose an action that maximizes an expected *sequence* of rewards, where future rewards are also accounted for. This sequence of rewards, also known as *utility*, quantifies the desirability of that sequence of interactions, where the most desirable leads the *agent* closer to its goal. Making the best decision is therefore the same as choosing the action that maximizes the expected utility (a weighted average of all its possible sequences of rewards), and is known as the *Maximum Expected Utility* (MEU) principle Russell and Norvig [2009].

Given that the goal of RL is that of intelligent decision making, it comes as no surprise that it has a very broad range of applications, from robotics Kober et al. [2013], to autonomous driving vehicles Kiran et al. [2021] and even healthcare Yu et al. [2019]. Nonetheless, in this work, no particular application is targeted. Rather, the RL decision making process is studied as a whole, making this study applicable to any area in which RL is a viable solution.

3.2 Fully observable environments

3.2.1 Markov decision processes

The *environment* plays a key role in RL as it is responsible for representing the world with which the *agent* interacts. It is a crucial part of the decision making process as it defines the context of the agent's actions, without which it wouldn't be possible to determine what good decisions are. As such, formally defining an environment is essential for solving RL problems. In this section, a description of *Markov decision processes* (MDPs) is presented as a way of modelling *fully observable* environments.

Let's suppose it is known that the t previous states of an environment are given by $H_t = S_0 S_1 \dots S_t$, which is called the *history* of states. This history is a very rich statistic of the environment, making it a good candidate to both model the transitions in the environment and plan for the agent's future. Still, it is generally not a very good idea to use it in practice. Firstly, as the time-step grows, the number of states in the history also grows, meaning that the memory needed to store it increases over time, which can be problematic for computing long sequences of decisions. Secondly, dynamics dependent on this whole history are hard to define. An easier and more computationally tractable approach are *Markov states*. Markov States can capture the information of the whole history, meaning that each state is a sufficient statistic of the future Sutton and Barto [2018]. This fundamental property of Markov States is known as the *Markov property*:

$$P\left(S_{t+1} | H_t\right) = P\left(S_{t+1} | S_t\right) \tag{3.1}$$

The Markov property states that the current state is enough to capture the whole history in the transition dynamics. An important note is that these dynamics are also generally dependent on the agent's actions, but that will come later in this section. Another useful reminder about this property that ties it with the previous Chapter is that it is equivalent to only considering *first order Markov processes* when defining DBNs. Given that a state RV at any time-step only depends on the state RVs of the previous time-step, Equation (3.1) holds.

This leads to a first stepping stone in defining an MDP, which is defining a *Markov process* or *Markov chain*. A *Markov process* is a tuple $\mathcal{M} = \langle \mathcal{S}, \mathcal{P} \rangle$ Sutton and Barto [2018] where:

- $\mathcal{S}$ is a set of states
- $\mathcal{P} : \mathcal{S} \times \mathcal{S} \mapsto [0, 1]$ is a probabilistic transition function such that:

$$\begin{aligned} \mathcal{P}_{ss'} &= P\left(S_{t+1} = s' | S_t = s\right) \\ \text{where} \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'} &= 1, \forall s \in \mathcal{S} \end{aligned} \tag{3.2}$$

As can be seen by the transition dynamics of Markov processes, they leverage the Markov property in order to define a dynamic system. Notice how the probabilistic transition function $\mathcal{P}$ and the *transition model* of a DBN defined in the previous Chapter are identical. In fact, if the observation RVs of a DBN are discarded, it can be seen as an alternative representation of a Markov process.

To fully define an environment, there are still two important aspects that a Markov process does not address:

- It does not contain any *rewards*, and it is therefore impossible to judge the value of each transition.
- The notion of *agency* is not captured either, due to the absence of *actions* in its definition.

The first issue is easily addressed with the addition of a reward for each transition in a Markov process. With this addition, it is also a good time to define the value of sequences of rewards or *utilities*. There are multiple ways this can be done, but generally these utilities are defined by assuming that preferences between state sequences are *stationary*, meaning that the preference ordering of the state sequences $[s_0, s_1, s_2, \dots]$ and $[s_0, s'_1, s'_2, \dots]$ should be the same as the preference ordering of the sequences $[s_1, s_2, \dots]$ and $[s'_1, s'_2, \dots]$, since they both start with the same state. Under this assumption, there are only two ways of defining a *utility* Russell and Norvig [2009] or, as it is usually called in the RL, a *return*:

- *Additive rewards*: the return G_t is the sum of all future rewards:

$$G_t = \sum_{k=0}^{T} R_{t+k+1} \tag{3.3}$$

- *Discounted rewards*: the future rewards are added but multiplied by a power of a discount factor $\gamma \in [0, 1]$:

$$G_t = \sum_{k=0}^{T} \gamma^k R_{t+k+1} \tag{3.4}$$

In this work, only the discounted rewards return is considered. The main reason is that with additive rewards, the return can diverge for infinite sequences of the MDP (sequences where the agent does not reach a terminal state of the environment, meaning $T = \infty$), making it difficult to compare two divergent sequences. In contrast, the discount factor ensures that the return G_t in Equation (3.4) is bounded, provided that the rewards are also bounded and that $\gamma \neq 1$, making the series represented in Equation (3.4) converges.

The value of the discount factor also weights the importance of future rewards. If $\gamma = 0$, an agent will be *myopic* and make decisions only to maximize its expected immediate reward. On the other extreme, with $\gamma = 1$, all the rewards at every time-step are of equal importance to the agent's decisions, falling back to the original *far-sighted* definition of return provided in Equation (3.3). Discount factors are generally chosen to fall somewhere between these two extremes, according to how much the agent should value its future rewards.

Adding rewards and the discount factor to a Markov process defines a *Markov reward process* as a tuple $\mathcal{M} = \langle \mathcal{S}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ Sutton and Barto [2018], where:

- $\mathcal{S}$ and $\mathcal{P}$ have the same definition as in a Markov process.
- $\mathcal{R} : \mathcal{S} \times \mathcal{S} \mapsto \mathbb{R}$ is a reward function such that:

$$\mathcal{R}_{ss'} = \mathbb{E}\left(R_{t+1} | S_{t+1} = s', S_t = s\right) \tag{3.5}$$

- $\gamma \in [0, 1]$ is a discount factor to define a bounded return G_t as in Equation (3.4).

Finally, to add agency to a Markov Reward Process, it can be equipped with a set of actions that are available to an agent, allowing those actions to also interfere with both the reward and transition dynamics of a Markov reward process. As such, a *Markov decision process* (MDP) is a tuple $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ Sutton and Barto [2018] such that:

- $\mathcal{S}$ is the set of possible states.
- $\mathcal{A}$ is the set of possible actions.

- $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto [0, 1]$ is a probabilistic transition function:

$$\begin{aligned} \mathcal{P}^a_{ss'} &= P\left(S_{t+1} = s' | S_t = s, A_t = a\right) \\ \text{where} \sum_{s' \in \mathcal{S}} \mathcal{P}^a_{ss'} &= 1, \forall s \in \mathcal{S},\, a \in \mathcal{A} \end{aligned} \tag{3.6}$$

- $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto \mathbb{R}$ is a reward function:

$$\mathcal{R}^a_{ss'} = \mathbb{E}\left(R_{t+1} | S_{t+1} = s', S_t = s, A_t = a\right) \tag{3.7}$$

- $\gamma \in [0, 1]$ is a discount factor to define a bounded return G_t.

There are, however, variations of this MDP that can be defined from this more general one. The reward function $\mathcal{R}$, for instance, can be dependent only on one state s [2] or on one state s and an action a.

As an example of an MDP, consider the gridworld environment of Figure 8. It has 9 states, each represented by individual positions of the grid, and therefore $\mathcal{S} = \{0, 1, \ldots, 8\}$. The agent can select between four actions $\mathcal{A} = \{\text{left, right, up, down}\}$ to move on the grid. The state transitions are deterministic, as in, if the agent is in the bottom left corner and decides to move up, then it moves to the middle square on the left column of the grid with 100% probability. The rewards can be selected in many different ways, such as the following:

- State dependent rewards: a reward of 10 for reaching the goal state and of -1 for any other state.
- State and action dependent rewards: a reward of 1 if the agent moves closer to the goal, -1 if it moves further or does not move, and a reward of 10 for reaching the goal state.

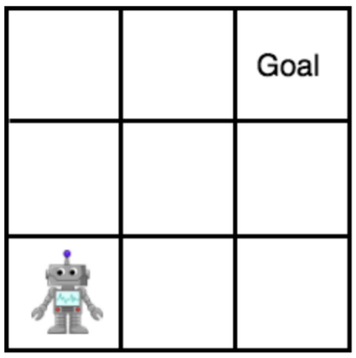


Figure 8: Gridworld environment representation, taken from Edwards et al. [2018].

[2] this one state can be either the origin state of the transition, meaning that $\mathcal{R}_s = \mathbb{E}\left(R_{t+1} | S_t = s\right)$ or the destination state of the transition, such that $\mathcal{R}_s = \mathbb{E}\left(R_{t+1} | S_{t+1} = s\right)$.

3.2.2 Optimal behaviour in MDPs

How does an agent decide what action to pick? Ideally, one would want it to pick the best action, but to get there, a formalism that captures an agent's behaviour is needed. Formally, this is done using a *policy* $\pi : \mathcal{A} \times \mathcal{S} \mapsto [0, 1]$, a distribution of actions given states:

$$\pi(a|s) = P(A_t = a|S_t = s) \tag{3.8}$$

A policy attributes a probability to every action given a state, and an agent following that policy picks actions according to that distribution. The goal of RL is to find the best policy, or a close approximation, for an agent given its environment, in this particular case represented as an MDP. Once again, it is important to recall that the definition of a "good action" depends entirely on the environment, and thus finding this optimal policy must depend on the MDP the agent lives in.

To find the optimal policy, the agent must leverage, in some way, the information it has about the environment. From Figure 7, it can be seen that, in a fully observable environment, at every time-step the agent receives a reward and knows its current state, and therefore this is the information it has to work with to make decisions. As previously stated, rewards can be used to define returns, which are a quantitative way of defining value for a given sequence of rewards. Due to the probabilistic nature of an environment, even while picking the same actions, many different trajectories (sequences of states and rewards) can occur. As such, an optimal policy is one that maximizes the agent's *expected return*, meaning that the return of each trajectory is weighted by the probability of its occurrence. To get to this point, it is easier to start with attributing a utility to single states and actions, which requires the introduction of *value functions*:

- The *state value function* $V : \mathcal{S} \mapsto \mathbb{R}$ defines the *utility* of a state s while following policy π:

$$V^\pi(s) = \mathbb{E}_\pi(G_t|S_t = s) = \sum_{a\in\mathcal{A}} \pi(a|s) \sum_{s'\in\mathcal{S}} \mathcal{P}^a_{ss'}\left(\mathcal{R}^a_{ss'} + \gamma V^\pi(s')\right) \tag{3.9}$$

- The *state-action value function* $Q : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$ defines the *utility* of taking an action a in a state s while following policy π:

$$Q^\pi(s, a) = \mathbb{E}_\pi(G_t|S_t = s, A_t = a) = \sum_{s'\in\mathcal{S}} \mathcal{P}^a_{ss'}\left(\mathcal{R}^a_{ss'} + \gamma \sum_{a'\in\mathcal{A}} \pi(a'|s')\, Q^\pi(s', a')\right) \tag{3.10}$$

The *value-functions* defined above are a way of telling the agent the value of states and actions in order to make decisions. These values are *relative to the agent's policy*, meaning that different policies have different values associated. This is because while the agent wants to maximize its expected return along the many possible trajectories of the MDP, the probability that each trajectory occurs is dependent on the policy itself, because it defines the probability that an agent picks a certain action, and the MDPs state transitions depend on these actions. A better way to evaluate the value of states and actions is to consider the values of an *optimal policy* (a policy which always picks the best action), which are also the optimal values for the states and actions. It might seem like a feedback loop to use an optimal policy to define optimal value functions to then be able to extract an optimal policy, but there are ways to estimate the optimal value functions without knowing optimal policies.

For any optimal policy $\pi^\star$ there is both an optimal state value function $V^\star(s) = \max_\pi V^\pi(s)$ and an optimal state-action value function $Q^\star(s,a) = \max_\pi Q^\pi(s,a)$. Most importantly, given the optimal value functions, it is possible to derive the optimal policy. Taking the state-action value function as an example, if the value of each state-action pair is known, then the best action to take in any given state is simply the one with the highest expected return:

$$\pi^\star(a|s) = \begin{cases} 1, & a = \operatorname{argmax}_{a' \in \mathcal{A}} Q^\star(s,a') \\ 0, & \text{otherwise} \end{cases} \tag{3.11}$$

If, on the other hand, only the optimal state value function is known, the optimal state-action value function can be derived from the state value function using Equation (3.12). After calculating the optimal state-action value function, one can use Equation (3.11) to get the optimal policy.

$$Q^\star(s,a) = \sum_{s' \in \mathcal{S}} \mathcal{P}^a_{ss'} \left(\mathcal{R}^a_{ss'} + \gamma V^\star(s')\right) \tag{3.12}$$

Solutions to RL problems are therefore ways of obtaining this optimal policy. Notice that the definition of an optimal policy presented above depends on the definition of value functions, which requires knowledge over the environment dynamics $\mathcal{P}^a_{ss'}$ and $\mathcal{R}^a_{ss'}$. These dynamics might not be known before-hand, meaning that an agent must learn them over time by interacting with the environment. This introduces two big families of methods for solving RL problems:

- *Model-based*: these methods use a known or learned model for the dynamics of the MDP. *Value iteration* and *policy iteration* are examples of such methods, detailed in Kaelbling et al. [1996]. For a more extensive review of these methods, the reader is referred to Moerland et al. [2020].

- *Model-free*: these methods do not rely on a known model of the MDP dynamics. Some notable examples are SARSA and Q-Learning, both described in Sutton and Barto [2018].

Model-free RL is a harder task when compared to model-based methods. The main reason is that the agent must also learn the environment over time. One of the biggest consequences of imperfect knowledge of the environment dynamics is an interesting trade-off in RL known as the *exploration-exploitation dilemma* Kaelbling et al. [1996]. On the one hand, the agent wants to increase its rewards, and therefore wants to choose the actions it *thinks* are optimal (exploitation). On the other hand, since the agent has imperfect knowledge of the environment, what it thinks to be the optimal actions to take may actually be sub-optimal. As such, it is often beneficial for the agent to choose what it perceives as sub-optimal actions, gathering new information about the environment (exploration), in order to make better decisions in the future.

As interesting as this trade-off might be, this work focuses on the simpler, model-based approach, using DDNs as the underlying model for the environment.

3.3 Partially observable environments

A partially observable environment has a subtle but impactful difference to a fully observable one. In a fully observable environment, the observations an agent receives fully characterize the state, and therefore the agent always knows in which state it is. However, that certainty is lost in a partially observable scenario, because observations only contain partial information about the state. Due to these differences, a partially observable environment is formalized using a *partially observable Markov decision process* (POMDP), instead of an MDP.

POMDPs are not so formally different from MDPs. They contain two more elements: the observations (omitted in the fully observable case because they represent the state) and a sensor model, similar to that of DBNs, relating observations, actions and states Littman [2009]. As such, a POMDP is a tuple $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \Omega, \mathcal{P}, \mathcal{R}, \mathcal{Z}, \gamma \rangle$ where:

- $\mathcal{S}$, $\mathcal{A}$, $\mathcal{P}$, $\mathcal{R}$ and γ have the same definition as in an MDP
- Ω is the set of possible observations o the agent can receive
- $\mathcal{Z} : \Omega \times \mathcal{S} \times \mathcal{A} \mapsto [0, 1]$ is the sensor model, representing a conditional distribution of observations given states and actions:

$$\mathcal{Z}_{sa}^{o} = P\left(O_{t+1} = o | S_{t+1} = s, A_t = a\right) \tag{3.13}$$

Solving a POMDP comes with the added challenge of the agent not knowing in what state it is while taking an action. Nevertheless, while the agent might not know exactly in which state it is, it can have some *beliefs* about in what states it could be. For example, in an environment with two states it could believe that it is 40% likely to be in state 1 and 60% likely to be in state 2. These beliefs can be formally defined as a probability distribution over states that represent the likelihood of being in a certain state for the agent, the *belief states* $b(s)$, also similar to these introduced for DBNs:

$$b(s) = P(S_t = s) \tag{3.14}$$

As the decision making process goes on, the agent will be receiving more and more observations about the environment and will be taking actions in the process. These observations and actions are important information for the agent to update its beliefs. Therefore, the definition of a belief state in Equation (3.14) is somewhat incomplete, because an agent's beliefs should depend on the history of observations $o_{1:t}$ and actions $a_{1:t-1}$ so far:

$$b(s) = P\left(S_t = s | O_{1:t} = o_{1:t}, A_{1:t-1} = a_{1:t-1}\right) \tag{3.15}$$

Belief states allow the agent to weight its decisions according to the likelihood of being in each of the possible states instead of considering only its current state. Due to the Markov Property, belief states in one time-step can be calculated from the belief states in the previous step in a recursive fashion Russell and Norvig [2009], similarly to the one represented in Equation (2.5) of Chapter 2, which introduces belief updating for DBNs. Consider that, at time-step t, an agent with a belief state $b(s)$ takes an action a, leading it to a new unknown state of the POMDP, which sends him an observation o. The agent can update its belief state $b(s)$ to a new belief state $b'(s')$ using the belief-update rule $\tau(b, a, o)$ Littman [2009]:

$$\tau\left(b, a, o\right)\left(s'\right) \equiv b'(s') \propto \mathcal{Z}^{o}_{s'a} \sum_{s \in \mathcal{S}} \mathcal{P}^{a}_{ss'} b(s) \tag{3.16}$$

This surprising results helps tremendously in the calculation of the probabilities of Equation (3.15). It states that the only thing needed for an agent to always know a belief state is to define it at the beginning of the decision making process. With an initial belief state defined, the agent can therefore take the best actions according to that belief state and use the new observations from the environment to update those beliefs.

In the partially observable case, the behaviour of an agent is captured by a policy π which is now redefined to be a distribution of actions given belief states:

$$\pi\left(a|b\right) = P\left(A_t = a | B_t = b\right) \tag{3.17}$$

Solving a POMDP also comes down to finding an optimal or near-optimal policy and, similarly to the previous section, defining such a policy is simpler with the help of value-functions. However, these value-functions also need a redefinition to depend not on a state, but on a belief state, yielding the expressions

Hauskrecht [2000]:

$$Q^{\pi}(b,a) = \sum_{s\in\mathcal{S}} b(s) \sum_{s'\in\mathcal{S}} \mathcal{P}^{a}_{ss'}\mathcal{R}^{a}_{ss'} + \gamma \sum_{o\in\Omega} V^{\pi}(\tau(b,a,o)) \sum_{s'\in\mathcal{S}} \mathcal{Z}^{o}_{s'a} \sum_{s\in\mathcal{S}} \mathcal{P}^{a}_{ss'} b(s) \tag{3.18}$$

$$V^{\pi}(b) = \sum_{a\in\mathcal{A}} \pi(a|b)\, Q^{\pi}(s,a) \tag{3.19}$$

Optimal value functions can be defined similarly to the MDP case. The optimal state value function is $V^{\star}(b) = \max_{\pi} V^{\pi}(b)$ and the optimal state-action value function is $Q^{\star}(b,a) = \max_{\pi} Q^{\pi}(b,a)$. Finally, solving a POMDP implies following an optimal policy $\pi^{\star}$:

$$\pi^{\star}(a|b) = \begin{cases} 1, & a = \text{argmax}_{a'\in\mathcal{A}} Q^{\star}(b,a) \\ 0, & \text{otherwise} \end{cases} \tag{3.20}$$

Unfortunately, it is not possible to construct a general algorithm to find exact solutions for any POMDP, since finding this general solution can be reduced to solving the halting problem Madani et al. [2003]. In any case, other methods still exist to find approximate, near-optimal policies, such as the one presented in the next section.

To connect this section with DDNs, introduced in Section 2.5, it is useful to understand how they can model a POMDP. To show this, notice from the definition of a POMDP $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \Omega, \mathcal{P}, \mathcal{R}, \mathcal{Z}, \gamma \rangle$ that all of its elements are present in a DDN, except for the discount factor γ, which must be separately defined:

- The sets $\mathcal{S}$, $\mathcal{A}$ and Ω are represented by all the possible values the RVs S_t, A_t and O_t can take, respectively, which are encoded in their CPTs.

- The transition function $\mathcal{P}$, whose values can be obtained by inferring $P(S_{t+1}|S_t, A_t)$, is encoded in the CPT of state RVs S_{t+1}.

- The reward function $\mathcal{R}$, whose values can be inferred from $\mathbb{E}(R_{t+1}|S_t, A_t)$ can be obtained by the CPT of the reward RVs R_{t+1}, which holds the probability distribution $P(R_{t+1}|S_t, A_t)$.

- The sensor model $\mathcal{Z}$, whose values can be inferred from $P(O_{t+1}|S_{t+1}, A_t)$, is encoded in the CPT of the observation RVs O_{t+1}.

Given that all of the elements of a POMDP are captured in a DDN, then a DDN can model partially observable environments.

Finally, Equation (3.18) can be re-written in a much more compact manner, using general probability distributions that can be extracted from DDNs, instead of the specific probability distributions that constitute the POMDP:

$$
\begin{aligned}
Q^{\star}(b_t, a_t) &= \mathbb{E}\left(r_{t+1} | b_t, a_t\right) + \gamma \sum_{o_{t+1}} P(o_{t+1} | b_t, a_t) \max_{a_{t+1} \in \mathcal{A}} Q^{\star}(b_{t+1}, a_{t+1}) \\
b_{t+1}(s_{t+1}) &= P(s_{t+1} | o_{t+1}, a_t, b_t)
\end{aligned}
\tag{3.21}
$$

3.4 POMDP lookahead

Although Section 3.3 introduces and formally defines POMDPs, no algorithm has been shown for this purpose. This section describes one algorithm for approximately solving model-based, partially observable RL problems using a *lookahead* method.

The lookahead algorithm does not return a policy for the agent to follow, but rather calculates a near optimal action for a particular time-step. Naturally, by applying this algorithm sequentially for each time-step, it is possible to make rational sequences of decisions.

The algorithm entails building a lookahead tree as in Figure 9: starting with the current belief-state as the root (belief) node of the tree, enumerate every possible action the agent can take to create the child (observation) nodes. Then, for every observation node, enumerate all possible observations the agent can receive as a response to create this node's child (belief) nodes [3]. This procedure is repeated until the desired horizon (or depth) H of the tree is reached.

1. Create the root (belief) node of the tree, holding the current belief state of the agent.

2. For each leaf belief node, enumerate every possible action the agent can take. For each action, create a (observation) node and make it a child of the previous belief node.

3. If the desired horizon H has already been reached[4], stop. Else, go to the next step.

4. For each leaf observation node, create a new belief node for each possible observation and make them children of this observation node.

5. At each leaf belief node, perform the belief update of Equation (2.6) to calculate its belief state. To do this, use this node's observation and the previous action and belief state that lead to this branch of the tree.

[3] Recall from the belief-update rule of Equation (3.16) that a belief-state, action and observation can be used to update a belief-state. This belief-update should be performed at every child belief node to calculate its respective belief-state.

[4] For an horizon H to be reached, the tree should be of depth $2H$.

6. Go back to step 2.

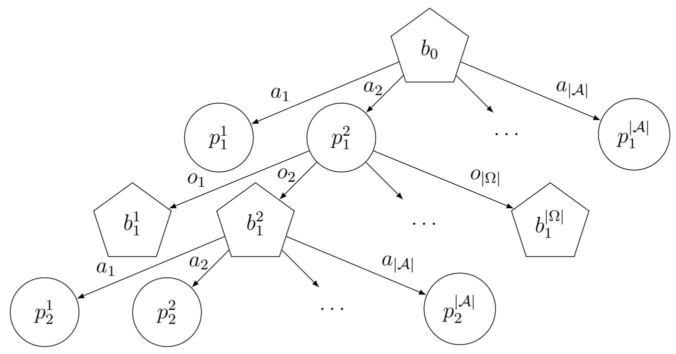


Figure 9: A two-step ($H = 2$) lookahead tree. Belief nodes are pentagon-ally shaped and observation nodes are circle shaped. For diagram simplicity, only one of the observation nodes and one of its belief nodes is expanded.

By following the aforementioned steps, the lookahead tree is finally complete. To perform the belief update along the tree, knowing the POMDP is a requirement. As such, this algorithm requires the use of a model for the POMDP, such as a DDN, hence being a model-based algorithm.

After the lookahead tree has been built, all that remains to be done is the extraction of an action. This can be done using the following steps:

1. Calculate the expected reward $\mathbb{E}\left(R_{t+H}|b_{t+H-1}, a_{t+H-1}\right)$[5] for each leaf observation node. The belief-state b_{t+H-1} refers to the belief-state of the leaf node's parent and the action a_{t+H-1} refers to the action of the current observation node. Assume this is an approximate Q value for the observation node, denoted by $Q^{\mathcal{L}}(b_{t+H-1}, a_{t+H-1})$ [6].

2. If the parent node of the nodes whose values were just calculated, go to step 6. Else, go to the next step.

3. Calculate the value of the parent belief node given the value of its children observation nodes, propagating the values upwards in the tree, using the following expression:

$$V^{\mathcal{L}}(b_t) = \max_{a_t \in \mathcal{A}} Q^{\mathcal{L}}(b_t, a_t) \tag{3.22}$$

4. Calculate the value of the parent observation node given the value of its children belief nodes,

[5] Here, the reward is considered to be dependent only on one state and action, the same assumption used for the definition of DDNs in Section 2.5.

[6] The superscript $\mathcal{L}$ states that the value function is calculated using the lookahead algorithm. This makes it easy to understand from the notation what method is being used in these calculations and is used throughout the remainder of this text.

propagating the values upwards in the tree, using the following expression:

$$Q^{\mathcal{L}}(b_t, a_t) = \mathbb{E}\left(R_{t+1}|b_t, a_t\right) + \gamma \sum_{o_{t+1}\in\Omega} P\left(o_{t+1}|b_t, a_t\right) V^{\mathcal{L}}(\tau(b_t, a_t, o_{t+1})) \tag{3.23}$$

5. Go back to step 2.

6. Using the values calculated for the child observation nodes of the root belief node, return the action $a^{\mathcal{L}}$ given by the following equation:

$$a^{\mathcal{L}} = \operatorname*{argmax}_{a_t\in\mathcal{A}} Q^{\mathcal{L}}(b_t, a_t) \tag{3.24}$$

The above procedure fully describes the lookahead algorithm. To summarize, two things are being done to calculate a good action for the agent:

- Approximations of optimal value functions are propagated along the tree by performing a max operation on belief nodes and taking a weighted average on observation nodes.

- The action with the highest (approximate) value is returned.

It is important to realize that the algorithm does not learn or get better over time, as in dynamic programming methods such as value or policy iteration Kaelbling et al. [1996]. It does not use its current knowledge to update knowledge on future time-steps, but instead uses brute force to select a near optimal action at any given time. The outputs of the algorithm naturally still vary over time, but that is due to a change of the belief state of the agent. This gives it both a disadvantage and an advantage relative to learning methods. On the one hand, if the algorithm performs poorly on a particular situation, it won't learn from its mistakes and adapt to it, but rather keep struggling in that particular situation. On the other hand, being an algorithm that does not learn over time, it comes with the benefit that it can get good results right after being deployed, as it does not require previous training.

Suppose that a DDN is used to model the POMDP, hence using inference to calculate every probability distribution. If these probability distributions were calculated using exact inference methods, the lookahead algorithm's error would only be due to its finite horizon H, because every reward after that horizon is not accounted for to choose the actions of the agent. However, using approximate inference yields another source of error, since every probability distribution is approximated via sampling and won't be calculated exactly. Chapter 6 performs both an analysis on the complexity of this algorithm and provides conditions under which this approximation error can be bounded.

3.5 Summary

In this Chapter, reinforcement learning was described as the interplay between a decision maker, the agent, and an environment, the world the decision maker lives in.

It is shown how environments can be formalized with either MDPs, in cases where the environment is fully observable, or POMDPs, for partially observable environments. Additionally, the behaviour of a decision maker can be mathematically described by a policy, a probability distribution of actions over states. To solve a RL problem is to find an optimal policy or optimal actions for an agent to take, such that its Expected Utility is maximized.

Furthermore, it was also shown in this Chapter that DDNs can be used to model both MDPs and POMDPs. They can be used together with a lookahead algorithm to build a model-based approach to solve partially observable RL problems.

Chapter 4

Quantum computing

Quantum computing presents itself as a new way of performing computations, vastly different to the way conventional classical computers work. For a variety of computational processes, quantum computers are theorized to be more efficient than classical ones.

As a way of exploring these algorithmic advantages in the domain of decision making problems, this Chapter is devoted to the introduction of Quantum Computation, presenting basic concepts of quantum computing with quantum circuits. This Chapter also explains Grover's algorithm and how it can be generalized to the amplitude amplification algorithm. Lastly, it covers how to perform inference via rejection sampling on a BN with a quantum computer, with a quadratic speedup over classical rejection sampling.

4.1 Context

In the last few years, quantum computing emerged as a very active area of research and has even been shown to be more efficient than classical computers for certain problems in idealized conditions. As a concrete example, calculating a discrete Fourier transform, widely used in a variety of engineering applications, has an exponential speedup through the quantum Fourier transform (QFT) algorithm Nielsen and Chuang [2011]. Another exponential speedup problem is Shor's Algorithm for prime number factorization Shor [1999], which has the potential to break cryptography protocols such as RSA encryption. Yet another quantum algorithm, and one of extreme importance to this work, is Grover's algorithm Grover [1996], which provides a quadratic speedup for unstructured search problems.

Despite this, there are still many problems associated with practical implementations of quantum algorithms, especially on large scale applications, with the most notable challenges being the scalability of quantum devices, the fragility and noise in both quantum gates and quantum states that affect the outcome of the computations and the fast decoherence of both quantum gates and quantum states that currently prevent the use of deep quantum circuits Everitt [2005]. These challenges are the main reason why today's quantum computer still use small numbers of qubits which are still noisy, and likely to be for yet some time. These devices are called *noisy intermediate-scale quantum* (NISQ) devices, and more and more algorithms are being developed to target this specific range of quantum computing devices, to try and make it possible to have some practical near-term applications for this new technology.

There have also been advancements in quantum computing in the specific context of learning, one of the most relevant topics to this dissertation. In Huang et al. [2021], the authors leverage the use of quantum information to find an exponential speedup in three static learning tasks: acquiring information about a large number of non-commuting observables, learning the symmetry class of a physical evolution operator and learning about the principal component of a noisy state. In the context of active learning, the authors of Saggio et al. [2021] found an experimental quadratic speedup in using a hybrid quantum-classical communication channel between the agent and the environment. A more theoretical result is that of Paparo et al. [2014], which shows that there is a quadratic speedup in projective simulation agents' decision making processes by using quantum hardware, enabling decisions to be computed faster. All of these results are very interesting, but have a very narrow relation to this work: they are all applied to active learning.

The two following works are much more similar to this one, focusing on quantum techniques to solve RL problems. They differ in their environment and algorithm choice, but the techniques used are very similar. Firstly, by using generative models (as described in Kakade [2003]), a category of models DDNs fall into, the authors of Wang et al. [2021] developed quantum algorithms, inspired in the value iteration RL algorithm, with a quadratic advantage for solving MDPs, as opposed to this dissertation which deals with POMDPs. Secondly, the authors of Wiedemann et al. [2022] developed a quantum policy iteration algorithm (the quantum analog to the classical policy iteration algorithm for solving MDPs) that leverages the amplitude amplification algorithm, similarly to this work, to yield a quadratic speedup in its sample complexity. This is in contrast to the use of amplitude amplification in this work, which keeps the sample complexity unchanged and only provides a speedup in the computational complexity.

On the topic of BNs, Moreira and Wichert [2016] propose a quantum analog of BNs that can handle quantum information, proving very useful to model human decision making, especially when it violates the laws of classical probability. Moreover, while this work focuses on computing complex decisions (as defined in Chapter 2) using DDNs, the authors of de Oliveira and Barbosa [2021] present a fully quantum algorithm to compute simple decisions in a static environment using decision networks (a static variant of DDNs), while also proving its computational advantage, which is dependent on the characteristics of the problem. Perhaps the most relevant result for this work is the quadratic speedup of the rejection sampling inference algorithm for BNs when performed in a quantum computer Low et al. [2014]. Unfortunately, as reported in Borujeni et al. [2020], quantum rejection sampling is not NISQ friendly, as its application in modern quantum computers still yield very noisy results even for small BNs, such that its pratical use will probably have to be reserved to more reliable generations of quantum computing devices. Moreover, the quantum rejection sampling algorithm was studied further by Borujeni and Nannapaneni [2021], showing that this algorithm can be applied to time dynamic systems by using DBNs instead of regular BNs. One of the main aims of this thesis is to extend this further by applying quantum rejection sampling to DDNs, making it possible to use them to solve RL problems.

4.2 Quantum computing with quantum circuits

4.2.1 Single qubit systems

There are many different models of quantum computation, such as quantum annealing Finnila et al. [1994], measurement based quantum computing Briegel et al. [2009] and the circuit model of quantum computation Nielsen and Chuang [2011]. Among all of these models, the one this work focuses on the circuit model of quantum computing and does not address any of the other alternatives.

In classical computing, the unit of information is the *bit*, which can be either 0 or 1 at any given time. Conversely, in quantum computing, the unit of information is the *qubit*, the quantum analog of a bit. The big difference between the two is that a qubit can be in a linear combination of 0 and 1, with complex coefficients representing *probability amplitudes*, which, moreover, can *interfere* with each other. A general single qubit quantum state has the following form:

$$|\psi\rangle = \alpha_0 |0\rangle + \alpha_1 |1\rangle \tag{4.1}$$

where $\alpha_i \in \mathbb{C}$ and, if $|\psi\rangle$ is a *normalized* state, $|\alpha_0|^2 + |\alpha_1|^2 = 1$.

Let $|0\rangle = (1,0)^T$ and $|1\rangle = (0,1)^T$ be basis vectors (the *computational basis*). The quantum state in the Equation (4.1) can also be rewritten as $|\psi\rangle = (\alpha_0, \alpha_1)^T$, but Dirac notation will be preferred throughout this text. The complex conjugate of a quantum state is $\langle\psi| = |\psi\rangle^\dagger$:

$$\langle\psi| = \alpha_0^* \langle 0| + \alpha_1^* \langle 1| \tag{4.2}$$

where α_i^* are the complex conjugates of α_i.

Such that the inner product between two quantum states $|\psi\rangle$ and $|\varphi\rangle$ is defined as:

$$\langle\varphi|\psi\rangle = (\beta_0^* \langle 0| + \beta_1^* \langle 1|)(\alpha_0 |0\rangle + \alpha_1 |1\rangle) = \beta_0^* \alpha_0 + \beta_1^* \alpha_1 \tag{4.3}$$

The inner product plays a crucial role in the description of quantum states, due to the following property: information about a quantum state can only be obtained through *measurements*. Measurements in quantum mechanics are called *observables* and are represented by a quantum *operator* that acts on quantum states. When a measurement is performed on $|\psi\rangle = \alpha_0 |0\rangle + \alpha_1 |1\rangle$, the state collapses to one of the *eigenstates* of the corresponding observable with some probability. In quantum circuits, the eigenstates for a measurement are typically the computational basis $\{|0\rangle, |1\rangle\}$. The probability that $|\psi\rangle$ collapses to state $|i\rangle, i \in \{0,1\}$ is given by the expression $|\langle i|\psi\rangle|^2 = |\alpha_i|^2$.

Another crucial part of quantum circuits is its elementary operations, denoted by *quantum gates*. Quantum gates are responsible for changing quantum states, allowing the circuits to perform quantum computations. One of the most notable examples is the *Hadamard* gate H, used to create uniform superpositions in qubits:

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \tag{4.4}$$

The Hadamard gate, when applied to the computational basis, has the following effect:

$$\begin{aligned} H|0\rangle &= \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) = |+\rangle \\ H|1\rangle &= \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) = |-\rangle \end{aligned} \tag{4.5}$$

Four other very important gates are the X, Y and Z Pauli gates and the identity $\mathbb{1}$:

$$\mathbb{1} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \quad X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \quad Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} \quad Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \tag{4.6}$$

These Pauli gates can also be used to generate *rotation gates* $R_q(\theta) \equiv e^{i\frac{\theta}{2}q}, q \in \{X, Y, Z\}$. Just as their name states, these quantum gates apply rotations to a quantum state. But how do rotations on quantum states work?

One other possible way to represent a generic single qubit state is using the *Bloch sphere* (see Figure 10). The bloch sphere is a sphere with three axis, allowing a generic single qubit state to be mathematically written as the following:

$$|\psi\rangle = \cos\frac{\theta}{2}|0\rangle + e^{i\phi}\sin\frac{\theta}{2}|1\rangle \tag{4.7}$$

Where the angles θ and ϕ are as represented in Figure 10.

The R_X, R_Y and R_Z rotation gates perform rotations on a quantum state ψ over the x, y and z axis of the Bloch sphere, respectively:

$$R_X(\theta) = \begin{pmatrix} \cos\frac{\theta}{2} & -i\sin\frac{\theta}{2} \\ -i\sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{pmatrix} \quad R_Y(\theta) = \begin{pmatrix} \cos\frac{\theta}{2} & -\sin\frac{\theta}{2} \\ \sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{pmatrix} \quad R_Z(\theta) = \begin{pmatrix} e^{-i\frac{\theta}{2}} & 0 \\ 0 & e^{i\frac{\theta}{2}} \end{pmatrix} \tag{4.8}$$

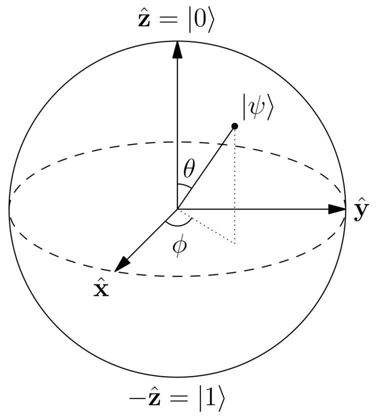


Figure 10: Representation of the qubit in Equation (4.7) in the Bloch Sphere.

The *phase-shift* gate $P(\theta)$ introduces a phase in a qubit whenever it is in state $|1\rangle$, which will be useful do define a *universal set* of quantum gates later on:

$$P(\theta) = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\theta} \end{pmatrix} \tag{4.9}$$

There are many other possible quantum gates (see Nielsen and Chuang [2011]). The aforementioned ones are just a few of the most notable and useful for the purposes of this Chapter, but the key idea is that every quantum gate changes the quantum state it is applied to (with the exception of the identity gate $\mathbb{1}$ which maps a state to itself $\mathbb{1}|\psi\rangle = |\psi\rangle$). Any quantum algorithm is composed of quantum gates that change the initial state in a meaningful way in order to get the correct final state for the task at hand, such that measuring that final state yields the expected outcome of the algorithm.

Single qubit systems are, however, limited and do not explore the full advantages and nuances of quantum computing. This leads to the next subsection, where multi-qubit quantum circuits are considered.

4.2.2 Multi-qubit systems

In single qubit systems, a quantum state lives in a two dimensional complex space $\mathbb{C}^2$. For a multi-qubit system with N qubits, this space is expanded to $\mathbb{C}^{2^N}$. Considering the set $\mathbb{B} = \{0, 1\}$, a general N qubit quantum state has the form:

$$|\psi\rangle = \sum_{x \in \mathbb{B}^N} \alpha_x |x\rangle \tag{4.10}$$

Once again, ensuring that the probability amplitudes obey the relationship $\sum_{x\in\mathbb{B}^N} |\alpha_x|^2 = 1$. A multi-qubit system can be a composition of single qubit ones. Constructing a multi-qubit quantum state $|\psi\rangle$ from single qubit states $|\psi_i\rangle$ can be done using the tensor product (represented by the mathematical symbol $\otimes$):

$$|\psi\rangle = |\psi_1\rangle \otimes |\psi_2\rangle \otimes \cdots \otimes |\psi_N\rangle = \bigotimes_{i=1}^{N} |\psi_i\rangle \tag{4.11}$$

For compactness, a tensor product $|\psi_1\rangle \otimes \cdots \otimes |\psi_N\rangle$ will be shorthanded as $|\psi_1 \ldots \psi_N\rangle$. In a similar fashion, a multi-qubit quantum gate can be composed of single qubit gates. As an example, a multi-qubit Hadamard gate $H^{\otimes N}$, to make uniform multi-qubit superpositions, is just the tensor product of single-qubit Hadamard gates. Considering the quantum state $|\psi\rangle$ in Equation (4.11):

$$H^{\otimes N} |\psi\rangle = \bigotimes_{i=1}^{N} H |\psi_i\rangle \tag{4.12}$$

Nevertheless, not all multi-qubit systems can be described in terms of tensor products of single-qubit ones. This happens because sometimes, a quantum system can only be described as a whole, and not by a description of each of its parts. When this is the case, a quantum state is said to be *entangled*. To illustrate this point, let's introduce the first two-qubit quantum gate, the $CNOT$:

$$CNOT \equiv |0\rangle\langle 0| \otimes \mathbb{1} + |1\rangle\langle 1| \otimes X = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \tag{4.13}$$

The $CNOT$ is a *controlled* gate, that flips the second qubit only if the first qubit is in the state $|1\rangle$:

$$\begin{aligned} CNOT\,|00\rangle &= |00\rangle\,, \quad CNOT\,|01\rangle = |01\rangle \\ CNOT\,|10\rangle &= |11\rangle\,, \quad CNOT\,|11\rangle = |10\rangle \end{aligned} \tag{4.14}$$

Now consider the application of a $CNOT$ to a 2 qubit quantum state where the first qubit is in uniform superposition:

$$CNOT\,(H \otimes \mathbb{1})\,|00\rangle = CNOT \frac{1}{\sqrt{2}}(|00\rangle + |10\rangle) = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle) = |\Psi^+\rangle \tag{4.15}$$

The final state $|\Psi^+\rangle$ can not be written as a tensor product of single qubit states. To prove this, consider the tensor product of two general single qubit quantum states:

$$\begin{aligned} |\psi\rangle \otimes |\phi\rangle &= (\alpha_0|0\rangle + \alpha_1|1\rangle) \otimes (\beta_0|0\rangle + \beta_1|1\rangle) \\ &= \alpha_0\beta_0|00\rangle + \alpha_0\beta_1|01\rangle + \alpha_1\beta_0|10\rangle + \alpha_1\beta_1|11\rangle \end{aligned} \tag{4.16}$$

Notice that for the tensor product of Equation (4.16) to equal the quantum state of Equation (4.15), then $(\alpha_0 = 0 \vee \beta_1 = 0) \wedge (\alpha_1 = 0 \vee \beta_0 = 0)$. Let's consider all of these four possibilities separately:

- If $\alpha_0 = 0$ and $\alpha_1 = 0$, then $|\psi\rangle \otimes |\phi\rangle$ has probability amplitude 0 for any of the four states $|00\rangle, |01\rangle, |10\rangle, |11\rangle$. The same is true if $\beta_0 = 0$ and $\beta_1 = 0$.

- If $\alpha_0 = 0$ and $\beta_0 = 0$, then $|\psi\rangle \otimes |\phi\rangle = \alpha_1\beta_1 |11\rangle \neq |\Psi^+\rangle$, independently of the choice of α_1 and β_1.

- If $\alpha_1 = 0$ and $\beta_1 = 0$, then $|\psi\rangle \otimes |\phi\rangle = \alpha_0\beta_0 |00\rangle \neq |\Psi^+\rangle$, independently of the choice of α_0 and β_0.

Therefore, it can be concluded that the quantum state of Equation (4.15) can not be written as the tensor product of two single qubit quantum states, and so it is called an *entangled* state.

Having introduced the $CNOT$ gate, it is an important fact that, together with $R_X(\theta)$, $R_Y(\theta)$, $R_Z(\theta)$ and $P(\theta)$, these gates form a *universal set* of quantum gates Williams [2011], meaning that any other quantum operation can be decomposed into these five ones.

Quantum gates can also be composed in various different ways to create more complex operations, denoted as *quantum operators*. As Solovay-Kitaev's Theorem Nielsen and Chuang [2011] states, any quantum operation can be efficiently approximated, with arbitrary accuracy, by using specific sets of quantum gates, namely the universal sets. These quantum gates are considered as elementary operations, while quantum operators are an abstraction of the underlying quantum gates that compose them and have a higher computational complexity. This is crucial to define the computational complexity of quantum circuits: the elementary operations are quantum gates, and the complexity of a quantum circuit counts the total number of quantum gates that decompose that quantum circuit.

4.3 Grover's algorithm and amplitude amplification

The *amplitude amplification* algorithm Brassard et al. [2002], is a generalization of the previously mentioned Grover's algorithm Grover [1996] for unstructured search problems. Consider the function $f : \mathbb{B}^N \mapsto \mathbb{B}$ that separates the set of bit-strings $\mathbb{B}^N$ into two sets: the *good elements* $\mathcal{X}$, for which $f(x) = 1, \forall x \in \mathcal{X}$, and the bad elements $\overline{\mathcal{X}}$ for which $f(x) = 0, \forall x \in \overline{\mathcal{X}}$. Both Grover and amplitude amplification algorithms solve the task of finding the good elements $\mathcal{X}$ in a probabilistic manner, only differing in the initial distribution of its inputs.

Classically, the worst case for solving this problem for an N bit-string has a complexity of $\mathcal{O}\left(2^N\right)$, representing the total number of elements 2^N to be searched over. Nonetheless, Grover's algorithm accomplishes the same task with a complexity of $\mathcal{O}\left(\sqrt{2^N}\right)$, yielding a quadratic speedup over the best classical algorithms. This is accomplished by *amplifying* the probability amplitudes of the qubits representing good elements, while decreasing the amplitudes of the rest.

Grover's algorithm consists in the successive application of *Grover's operator* G:

$$G = H^{\otimes N} \mathcal{S}_0 H^{\otimes N} \mathcal{S}_f \tag{4.17}$$

The number of applications of Grover's operator depends on the number of good states $|\mathcal{X}|$, as detailed later in this section. Moreover, the first application of this operator should be performed on a uniform superposition state $|\psi\rangle = \frac{1}{\sqrt{2^N}} \sum_{x \in \mathbb{B}^N} |x\rangle$. To better understand Grover's operator, it is important to take a closer look at the three distinct operators that compose it:

1. The oracle $\mathcal{S}_f$ that flips the phase of the good states $\mathcal{X}$ and does so by using the Boolean function f defined above:
$$\mathcal{S}_f |x\rangle = (-1)^{f(x)} |x\rangle, \forall x \in \mathbb{B}^N \tag{4.18}$$

2. The multi-qubit Hadamard $H^{\otimes N}$ that creates an N qubit uniform superposition.

3. The reflection operator $\mathcal{S}_0 = 2|0\rangle^{\otimes N} \langle 0|^{\otimes N} - \mathbb{1}$ that flips the phase of all non-zero states:
$$\mathcal{S}_0 |x\rangle = \begin{cases} |x\rangle, & x = 00\dots0 \\ -|x\rangle, & x \neq 00\dots0 \end{cases}, \forall x \in \mathbb{B}^N \tag{4.19}$$

The oracle operator varies according to the Boolean function f and therefore changes according to the problem at hands. The description of this operator presented above is therefore sufficient for the purposes of this section and is revisited in the next section for the application of this algorithm to a specific problem. For now, the main focus is shifted to the remaining gates in Grover's operator, shorthanded as $\mathcal{W} = H^{\otimes N} \mathcal{S}_0 H^{\otimes N}$, to build some additional insight on how Grover's algorithm works. This operator can be re-expressed using the initial uniform superposition state $|\psi\rangle = \frac{1}{\sqrt{2^N}} \sum_{x \in \mathbb{B}^N} |x\rangle$:

$$\mathcal{W} = H^{\otimes N} \left(2|0\rangle^{\otimes N} \langle 0|^{\otimes N} - \mathbb{1}\right) H^{\otimes N} = 2|\psi\rangle\langle\psi| - \mathbb{1} \tag{4.20}$$

And now consider its effect on an N qubit state, as expressed by Equation (4.10):

$$\begin{aligned}
\mathcal{W}\sum_{x\in\mathbb{B}^N}\alpha_x|x\rangle &= (2|\psi\rangle\langle\psi| - \mathbb{1})\left(\sum_{x\in\mathbb{B}^N}\alpha_x|x\rangle\right) \\
&= \frac{2}{\sqrt{2^N}}|\psi\rangle\sum_{x\in\mathbb{B}^N}\sum_{y\in\mathbb{B}^N}\alpha_x\langle y|x\rangle - \sum_{x\in\mathbb{B}^N}\alpha_x|x\rangle \\
&= \frac{2}{\sqrt{2^N}}|\psi\rangle\sum_{x\in\mathbb{B}^N}\sum_{y\in\mathbb{B}^N}\alpha_x\delta_{xy} - \sum_{x\in\mathbb{B}^N}\alpha_x|x\rangle \\
&= \frac{2}{\sqrt{2^N}}|\psi\rangle\sum_{x\in\mathbb{B}^N}\alpha_x - \sum_{x\in\mathbb{B}^N}\alpha_x|x\rangle \\
&= \sum_{x\in\mathbb{B}^N}2\overline{\alpha}|x\rangle - \sum_{x\in\mathbb{B}^N}\alpha_x|x\rangle \\
&= \sum_{x\in\mathbb{B}^N}(2\overline{\alpha} - \alpha_x)|x\rangle
\end{aligned}$$

Hence:

$$\mathcal{W}\sum_{x\in\mathbb{B}^N}\alpha_x|x\rangle = \sum_{x\in\mathbb{B}^N}(2\overline{\alpha} - \alpha_x)|x\rangle \tag{4.21}$$

Where $\overline{\alpha} = \frac{1}{2^N}\sum_{x\in\mathbb{B}^N}\alpha_x$ is the *mean amplitude* of the initial state. Equation (4.21) asserts that the operator $\mathcal{W}$ inverts every probability amplitude α_x of the initial state about the *mean* probability amplitude $\overline{\alpha}$. With this in mind, it can be said that the Grover operator performs two operations:

- $\mathcal{S}_f$ flips the phase of the good states,
- $\mathcal{W}$ then performs an inversion about the mean on the probability amplitudes.

Consider figure 11 that represents the probability amplitudes of a quantum state when applied the Grover operator. In this picture, states are represented in decimal instead of binary notation for compactness and there is only one good state $\mathcal{X} = \{1\}$. For this particular example, only one application of Grover's operator is needed to find the solution, as the only solution is guaranteed to be the result of a measurement on the final state, as represented in the sub-figure on the right, since it has maximum probability amplitude associated.

The final question that remains unanswered is how many times Grover's operator needs to be applied. To understand this question, it is useful to consider the initial state $|\psi\rangle$ to which the first Grover operator is applied to, and express it as a superposition of a solution state $|\mathcal{X}\rangle$ and remainder state $|\overline{\mathcal{X}}\rangle$:

$$|\psi\rangle = \frac{1}{\sqrt{2^N}}\sum_{x\in\mathbb{B}^N}|x\rangle = \sqrt{\frac{|\mathcal{X}|}{2^N}}|\mathcal{X}\rangle + \sqrt{\frac{2^N - |\mathcal{X}|}{2^N}}|\overline{\mathcal{X}}\rangle \tag{4.22}$$

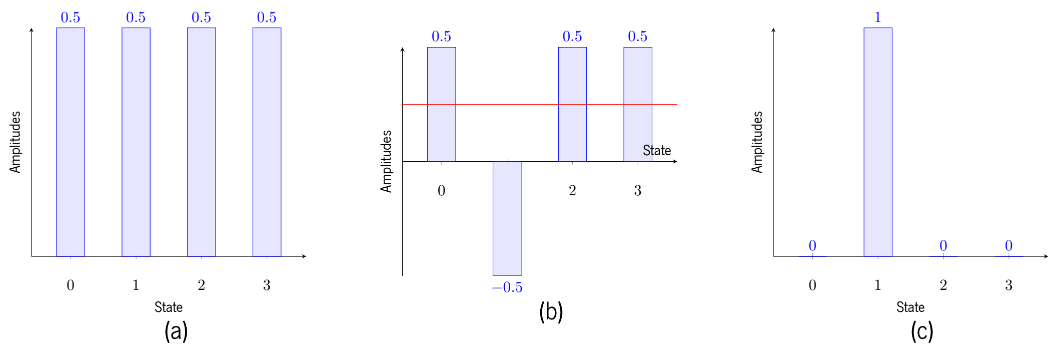


Figure 11: Representation of the probability amplitudes of a quantum state during the application of the Grover Operator. a) represents the probability amplitudes of the initial state in a uniform superposition, b) represents the probability amplitudes of the initial state after the phase flip of the good state $|1\rangle$, and the red line represents the mean amplitude $\overline{\alpha} = 0.25$. Finally, c) represents the probability amplitudes after the inversion about the mean.

where the solution state $|\mathcal{X}\rangle$ is a uniform superposition of every state that is a solution to the problem and the remainder state $|\overline{\mathcal{X}}\rangle$ a uniform superposition of the remaining states:

$$|\mathcal{X}\rangle = \frac{1}{\sqrt{|\mathcal{X}|}} \sum_{x \in \mathcal{X}} |x\rangle \, , \; |\overline{\mathcal{X}}\rangle = \frac{1}{\sqrt{2^N - |\mathcal{X}|}} \sum_{x \in \overline{\mathcal{X}}} |x\rangle \, , \tag{4.23}$$

As such, the probability amplitudes in Equation (4.22) can be expressed in terms of a rotation angle θ:

$$|\psi\rangle = \sin\frac{\theta}{2} |\mathcal{X}\rangle + \cos\frac{\theta}{2} |\overline{\mathcal{X}}\rangle \, , \; \theta = 2 \arcsin \sqrt{\frac{|\mathcal{X}|}{2^N}} \tag{4.24}$$

It just so happens that the Grover operator G performs a θ angle rotation on state $|\psi\rangle$ Nielsen and Chuang [2011], giving it a geometrical interpretation, such that:

$$G |\psi\rangle = \sin\left(\frac{\theta}{2} + \theta\right) |\mathcal{X}\rangle + \cos\left(\frac{\theta}{2} + \theta\right) |\overline{\mathcal{X}}\rangle \tag{4.25}$$

And therefore, k applications of Grover's operator rotate state $|\psi\rangle$ by an angle of $k\theta$:

$$G^k |\psi\rangle = \sin\left(\frac{2k+1}{2}\theta\right) |\mathcal{X}\rangle + \cos\left(\frac{2k+1}{2}\theta\right) |\overline{\mathcal{X}}\rangle \tag{4.26}$$

The number of times k the Grover operator has to be applied is such that it rotates state $|\psi\rangle$ as close as possible to the solution state $|\mathcal{X}\rangle$ (see figure 12).

In view of the fact that states $|\mathcal{X}\rangle$ and $|\overline{\mathcal{X}}\rangle$ are orthogonal, the angle between the solution state and $|\psi\rangle$ is $\frac{\pi-\theta}{2}$, and since the Grover operator rotates $|\psi\rangle$ by an angle of θ, the number of times k it has to be applied is:

$$k = \text{round}\left(\frac{\pi - \theta}{2\theta}\right) = \text{round}\left(\sqrt{\frac{2^N}{|\mathcal{X}|}} \arccos\left(\sqrt{\frac{|\mathcal{X}|}{2^N}}\right)\right) \tag{4.27}$$

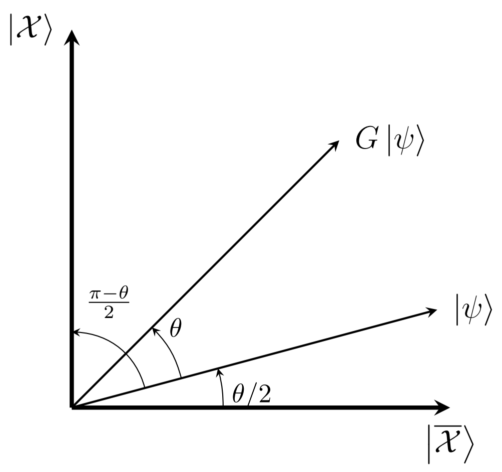


Figure 12: Representation of the rotational effect of the Grover operator G.

As such, Grover's algorithm amounts to applying the quantum operator $G = H^{\otimes N}\mathcal{S}_0 H^{\otimes N}\mathcal{S}_f$ k times as in Equation (4.27), followed by a measurement to retrieve a good state.

Grover's algorithm, considers a uniform distribution in the initial quantum state $|\psi\rangle = \frac{1}{\sqrt{2^N}}\sum_{x\in\mathbb{B}^N}|x\rangle$, resulting from the Hadamard gate applied to every qubit $H^{\otimes N}$. What if instead we want the search to be performed on a general, possibly non-uniform, distribution p? This is the idea of amplitude amplification: generalize Grover's algorithm to arbitrary distributions by substituting the N qubit Hadamards $H^{\otimes N}$ in the Grover operator by an arbitrary state preparation circuit $\mathcal{B}$. If the desired distribution for the initial quantum state is p, then the state preparation circuit $\mathcal{B}$ performs the following operation:

$$\mathcal{B}\,|0\rangle^{\otimes N} = \sum_{x\in\mathbb{B}^N}\sqrt{p(x)}\,|x\rangle \tag{4.28}$$

Therefore, an operator for the amplitude amplification algorithm can be defined by:

$$G = \mathcal{B}\mathcal{S}_0\mathcal{B}^\dagger\mathcal{S}_f \tag{4.29}$$

This generalization allows the initial state to be prepared by an arbitrary operator $\mathcal{B}$, removing the restriction of a uniform superposition imposed in Grover's algorithm due to the use of the Hadamard gate $H^{\otimes N}$. A consequence of this is a wider range of applications for the algorithm, as now more complex distributions aside from uniform ones can be searched over.

Non-withstanding, calculating the optimal number of iterations k using Equation (4.27) requires knowing before-hand how many solutions $|\mathcal{X}|$ there are to the problem. In some cases, however, this just isn't possible. Thankfully, using the QSearch algorithm in Brassard et al. [2002], as in Algorithm 3, it is still possible to construct a classical-quantum algorithm to find solutions with a quadratic speedup on average, even when $|\mathcal{X}|$ is unknown:

Algorithm 3 QSearch.

Require:

$\mathcal{B}$, a state preparation quantum circuit with N qubits;

G, the grover operator for the problem at hand;

f, the boolean function that outputs the ”goodness” of a state

```
$l \leftarrow 0$
$c \leftarrow C$                                                    ▷ $C \in (1, 2)$
go $\leftarrow 1$
while go $\neq 0$ do
    $l \leftarrow l + 1$
    $m \leftarrow \lceil c^l \rceil$
    $\mathcal{Q} \leftarrow \mathcal{B} |0\rangle^{\otimes N}$
    $r \leftarrow$ measure $\mathcal{Q}$
    go $\leftarrow 1 - f(r)$
    if go $\neq 0$ then
        $k \leftarrow$ sample_uniform_distribution$(1, m)$    ▷ $k$ is a random integer such that $1 \leq k \leq m$
        $r \leftarrow$ measure $G^k \mathcal{B} |0\rangle^{\otimes N}$
        go $\leftarrow 1 - f(r)$
    end if
end while
return $r$
```

Next section details an application of the amplitude amplification algorithm to BN inference, where the state preparation circuit $\mathcal{B}$ is responsible for encoding the BN into a quantum circuit.

4.4 Quantum Bayesian inference

As already stated in Chapter 2, BNs are a powerful tool for representing probability distributions in a compact way and, as seen in Chapter 3, they can also be used to solve RL problems when regarding that BN as a DDN. Nonetheless, inference is required to use BNs in order to solve these problems, because it allows the retrieval of a conditional probability distribution $P\left(\mathcal{Q}|\mathcal{E}=e\right)$, with query variables $\mathcal{Q}$ and evidence variables $\mathcal{E}$ with value e, from the joint probability distribution $P\left(X_1, X_2, \ldots, X_N\right)$ that the BN encodes.

This section presents a quantum version of the rejection sampling inference algorithm described in Section 2.3.2, which provides a quadratic speedup over its classical counterpart due to the use of amplitude amplification, detailed in the Section 4.3.

Quantum rejection sampling inference works by encoding the BN in a quantum operator $\mathcal{B}$ and using amplitude amplification to amplify quantum states with evidence $\mathcal{E}=e$. To further explain this algorithm, this section in broken into two subsections, the first one explaining how to construct the quantum operator $\mathcal{B}$, and the second detailing how to build the evidence phase flip operator $\mathcal{S}_e$ that performs the phase flip on the states with the right evidence, similar to the oracle operator $\mathcal{S}_f$.

4.4.1 Quantum embeddings of Bayesian networks

The problem of encoding a BN in a quantum circuit can be easily understood. A BN encodes a probability distribution $P\left(X_1, X_2, \ldots, X_N\right)$ over its RVs, and encoding it in a quantum circuit means being able to construct a quantum state $|\Psi\rangle = \mathcal{B}|0\rangle^{\otimes N}$ whose qubits represent the RVs. Once the circuit is properly encoded, sampling from the BN joint distribution in the quantum setting is equivalent to performing measurements in the qubits of the circuit. Nevertheless, for this method to work, the BN can only have *discrete* RVs due to used the encoding scheme. Additionally, for simplicity, this discussion only considers the binary RVs case, where each RV can only take two values and is encoded in a single qubit. For a description of this process for arbitrary discrete RVs, the reader is referred to Borujeni et al. [2021]. Hence, to successfully encode the BN, the following equation must hold:

$$\left|\langle x_1 x_2 \ldots x_N|\Psi\rangle\right|^2 = P\left(X_1 = x_1, X_2 = x_2, \ldots, X_N = x_N\right) \tag{4.30}$$

This can be achieved by encoding each of the values of the CPTs in the BN into the quantum circuit.

To understand how this is done in Low et al. [2014], consider a quantum circuit with N qubits q_i, each representing a binary RV X_i. As previously seen in Section 2.2, each entry of the CPT encodes a probability $P\left(X_i = x_i | \text{parents}\left(X_i\right) = x_p\right)$ of the occurrence of a value x_i in a RV X_i, given a value x_p for the RV's parents, parents $\left(X_i\right)$. In the quantum setting, for each RV X_i, a controlled $R_Y(\theta)$ rotation gate is applied for each possible value x_p its parent RVs can take, where the control qubits of this gate are the qubits q_i representing RVs parents $\left(X_i\right)$. This ensures that any RVs that share an edge in the BN are entangled in the quantum circuit, expressing the information they share.

Consider the BN from Figure 1 as an example. To encode it in a quantum circuit, the following gates have to be applied (see Figure 13):

- For the variable Rain R, with no parents, an uncontrolled rotation gate is applied.
- For Sprinkler S, with R as a parent, two rotation gates are applied, controlled by the qubit representing R. The first one represents the case when $|R\rangle = |0\rangle$ [1], and the second one when $|R\rangle = |1\rangle$.
- For WetGrass W, which has two parents, four rotation gates, controlled by both $|R\rangle$ and $|S\rangle$, are needed. The first one for when $|RS\rangle = |00\rangle$, the second for when $|RS\rangle = |01\rangle$, the third for $|RS\rangle = |10\rangle$ and a final one for $|RS\rangle = |11\rangle$.

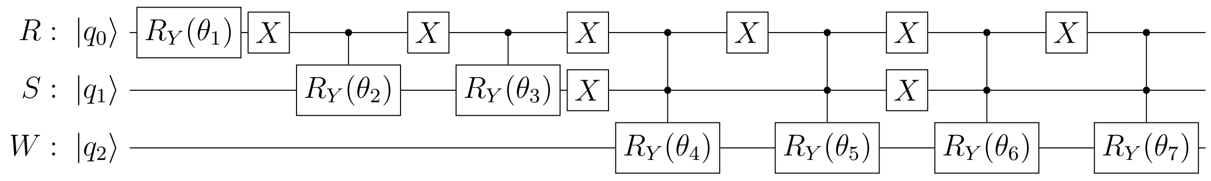


Figure 13: Quantum circuit encoding the BN of Figure 1.

Finally, the only missing step for the encoding is knowing the angle for each rotation gate. Consider the controlled rotation gate for a binary RV X_i with value x_p for its parents. The angle of rotation θ is given by:

$$\theta = 2 \arctan\left(\sqrt{\frac{P\left(X_i = 1 | \text{parents}\left(X_i\right) = x_p\right)}{P\left(X_i = 0 | \text{parents}\left(X_i\right) = x_p\right)}}\right) \tag{4.31}$$

As Low et al. [2014] details, this encoding can be done with a complexity $\mathcal{O}\left(N2^M\right)$, where N represents the number of RVs in the BN and M represents the greatest number of parents of any RV. Given

[1] For $|R\rangle = |0\rangle$, an X gate must be applied *before* and *after* the control operation on qubit $|R\rangle$. This is because a controlled gate, by definition, only changes the target state when its control qubit is in state $|1\rangle$, but in this case the opposite effect is wanted. Therefore, the first X gate flips state $|R\rangle$ before the control operation, and the second X gate returns the $|R\rangle$ state to the way it was before the first X gate was applied, leaving it unchanged.

that this encoding allows the sampling from the joint probability distribution, it can be directly compared to direct sampling, explained in Subsection 2.3.2, which has a complexity of $\mathcal{O}(NM)$. The quantum encoding is therefore generally slower than direct sampling, due to the at most M qubit uniformly controlled rotations, each resulting in a decomposition of $\mathcal{O}(2^M)$ elementary gates Bergholm et al. [2005].

4.4.2 The evidence phase flip operator

Having explained the state preparation circuit $\mathcal{B}$, the missing part of the algorithm is the oracle operator, which, in this particular example, is the evidence phase flip operator $\mathcal{S}_e$. In quantum Bayesian inference, the good states are states where the evidence qubits $\mathcal{E}$ matches the evidence value e of the conditional distribution $P(\mathcal{Q}|\mathcal{E} = e)$ that is to be inferred.

In this sense, the amplitude amplification technique turns the joint probability distribution of the BN into the aforementioned conditional distribution by decreasing the amplitudes of states with incorrect evidence $\mathcal{E} \neq e$ and amplifying the amplitudes of states with the right evidence $\mathcal{E} = e$. Formally, the resulting state $|\Psi\rangle = \mathcal{B}|0\rangle^{\otimes N}$ of the BN quantum encoding can be expressed as:

$$|\Psi\rangle = \sqrt{P(e)}\,|\mathcal{Q}, e\rangle + \sqrt{1 - P(e)}\,|\mathcal{Q}, \overline{e}\rangle \tag{4.32}$$

where $|\mathcal{Q}, e\rangle$ represents the quantum states with the correct evidence e, $|\mathcal{Q}, \overline{e}\rangle$ the quantum states with the wrong evidence and $P(e)$ the probability of occurrence of evidence e. Therefore, the effect of $\mathcal{S}_e$ on this quantum state is given by:

$$\mathcal{S}_e\,|\Psi\rangle = -\sqrt{P(e)}\,|\mathcal{Q}, e\rangle + \sqrt{1 - P(e)}\,|\mathcal{Q}, \overline{e}\rangle \tag{4.33}$$

Let $e = e_1 e_2 \ldots e_k$ represent the evidence bit string, with $e_i \in \mathbb{B}$. Constructing this operator requires the help of two other operators:

- The operator $\mathcal{X}_i = \mathbb{1} \otimes \cdots \otimes X \otimes \cdots \otimes \mathbb{1}$, which flips the i-th evidence qubit by applying an X gate to it and leaving every other qubit unchanged,
- The controlled phase operator $P_{1\ldots k}$, which flips the phase of a quantum state if all evidence qubits are in state $|1\rangle$.

With the above two operators defined, the evidence phase flip operator $\mathcal{S}_e$ can be defined as follows:

$$\mathcal{S}_e = \mathcal{F} P_{1\ldots k} \mathcal{F}, \; \mathcal{F} = \prod_{i=1}^{k} \mathcal{X}_i^{1-e_i} \tag{4.34}$$

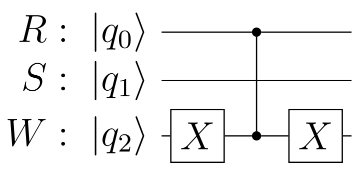


Figure 14: Quantum circuit for the evidence phase flip operator $\mathcal{S}_e$ for the BN of Figure 1 with evidences $R = 1$ and $W = 0$.

Consider once more the example of Figure 1 with evidence qubits R and W and values 1 and 0, respectively, in order to infer the probability distribution $P\left(S|R = 1, W = 0\right)$. The evidence phase flip operator for this specific case is represented in Figure 14

Notice from Equation (4.34) that the quantum operator $\mathcal{F}$ only flips the evidence qubits with evidence $e_i = 0$, otherwise it leaves them unchanged. This allows for the operator $\mathcal{S}_e$ to behave as expressed by Equation (4.33). To show this, let $|\mathcal{Q}, \tilde{e}\rangle = \mathcal{F}\,|\mathcal{Q}, \overline{e}\rangle$, where $|\tilde{e}\rangle \neq |1\rangle^{\otimes k}$, represent the application of the quantum operator $\mathcal{F}$ to a superposition of quantum states whose evidence does not match e. Then:

$$\begin{aligned}
\mathcal{S}_e\,|\Psi\rangle &= \mathcal{F}P_{1\ldots k}\mathcal{F}\left(\sqrt{P(e)}\,|\mathcal{Q}, e\rangle + \sqrt{1 - P(e)}\,|\mathcal{Q}, \overline{e}\rangle\right)\\
&= \mathcal{F}P_{1\ldots k}\left(\sqrt{P(e)}\,|\mathcal{Q}, 1\rangle + \sqrt{1 - P(e)}\,|\mathcal{Q}, \tilde{e}\rangle\right)\\
&= \mathcal{F}\left(-\sqrt{P(e)}\,|\mathcal{Q}1\rangle + \sqrt{1 - P(e)}\,|\mathcal{Q}, \tilde{e}\rangle\right)\\
&= -\sqrt{P(e)}\,|\mathcal{Q}, e\rangle + \sqrt{1 - P(e)}\,|\mathcal{Q}, \overline{e}\rangle
\end{aligned} \tag{4.35}$$

This operator, together with the BN encoding, leads to an amplitude amplification operator $G = \mathcal{B}\mathcal{S}_0\mathcal{B}^{\dagger}\mathcal{S}_e$, making the quantum inference in BNs quadratically faster than in the classical case, provided that the BN has a small maximum number of parents M:

	Classical	Quantum
Complexity	$\mathcal{O}\left(NMP(e)^{-1}\right)$	$\mathcal{O}\left(N2^M P(e)^{-\frac{1}{2}}\right)$

Table 1: Comparison between classical and quantum complexity for rejection sampling inference in BNs, taken from Low et al. [2014].

Contrary to many other speedups in quantum algorithms, this particular one does not require any oracle queries. As the authors state, the complexity of this algorithm is completely determined by counting primitive quantum operations. Another important mention is that this algorithm can also be applied to DBNs, as Borujeni and Nannapaneni [2021] show, by also encoding observation nodes in the quantum circuit and considering a finite number of time-slices of the network at a time, performing inference over those finite slices. The only difference with respect to performing inference in a regular BN is their relative size. Moreover, the same concept can be applied to DDNs by also encoding action and reward nodes into qubits, allowing this inference to be performed in the context of solving RL problems.

4.5 Summary

This Chapter presents a brief introduction to quantum computing with quantum circuits, from single-qubit systems to multi-qubit ones. Grover's algorithm and its generalization, amplitude amplification, are also discussed in detail.

Finally, as a way to tie this Chapter together with the previous ones, regarding BNs and RL, an algorithm for performing inference over a BN in a quantum circuit is described. This quantum algorithm has a classical counterpart described in Subsection 2.3.2 and has a quadratic speedup over it, allowing for faster inference times. Moreover, as shown in Chapter 6, quantum rejection sampling also provides a computational advantage over classical rejection sampling for solving RL partially observable problems.

Part II

Core of the Dissertation

Chapter 5

Quantum sampling subroutines

In Section 3.4 it is shown how to extract a near-optimal action in a POMDP using a lookahead based algorithm. This algorithm must have access to three different probability distributions encoding the environment dynamics. This Chapter presents quantum circuits based on the quantum rejection sampling algorithm of Section 4.4 for sampling these probability distributions, allowing for the definition of a classical quantum lookahead algorithm by leveraging these subroutines. Furthermore, this chapter also contains formal proofs along each section, showing that the presented quantum circuits do encode the desired probability distributions.

5.1 Quantum direct sampling

Performing inference in a BN can be tricky in its details. Consider, for example, the arbitrary conditional probability distribution $P\left(\mathcal{Q}|\mathcal{E}=e\right)$. Given that it is a conditional probability distribution, it should be necessary to extract it from the BN using rejection sampling, right? Not quite. Suppose that this BN is a DDN and that the evidence e corresponds to a belief state b, such that the probability distribution now is $P\left(\mathcal{Q}|b\right)$. The belief state of an agent is always updated, and used as the CPT of the root state node of a DDN (see Figure 15). As such, the belief-state is already encoded in the DDN, and therefore all samples collected from it are sampled from the probability distribution $P\left(\mathcal{Q}|b\right)$, even when using direct sampling!

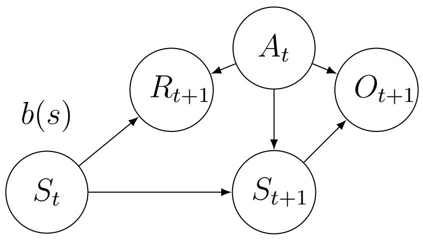


Figure 15: Representation of a DDN for performing the belief update operation.

And what if the conditioning RV is the action, and therefore the distribution is $P\left(\mathcal{Q}|a\right)$? The value for the action has to be directly encoded into its CPT every time-step, for two possible reasons:

- The best action is not known, and therefore all actions have to be tested (as in the lookahead algorithm). In this case, every action is encoded into the DDN individually to test their performance.
- The best action has already been determined, and therefore has to be encoded directly into the DDN.

Either way, the value of the action is encoded into the DDN, and therefore all samples extracted from it already satisfy the conditioning of the distribution $P\left(\mathcal{Q}|a\right)$. Once again, only direct sampling is needed to extract this distribution. Of course, the same reasoning can be applied to any distribution that is conditioned on both a belief state and an action: both of them just have to be encoded into the DDN's CPTs.

Can this same reasoning be applied to the observation O_{t+1}, reward R_{t+1} and state S_{t+1} RVs of Figure 15? No, otherwise rejection sampling would have no purpose for DDNs. The reason why this encoding trick works for the belief and action RVs is because they are root RVs of the DDN, and so their CPTs are not dependent on any other RVs. The remaining RVs of the DDN are not root RVs, and therefore the same does not apply to them.

With this encoding trick covered, it is important to check whether it can be used to calculate any of the distributions of the lookahead algorithm of Section 3.4:

- Reward sampling distribution $P\left(R_{t+1}|b_t, a_t\right)$: conditioned on the belief state and action, and therefore can be performed with direct sampling.
- Observation sampling distribution $P\left(O_{t+1}|b_t, a_t\right)$: conditioned on the belief state and action, and therefore can also be performed with direct sampling.
- Belief-update distribution $P\left(S_{t+1}|o_{t+1}, b_t, a_t\right)$: conditioned not just on the belief state and action, which can be directly encoded into the DDN, but also on the observation. This distribution must be extracted using rejection sampling.

Two of the three probability distributions can be extracted using direct sampling, and therefore should, as direct sampling is a more efficient algorithm when compared to rejection sampling. Seeing as encoding a BN into a quantum circuit, the quantum analog of direct sampling, has a complexity of $\mathcal{O}\left(N2^M\right)$, which is more costly than direct sampling $\mathcal{O}\left(NM\right)$. Thus, the classical alternative should be preferred. A detailed proof that both reward and observation sampling can be performed on a quantum computer by only encoding the corresponding DDN is provided in Appendices A.1 and A.2, respectively.

This entails that the belief-update is the only possible source of quantum advantage, as it is the only probability distribution used in the lookahead algorithm that is elligible to use quantum rejection sampling and have a possible speedup over its classical alternative. This is discussed in Section 5.2, showing how to use quantum rejection sampling to perform the belief-update operation on a DDN.

Another very important observation is that this encoding trick can be applied to any MDP distribution. Recall from Section 3.2 that the distributions in an MDP are the transition dynamics $P\left(S_{t+1}|s_t, a_t\right)$ and the reward dynamics $P\left(R_{t+1}|s_t, a_t\right)$, both of which are only dependent on a state and an action, and therefore can be extracted using direct sampling as described above. A consequence is that quantum rejection sampling does not provide a quantum advantage for MDPs, but only for POMDPs.

5.2 Quantum belief update

When using either DBNs or DDNs to model dynamic systems, one of the most important operations is the belief update, as discussed in Chapters 2 and 3. This section shows how this procedure can be performed in a quantum circuit by leveraging the amplitude amplification algorithm.

Performing a belief-update requires two time-steps of a DDN, as shown in Figure 15. Consider the scenario where the value of S_t is unknown, but the agent has a belief-state $b(s)$ at time t. Also suppose that action $A_t = a$ has been chosen and that the environment sent back observation $O_{t+1} = o$. In this scenario, the new belief state $\tau_c\left(b, a, o\right)\left(s'\right)$ of the agent can be determined by the following equation:

$$\begin{aligned}\tau_c\left(b, a, o\right)\left(s'\right) &\equiv P\left(s'|o, a, b\right) \\ &\propto P\left(o|s, a\right)\sum_{s\in\mathcal{S}} P\left(s'|a, s\right) b(s)\end{aligned} \tag{5.1}$$

Classically, a belief-update requires performing the computation expressed in Equation (5.1). Let us show, however, that a belief update can be performed in a quantum setting by encoding the DDN in Figure 16 with the right action $A_t = a$, a superposition on RV S_t according to its belief-state b and performing amplitude amplification on observation $O_{t+1} = o$. If $\tau_q(b, a, o)$ represents the quantum belief update, then the following reasoning seeks to show that the following equation holds:

$$\tau_q(b, a, o)(s') = \tau_c(b, a, o)(s'),\ \forall s' \in \mathcal{S} \tag{5.2}$$

To prove this claim, first consider the quantum circuit corresponding to a belief update represented in Figure 16:

The various $\mathcal{U}$ gates perform the encoding of the CPTs of the DDN, while the operator $G^k(o)$ performs the amplitude amplification of the quantum states with the observation o. To formally define these

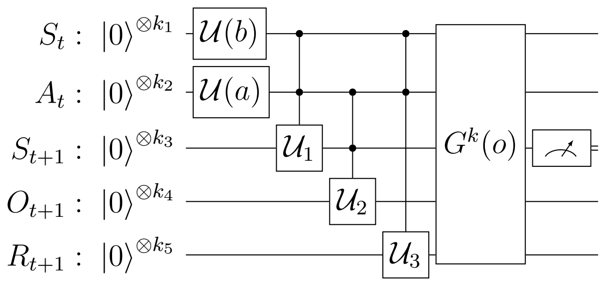

S_t : $|0\rangle^{\otimes k_1}$
A_t : $|0\rangle^{\otimes k_2}$
S_{t+1} : $|0\rangle^{\otimes k_3}$
O_{t+1} : $|0\rangle^{\otimes k_4}$
R_{t+1} : $|0\rangle^{\otimes k_5}$
$\mathcal{U}(b)$
$\mathcal{U}(a)$
$\mathcal{U}_1$
$\mathcal{U}_2$
$\mathcal{U}_3$
$G^k(o)$

Figure 16: Quantum circuit for a belief update in the DDN of Figure 15

operators, a number of definitions are required.

To encode the belief state of the DDN, the operator $\mathcal{U}(b)$ is applied as in Definition 5.2.1:

Definition 5.2.1. *The quantum operator $\mathcal{U}(b)$ encodes belief state b into the qubits of S_t:*

$$\mathcal{U}(b)\,|0\rangle^{\otimes k_1} = \sum_{s\in\mathcal{S}} \sqrt{b(s)}\,|s\rangle$$

The quantum operator $\mathcal{U}(a)$ is used to encode action a into the quantum circuit of the DDN, as in Definition 5.2.2:

Definition 5.2.2. *The quantum operator $\mathcal{U}(a)$ encodes action a into qubits representing A_t, such that:*

$$\mathcal{U}(a)\,|0\rangle^{\otimes k_2} = |a\rangle$$

To encode the transition dynamics of the DDN, a controlled operator, $\mathcal{U}_1$ is applied to the quantum circuit, as in Definition 5.2.3:

Definition 5.2.3. *The operator $\mathcal{U}_1$ is a controlled quantum operator that encodes the transition dynamics $P\left(S_{t+1}|s_t, a_t\right)$ on the qubits of RV S_{t+1}, based on the values s_t and a_t of RVs S_t and A_t, respectively:*

$$\mathcal{U}_1\,|sa\rangle\,|0\rangle^{\otimes k_3} = \sum_{s'\in\mathcal{S}} \sqrt{P\left(s'|s,a\right)}\,|sas'\rangle$$

For the encoding of the sensor model of the DDN, the quantum operator $\mathcal{U}_2$ is used as in Definition 5.2.4:

Definition 5.2.4. *$\mathcal{U}_2$ is a controlled quantum operator encoding the sensor model $P\left(O_{t+1}|s_{t+1}, a_t\right)$ on the qubits of RV O_{t+1} given the values s_{t+1} and a_t of RVs S_{t+1} and A_t, respectively:*

$$\mathcal{U}_2 \left|as'\right\rangle \left|0\right\rangle^{\otimes k_4} = \sum_{o\in\Omega} \sqrt{P\left(o|s', a\right)} \left|as'o\right\rangle$$

The final encoding operator is $\mathcal{U}_3$, which encodes the reward dynamics of the DDN into the quantum circuit, as expressed by Definition 5.2.5:

Definition 5.2.5. *The quantum operator $\mathcal{U}_3$ encodes the reward dynamics $P\left(R_{t+1}|s_t, a_t\right)$ into the qubits of RV R_{t+1} given the values s_{t+1} and a_t of RVs S_t and A_t, respectively:*

$$\mathcal{U}_3 \left|sa\right\rangle \left|0\right\rangle^{\otimes k_5} = \sum_{r\in\mathcal{R}} \sqrt{P\left(r|s, a\right)} \left|sar\right\rangle$$

All of the aforementioned quantum operators can be implemented by the encoding schemes discussed in Section 4.4. The rotation operators are responsible for encoding the CPTs of the DDN in Figure 15 into the quantum circuit according to the discussion in Subsection 4.4.1.

Applying all of these encoding operators, in the order they were presented in the definitions above, is the same as encoding the DDN. They can also be grouped into another quantum operator $\mathcal{B}$, the DDN encoding operator, as in Lemma 5.2.6.

Lemma 5.2.6. *The DDN encoding operator $\mathcal{B}$ is composed of the sequential application of quantum operators $\mathcal{U}(b)$, $\mathcal{U}(b)$, $\mathcal{U}_1$, $\mathcal{U}_2$ and $\mathcal{U}_3$ fom Definitions 5.2.1 up to 5.2.5, in this order. Its application results in the following quantum state:*

$$\mathcal{B}\left|0\right\rangle^{\otimes N} = \sum_{s\in\mathcal{S}} \sqrt{b(s)} \sum_{s'\in\mathcal{S}} \sqrt{P(s'|s,a)} \sum_{o\in\Omega} \sqrt{P(o|s',a)} \sum_{r\in\mathcal{R}} \sqrt{P(r|s',a)} \left|sas'or\right\rangle$$

Proof. First, the operators $\mathcal{U}(b)$ and $\mathcal{U}(a)$ are applied to the initial quantum state $\left|0\right\rangle^{\otimes n}$, according to Definitions 5.2.1 and 5.2.2:

$$\left|\psi_1(b,a)\right\rangle = \left(\mathcal{U}(b) \otimes \mathcal{U}(a) \otimes \mathbb{1}\right)\left|0\right\rangle^{\otimes n} = \sum_{s\in\mathcal{S}} \sqrt{b(s)} \left|sa\right\rangle \left|0\right\rangle^{\otimes (k_3+k_4+k_5)}$$

Then, the transition dynamics encoding operator $\mathcal{U}_1$ is applied to the quantum state $\left|\psi_1(b,a)\right\rangle$ as in Definition 5.2.3:

$$\left|\psi_2(b,a)\right\rangle = \left(\mathcal{U}_1 \otimes \mathbb{1}\right)\left|\psi_1(b,a)\right\rangle \overset{(5.2.3)}{=} \sum_{s\in\mathcal{S}} \sqrt{b(s)} \sum_{s'\in\mathcal{S}} \sqrt{P(s'|s,a)} \left|sas'\right\rangle \left|0\right\rangle^{\otimes (k_4+k_5)}$$

Afterwards, the sensor model encoding operator $\mathcal{U}_2$ is applied to $|\psi_2(b,a)\rangle$, according to Definition 5.2.4, yielding:

$$\begin{aligned}|\psi_3(b,a)\rangle &= (\mathbb{1}\otimes\mathcal{U}_2\otimes\mathbb{1})\,|\psi_2(b,a)\rangle\\ &\overset{(5.2.4)}{=} \sum_{s\in\mathcal{S}}\sqrt{b(s)}\sum_{s'\in\mathcal{S}}\sqrt{P(s'|s,a)}\sum_{o\in\Omega}\sqrt{P(o|s')}\,|sas'o\rangle\,|0\rangle^{\otimes k5}\end{aligned}$$

Finally, using Definition 5.2.5, the reward dynamics encoding operator $\mathcal{U}_3$ is applied to $|\psi_3(b,a)\rangle$. The output quantum state is given by:

$$\begin{aligned}|\psi_4(b,a)\rangle &= (\mathbb{1}\otimes\mathcal{U}_3)\,|\psi_3(b,a)\rangle\\ &\overset{(5.2.5)}{=} \sum_{s\in\mathcal{S}}\sqrt{b(s)}\sum_{s'\in\mathcal{S}}\sqrt{P(s'|s,a)}\sum_{o\in\Omega}\sqrt{P(o|s',a)}\sum_{r\in\mathcal{R}}\sqrt{P(r|s',a)}\,|sas'or\rangle\end{aligned}$$

□

The DDN encoding operator is therefore used to create the amplitude amplification operator $G(o) = \mathcal{B}\mathcal{S}_0\mathcal{B}^\dagger\mathcal{S}_e$, as described in Sections 4.3 and 4.4. This operator is then applied a certain amount k of times to transform the quantum states into only goal states. For some cases, estimating a number k of applications of the amplitude amplification operator that guarantees the measurement outcome to be a solution state might not be possible. It is often the case that a solution is only measured up to some probability. However, using Algorithm 3, it is always possible to obtain a solution state, not by choosing the perfect k, but by applying many different k's until one of them works. As such, in this section, it is assumed that the number k always gives a solution state with 100% probability, as it still proves that the quantum circuit of Figure 16 is correct. The only difference is that in a real practical application, the quantum circuit of Figure 16 would not be used on its own, but applied within Algorithm 3.

Thus, assuming a perfect parameter k for the number of amplitude amplification operator applications, this operator will completely zero all the amplitudes of the quantum states with the wrong observation $O_{t+1}\neq o$, amplifying the amplitudes of quantum states with the observation $O_{t+1} = o$, as follows:

Assertion 5.2.7. *Suppose the amplitude amplification operator $G(o)$ is applied to a quantum state $|\psi\rangle = \sum_{x\in\mathcal{X}}\sum_{o'\in\Omega}\alpha_{x,o'}\,|x,o'\rangle$ (a superposition over values of the sets $\mathcal{X}$ and observations Ω) a perfect number k of times. In this scenario, this operator amplifies the amplitudes of quantum states with observations o, while leaving all other states with zero amplitude:*

$$G^k(o)\,|\psi\rangle = \frac{1}{\sqrt{\eta}}\sum_{x\in\mathcal{X}}\alpha_{x,o}\,|x,o\rangle$$

where $\eta = \sum_{x\in\mathcal{X}}|\alpha_{x,o}|^2$ is a normalization constant such that the resulting quantum state $|\psi'\rangle = G^k(o)\,|\psi\rangle$ obeys the relationship $\langle\psi'|\psi'\rangle = 1$.

The amplitude amplification operator can be implemented using the phase flip operator discussed in Subsection 4.4.2. Now, with every operator defined, Lemma 5.2.8 derives the final quantum state after applying the encoding circuit depicted in Figure 16.

Lemma 5.2.8. *The final quantum state $|\psi_{\textit{final}}(b,a,o)\rangle$ of the quantum circuit depicted in Figure 16 is given by the following equation:*

$$|\psi_{\textit{final}}(b,a,o)\rangle = \frac{1}{\sqrt{\eta}} \sum_{s\in\mathcal{S}} \sqrt{b(s)} \sum_{s'\in\mathcal{S}} \sqrt{P(s'|s,a)}\sqrt{P(o|s',a)} \sum_{r\in\mathcal{R}} \sqrt{P(r|s',a)}\, |sas'or\rangle$$

where $\eta = \sum_{s'\in\mathcal{S}} P(o|s',a) \sum_{s\in\mathcal{S}} P(s'|s,a)b(s)$ is a normalization constant.

Proof. The final quantum state is the application of the amplitude amplification operator to the quantum state resulting of the the DDN encoding operator of Lemma 5.2.6:

$$\begin{aligned}|\psi_{\text{final}}(b,a,o)\rangle &= G^k(o) \sum_{s\in\mathcal{S}} \sqrt{b(s)} \sum_{s'\in\mathcal{S}} \sqrt{P(s'|s,a)} \sum_{o\in\Omega} \sqrt{P(o|s',a)} \sum_{r\in\mathcal{R}} \sqrt{P(r|s',a)}\, |sas'or\rangle \\ &\overset{(5.2.7)}{=} \frac{1}{\sqrt{\eta}} \sum_{s\in\mathcal{S}} \sqrt{b(s)} \sum_{s'\in\mathcal{S}} \sqrt{P(s'|s,a)}\sqrt{P(o|s',a)} \sum_{r\in\mathcal{R}} \sqrt{P(r|s',a)}\, |sas'or\rangle\end{aligned}$$

□

The quantum state $|\psi_{\text{final}}\rangle$ is the result of the application of every quantum operator in the circuit of Figure 16. The measurement, the final operation of this circuit, is performed only on the qubit of S_{t+1}. The likelihood of an outcome of this measurement can be derived from the partial density matrix $\rho(b,a,o)$ only representing qubit S_{t+1}. In turn, this matrix can be calculated by executing a partial trace (over every other RV except S_{t+1}) in the density matrix $|\psi_{\text{final}}(b,a,o)\rangle\langle\psi_{\text{final}}(b,a,o)|$ representing the final quantum state. Using this approach, Lemma 5.2.9 calculates the measurement probabilities of the quantum circuit of Figure 16:

Lemma 5.2.9. *The probability of measuring $S_{t+1} = s'$ in the quantum circuit of Figure 16 is given by the following expression:*

$$\langle s'|\rho(b,a,o)|s'\rangle = \frac{1}{\eta} P(o|s',a) \sum_{s\in\mathcal{S}} P(s'|s,a)b(s) \tag{5.3}$$

Proof. First, let us compute the partial density matrix:

$$\rho(b,a,o) = \sum_{s,a',o',r} \langle sa'o'r|\psi_{\text{final}}(b,a,o)\rangle \langle\psi_{\text{final}}(b,a,o)|sa'o'r\rangle$$

$$= \frac{1}{\eta} \sum_{s,a',o',r} \left(\sum_{s_\star \in \mathcal{S}} \sqrt{b(s_\star)} \sum_{s' \in \mathcal{S}} \sqrt{P(s'|s_\star, a)} \sqrt{P(o|s', a)} \sum_{r' \in \mathcal{R}} \sqrt{P(r'|s_\star, a)} \langle s a' o' r | s_\star a s' o r' \rangle \right)$$

$$\left(\sum_{s_\star \in \mathcal{S}} \sqrt{b(s_\star)} \sum_{s' \in \mathcal{S}} \sqrt{P(s'|s_\star, a)} \sqrt{P(o|s', a)} \sum_{r' \in \mathcal{R}} \sqrt{P(r'|s_\star, a)} \langle s_\star a s' o r' | s a' o' r \rangle \right)$$

$$= \frac{1}{\eta} \sum_{s,a',o',r} \left(\sum_{s_\star \in \mathcal{S}} \sqrt{b(s_\star)} \sum_{s' \in \mathcal{S}} \sqrt{P(s'|s_\star, a)} \sqrt{P(o|s', a)} \sum_{r' \in \mathcal{R}} \sqrt{P(r'|s_\star, a)} \delta_{ss_\star} \delta_{aa'} \delta_{oo'} \delta_{rr'} |s'\rangle \right)$$

$$\left(\sum_{s_\star \in \mathcal{S}} \sqrt{b(s_\star)} \sum_{s' \in \mathcal{S}} \sqrt{P(s'|s_\star, a)} \sqrt{P(o|s', a)} \sum_{r' \in \mathcal{R}} \sqrt{P(r'|s_\star, a)} \delta_{ss_\star} \delta_{aa'} \delta_{oo'} \delta_{rr'} \langle s'| \right)$$

$$= \frac{1}{\eta} \sum_{s \in \mathcal{S}} b(s) \sum_{r \in \mathcal{R}} \left(\sum_{s' \in \mathcal{S}} \sqrt{P(s'|s, a)} \sqrt{P(o|s', a)} \sqrt{P(r|s, a)} |s'\rangle \right)$$

$$\left(\sum_{s' \in \mathcal{S}} \sqrt{P(s'|s, a)} \sqrt{P(o|s', a)} \sqrt{P(r|s, a)} \langle s'| \right)$$

Then, the probability of measuring state S_{t+1} with value s' is computed as follows:

$$\langle s'|\rho(b,a,o)|s'\rangle = \frac{1}{\eta} \sum_{s \in \mathcal{S}} b(s) \sum_{r \in \mathcal{R}} \left(\sum_{s^\star \in \mathcal{S}} \sqrt{P(s^\star|s, a)} \sqrt{P(o|s^\star, a)} \sqrt{P(r|s, a)} \langle s'|s^\star \rangle \right)$$

$$\left(\sum_{s^\star \in \mathcal{S}} \sqrt{P(s^\star|s, a)} \sqrt{P(o|s^\star, a)} \sqrt{P(r|s, a)} \langle s^\star|s' \rangle \right)$$

$$= \frac{1}{\eta} \sum_{s \in \mathcal{S}} b(s) \sum_{r \in \mathcal{R}} \left(\sum_{s^\star \in \mathcal{S}} \sqrt{P(s^\star|s, a)} \sqrt{P(o|s^\star, a)} \sqrt{P(r|s, a)} \delta_{s's^\star} \right)$$

$$\left(\sum_{s^\star \in \mathcal{S}} \sqrt{P(s^\star|s, a)} \sqrt{P(o|s^\star, a)} \sqrt{P(r|s, a)} \delta_{s^\star s'} \right)$$

$$= \frac{1}{\eta} \sum_{s \in \mathcal{S}} b(s) \sum_{r \in \mathcal{R}} P(s'|s,a) P(o|s',a) P(r|s,a)$$

$$= \frac{1}{\eta} P(o|s',a) \sum_{s \in \mathcal{S}} P(s'|s,a) b(s) \sum_{r \in \mathcal{R}} P(r|s,a)$$

$$= \frac{1}{\eta} P(o|s',a) \sum_{s \in \mathcal{S}} P(s'|s,a) b(s)$$

□

Once the probabilities of measurement of the final quantum state of the belief update quantum circuit are known, let us rename $\tau_q(b,a,o)(s') = \langle s'|\rho(b,a,o)|s'\rangle$ as the quantum belief update rule. Thus, Theorem 5.2.10 proves the initial claim, that the quantum and classical belief update rules, as in Equation (5.2), are equivalent:

Theorem 5.2.10. *The quantum and classical belief update rules $\tau_q(b,a,o)$ and $\tau_c(b,a,o)$, for DDNs* *using rejection sampling, are equivalent:*

$$\tau_q(b,a,o)(s') = \tau_c(b,a,o)(s'),\ \forall s' \in \mathcal{S}$$

Proof. Equation (5.3) can be re-written as:

$$\tau_q(b,a,o)(s') = \frac{1}{\eta} P(o|s',a) \sum_{s\in\mathcal{S}} P(s'|s,a)b(s)$$

It is also known from the classical case that the classical belief update can be re-written to incorporate a proportionality constant:

$$\tau_c(b,a,o)(s') \overset{((5.1))}{=} \frac{1}{\eta'} P(o|s',a) \sum_{s\in\mathcal{S}} P(s'|s,a)b(s)$$

Since both $\tau_q(b,a,o)(s')$ and $\tau_c(b,a,o)(s')$ are probability distributions, their sum over s' is the same, which is equal to one, and therefore:

$$\begin{gathered}\sum_{s'} \tau_q(b,a,o)(s') = \sum_{s'} \tau_c(b,a,o)(s') \\ \Leftrightarrow \frac{1}{\eta} \sum_{s'\in\mathcal{S}} P(o|s',a) \sum_{s\in\mathcal{S}} P(s'|s,a)b(s) = \frac{1}{\eta'} \sum_{s'\in\mathcal{S}} P(o|s',a) \sum_{s\in\mathcal{S}} P(s'|s,a)b(s) \\ \Leftrightarrow \eta = \eta'\end{gathered}$$

Therefore, the quantum and the classical belief update are equivalent:

$$\tau_q(b,a,o)(s') = \tau_c(b,a,o)(s'),\ \forall s' \in \mathcal{S}$$

□

As a consequence of Theorem 5.2.10, the belief updating procedure on the lookahead algorithm described in Section 3.4 (or any other algorithm that uses the belief update procedure) can be performed in a quantum setting, leveraging the quadratic speedup of quantum rejection sampling.

Naturally, the way to perform this quantum belief update operation is not to just run the circuit once and retrieve a measurement. What is shown in Theorem 5.2.10 is that the probabilities of obtaining a certain measurement match the probabilities of the classical belief update rule. Nonetheless, obtaining an approximation of the next belief state using this circuit should be done with a certain number of samples n.

The simplest way to proceed is to run the quantum circuit of Figure 16 n times (n being the desired number of samples), extracting the samples $s'_i \sim \tau_q(b, a, o),\ i = \{1, \dots, n\}$ for the next state, and approximating the next belief state $b'(s')$ by the following equation:

$$b'_n(s') = \frac{1}{n} \sum_{i=1}^{n} \delta_{s' s'_i} \tag{5.4}$$

where b'_n is an approximation of the belief state, computed resorting to n samples. Equation (5.4) simply states that the next belief state value for state s' is the fraction of extracted samples that match state s'.

5.3 A classical quantum lookahead algorithm

All of the three subroutines of this Chapter (belief update, reward sampling and observation sampling) are necessary for the lookahead algorithm described in Section 3.4. More specifically, these are the only three distributions needed to calculate the values for each belief state or belief state action pair, according to Equations (3.22) and (3.23), which in turn allows the selection of a near-optimal action, as in Equation (3.24).

Using these quantum subroutines in the lookahead algorithm defines a new, hybrid classical quantum algorithm to solve partially observable RL problems. However, as shown in this section, a better option is to only use the quantum belief update subroutine, while resorting to the classical versions of reward sampling and observation sampling, because their quantum versions are more costly in terms of time complexity.

Finally, it is interesting to note that for a fully observable MDP where the reward depends on state S_t and action A_t, there is no advantage in using quantum rejection sampling, because the environment dynamics $P(r|s, a)$ and $P(s'|s, a)$ are distributions where direct CPT encoding can be applied, which in turn means that they can be extracted using direct sampling (rather than rejection sampling), which is more efficient if performed classically. As such, using quantum rejection sampling is only advantageous over its classical counterpart for POMDPs.

5.4 Summary

This Chapter details the use of quantum circuits for sampling subroutines used in the RL lookahead algorithm of Section 3.4.

It is shown how these quantum circuits encode the desired probability distributions and how they tie with both quantum direct and rejection sampling, as introduced in Section 4.4.

Finally, Section 5.3 mentions how these quantum sampling subroutines can be applied to the lookahead algorithm to make a hybrid classical quantum lookahead algorithm, which is possibly more efficient than the fully classical one. This section also establishes that quantum rejection sampling is only advantageous when compared to classical rejection sampling for partially observable environments, making classical rejection sampling better than its quantum equivalent for dealing with MDPs.

Chapter 6

Complexity analysis

Analysing algorithms, either in terms of memory consumption or execution time, typically entails performing a complexity analysis. This Chapter is devoted to performing an execution time complexity analysis of both the classical and quantum-classical lookahead algorithms in order to compare the two and check for potential speedups in the quantum approach.

Both of these lookahead algorithms, however, are approximate and probabilistic algorithms. To execute them, samples have to be extracted and used to approximate probability distributions, which in turn are used to calculate a near-optimal action. As such, to determine their computational complexity, one should also determine their *sample complexity*: the total number of samples needed to execute the algorithm. This Chapter is therefore divided into two sections: one discussing the sample complexity analysis and another carrying on the computational complexity analysis using the sample complexity results.

6.1 Sample complexity

The sample complexity of the quantum-classical algorithm counts the total number of samples [1] required to achieve a certain goal. In this analysis, the chosen goal is to reach an ϵ approximation of the action value function of the optimal action, with a confidence interval of $1-\delta$.

To exemplify, consider that the value r is approximated by the output $r^{\mathcal{L}}$ of a probabilistic algorithm $\mathcal{L}$. For a given precision ϵ, the output of a single execution of $\mathcal{L}$ is said to be ϵ-approximate if:

$$\left|r-r^{\mathcal{L}}\right| \leq \epsilon \tag{6.1}$$

However, if algorithm $\mathcal{L}$ is probabilistic, the error of the approximation varies from one execution to another. As a consequence, if this algorithm is executed an extremely high number of times, the approximation error of some of these executions is likely to be higher than the defined precision. Therefore, most probabilistic algorithms can not be called ϵ-approximate, as not every execution obeys Equation (6.1).

[1] which corresponds to the total amount of times that the dynamic decision network (DDN) has to be called

A better alternative is not to force the approximation of every execution to have a precision of ϵ, but to guarantee instead that each execution has at least some probability $1-\delta$ (confidence interval) of having an error no greater than ϵ:

$$P\left(\left|r - r^{\mathcal{L}}\right| \leq \epsilon\right) \geq 1-\delta \tag{6.2}$$

To start off, recall the definition of the optimal action value function $Q^{\star}(\boldsymbol{b}_t, a_t)$[2] from Equation (3.21):

$$Q^{\star}(\boldsymbol{b}_t, a_t) = \mathbb{E}\left(r_{t+1}|\boldsymbol{b}_t, a_t\right) + \gamma \sum_{o_{t+1}} P(o_{t+1}|\boldsymbol{b}_t, a_t) \max_{a_{t+1}\in\mathcal{A}} Q^{\star}(\boldsymbol{b}_{t+1}, a_{t+1})$$

$$\boldsymbol{b}_{t+1}(s_{t+1}) = P(s_{t+1}|o_{t+1}, a_t, \boldsymbol{b}_t)$$

The analysis of the sample complexity of the algorithm entails many manipulations of these expressions. So, for the sake of compactness and comprehensibility of the proofs of the results ahead, it is useful to resort to vectorial notation [3]. As such, let $\boldsymbol{p}_{r_t} \in \Delta_{\mathcal{R}}$ [4] represent the probability distribution $P(r_{t+1}|\boldsymbol{b}_t, a_t), \forall r_{t+1} \in \mathcal{R}$, $\boldsymbol{p}_{o_t} \in \Delta_{\Omega}$ represent the probability distribution $P(o_{t+1}|\boldsymbol{b}_t, a_t), \forall o_{t+1} \in \Omega$, $\boldsymbol{r} \in \mathbb{R}^{|\mathcal{R}|}$ be a vector whose entries enumerate every possible reward and $\boldsymbol{V}^{\star}_{t+1} \in \mathbb{R}^{|\Omega|}$ be a vector whose entries are the values $\max_{a_{t+1}\in\mathcal{A}} Q^{\star}(\boldsymbol{b}_{t+1}, a_{t+1})$ for every belief update of $\boldsymbol{b}_t$ after taking action a_t [5]. The previous equation can be re-written, using this notation, as[6]:

$$Q^{\star}(\boldsymbol{b}_t, a_t) = \boldsymbol{p}_{r_t}^{\top}\boldsymbol{r} + \gamma \boldsymbol{p}_{o_t}^{\top}\boldsymbol{V}^{\star}_{t+1} \tag{6.3}$$

The lookahead algorithm $\mathcal{L}$ calculates a near-optimal value function $Q^{\mathcal{L}}$ similarly to that of Equation (6.3), but replacing $\boldsymbol{p}_{r_t}$ with its empirical estimation $\boldsymbol{p}_{r_t,n_t}$ using n_t samples, $\boldsymbol{p}_{o_t}$ with its empirical estimation $\boldsymbol{p}_{o_t,m_t}$ using m_t samples and $\boldsymbol{V}^{\star}_{t+1}$ with its empirical estimation $\boldsymbol{V}^{\mathcal{L}}_{t+1}$ calculated recursively. Thus, the lookahead algorithm, starting with root belief node $\boldsymbol{b}_t$ at time-step t obeys the following recursive Equation (6.4) with a terminal case described by Equation (6.5) (according to the lookahead horizon H):

$$Q^{\mathcal{L}}(\boldsymbol{b}_t, a_t) = \boldsymbol{p}_{r_t,n_t}^{\top}\boldsymbol{r} + \gamma \boldsymbol{p}_{o_t,m_t}^{\top}\boldsymbol{V}^{\mathcal{L}}_{t+1} \tag{6.4}$$

$$Q^{\mathcal{L}}(\boldsymbol{b}_{t+H}, a_{t+H}) = 0 \tag{6.5}$$

So far, three sources of error for the lookahead algorithm have been mentioned:

- Approximating the reward distribution $\boldsymbol{p}_{r_t}$ by its empirical estimation $\boldsymbol{p}_{r_t,n_t}$, using n_t samples.

[2] In this section, vector quantities are represented in bold.

[3] Although vectorial notation in preferred in this section, explicitly writing out the probability distributions is sometimes clearer and occasionally used.

[4] In this notation, $\boldsymbol{p} \in \Delta_{\mathcal{V}}$ means that $\boldsymbol{p}$ is a probability distribution vector over the space $\mathcal{V}$.

[5] Since the belief update rule is $\boldsymbol{b}_{t+1}(s_{t+1}) = P(s_{t+1}|o_{t+1}, a_t, \boldsymbol{b}_t)$ and both a_t and $\boldsymbol{b}_t$ are fixed, there are $|\Omega|$ different possible $\boldsymbol{b}_{t+1}$, one for each observation $o_{t+1} \in \Omega$.

[6] In vector notation, $\boldsymbol{v}^{\top}$ is the transpose of $\boldsymbol{v}$, such that the inner-product of two vectors $\boldsymbol{u}$ and $\boldsymbol{v}$ can be written as $\boldsymbol{u}^{\top}\boldsymbol{v}$.

- Approximating the observation distribution $\boldsymbol{p}_{o_t}$ by its empirical estimation $\boldsymbol{p}_{o_t, m_t}$, using m_t samples.
- The stopping case for the lookahead algorithms in the horizon H limit.

There is yet another source of error, coming from the estimation of the belief state. This estimation, however, differs slightly from the estimations of the two distributions mentioned above. Suppose we have an empirical estimation $\boldsymbol{b}_{l_t}$ of the belief state $\boldsymbol{b}_t$, using l_t samples. When performing the belief update to get the next belief state $\boldsymbol{b}_{l_{t+1}}(s_{t+1}) = P_{l_{t+1}}(s_{t+1}|o_{t+1}, a_t, \boldsymbol{b}_{l_t})$, this quantity estimates $P(s_{t+1}|o_{t+1}, a_t, \boldsymbol{b}_{l_t})$ rather than the belief-update $P(s_{t+1}|o_{t+1}, a_t, \boldsymbol{b}_t)$ of the true belief-state $\boldsymbol{b}_t$. As a consequence, $\boldsymbol{b}_{l_{t+1}}$ has two sources of error: the fact that the belief update is performed on a previous estimator and the sampling process that comes after it.

Definition 6.1.1 introduces the lookahead error using the notion of *regret* Liu et al. [2021], a scalar quantity generally used in RL to define an algorithm's error. It is defined as the absolute difference between the utility of the best action that could be taken and the utility of the action that the agent actually took. The remainder of this section serves to provide and analyze an upper bound to this error in order to give theoretical guarantees concerning the performance of the algorithm. These theoretical guarantees are the conditions under which the algorithm can be considered ϵ-approximate with a confidence interval of $1-\delta$, as per Equation (6.2).

Definition 6.1.1. *(Lookahead error). Let* $a_t^{\mathcal{L}} = \text{argmax}_{a_t \in \mathcal{A}} Q^{\mathcal{L}}(\boldsymbol{b}_{l_t}, a_t)$ *be the near optimal action chosen by the algorithm at time* t*. The lookahead error* ϵ_t *at time* t *is defined as the regret:*

$$\epsilon_t = \left| \max_{a_t \in \mathcal{A}} Q^{\star}(\boldsymbol{b}_t, a_t) - Q^{\star}(\boldsymbol{b}_{l_t}, a_t^{\mathcal{L}}) \right| \tag{6.6}$$

Using Lemmas B.0.1 and B.0.2, Lemma 6.1.2 decomposes the lookahead error into three distinct errors, each one easier to bound than the original definition of the error:

Lemma 6.1.2. *The lookahead error* ϵ_t *can be bounded by the following expression:*

$$\epsilon_t \leq \max_{a_t \in \mathcal{A}} \left(\left| Q^{\star}(\boldsymbol{b}_t, a_t) - Q^{\mathcal{L}}(\boldsymbol{b}_t, a_t) \right| + \left| Q^{\mathcal{L}}(\boldsymbol{b}_t, a_t) - Q^{\mathcal{L}}(\boldsymbol{b}_{l_t}, a_t) \right| \right) + \left| Q^{\star}(\boldsymbol{b}_{l_t}, a_t^{\mathcal{L}}) - Q^{\mathcal{L}}(\boldsymbol{b}_{l_t}, a_t^{\mathcal{L}}) \right|$$

Proof.

$$\begin{aligned}
\epsilon_t &\overset{(6.1.1)}{=} \left|\max_{a_t\in\mathcal{A}} Q^\star(\boldsymbol{b}_t, a_t) - Q^\star(\boldsymbol{b}_{l_t}, a_t^{\mathcal{L}})\right| \\
&\overset{(B.0.2)}{\leq} \left|\max_{a_t\in\mathcal{A}} Q^\star(\boldsymbol{b}_t, a_t) - \max_{a_t\in\mathcal{A}} Q^{\mathcal{L}}(\boldsymbol{b}_{l_t}, a_t)\right| + \left|Q^\star(\boldsymbol{b}_{l_t}, a_t^{\mathcal{L}}) - \max_{a_t\in\mathcal{A}} Q^{\mathcal{L}}(\boldsymbol{b}_{l_t}, a_t)\right| \\
&\overset{(B.0.1)}{\leq} \max_{a_t\in\mathcal{A}} \left|Q^\star(\boldsymbol{b}_t, a_t) - Q^{\mathcal{L}}(\boldsymbol{b}_{l_t}, a_t)\right| + \left|Q^\star(\boldsymbol{b}_{l_t}, a_t^{\mathcal{L}}) - Q^{\mathcal{L}}(\boldsymbol{b}_{l_t}, a_t^{\mathcal{L}})\right| \\
&= \max_{a_t\in\mathcal{A}} \left|\left(Q^\star(\boldsymbol{b}_t, a_t) - Q^{\mathcal{L}}(\boldsymbol{b}_t, a_t)\right) + \left(Q^{\mathcal{L}}(\boldsymbol{b}_t, a_t) - Q^{\mathcal{L}}(\boldsymbol{b}_{l_t}, a_t)\right)\right| + \left|Q^\star(\boldsymbol{b}_{l_t}, a_t^{\mathcal{L}}) - Q^{\mathcal{L}}(\boldsymbol{b}_{l_t}, a_t^{\mathcal{L}})\right| \\
&\overset{(B.0.2)}{\leq} \max_{a_t\in\mathcal{A}} \left(\left|Q^\star(\boldsymbol{b}_t, a_t) - Q^{\mathcal{L}}(\boldsymbol{b}_t, a_t)\right| + \left|Q^{\mathcal{L}}(\boldsymbol{b}_t, a_t) - Q^{\mathcal{L}}(\boldsymbol{b}_{l_t}, a_t)\right|\right) + \left|Q^\star(\boldsymbol{b}_{l_t}, a_t^{\mathcal{L}}) - Q^{\mathcal{L}}(\boldsymbol{b}_{l_t}, a_t^{\mathcal{L}})\right|
\end{aligned}$$

□

Lemma 6.1.2 states that the total lookahead error can be decomposed into three distinct errors:

- $\left|Q^\star(\boldsymbol{b}_t, a_t) - Q^{\mathcal{L}}(\boldsymbol{b}_t, a_t)\right|$: the error between the true action value function and the approximate one, both evaluated at belief state $\boldsymbol{b}_t$.
- $\left|Q^{\mathcal{L}}(\boldsymbol{b}_t, a_t) - Q^{\mathcal{L}}(\boldsymbol{b}_{l_t}, a_t)\right|$: the error between the approximate action value function evaluated at the true belief state $\boldsymbol{b}_t$ and at the approximate belief-state $\boldsymbol{b}_{l_t}$.
- $\left|Q^\star(\boldsymbol{b}_{l_t}, a_t^{\mathcal{L}}) - Q^{\mathcal{L}}(\boldsymbol{b}_{l_t}, a_t^{\mathcal{L}})\right|$: the error between the true and approximate action value functions, both evaluated at the approximate belief $\boldsymbol{b}_{l_t}$ and the near optimal action $a_t^\star$.

The remainder of this section determines upper bounds to all error terms in order to define an upper bound for the lookahead error. The main result that allows to define these bounds is Hoeffding's inequality, stated in Theorem 6.1.3[7].

Theorem 6.1.3. *(Hoeffding's Inequality Sidford et al. [2018]). Let $\boldsymbol{p} \in \Delta_{\mathcal{V}}$ be a probability vector and $\boldsymbol{V} \in \mathbb{R}^{|\mathcal{V}|}$ a vector. Let $\boldsymbol{p}_m \in \Delta_{\mathcal{V}}$ be an empirical estimation of $\boldsymbol{p}$ using m i.i.d. [8] samples and $\delta \in (0,1)$ a parameter. Then, with probability at least $1-\delta$:*

$$\left|\boldsymbol{p}^\top \boldsymbol{V} - \boldsymbol{p}_m^\top \boldsymbol{V}\right| \leq \|V\|_\infty \mathcal{H}(m,\delta)$$

Where $\mathcal{H}(m,\delta) = \sqrt{\frac{1}{2m}\log\left(\frac{2}{\delta}\right)}$.

The lemmas and theorems above already provide the necessary tools to bound two of the error sources that compose the lookahead error as in Lemma 6.1.2. Using these results, Lemma 6.1.4 bounds the error between the true value function and the approximate one, both evaluated at the same belief state and action:

[7] $\|\cdot\|_\infty$ denotes the l_∞ norm of a vector, which extracts its maximum element. If $u = \|\boldsymbol{U}\|_\infty$, then $u \geq u', \forall u' \in \boldsymbol{U}$.

[8] Independent and identically distributed

Lemma 6.1.4. *Let $\Gamma = (1-\gamma)^{-1}$ be the effective horizon, $r_{\textsf{max}} = \|\boldsymbol{r}\|_\infty$ be the maximum reward and $\sigma_t \in (0,1)$ a parameter defining the confidence interval $1-\sigma_t$ for sampling both $\boldsymbol{p}_{r_t}$ and $\boldsymbol{p}_{o_t}$ at time-step t. The error $\left|Q^\star(\boldsymbol{b}_t, a_t) - Q^{\mathcal{L}}(\boldsymbol{b}_t, a_t)\right|$ has the following upper bound:*

$$\left|Q^\star(\boldsymbol{b}_t, a_t) - Q^{\mathcal{L}}(\boldsymbol{b}_t, a_t)\right| \leq r_{\textsf{max}} \sum_{k=0}^{H-1} \gamma^k \left(\mathcal{H}(n_{t+k}, \sigma_{t+k}) + \gamma\Gamma\mathcal{H}(m_{t+k}, \sigma_{t+k})\right) + \gamma^H \Gamma r_{\textsf{max}}$$

Proof. Using the definitions for $Q^\star$ and $Q^{\mathcal{L}}$ from Equations (6.3) and (6.4), respectively:

$$\left|Q^\star(\boldsymbol{b}_t, a_t) - Q^{\mathcal{L}}(\boldsymbol{b}_t, a_t)\right| \overset{(B.0.2)}{\leq} \left|\boldsymbol{r}^\top \boldsymbol{p}_{r_t} - \boldsymbol{r}^\top \boldsymbol{p}_{r_t,n_t}\right| + \gamma \left|\boldsymbol{p}_{o_t}^\top \boldsymbol{V}_{t+1}^\star - \boldsymbol{p}_{o_t,m_t}^\top \boldsymbol{V}_{t+1}^{\mathcal{L}}\right|$$

Identity $\boldsymbol{p}_{o_t} - \boldsymbol{p}_{o_t,m_t} \leq |\boldsymbol{p}_{o_t} - \boldsymbol{p}_{o_t,m_t}| \Leftrightarrow -\boldsymbol{p}_{o_t,m_t} \leq |\boldsymbol{p}_{o_t} - \boldsymbol{p}_{o_t,m_t}| - \boldsymbol{p}_{o_t}$ yields:

$$\left|Q^\star(\boldsymbol{b}_t, a_t) - Q^{\mathcal{L}}(\boldsymbol{b}_t, a_t)\right| \overset{(B.0.2)}{\leq} \left|\boldsymbol{r}^\top \boldsymbol{p}_{r_t} - \boldsymbol{r}^\top \boldsymbol{p}_{r_t,n_t}\right| + \gamma \left|\boldsymbol{p}_{o_t}^\top \boldsymbol{V}_{t+1}^{\mathcal{L}} - \boldsymbol{p}_{o_t,m_t}^\top \boldsymbol{V}_{t+1}^{\mathcal{L}}\right| + \gamma \left|\boldsymbol{p}_{o_t}^\top \boldsymbol{V}_{t+1}^\star - \boldsymbol{p}_{o_t}^\top \boldsymbol{V}_{t+1}^{\mathcal{L}}\right|$$

The terms $\left|\boldsymbol{r}^\top \boldsymbol{p}_{r_t} - \boldsymbol{r}^\top \boldsymbol{p}_{r_t,n_t}\right|$ and $\left|\boldsymbol{p}_{o_t}^\top \boldsymbol{V}_{t+1}^{\mathcal{L}} - \boldsymbol{p}_{o_t,m_t}^\top \boldsymbol{V}_{t+1}^{\mathcal{L}}\right|$ can be bounded through Hoeffding's inequality in Theorem 6.1.3 and the fact that, by definition, $\|\boldsymbol{V}_{t+1}^{\mathcal{L}}\|_\infty \leq Q_{\textsf{max}} = \Gamma r_{\textsf{max}}$. The other term can be decomposed into a sum:

$$\begin{aligned}\left|Q^\star(\boldsymbol{b}_t, a_t) - Q^{\mathcal{L}}(\boldsymbol{b}_t, a_t)\right| &\overset{(6.1.3)}{\leq} r_{\textsf{max}} \left(\mathcal{H}(n_t, \sigma_t) + \gamma\Gamma\mathcal{H}(m_t, \sigma_t)\right) \\ &\quad + \gamma \max_{a_{t+1}\in\mathcal{A}} \sum_{o_{t+1}\in\Omega} P(o_{t+1}|\boldsymbol{b}_t, a_t) \left|Q^\star(\boldsymbol{b}_{t+1}, a_{t+1}) - Q^{\mathcal{L}}(\boldsymbol{b}_{t+1}, a_{t+1})\right|\end{aligned}$$

Let $\boldsymbol{\mu}_{t+1}$ be a belief-state and a'_{t+1} an action such that $\left|Q^\star(\boldsymbol{\mu}_{t+1}, a'_{t+1}) - Q^{\mathcal{L}}(\boldsymbol{\mu}_{t+1}, a'_{t+1})\right| = \max_{a_{t+1},\boldsymbol{b}_{t+1}} \left|Q^\star(\boldsymbol{b}_{t+1}, a_{t+1}) - Q^{\mathcal{L}}(\boldsymbol{b}_{t+1}, a_{t+1})\right|$. Then:

$$\begin{aligned}\left|Q^\star(\boldsymbol{b}_t, a_t) - Q^{\mathcal{L}}(\boldsymbol{b}_t, a_t)\right| &\leq r_{\textsf{max}} \left(\mathcal{H}(n_t, \sigma_t) + \gamma\Gamma\mathcal{H}(m_t, \sigma_t)\right) \\ &\quad + \gamma \left|Q^\star(\boldsymbol{\mu}_{t+1}, a'_{t+1}) - Q^{\mathcal{L}}(\boldsymbol{\mu}_{t+1}, a'_{t+1})\right| \sum_{o_{t+1}\in\Omega} P(o_{t+1}|\boldsymbol{b}_t, a_t) \\ &= r_{\textsf{max}} \left(\mathcal{H}(n_t, \sigma_t) + \gamma\Gamma\mathcal{H}(m_t, \sigma_t)\right) \\ &\quad + \gamma \left|Q^\star(\boldsymbol{\mu}_{t+1}, a'_{t+1}) - Q^{\mathcal{L}}(\boldsymbol{\mu}_{t+1}, a'_{t+1})\right|\end{aligned}$$

Finally, with the recursive application of this inequality to $\left|Q^\star(\boldsymbol{\mu}_{t+1}, a'_{t+1}) - Q^{\mathcal{L}}(\boldsymbol{\mu}_{t+1}, a'_{t+1})\right|$, with Equation (6.5) as a stopping case to the approximate action value function, the result of this lemma is attained:

$$\begin{aligned}
\left|Q^\star(\boldsymbol{b}_t,a_t)-Q^{\mathcal{L}}(\boldsymbol{b}_t,a_t)\right| &\leq r_{\text{max}}\sum_{k=0}^{H-1}\gamma^k\left(\mathcal{H}(n_{t+k},\sigma_{t+k})+\gamma\Gamma\mathcal{H}(m_{t+k},\sigma_{t+k})\right)\\
&\quad+\gamma^H\left|Q^\star(\boldsymbol{\mu}_{t+H},a'_{t+H})-Q^{\mathcal{L}}(\boldsymbol{\mu}_{t+H},a'_{t+H})\right|\\
&\overset{((6.5))}{=} r_{\text{max}}\sum_{k=0}^{H-1}\gamma^k\left(\mathcal{H}(n_{t+k},\sigma_{t+k})+\gamma\Gamma\mathcal{H}(m_{t+k},\sigma_{t+k})\right)\\
&\quad+\gamma^H\left|Q^\star(\boldsymbol{\mu}_{t+H},a^\star_{t+H})\right|\\
&\leq r_{\text{max}}\sum_{k=0}^{H-1}\gamma^k\left(\mathcal{H}(n_{t+k},\sigma_{t+k})+\gamma\Gamma\mathcal{H}(m_{t+k},\sigma_{t+k})\right)+\gamma^H\Gamma r_{\text{max}}
\end{aligned}$$

□

Lemma 6.1.4 gives a bound to the difference between the true value function and the value function calculated by the lookahead algorithm, both evaluated at the same action and belief-state. Notice, however, that the bound holds for any choice of action and belief-state, because it is independent of both these quantities. Thus, both $\left|Q^\star(\boldsymbol{b}_t,a_t)-Q^{\mathcal{L}}(\boldsymbol{b}_t,a_t)\right|$ and $\left|Q^\star(\boldsymbol{b}_{l_t},a^{\mathcal{L}}_t)-Q^{\mathcal{L}}(\boldsymbol{b}_{l_t},a^{\mathcal{L}}_t)\right|$ can be bounded by this lemma.

Before moving on to the next lemma, it will prove useful to first count the total number of Hoeffding inequalities that were used in the proof of Lemma 6.1.4. This count will be useful later on to calculate the confidence interval of the lookahead algorithm's bounds. The bound in Lemma 6.1.4 was attained by splitting the error source $\left|Q^\star(\boldsymbol{b}_t,a_t)-Q^{\mathcal{L}}(\boldsymbol{b}_t,a_t)\right|$ in two terms, one where the Hoeffding bounds are applied ($\left|\boldsymbol{r}^\top\boldsymbol{p}_{r_t}-\boldsymbol{r}^\top\boldsymbol{p}_{r_t,n_t}\right|+\gamma\left|\boldsymbol{p}_{o_t}^\top\boldsymbol{V}^{\mathcal{L}}_{t+1}-\boldsymbol{p}_{o_t,m_t}^\top\boldsymbol{V}^{\mathcal{L}}_{t+1}\right|$) and a recursive term ($\gamma\left|\boldsymbol{p}_{o_t}^\top\boldsymbol{V}^\star_{t+1}-\boldsymbol{p}_{o_t}^\top\boldsymbol{V}^{\mathcal{L}}_{t+1}\right|$) where the same procedure is applied until a stopping case is found. Let's call these two terms *immediate* and *recursive*, respectively. To bound the immediate term, 2 Hoeffding inequalities were used. For the first time-step in the recursive term, $2\mathcal{A}\Omega$ Hoeffding inequalities are needed (2 Hoeffding inequalities as per the immediate term, but for every action and every observation). As such, a total of $2\sum_{k=0}^{H-1}\mathcal{A}^k\Omega^k$ Hoeffding inequalities are used to derive the bound of Lemma 6.1.4.

To bound the full lookahead error, only the error of approximating the belief-state remains to be bounded. As mentioned above, the belief-states for the lookahead algorithm are approximated in a different manner to the reward and observation distributions. The only true belief state that is known to the lookahead algorithm is the prior $\boldsymbol{b}_0$, and therefore, the belief estimations $\boldsymbol{b}_{l_t}$ are calculated by performing a belief-update on the previous estimation $\boldsymbol{b}_{l_{t-1}}$, yielding the distribution $\boldsymbol{p}_t\in\Delta_{\mathcal{S}}$ with entries $P(s_t|o_t,a_{t-1},\boldsymbol{b}_{l_{t-1}})$, and extracting l_t samples from it, creating an empirical estimation distribution $\boldsymbol{p}_{t,l_t}=\boldsymbol{b}_{l_t}\in\Delta_{\mathcal{S}}$ with entries $P_{l_t}(s_t|o_t,a_{t-1},\boldsymbol{b}_{l_{t-1}})$. As such, Hoeffding's inequality can not be applied to the distributions $\boldsymbol{b}_t$ and $\boldsymbol{b}_{l_t}$, because they are not direct estimations of one another. It can be applied, however, to $\boldsymbol{p}_t$ and $\boldsymbol{p}_{t,l_t}$, since the latter approximates the former distribution via sampling, as stated above.

Lemma 6.1.5 uses an indirect approach to applying the Hoeffding inequalities and still be able to bound the error between the true belief state $\boldsymbol{b}_t$ and the approximate belief state $\boldsymbol{b}_{l_t}$ of the algorithm at time t:

Lemma 6.1.5. *Given a distribution $\boldsymbol{p}_{s_t}$ from a family of distributions $P(\cdot|s_t,\cdot)$ conditioned on the state $s_t \in \mathcal{S}$* [9]*, the following inequality bounds the error of approximating the belief-state at time t, using l_k samples and a confidence interval of $1-\sigma_k$ to estimate it at time-step k:*

$$\left|\boldsymbol{p}_{s_t}^\top \boldsymbol{b}_t - \boldsymbol{p}_{s_t}^\top \boldsymbol{b}_{l_t}\right| \leq \mathcal{S}^t \sum_{k=0}^{t} \frac{1}{\mathcal{S}^k} \mathcal{H}(l_k, \sigma_k)$$

Proof.

$$\left|\boldsymbol{p}_{s_t}^\top \boldsymbol{b}_t - \boldsymbol{p}_{s_t}^\top \boldsymbol{b}_{l_t}\right| = \left|\sum_{s_t \in \mathcal{S}} P(\cdot|s_t,\cdot)\left(P(s_t|o_t, a_{t-1}, \boldsymbol{b}_{t-1}) - P_{l_t}(s_t|o_t, a_{t-1}, \boldsymbol{b}_{l_{t-1}})\right)\right|$$

Notice that the difference $P(s_t|o_t, a_{t-1}, \boldsymbol{b}_{t-1}) - P_{l_t}(s_t|o_t, a_{t-1}, \boldsymbol{b}_{l_{t-1}})$ in the equation above is between two distributions conditioned on different belief state. As such, Hoeffding's inequality can't be directly applied. To overcome this issue, the triangle inequality $P(s_t|o_t, a_{t-1}, \boldsymbol{b}_{t-1}) \leq P(s_t|o_t, a_{t-1}, \boldsymbol{b}_{l_{t-1}}) + \left|P(s_t|o_t, a_{t-1}, \boldsymbol{b}_{t-1}) - P(s_t|o_t, a_{t-1}, \boldsymbol{b}_{l_{t-1}})\right| = P(s_t|o_t, a_{t-1}, \boldsymbol{b}_{l_{t-1}}) + \left|\boldsymbol{p}_{s_{t-1}}^\top \boldsymbol{b}_{t-1} - \boldsymbol{p}_{s_{t-1}}^\top \boldsymbol{b}_{l_{t-1}}\right|$ is used, allowing the expression above can be re-written as:

$$\begin{aligned}\left|\boldsymbol{p}_{s_t}^\top \boldsymbol{b}_t - \boldsymbol{p}_{s_t}^\top \boldsymbol{b}_{l_t}\right| &\leq \sum_{s_t \in \mathcal{S}} P(\cdot|s_t,\cdot)\left(P(s_t|o_t, a_{t-1}, \boldsymbol{b}_{l_{t-1}}) - P_{l_t}(s_t|o_t, a_{t-1}, \boldsymbol{b}_{l_{t-1}}) + \left|\boldsymbol{p}_{s_{t-1}}^\top \boldsymbol{b}_{t-1} - \boldsymbol{p}_{s_{t-1}}^\top \boldsymbol{b}_{l_{t-1}}\right|\right) \\ &\leq \left|\boldsymbol{p}_{s_t}^\top \boldsymbol{p}_t - \boldsymbol{p}_{s_t}^\top \boldsymbol{p}_{t,l_t}\right| + \mathcal{S}\left|\boldsymbol{p}_{s_{t-1}}^\top \boldsymbol{b}_{t-1} - \boldsymbol{p}_{s_{t-1}}^\top \boldsymbol{b}_{l_{t-1}}\right|\end{aligned}$$

Applying the inequality above in a recursive manner and using Hoeffding's inequality:

$$\begin{aligned}\left|\boldsymbol{p}_{s_t}^\top \boldsymbol{b}_t - \boldsymbol{p}_{s_t}^\top \boldsymbol{b}_{l_t}\right| &\leq \sum_{k=0}^{t} \mathcal{S}^{t-k}\left|\boldsymbol{p}_{s_k}^\top \boldsymbol{p}_k - \boldsymbol{p}_{s_k}^\top \boldsymbol{p}_{k,l_k}\right| \\ &\overset{(6.1.3)}{\leq} \mathcal{S}^t \sum_{k=0}^{t} \frac{1}{\mathcal{S}^k} \mathcal{H}(l_k, \sigma_k)\end{aligned}$$

□

[9] In the results of this lemma and the lemmas and theorems that follow, the notation $\mathcal{X}$ for a set $\mathcal{X}$ can have two meanings, that depend on the context of use: it can refer to the set itself, or to the size of the set $|\mathcal{X}|$.

This lemma states that the belief-approximation error bound is cumulative over time. This is intuitive, given that the approximation of a belief-state at time-step t is attained by estimating over the previous estimation, and therefore the estimation errors pile up as time progresses.

A few more bounds are useful to derive the full upper bound of the belief-error. Due to the fact that an inaccurate belief-state is used throughout every time-step of the algorithm, not only does this cause errors in the algorithm's belief-states themselves, but also in calculating other probability distributions that are conditioned on the belief-state (because they should be conditioned on the true belief-state rather than an estimation of it). This implies that both reward sampling and observation sampling also suffer from this belief-state approximation error, and so these error contributions should be accounted for. Lemma 6.1.6 bounds the reward sampling error caused by this belief-approximation:

Lemma 6.1.6. *Let $\boldsymbol{p} \in \Delta_{\mathcal{R}}$ represent the probability distribution $P(r_{t+1}|\boldsymbol{b}_t)$ and $\hat{\boldsymbol{p}} \in \Delta_{\mathcal{R}}$ represent distribution $P(r_{t+1}|\boldsymbol{b}_{l_t})$. Also, let $\boldsymbol{p}_{n_t}$ and $\hat{\boldsymbol{p}}_{n_t}$ be their respective estimations using n_t samples. Then, the following inequality holds:*

$$\left|\boldsymbol{p}_{n_t}^\top \boldsymbol{r} - \hat{\boldsymbol{p}}_{n_t}^\top \boldsymbol{r}\right| \leq r_{\mathit{max}} \left(2\mathcal{H}(n_t, \sigma_t) + \mathcal{R}\mathcal{S}^t \sum_{k=0}^{t} \frac{1}{\mathcal{S}^k} \mathcal{H}(l_k, \sigma_k)\right)$$

Proof.

$$\begin{aligned}
\left|\boldsymbol{p}_{n_t}^\top \boldsymbol{r} - \hat{\boldsymbol{p}}_{n_t}^\top \boldsymbol{r}\right| &\leq \left|\left(\boldsymbol{p}^\top + \left|\boldsymbol{p}^\top - \boldsymbol{p}_{n_t}^\top\right|\right)\boldsymbol{r} + \left(-\hat{\boldsymbol{p}}^\top + \left|\hat{\boldsymbol{p}}^\top - \hat{\boldsymbol{p}}_{n_t}^\top\right|\right)\boldsymbol{r}\right| \\
&\overset{(B.0.2)}{\leq} \left|\boldsymbol{p}^\top \boldsymbol{r} - \boldsymbol{p}_{n_t}^\top \boldsymbol{r}\right| + \left|\hat{\boldsymbol{p}}^\top \boldsymbol{r} - \hat{\boldsymbol{p}}_{n_t}^\top \boldsymbol{r}\right| + \left|\boldsymbol{p}^\top \boldsymbol{r} - \hat{\boldsymbol{p}}^\top \boldsymbol{r}\right| \\
&\overset{(6.1.3)}{\leq} 2r_{\text{max}}\mathcal{H}(n_t, \sigma_t) + \sum_{r_t \in \mathcal{R}} r_t \left|\boldsymbol{p}_{s_t}^\top \boldsymbol{b}_t - \boldsymbol{p}_{s_t}^\top \boldsymbol{b}_{l_t}\right| \\
&\overset{(6.1.5)}{\leq} r_{\text{max}} \left(2\mathcal{H}(n_t, \sigma_t) + \mathcal{R}\mathcal{S}^t \sum_{k=0}^{t} \frac{1}{\mathcal{S}^k} \mathcal{H}(l_k, \sigma_k)\right)
\end{aligned}$$

□

Analogously to the Lemma 6.1.6, Lemma 6.1.7 bounds the observation sampling error due to the belief-state approximation:

Lemma 6.1.7. *Let $\boldsymbol{p}_{o_t} \in \Delta_\Omega$ be a probability distribution $P(o_{t+1}|\boldsymbol{b}_t, a_t)$ and $\hat{\boldsymbol{p}}_{o_t} \in \Delta_\Omega$ be $P(o_{t+1}|\boldsymbol{b}_{l_t}, a_t)$, such that $\boldsymbol{p}_{o_t,m_t}$ and $\hat{\boldsymbol{p}}_{o_t,m_t}$ are their empirical estimations with m_t samples. Also, let $\boldsymbol{V} \in \mathbb{R}^{|\Omega|}$ be a vector of value functions with entries $V(\boldsymbol{b}_t)$ and $\boldsymbol{V}^{\mathcal{L}}$ a similar vector with entries $V^{\mathcal{L}}(\boldsymbol{b}_{l_t})$. Then, the following inequality holds:*

$$\left|\boldsymbol{p}_{o_t,m_t}^\top \boldsymbol{V} - \hat{\boldsymbol{p}}_{o_t,m_t}^\top \boldsymbol{V}^{\mathcal{L}}\right| \leq \Gamma r_{\mathit{max}} \left(2\mathcal{H}(m_t, \sigma_t) + \Omega\mathcal{S}^t \sum_{k=0}^{t} \frac{1}{\mathcal{S}^k} \mathcal{H}(l_k, \sigma_k)\right) + \left|\hat{\boldsymbol{p}}_{o_t,m_t}^\top \boldsymbol{V} - \hat{\boldsymbol{p}}_{o_t,m_t}^\top \boldsymbol{V}^{\mathcal{L}}\right|$$

Proof.

$$
\begin{aligned}
\left|\boldsymbol{p}_{o_t,m_t}^\top \boldsymbol{V} - \hat{\boldsymbol{p}}_{o_t,m_t}^\top \boldsymbol{V}^{\mathcal{L}}\right| &\le \left|\left(\boldsymbol{p}_{o_t}^\top + \left|\boldsymbol{p}_{o_t}^\top - \boldsymbol{p}_{o_t,m_t}^\top\right|\right)\boldsymbol{V} + \left(-\hat{\boldsymbol{p}}_{o_t}^\top + \left|\hat{\boldsymbol{p}}_{o_t}^\top - \hat{\boldsymbol{p}}_{o_t,m_t}^\top\right|\right)\boldsymbol{V}^{\mathcal{L}}\right| \\
&\overset{(B.0.2)}{\le} \left|\boldsymbol{p}_{o_t}^\top \boldsymbol{V} - \boldsymbol{p}_{o_t,m_t}^\top \boldsymbol{V}\right| + \left|\hat{\boldsymbol{p}}_{o_t}^\top \boldsymbol{V}^{\mathcal{L}} - \hat{\boldsymbol{p}}_{o_t,m_t}^\top \boldsymbol{V}^{\mathcal{L}}\right| + \left|\boldsymbol{p}_{o_t}^\top \boldsymbol{V} - \hat{\boldsymbol{p}}_{o_t}^\top \boldsymbol{V}^{\mathcal{L}}\right| \\
&\overset{(6.1.3)}{\le} 2\Gamma r_{\mathsf{max}} \mathcal{H}(m_t,\sigma_t) + \left|\boldsymbol{p}_{o_t}^\top \boldsymbol{V} - \hat{\boldsymbol{p}}_{o_t}^\top \boldsymbol{V}\right| + \left|\hat{\boldsymbol{p}}_{o_t}^\top \boldsymbol{V} - \hat{\boldsymbol{p}}_{o_t}^\top \boldsymbol{V}^{\mathcal{L}}\right| \\
&= 2\Gamma r_{\mathsf{max}} \mathcal{H}(m_t,\sigma_t) + \sum_{o_{t+1}\in\Omega} V^{\mathcal{L}}(\boldsymbol{b}_{t+1}) \left|\boldsymbol{p}_{s_t}^\top \boldsymbol{b}_t - \boldsymbol{p}_{s_t}^\top \boldsymbol{b}_{l_t}\right| + \left|\hat{\boldsymbol{p}}_{o_t}^\top \boldsymbol{V} - \hat{\boldsymbol{p}}_{o_t}^\top \boldsymbol{V}^{\mathcal{L}}\right| \\
&\le 2\Gamma r_{\mathsf{max}} \mathcal{H}(m_t,\sigma_t) + \Gamma r_{\mathsf{max}} \Omega \left|\boldsymbol{p}_{s_t}^\top \boldsymbol{b}_t - \boldsymbol{p}_{s_t}^\top \boldsymbol{b}_{l_t}\right| + \left|\hat{\boldsymbol{p}}_{o_t}^\top \boldsymbol{V} - \hat{\boldsymbol{p}}_{o_t}^\top \boldsymbol{V}^{\mathcal{L}}\right| \\
&\overset{(6.1.5)}{\le} \Gamma r_{\mathsf{max}} \left(2\mathcal{H}(m_t,\sigma_t) + \Omega \mathcal{S}^t \sum_{k=0}^{t} \frac{1}{\mathcal{S}^k} \mathcal{H}(l_k,\sigma_k)\right) + \left|\hat{\boldsymbol{p}}_{o_t,m_t}^\top \boldsymbol{V} - \hat{\boldsymbol{p}}_{o_t,m_t}^\top \boldsymbol{V}^{\mathcal{L}}\right|
\end{aligned}
$$

The final recursive term $\left|\hat{\boldsymbol{p}}_{o_t,m_t}^\top \boldsymbol{V} - \hat{\boldsymbol{p}}_{o_t,m_t}^\top \boldsymbol{V}^{\mathcal{L}}\right|$ is expanded and used to give an upper bound to the full belief error in Lemma 6.1.8. □

Finally, using the results of Lemmas 6.1.6 and 6.1.7, Lemma 6.1.8 provides an upper bound to the belief approximation error:

Lemma 6.1.8. *The belief error $\left|Q^{\mathcal{L}}(\boldsymbol{b}_t,a_t) - Q^{\mathcal{L}}(\boldsymbol{b}_{l_t},a_t)\right|$ is bounded by the following expression* [10]*:*

$$
\begin{aligned}
&\left|Q^{\mathcal{L}}(\boldsymbol{b}_t,a_t) - Q^{\mathcal{L}}(\boldsymbol{b}_{l_t},a_t)\right| \le r_{max} \\
&\sum_{k=0}^{H-1} \gamma^k \left(2\mathcal{H}(n_{t+k},\sigma_{t+k}) + 2\gamma\Gamma\mathcal{H}(m_{t+k},\sigma_{t+k}) + (\mathcal{R}+\gamma\Gamma\Omega)\,\mathcal{S}^{t+k} \sum_{j=0}^{t+k} \frac{1}{\mathcal{S}^j}\mathcal{H}(l_j,\sigma_j)\right)
\end{aligned}
$$

Proof.

$$
\left|Q^{\mathcal{L}}(\boldsymbol{b}_t,a_t) - Q^{\mathcal{L}}(\boldsymbol{b}_{l_t},a_t)\right| \le \left|\boldsymbol{p}_{n_t}^\top \boldsymbol{r} - \hat{\boldsymbol{p}}_{n_t}^\top \boldsymbol{r}\right| + \gamma \left|\boldsymbol{p}_{o_t,m_t}^\top \boldsymbol{V} - \hat{\boldsymbol{p}}_{o_t,m_t}^\top \boldsymbol{V}^{\mathcal{L}}\right|
$$

using Lemmas 6.1.6 and 6.1.7:

$$
\begin{aligned}
\left|Q^{\mathcal{L}}(\boldsymbol{b}_t,a_t) - Q^{\mathcal{L}}(\boldsymbol{b}_{l_t},a_t)\right| &\le r_{\mathsf{max}} \left(2\mathcal{H}(n_t,\sigma_t) + 2\gamma\Gamma\mathcal{H}(m_t,\sigma_t) + (\mathcal{R}+\gamma\Gamma\Omega)\,\mathcal{S}^t \sum_{j=0}^{t} \frac{1}{\mathcal{S}^j}\mathcal{H}(l_j,\sigma_j)\right) \\
&\quad + \gamma \left|\hat{\boldsymbol{p}}_{o_t,m_t}^\top \boldsymbol{V} - \hat{\boldsymbol{p}}_{o_t,m_t}^\top \boldsymbol{V}^{\mathcal{L}}\right| \\
&\le r_{\mathsf{max}} \left(2\mathcal{H}(n_t,\sigma_t) + 2\gamma\Gamma\mathcal{H}(m_t,\sigma_t) + (\mathcal{R}+\gamma\Gamma\Omega)\,\mathcal{S}^t \sum_{j=0}^{t} \frac{1}{\mathcal{S}^j}\mathcal{H}(l_j,\sigma_j)\right) \\
&\quad + \gamma \sum_{o_{t+1}\in\Omega} P_{m_t}(o_{t+1}|\boldsymbol{b}_{l_t},a_t) \max_{a_{t+1}} \left|Q^{\mathcal{L}}(\boldsymbol{b}_{t+1},a_{t+1}) - Q^{\mathcal{L}}(\boldsymbol{b}_{l_{t+1}},a_{t+1})\right|
\end{aligned}
$$

[10] In this bound, and throughout the remaining of this section, it is assumed that every sampling operation on the same time-step t (be it reward, observation or belief-state sampling) has the same confidence interval $1-\sigma_t$.

let $\boldsymbol{\mu}_{t+1}$ and a'_{t+1} be a belief-state and an action, respectively, such that $\left|Q^{\mathcal{L}}(\boldsymbol{\mu}_{t+1}, a'_{t+1}) - Q^{\mathcal{L}}(\boldsymbol{\mu}_{l_{t+1}}, a'_{t+1})\right| \geq \left|Q^{\mathcal{L}}(\boldsymbol{b}_{t+1}, a_{t+1}) - Q^{\mathcal{L}}(\boldsymbol{b}_{l_{t+1}}, a_{t+1})\right|, \forall \boldsymbol{b}_{t+1}, a_{t+1}$. Then:

$$\begin{aligned}\left|Q^{\mathcal{L}}(\boldsymbol{b}_t, a_t) - Q^{\mathcal{L}}(\boldsymbol{b}_{l_t}, a_t)\right| \leq\ & \gamma \left|Q^{\mathcal{L}}(\boldsymbol{\mu}_{t+1}, a'_{t+1}) - Q^{\mathcal{L}}(\boldsymbol{\mu}_{l_{t+1}}, a'_{t+1})\right| \\ & + r_{\text{max}} \left(2\mathcal{H}(n_t, \sigma_t) + 2\gamma\Gamma\mathcal{H}(m_t, \sigma_t) + (\mathcal{R} + \gamma\Gamma\Omega)\, \mathcal{S}^t \sum_{j=0}^{t} \frac{1}{\mathcal{S}^j} \mathcal{H}(l_j, \sigma_j)\right)\end{aligned}$$

Using the inequality above recursively, and the stopping case from Equation (6.5):

$$\begin{aligned}&\left|Q^{\mathcal{L}}(\boldsymbol{b}_t, a_t) - Q^{\mathcal{L}}(\boldsymbol{b}_{l_t}, a_t)\right| \leq r_{\text{max}} \\ &\sum_{k=0}^{H-1} \gamma^k \left(2\mathcal{H}(n_{t+k}, \sigma_{t+k}) + 2\gamma\Gamma\mathcal{H}(m_{t+k}, \sigma_{t+k}) + (\mathcal{R} + \gamma\Gamma\Omega)\, \mathcal{S}^{t+k} \sum_{j=0}^{t+k} \frac{1}{\mathcal{S}^j} \mathcal{H}(l_j, \sigma_j)\right)\end{aligned}$$

□

Once again, let us count the number of Hoeffding inequalities used to prove this result. Analogously to the count done for Lemma 6.1.4, consider the *immediate* and *recursive* components. Notice that, for the immediate term, $(t+1)+2$ Hoeffding inequalities are used. The first time-step in the recursive term needs $\mathcal{A}\Omega\,((t+2)+2)$ Hoeffding inequalities (it should hold for every action and observation, and the $t+1$ Hoeffding inequalities from the belief-error bound now become $t+2$ because the time-step has increased by one). In total, $\sum_{k=0}^{H-1} \mathcal{A}^k\Omega^k\,(t+k+2)$ Hoeffding inequalities are needed to prove this bound.

Finally, with the two error terms of the lookahead already bounded, the full lookahead error can be derived:

Lemma 6.1.9. *The lookahead error ϵ_t has the following upper bound:*

$$\begin{aligned}\epsilon_t \leq r_{max} \sum_{k=0}^{H-1} \gamma^k \left(4\mathcal{H}(n_{t+k}, \sigma_{t+k}) + 4\gamma\Gamma\mathcal{H}(m_{t+k}, \sigma_{t+k}) + (\mathcal{R} + \gamma\Gamma\Omega)\, \mathcal{S}^{t+k} \sum_{j=0}^{t+k} \frac{1}{\mathcal{S}^j} \mathcal{H}(l_j, \sigma_j)\right) \\ + 2\gamma^H \Gamma r_{max}\end{aligned}$$

Proof. Lemma 6.1.2 yields:

$$\epsilon_t \leq \max_{a_t \in \mathcal{A}} \left(\left|Q^\star(\boldsymbol{b}_t, a_t) - Q^{\mathcal{L}}(\boldsymbol{b}_t, a_t)\right| + \left|Q^{\mathcal{L}}(\boldsymbol{b}_t, a_t) - Q^{\mathcal{L}}(\boldsymbol{b}_{l_t}, a_t)\right|\right) + \left|Q^\star(\boldsymbol{b}_{l_t}, a_t^{\mathcal{L}}) - Q^{\mathcal{L}}(\boldsymbol{b}_{l_t}, a_t^{\mathcal{L}})\right|$$

Applying Lemmas 6.1.4 and 6.1.8 to the previous expression concludes the proof. □

The error bound estimated in Lemma 6.1.9 encompasses all four errors of the quantum-classical lookahead algorithm. They can easily be identified from the expression in Lemma 6.1.9:

- *Approximating the reward distribution*: this error term is captured by the n_{t+k} term in the sum.
- *Approximating the observation distribution*: this error term is captured by the m_{t+k} term in the sum.
- *Approximating the belief states*: this error is captured by the l_j term of the bound and increases over time due to repeated approximation of the belief state.
- *The finite horizon of the lookahead*: the lookahead tree has a fixed depth, given by the finite horizon H, introducing a systematic error that can't be compensated by increasing the number of samples. It is captured by the $2\gamma^H \Gamma R_{\text{max}}$ term in the bound.

Lemma 6.1.9 provides an upper bound to the lookahead error of Definition 6.1.1. This upper bound takes into account the use of multiple sampling operations throughout the lookahead tree: reward sampling, observation sampling and belief-state sampling. Given that the bounds used for each sampling operator fail with a probability of σ_t, the bound for the whole lookahead error also has a non-zero chance of failure, that depends on the confidence interval of each sampling operation and is yet to be defined. This is because the confidence interval $1 - \sigma_t$ only holds locally for each sampling operation, and not for the whole expression presented in Lemma 6.1.9. Naturally, if none of the sampling operation bounds fail, the lookahead error bound holds. As such, the probability of failure δ_t of the lookahead error bound is at least as large as the probability that any one of the sampling bounds fails.

One way to relate the confidence interval $1-\delta_t$ of the bound of Lemma 6.1.9 to the confidence interval $1 - \sigma_t$ of each sampling operation is by using Theorem 6.1.10. It states that the probability of the union of multiple events is at least as small as the sum of the probabilities of each individual event:

Theorem 6.1.10. *(Boole's Inequality Seneta [1992]). Let $Z_1, Z_2, \ldots, Z_n$ be probabilistic events that occur with probability $P(Z_1), P(Z_2), \ldots, P(Z_n)$. Then:*

$$P\left(\bigcup_{i=1}^{n} Z_i\right) \leq \sum_{i=1}^{n} P(Z_i)$$

In the formulation of Theorem 6.1.10, the probabilistic events Z_i could have a higher sampling error than the one provided in Theorem 6.1.3. By attributing this meaning to the probabilistic events, Boole's inequality can be used to relate the sampling bound's confidence interval $1-\sigma_t$ to the confidence interval of the algorithm's upper bound presented in Lemma 6.1.9.

Lemma 6.1.11. *The confidence interval $1-\delta_t$ of the algorithm's upper bound given in Lemma 6.1.9 and the confidence interval $1-\sigma_t$ of the Hoeffding's bound for each sampling operation obey the following inequality:*

$$\delta_t \leq \sum_{i=0}^{H-1} \mathcal{A}^i \Omega^i \left(2 + \mathcal{A}\left(t+i+4\right)\right) \sigma_{t+i}$$

Proof. Recall, from Lemma 6.1.2, that the lookahead error is bounded by the following expression:

$$\epsilon_t \leq \max_{a_t \in \mathcal{A}} \left(\left| Q^\star(\boldsymbol{b}_t, a_t) - Q^\mathcal{L}(\boldsymbol{b}_t, a_t) \right| + \left| Q^\mathcal{L}(\boldsymbol{b}_t, a_t) - Q^\mathcal{L}(\boldsymbol{b}_{l_t}, a_t) \right| \right) + \left| Q^\star(\boldsymbol{b}_{l_t}, a_t^\mathcal{L}) - Q^\mathcal{L}(\boldsymbol{b}_{l_t}, a_t^\mathcal{L}) \right|$$

The terms $\left|Q^\star(\boldsymbol{b}_t, a_t) - Q^\mathcal{L}(\boldsymbol{b}_t, a_t)\right|$ and $\left|Q^\star(\boldsymbol{b}_{l_t}, a_t^\mathcal{L}) - Q^\mathcal{L}(\boldsymbol{b}_{l_t}, a_t^\mathcal{L})\right|$ can both be bounded using Lemma 6.1.4, which needs $2\sum_{i=0}^{H-1} \mathcal{A}^i\Omega^i$ Hoeffding inequalities to be derived. The term $\left|Q^\mathcal{L}(\boldsymbol{b}_t, a_t) - Q^\mathcal{L}(\boldsymbol{b}_{l_t}, a_t)\right|$ can be bounded using Lemma 6.1.8, which requires $\sum_{i=0}^{H-1} \mathcal{A}^i\Omega^i(t+i+2)$ Hoeffding inequalities. However, both the terms $\left|Q^\star(\boldsymbol{b}_t, a_t) - Q^\mathcal{L}(\boldsymbol{b}_t, a_t)\right|$ and $\left|Q^\mathcal{L}(\boldsymbol{b}_t, a_t) - Q^\mathcal{L}(\boldsymbol{b}_{l_t}, a_t)\right|$ are under a $\max_{a_t \in \mathcal{A}}$ operation in the lookahead error bound above, so their number of Hoeffding inequalities should be multiplied by a factor $\mathcal{A}$, since these bounds should hold for any possible action. As such, the total number of Hoeffding inequalities to bound the lookahead error is given by:

$$2\sum_{i=0}^{H-1} \mathcal{A}^i\Omega^i + 2\mathcal{A}\sum_{i=0}^{H-1} \mathcal{A}^i\Omega^i + \mathcal{A}\sum_{i=0}^{H-1} \mathcal{A}^i\Omega^i(t+i+2) = \sum_{i=0}^{H-1} \mathcal{A}^i\Omega^i \left(2 + \mathcal{A}\left(t+i+4\right)\right)$$

Each term of the previous sum represents the total number of Hoeffding inequalities at that particular depth in the lookahead tree (if $i=2$, the corresponding term represents the number of Hoeffding inequalities at depth 2). Given that, for a depth i at time-step t, the probability of a sampling operation failing is σ_{t+1}, Boole's inequality in Theorem 6.1.10 yields:

$$\delta_t \leq \sum_{i=0}^{H-1} \mathcal{A}^i \Omega^i \left(2 + \mathcal{A}\left(t+i+4\right)\right) \sigma_{t+i}$$

□

The bounds derived in Lemmas 6.1.9 and 6.1.11 depend on the definition of four successions:

- n_t, m_t and l_t: the successions that define how the number of samples for each sampling operation varies over time.

- σ_t: the succession defining how the confidence interval of each sampling operation varies over time.

Unfortunately, due to the cumulative nature of the belief-state error, the lookahead error grows exponentially as time progresses (it grows with $\mathcal{S}^t$, and the state size $\mathcal{S}$ of the POMDP can be extremely large by itself!). This effect could be counteracted by an appropriate choice for both l_t and σ_t, but this work does not try to answer this question. Instead, a much simpler case is considered: both n_t, m_t, l_t and σ_t are fixed, such that they do not change over time, as stated in Definition 6.1.12.

Definition 6.1.12. *The lookahead algorithm's samples n_t, m_t and l_t and the confidence interval $1 - \sigma_t$ are considered to be time-invariant:*

$$n_t = n, \quad m_t = m, \quad l_t = l, \quad \sigma_t = \sigma$$

Moreover, a further simplification is assumed to reduce the degrees of freedom in the choice of the number of samples: the contribution of each sampling operation to the whole lookahead error at time-step $t = 0$ is assumed to be the same, such that the number of samples n, m and l can be related to each other according to Lemma 6.1.13:

Lemma 6.1.13. *If the contribution to the error bound of Lemma 6.1.9 of each sampling operation is considered to be the same at time-step $t = 0$, the number of samples can be related to each other according to the following expressions:*

$$m = (\gamma\Gamma)^2 n, \quad l = \left(\frac{1}{4}\frac{\mathcal{R} + \gamma\Gamma\Omega}{\mathcal{S} - 1}\left(\frac{\mathcal{S}}{\Gamma\left(1 - \gamma^H\right)}\frac{(\gamma\mathcal{S})^H - 1}{\gamma\mathcal{S} - 1} - 1\right)\right)^2 n$$

Proof. According to Lemma 6.1.9 and Definition 6.1.12, the lookahead error at time $t = 0$ has the following upper-bound:

$$\epsilon_0 \leq r_{\text{max}} \sum_{k=0}^{H-1} \gamma^k \left(4\mathcal{H}(n,\sigma) + 4\gamma\Gamma\mathcal{H}(m,\sigma) + (\mathcal{R} + \gamma\Gamma\Omega)\,\mathcal{S}^k\mathcal{H}(l,\sigma)\sum_{j=0}^{k}\frac{1}{\mathcal{S}^j}\right)$$
$$+ 2\gamma^H\Gamma r_{\text{max}} = S_n + S_m + S_l + 2\gamma^H\Gamma r_{\text{max}}$$

where S_n, S_m and S_l represent the sums over the number of samples n, m and l, respectively. For their contributions to be the same, as stated in Lemma 6.1.13, all of these sums must be equal ($S_n = S_m = S_l$).

Letting $S_n = S_m$[11] yields:

$$4r_{\text{max}}\mathcal{H}(n,\sigma)\sum_{k=0}^{H-1}\gamma^k = 4\gamma\Gamma r_{\text{max}}\mathcal{H}(m,\sigma)\sum_{k=0}^{H-1}\gamma^k$$
$$\sqrt{n^{-1}} = \gamma\Gamma\sqrt{m^{-1}}$$
$$m = (\gamma\Gamma)^2 n$$

Finally, making $S_n = S_l$ and resorting to Lemma B.0.3:

$$\begin{aligned}
4r_{\text{max}}\mathcal{H}(n,\sigma)\sum_{k=0}^{H-1}\gamma^k =& r_{\text{max}}\mathcal{H}(l,\sigma)\left(\mathcal{R}+\gamma\Gamma\Omega\right)\sum_{k=0}^{H-1}\gamma^k\mathcal{S}^k\sum_{j=0}^{k}\frac{1}{\mathcal{S}^j}\\
4\sqrt{n^{-1}}\sum_{k=0}^{H-1}\gamma^k \overset{(B.0.3)}{=}& \sqrt{l^{-1}}\left(\mathcal{R}+\gamma\Gamma\Omega\right)\sum_{k=0}^{H-1}\gamma^k\frac{\mathcal{S}^{k+1}-1}{\mathcal{S}-1}\\
4\sqrt{n^{-1}}\Gamma\left(1-\gamma^H\right) =& \sqrt{l^{-1}}\left(\mathcal{R}+\gamma\Gamma\Omega\right)\frac{1}{\mathcal{S}-1}\left(\mathcal{S}\sum_{k=0}^{H-1}(\gamma\mathcal{S})^k-\sum_{k=0}^{H-1}\gamma^k\right)\\
4\sqrt{n^{-1}}\Gamma\left(1-\gamma^H\right) \overset{(B.0.3)}{=}& \sqrt{l^{-1}}\left(\mathcal{R}+\gamma\Gamma\Omega\right)\frac{1}{\mathcal{S}-1}\left(\mathcal{S}\frac{(\gamma\mathcal{S})^H-1}{\gamma\mathcal{S}-1}-\Gamma\left(1-\gamma^H\right)\right)\\
l =& \left(\frac{1}{4}\frac{\mathcal{R}+\gamma\Gamma\Omega}{\mathcal{S}-1}\left(\frac{\mathcal{S}}{\Gamma\left(1-\gamma^H\right)}\frac{(\gamma\mathcal{S})^H-1}{\gamma\mathcal{S}-1}-1\right)\right)^2 n
\end{aligned}$$

□

Given Definition 6.1.12 and Lemma 6.1.13, the bounds in Lemmas 6.1.9 and 6.1.11 can be re-written as follows:

Theorem 6.1.14. *The algorithm's error ϵ_t has the following upper bound, with a confidence interval of $1-\sigma_t$:*

$$\epsilon_t \leq r_{max}\sqrt{\frac{1}{2n}\log\left(\frac{2}{\sigma}\right)}\left(8\Gamma+4\mathcal{S}^t\right)+2\gamma\Gamma r_{max}$$
$$\delta_t \leq \sigma\left(\mathcal{A}\Omega\right)^H\left(t+\mathcal{A}\left(8+\Omega\frac{H-1}{(\mathcal{A}\Omega-1)^2}\right)\right)$$

Proof. The bound on the error ϵ_t, follows directly from Lemma 6.1.9, by substituting m and l using the expressions in Lemma 6.1.13.

[11] Recall that $\mathcal{H}(n,\sigma) = \sqrt{\frac{1}{2n}\log\left(\frac{2}{\sigma}\right)}$

As for the bound for the confidence δ_t, it follows from Lemma 6.1.11 that:

$$\begin{aligned}\delta_t &\leq \sum_{i=0}^{H-1} \mathcal{A}^i\Omega^i \left(2+\mathcal{A}\left(t+i+4\right)\right)\sigma_{t+i}\\ &= \sigma\sum_{i=0}^{H-1} \mathcal{A}^i\Omega^i \left(2+\mathcal{A}\left(t+4\right)+i\mathcal{A}\right)\end{aligned}$$

By using the geometric series of Lemma B.0.3 on the $2+\mathcal{A}\left(t+4\right)$ multiplicative term in the sum:

$$\begin{aligned}\delta_t &\overset{(B.0.3)}{\leq} \sigma\left(\mathcal{A}\sum_{i=0}^{H-1} i\mathcal{A}^i\Omega^i + \left(2+\mathcal{A}\left(t+4\right)\right)\frac{\left(\mathcal{A}\Omega\right)^H-1}{\mathcal{A}\Omega-1}\right)\\ &\leq \sigma\left(\mathcal{A}\sum_{i=0}^{H-1} i\mathcal{A}^i\Omega^i + \left(2+\mathcal{A}\left(t+4\right)\right)\left(\mathcal{A}\Omega\right)^H\right)\end{aligned}$$

Analogously applying a variation of the finite geometric series given by Lemma B.0.4 to the sum $\sum_{i=0}^{H-1} i\mathcal{A}^i\Omega^i$, the bound becomes:

$$\begin{aligned}\delta_t &\overset{(B.0.4)}{\leq} \sigma\left(\mathcal{A}\frac{\left(H-1\right)\left(\mathcal{A}\Omega\right)^{H+1}-H\left(\mathcal{A}\Omega\right)^H+\mathcal{A}\Omega}{\left(\mathcal{A}\Omega-1\right)^2} + \left(2+\mathcal{A}\left(t+4\right)\right)\left(\mathcal{A}\Omega\right)^H\right)\\ &\leq \sigma\left(\mathcal{A}\frac{\left(H-1\right)\left(\mathcal{A}\Omega\right)^{H+1}}{\left(\mathcal{A}\Omega-1\right)^2} + \left(2+\mathcal{A}\left(t+4\right)\right)\left(\mathcal{A}\Omega\right)^H\right)\\ &\leq \sigma\left(\mathcal{A}^2\Omega\right)^H\left(\mathcal{A}\Omega\frac{\left(H-1\right)}{\left(\mathcal{A}\Omega-1\right)^2}+\mathcal{A}\left(t+4\right)+2\right)\\ &\leq \sigma\left(\mathcal{A}\Omega\right)^H\left(2+\mathcal{A}\left(t+4+\mathcal{A}\Omega\frac{H-1}{\left(\mathcal{A}\Omega-1\right)^2}\right)\right)\end{aligned}$$

□

As can be seen in Theorem 6.1.14, both the bounds for ϵ_t and δ_t grow over time. Suppose that the designed algorithm accepts three arguments as input: the maximum error ϵ, the minimum confidence interval $1-\delta$ and the stopping time T for the decision making algorithm. In this scenario, both ϵ_t and δ_t will have a maximum bound for $t=T$. To ensure the error ϵ and the confidence interval $1-\delta$, Theorem 6.1.14 can be re-expressed to derive a lower-bound for the number of samples and an upper bound for σ (the probability of failure for the sampling bounds):

$$\begin{aligned}n &\geq \frac{1}{2}\log\left(\frac{2}{\sigma}\right)\left(\frac{r_{\text{max}}}{\epsilon-2\gamma^H\Gamma r_{\text{max}}}\left(8\Gamma+4\mathcal{S}^T\right)\right)^2\\ \sigma &\leq \delta\left(\mathcal{A}\Omega\right)^{-H}\left(2+\mathcal{A}\left(T+4+\mathcal{A}\Omega\frac{H-1}{\left(\mathcal{A}\Omega-1\right)^2}\right)\right)^{-1}\end{aligned} \quad (6.7)$$

The above conditions, however, are not enough to ensure the error ϵ. This is because no matter how many samples one extracts for approximating the probability distributions, the number of samples can't compensate for the error of performing a finite lookahead if the chosen horizon H is too shallow [12]. As such, it should also be assured that the lookahead horizon H is large enough to make the desired error ϵ attainable. This condition is met when $2\gamma^H\Gamma r_{\text{max}} < \epsilon$, such that:

$$H > \log_\gamma\left(\frac{\epsilon}{2\Gamma r_{\text{max}}}\right) \tag{6.8}$$

The three bounds presented in Equations (6.7) and (6.8) sums up the whole sample complexity analysis of the lookahead algorithm. They are captured in Theorem 6.1.15, the main theorem of this analisys:

Theorem 6.1.15. *Consider a stopping time T, horizon H and Definition 6.1.12 and Lemma 6.1.13 for the lookahead algorithm. A near optimal action is guaranteed to be taken at each time-step of the decision making process with a regret of at most ϵ and confidence interval of at least $1-\delta$ if all of the following conditions are met:*

$$n \geq \frac{1}{2}\log\left(\frac{2}{\sigma}\right)\left(\frac{r_{max}}{\epsilon - 2\gamma^H\Gamma r_{max}}\left(8\Gamma + 4\mathcal{S}^T\right)\right)^2$$
$$\sigma \leq \delta\left(\mathcal{A}\Omega\right)^{-H}\left(2 + \mathcal{A}\left(T + 4 + \mathcal{A}\Omega\frac{H-1}{\left(\mathcal{A}\Omega - 1\right)^2}\right)\right)^{-1}$$
$$H > \log_\gamma\left(\frac{\epsilon}{2\Gamma r_{max}}\right)$$

6.2 Computational complexity

To calculate the computational complexity of the lookahead algorithm, in both the classical and quantum-classical versions, one must multiply the total number of samples required by the computational complexity necessary to compute each sample. Given that the computational complexity for each sample has already been defined in Section 4.4, the only task left is to define the total number of samples that need to be extracted.

In order to do so, it is important to recall that there are three types of sampling operations that the algorithm performs:

- *Reward sampling*: this sampling operation is performed at every observation node and requires n samples.
- *Observation sampling*: performed at every observation node and requires m samples.

[12] Using a finite lookahead introduces an error of $2\gamma\Gamma r_{\text{max}}$, as in Theorem 6.1.14.

- *Belief sampling*: performed at every belief node and requires l samples.

As such, the total number of samples N_s can be defined in terms of the number of belief nodes N_b and the number of observation nodes N_o as:

$$N_s = lN_b + (n+m)\,N_o$$

where the number of belief nodes is computed as

$$N_b = \sum_{i=0}^{H-1} (\mathcal{A}\Omega)^i = \frac{(\mathcal{A}\Omega)^H - 1}{\mathcal{A}\Omega - 1}$$

and the number of observation nodes as

$$N_o = \mathcal{A}\sum_{i=0}^{H-1} (\mathcal{A}\Omega)^i = \mathcal{A}\frac{(\mathcal{A}\Omega)^H - 1}{\mathcal{A}\Omega - 1}$$

such that the total number of samples can be re-expressed as

$$N_s = \frac{(\mathcal{A}\Omega)^H - 1}{\mathcal{A}\Omega - 1}\left(l + \mathcal{A}\,(n+m)\right)$$

Let C_n, C_m and C_l represent the computational complexity of extracting each of the three types of samples considered. In asymptotic terms, where $\mathcal{S}$, $\mathcal{A}$, $\mathcal{R}$, Ω, H and T are all assumed to be very large, the computational complexity of the algorithm can be expressed as:

$$\mathcal{O}\left(\mathcal{A}^{H-1}\Omega^{H-1}\left(lC_l + \mathcal{A}\left(nC_n + mC_m\right)\right)\right)$$

Recalling the relation between the number of samples n, m and l established in Lemma 6.1.13, this complexity becomes:

$$\begin{aligned}&\mathcal{O}\left(n\mathcal{A}^{H-1}\Omega^{H-1}\left(\left(\frac{1}{4}\frac{\mathcal{R}+\gamma\Gamma\Omega}{\mathcal{S}-1}\left(\frac{\mathcal{S}}{\Gamma\left(1-\gamma^H\right)}\frac{(\gamma\mathcal{S})^H - 1}{\gamma\mathcal{S}-1} - 1\right)\right)^2 C_l + \mathcal{A}\left(C_n + \gamma^2\Gamma^2 C_m\right)\right)\right)\\ &= \mathcal{O}\left(n\mathcal{A}^{H-1}\Omega^{H-1}\left(\mathcal{A}\left(C_n + C_m\right) + \left(\left(\mathcal{R}+\Omega\right)\mathcal{S}^{H-1}\right)^2 C_l\right)\right)\end{aligned} \tag{6.9}$$

To substitute the computational complexities C_n, C_m and C_l, a distinction has to be made for the classical and quantum-classical algorithms, since they use different inference algorithms to extract probability distributions.

In the classical case, direct sampling is used for performing inference on the rewards and observations, while belief updating uses rejection sampling. The computational complexities are of order $\mathcal{O}\left(NM\right)$ for direct sampling and $\mathcal{O}\left(NMP(e)^{-1}\right)$ for rejection sampling.

What does the evidence stand for in the case of rejection sampling? As this operation is performed at every observation node in the lookahead tree, to account for all these sampling operations, it is the mean inverse probability of evidences for belief updating. Since belief updating only uses observation RVs as evidence, then it is given by $c_l = \sum_{o_{t+1}} P(o_{t+1}|\boldsymbol{b}_t, a_t)^{-1}$, where the sum is over all observations in the lookahead tree, conditioned on the previous belief state $\boldsymbol{b}_t$ and previous action a_t.

For the quantum-classical case, direct sampling is still used for reward and observation sampling, having the same complexity of the classical case. Quantum rejection sampling is, however, applied for belief updating. Therefore, in the quantum-classical case, the complexity for C_l is of the order $\mathcal{O}\left(N2^M q_l\right)$, where $q_l = P(e)^{-\frac{1}{2}} = \sum_{o_{t+1}} P(o_{t+1}, \boldsymbol{b}_t, a_t)^{-\frac{1}{2}}$. In this scenario, c_l is no longer adequate to represent $P(e)^{-\frac{1}{2}}$, because the latter is an average of the inverse square rooted probabilities $P\left(o_{t+1}|\boldsymbol{b}_t, a_t\right)^{-\frac{1}{2}}$ and not of the inverse probabilities $P\left(o_{t+1}|\boldsymbol{b}_t, a_t\right)^{-1}$.

It is also important to mention that the complexity of the algorithm can be written both in terms of the number of samples n, or in terms of the precision ϵ and confidence interval $1-\delta$, since they can be related according to Theorem 6.1.15:

$$\mathcal{O}\left(n\right) \leq \mathcal{O}\left(\left(\frac{\mathcal{S}^T}{\epsilon}\right)^2 \log\left(\frac{1}{\sigma}\right)\right) \leq \mathcal{O}\left(\left(\frac{\mathcal{S}^T}{\epsilon}\right)^2 \log\left(\left(T+\frac{H}{\mathcal{A}\Omega}\right)\frac{\mathcal{A}^{H+1}\Omega^H}{\delta}\right)\right) \tag{6.10}$$

Proof. This result is directly attained with Equations (6.9) and (6.10) and using $C_n = C_m = \mathcal{O}(NM)$ and $C_l = \mathcal{O}\left(N2^M c_l\right)$. □

In this context, Theorem 6.2.1 characterizes the computational complexity of the classical lookahead algorithm:

Theorem 6.2.1. *The computational complexity of the classical lookahead algorithm is given by any of the following expressions:*

$$\mathcal{O}\left(nNM\mathcal{A}^{H-1}\Omega^{H-1}\left(\mathcal{A}+\left(\left(\mathcal{R}+\Omega\right)\mathcal{S}^{H-1}\right)^2 c_l\right)\right)$$
$$\mathcal{O}\left(NM\mathcal{A}^{H-1}\Omega^{H-1}\left(\mathcal{A}+\left(\left(\mathcal{R}+\Omega\right)\mathcal{S}^{H-1}\right)^2 c_l\right)\left(\frac{\mathcal{S}^T}{\epsilon}\right)^2 \log\left(\left(T+\frac{H}{\mathcal{A}\Omega}\right)\frac{\mathcal{A}^{H+1}\Omega^H}{\delta}\right)\right)$$

Proof. This result is directly attained with Equations (6.9) and (6.10) and using $C_n = C_m = \mathcal{O}(NM)$ and $C_l = \mathcal{O}\left(N2^M q_l\right)$. □

Theorem 6.2.2 is analogous to the previous theorem, but characterizes the computational complexity of the quantum-classical lookahead algorithm:

Theorem 6.2.2. *The computational complexity of the quantum-classical lookahead algorithm is given by any of the following expressions, whenever $2^M \approx M$:*

$$\mathcal{O}\left(nNM\mathcal{A}^{H-1}\Omega^{H-1}\left(\mathcal{A}+\left((\mathcal{R}+\Omega)\,\mathcal{S}^{H-1}\right)^2 q_l\right)\right)$$
$$\mathcal{O}\left(NM\mathcal{A}^{H-1}\Omega^{H-1}\left(\mathcal{A}+\left((\mathcal{R}+\Omega)\,\mathcal{S}^{H-1}\right)^2 q_l\right)\left(\frac{\mathcal{S}^T}{\epsilon}\right)^2 \log\left(\left(T+\frac{H}{\mathcal{A}\Omega}\right)\frac{\mathcal{A}^{H+1}\Omega^H}{\delta}\right)\right)$$

As can be seen from Theorems 6.2.1 and 6.2.2, a partial speedup is obtained in the quantum case for the belief-state sampling operations, while the computational complexity of the rest of the algorithm is similar across both classical and quantum algorithms. This is so whenever the considered dynamic decision networks is sparse enough (such that $2^M \approx M$), just as in the case of quantum rejection sampling. However, for some specific situations, the portion of the algorithm that uses a quantum subroutine dominates the complexity of the algorithm so much, that the contribution of the classical subroutines to the computational complexity of the algorithm is negligible. In these scenarios, detailed in Corollaries 6.2.3 and 6.2.4, the quantum-classical lookahead has the greatest possible speedup over its classical counterpart:

Corollary 6.2.3. *Under the assumption that $\frac{1}{q_l} \ll \frac{\left((\mathcal{R}+\Omega)\mathcal{S}^{H-1}\right)^2}{\mathcal{A}}$, the expressions for computational complexity of the classical lookahead algorithm of Theorem 6.2.1 can be re-written as:*

$$\mathcal{O}\left(nNM\mathcal{A}^{H-1}\Omega^{H-1}\left((\mathcal{R}+\Omega)\,\mathcal{S}^{H-1}\right)^2 c_l\right)$$
$$\mathcal{O}\left(NM\mathcal{A}^{H-1}\Omega^{H-1}\left((\mathcal{R}+\Omega)\,\frac{\mathcal{S}^{T+H-1}}{\epsilon}\right)^2 c_l \log\left(\left(T+\frac{H}{\mathcal{A}\Omega}\right)\frac{\mathcal{A}^{H+1}\Omega^H}{\delta}\right)\right)$$

Proof. Given that $q_l \leq c_l$, it follows that $\frac{1}{c_l} \leq \frac{1}{q_l}$, and therefore $\frac{1}{q_l} \ll \frac{\left((\mathcal{R}+\Omega)\mathcal{S}^{H-1}\right)^2}{\mathcal{A}} \Rightarrow \frac{c_n}{c_l} \ll \frac{\left((\mathcal{R}+\Omega)\mathcal{S}^{H-1}\right)^2}{\mathcal{A}}$. Thus, the result of this corollary follows directly from Theorem 6.2.1. $\square$

Corollary 6.2.4. *Under the assumption that $\frac{c_n}{q_l} \ll \frac{\left((\mathcal{R}+\Omega)\mathcal{S}^{H-1}\right)^2}{\mathcal{A}}$, the expressions for the computational complexity of the quantum-classical lookahead algorithm of Theorem 6.2.2 can be re-written as:*

$$\mathcal{O}\left(nN2^M\mathcal{A}^{H-1}\Omega^{H-1}\left((\mathcal{R}+\Omega)\,\mathcal{S}^{H-1}\right)^2 q_l\right)$$
$$\mathcal{O}\left(N2^M\mathcal{A}^{H-1}\Omega^{H-1}\left((\mathcal{R}+\Omega)\,\frac{\mathcal{S}^{T+H-1}}{\epsilon}\right)^2 q_l \log\left(\left(T+\frac{H}{\mathcal{A}\Omega}\right)\frac{\mathcal{A}^{H+1}\Omega^H}{\delta}\right)\right)$$

Corollaries 6.2.3 and 6.2.4 compares the classical and quantum algorithm's complexity in the best case scenario for the quantum algorithm, where $\frac{c_n}{q_l} \ll \frac{\left((\mathcal{R}+\Omega)\mathcal{S}^{H-1}\right)^2}{\mathcal{A}}$. It remains to be answered as to what RL problems this assumption can be applied to in practice, as this work does not address this question. Nonetheless, it gives a benchmark as to what speedup one can expect from the quantum algorithm in a best case scenario.

What speedups can therefore be expected? Although one might initially think that using quantum rejection sampling would provide a quadratic speedup, this is not always the case. Dividing the classical complexity by the quantum complexity (and assuming that M and 2^M have about the same order of magnitude), this fraction reduces to $\frac{c_l}{q_l} = \frac{\sum_x P^{-1}(x)}{\sum_x P^{-\frac{1}{2}}(x)}$, where the sum over x represents the sum over the different evidences. From inequality $\sqrt{\sum_x a_x} \leq \sum_x \sqrt{a_x} \leq \sum_x a_x$, it is easy to derive that:

$$1 \leq \frac{c_l}{q_l} \leq q_l \tag{6.11}$$

The inequality in Equation (6.11) entails that the complexity of the quantum algorithm relative to the classical one can range between no speedup to a quadratic one, but it is most likely to fall somewhere in between these two extremes. This comes from the notion that, although using quantum rejection sampling yields a quadratic advantage over classical rejection sampling, the same cannot be said about using it sequentially, due to the mathematical property that the sum of square roots can be (and usually is) larger than the square root of the sum.

6.3 Summary

The first section of this Chapter contains a detailed analysis of the sample complexity of the lookahead algorithm, giving conditions under which there are theoretical guarantees for the algorithm to run with an error of at most ϵ, and a confidence interval of at least $1 - \delta$. This analysis is applicable to both the classical and quantum algorithms, as the number of samples that need to be extracted is the same in both cases.

The second and final section of this Chapter uses the sample complexity analysis to determine the computational complexity of both algorithms. In this analysis, it is possible to conclude that, despite the use of a quantum subroutine with a quadratic speedup over the classical counterpart, the complexity of quantum algorithm when compared to the classical one can range between no speedup to a quadratic one, most of the times falling somewhere in between.

Chapter 7

Experimental results and discussion

In Chapter 6, it is shown that the quantum lookahead algorithm has a time complexity advantage over its classical counterpart, which depends on the problem it is applied to.

This Chapter leverages this theoretically proven speedup to experimentally verify whether this advantage is noticeable on agents applied to a range of different partially observable RL problems.

7.1 Comparison metrics

In Sections 3.4 and 5.3, two similar lookahead algorithms were discussed. Both use a tree structure to brute-search every possible situation an agent can encounter in its environment, a few time-steps in the future (the horizon). Leveraging this information, the algorithms then choose an action that maximizes the expected return in those time-steps. To perform these algorithms, subroutines for performing sampling operations are needed, and this is where the two algorithm's differ: one uses classical rejection sampling, while the other uses quantum rejection sampling.

As derived in Chapter 6, the quantum lookahead algorithm can have a time complexity advantage over its classical counterpart, but what kind of benefits can this time complexity advantage yield for the quantum algorithm? This chapter tries to experimentally elucidate this question by comparing the performance of the two algorithms.

Consider that both algorithms are applied to the same problem, with the restriction that each time-step must be calculated in a finite amount of time (the same for both algorithms). Given that the quantum version has a possibly lower time-complexity, this implies that it should be able to be executed with an increased number of samples, improving the precision of the calculations, consequently also improving its decision making capabilities.

By dividing the time complexities of the classical lookahead algorithm in Corollary 6.2.3 by the quantum lookahead algorithm in Corollary 6.2.4, the following expression is obtained:

$$\frac{n_c N M \mathcal{A}^{H-1}\Omega^{H-1}\left(\mathcal{R}+\Omega\right)^2 c_l}{n_q N 2^M \mathcal{A}^{H-1}\Omega^{H-1}\left(\mathcal{R}+\Omega\right)^2 q_l} = \frac{n_c M c_l}{n_q 2^M q_l}$$

Under the assumption that both these algorithm take the same time to finish, the fraction above must be equal to one. With the further assumption that the number of parents M in the DDN is small, meaning that $2^M \approx M$, then the following relation between the number of quantum samples n_q and the number of classical samples n_c can be established as:

$$n_q = \frac{c_l}{q_l} n_c \tag{7.1}$$

Calculating $\frac{c_l}{q_l}$ implies calculating every probability $P(o_{t+1}|\boldsymbol{b}_t, a_t)$ in the lookahead tree, as descrided in Section 6.2. Notice that, as the lookahead algorithm is executed over multiple time-steps, and the belief state of the agent is updated over time, $\frac{c_l}{q_l}$ also varies over time (because the belief states in the conditioning of $P(o_{t+1}|\boldsymbol{b}_t, a_t)$ also change). Therefore, to get the results for the experiments, the number of quantum samples is calculated at each time-step using Equation (7.1). Since the ratio $\frac{c_l}{q_l}$ depends on both the POMDP of the problem and the current belief state of the agent, calculating it at every time-step ensures that the adequate quantum advantage in the number of samples is always given for the quantum lookahead algorithm.

To compare the classical and quantum algorithms, the chosen metric is the expected cumulative reward of the agent. This metric quantifies the amount of reward that the agent has been able to collect, on average, since the beginning of each experiment. Supposing that an agent starts at time 0 and is now at time t, its expected cumulative reward is given by:

$$\sum_{i=0}^{t} \mathbb{E}\left(R_{i+1}|\boldsymbol{b}_i, a_i\right) \tag{7.2}$$

Agents that perform better have a higher expected cumulative reward than others, and therefore the advantage of agents using the quantum classical lookahead should be reflected on this metric.

Finally, the two algorithms can also be executed with the same number of samples. If this is the case, then the quantum approach should have a time advantage due to its decrease time complexity. Since the quantum algorithm has to be simulated (because quantum rejection sampling is not a NISQ friendly algorithm, as explained in Section 4.4), the execution time can not be calculated experimentally, at least in a direct manner. This would be unfair to the quantum algorithm, as simulating it is more costly than actually using a quantum device to execute it. Therefore, the times for executing the algorithms are calculated using the time-complexities derived in Section 6.2 and the experimental values for c_l and q_l on each time-step.

To summarize, two different metrics are used to compare the two algorithms:

- The cumulative expected reward, under the condition that both the classical and quantum algorithms take the same time to calculate decisions.
- The time taken to execute the classical and quantum algorithms, under the condition that they both use the same number of samples.

7.2 Experiments

To compare the two lookahead algorithms, some small problems were selected. Firstly, this was due to the lack of computational power to run big problems. Secondly, because the quantum algorithms were run in a quantum simulator using qiskit, and the cost of simulation grows very quickly with the increase on the number of qubits. Using a simulator is preferred because today's NISQ devices are still very noisy to run quantum circuits as deep as the ones considered here. As such, the following experiments were chosen to compare the quantum and classical lookahead algorithms:

- Tiger problem
- Robot exploration problem

7.2.1 Tiger problem

The tiger problem is one of the simplest POMDPs that can be conceived. There are two doors, where behind one of them is a tiger and behind the other a treasure (see Figure 17). The agent must choose between three possible actions: pick the left door, the right door or listen to try and figure out in what door the tiger is. If the agent finds the treasure, it gets a reward of 5, whereas if he picks the tiger he gets a reward of -10. Listening is not free, as it also comes with a negative reward of -1. Furthermore, listening does not always give away the position of the tiger correctly: it fails 15% of the time.

After the agent has chosen a door, the tiger and treasure are randomly re-assigned to different doors to make the decision process continue. In this problem, the sets of states, actions, observations and rewards are the following:

- $\mathcal{S} = \{\text{tiger left}, \text{tiger right}\}$
- $\mathcal{A} = \{\text{left door}, \text{right door}, \text{listen}\}$

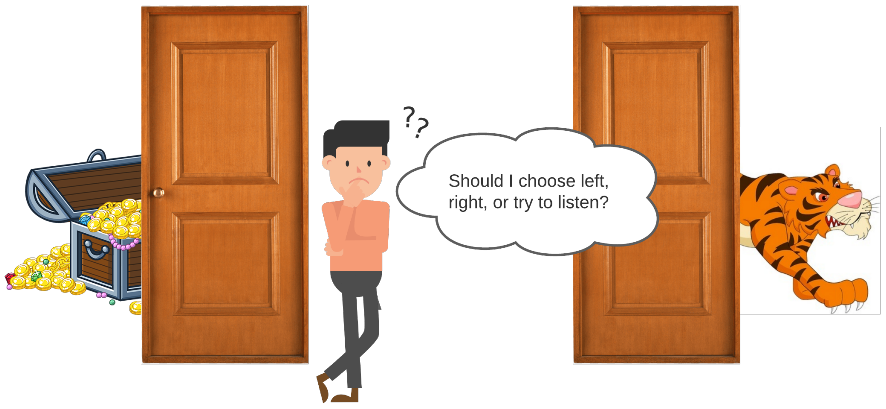


Figure 17: Representation of the tiger problem.

- $\Omega = \{\text{tiger left, tiger right}\}$
- $\mathcal{R} = \{-10, -1, 5\}$

To excel at this problem, the agent must balance obtaining information about where the tiger is, keeping in mind that listening negatively affects its rewards, and picking the right door.

7.2.2 Robot exploration problem

The robot exploration problem, represented in Figure 18, encompasses a robot that is trying to find a treasure chest. The robot can move between four rooms, where each room can be a hall or the treasure room (TR), and are circularly connected to each other. There are two possibilities for the agent to move: clockwise (CW) or counterclockwise (CCW). Either way, moving is time consuming and gives the agent a reward of -1. However, once the agent reaches the treasure room, it does not immediately find the treasure. Instead, it can choose to pull between two levers:

- lever$_A$: gives a 70% chance of lowering the treasure, giving the robot a reward of 10. The other 30% of the time, it drops a weight on the robot, slightly damaging it, giving it a reward of -5.
- lever$_B$: gives a 90% chance of lowering the treasure, giving the robot a reward of 10. The other 10% of the time, it drops a very heavy weight on the robot, causing it significant damage, giving it a reward of -20.

Naturally, if the robot tries to pull either of the levers when it is not in the treasure room, nothing happens. As for the observations, the agent can only receive two of them: either it is in a hall, or in the

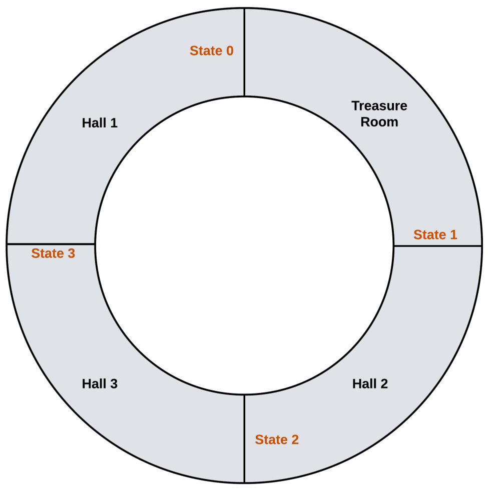


Figure 18: Diagram of the robot exploration problem

treasure room. Nevertheless, the robot's sensors are faulty, and give it the wrong observation 10% of the time.

Finally, when the robot pulls a lever, it is sent back into Hall_1, to continue the decision making process. The sets of states, actions, observations and rewards are therefore the following:

- $\mathcal{S} = \{\text{Hall}_1, \text{Hall}_2, \text{Hall}_3, \text{TR}\}$
- $\mathcal{A} = \{\text{CW}, \text{CCW}, \text{lever}_A, \text{lever}_B\}$
- $\Omega = \{\text{Hall}, \text{TR}\}$
- $\mathcal{R} = \{-20, -5, -1, 10\}$

This problem mixes exploring the environment to find the treasure room and choosing the best lever to maximize the agent's rewards. The best lever turns out to be lever_B, because it has the highest expected reward:

- For lever_A: $\mathbb{E}\left(R_{t+1} | \text{TR}, \text{lever}_A\right) = -5 \times 0.3 + 10 \times 0.7 = 5.5$
- For lever_B: $\mathbb{E}\left(R_{t+1} | \text{TR}, \text{lever}_B\right) = -20 \times 0.1 + 10 \times 0.9 = 7$

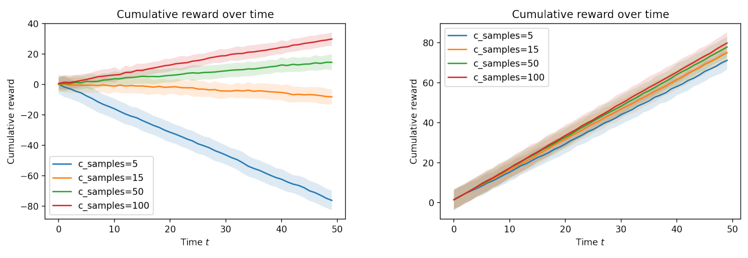
7.3 Results

All of the data collection was performed for multiple configurations of each experiment, varying two quantities of the lookahead algorithms: the horizon H of the lookahead, and the number of classical samples

n_c for the sampling processes. For each configuration of each experiment, a total of 40 different runs over 50 time-steps were performed, in order to collect data on multiple different ways that the experiment could have played out over those 50 time-steps, should small deviations change the course of the experiment. Moreover, although the number of samples for performing the algorithms varies depending on the configuration that is being run, the average rewards (and standard deviations) used as metrics for the algorithm evaluation are collected using 250 samples for every configuration, to assure a high degree of confidence on the results obtained. The same is true for the calculations of the classical and quantum coefficients c_l and q_l, respectively.

7.3.1 Cumulative reward comparison

Firstly, comparing only the classical algorithm with $H = 1$ to itself over the various different numbers of classical samples n_c, as in Figure 19, it is possible to check whether each algorithm has a big separation in its cumulative rewards just by varying the number of samples:



(a) (b)

Figure 19: Comparison of the performance of the classical lookahead with horizon $H = 1$ over different numbers of classical samples for: a) the tiger problem, b) the robot exploration problem. The higher the cummulative reward, the better the performance.

From Figure 19 it can be seen that the Tiger problem is the one most likely to have a good separation of cumulative rewards, since its curves are the ones that are most spread out in all of the experiments. This is not the only factor that determines whether or not a good separation of the classical and quantum algorithms is found, as the ratio $\frac{c_l}{q_l}$ also determines how many extra quantum samples can be leveraged against the classical lookahead algorithm. The separation between the two algorithms in the following results is a trade-off between these two possible sources of separation.

For the same setting, if the classical and quantum algorithms are compared, the following results are obtained:

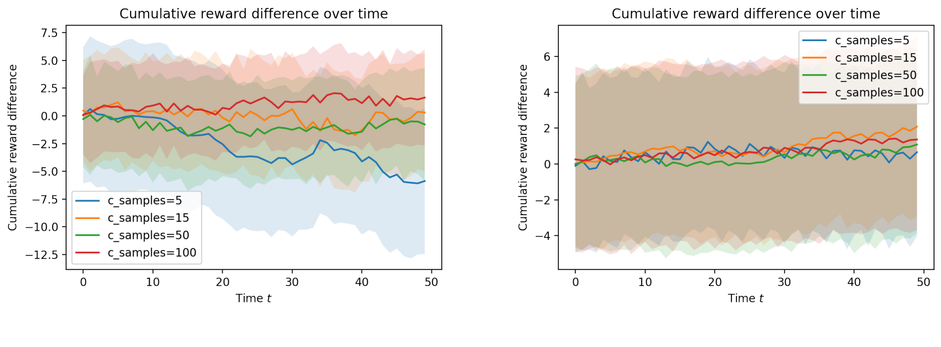


Figure 20: Comparison of the performance of the classical lookahead with horizon $H = 1$ against the quantum lookahead over different numbers of classical samples for: a) the tiger problem, b) the robot exploration problem. Positive difference implies advantage to the quantum lookahead.

As expected, results from Figure 20 show that there is no separation between the classical and quantum algorithm for $H = 1$, independently of the number of classical samples used, for any experiment. This is because the number of classical and quantum samples is the same for $H = 1$, as this horizon only considers the immediate reward for each action, and does not need to perform the belief-update necessary for the quantum speedup. As such, the performance of both these algorithms, independently of the experiment, is identical for an horizon of 1, as the results confirm.

Moving on to an horizon of $H = 2$, agents are able to look two time-step into the future in order to perform their actions. The cumulative reward difference between the quantum and classical lookahead algorithms is shown in Figure 21:

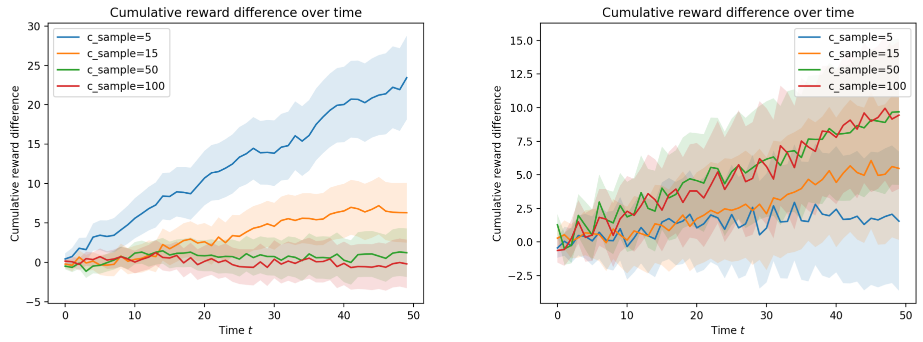


Figure 21: Cumulative reward difference of the quantum and classical lookahead with horizon $H = 2$ over different numbers of classical samples for: a) the tiger problem, b) the robot exploration problem. A positive difference implies quantum advantage.

As Figure 21 clearly depicts, there is a noticeable separation between the quantum and classical lookahead algorithms, with the quantum algorithm having an advantage. For the Tiger problem, the expected cumulative reward is significantly higher in the quantum case when using 5 classical samples, with this difference dropping as the number of classical samples increases. This likely occurs because the tiger problem has very simple dynamics, and therefore adding more and more samples to get better decisions has diminishing returns. However, for the robot problem, the reward difference increases with the number of classical samples. It is possible that this increase is due to the more complex dynamics of this environment, requiring a higher number of samples to extract good decisions from the lookahead algorithm. Nonetheless, both problems agree in a better decision making capacity for quantum agents.

Another relevant distinction is that the reward difference is higher in the tiger problem than in the robot exploration problem. There are three main aspects that can account for these differences:

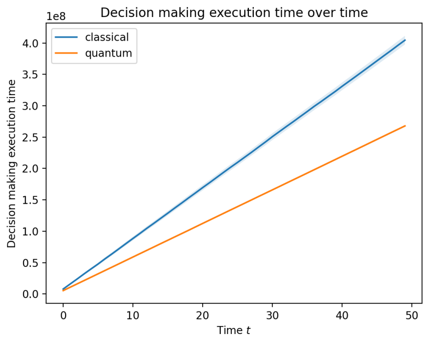
1. The ratio $\frac{c_l}{q_l}$. The higher the ratio, the higher the quantum advantage.
2. The rewards of the environments. Environments with higher (and more frequent) rewards will generally lead to higher expected rewards.
3. How sensitive the problem is to sample variations. If a small difference in the number of samples of a problem provides a large improvement in the rewards of the agent, that problem is likely to have a large quantum advantage in its rewards.

All of these experiments have somewhat similar rewards, and so point number 2 should not be too impactful. From the two experiments, the robot exploration problem has the higher ratio, averaging 2.76, with the tiger problem having a smaller ratio of 1.54. This means that the robot problem gets more extra samples compared to the tiger problem, according to Equation (7.1). In contrast, as seen in Figure 19, the tiger problem presents a higher reward sensitivity with respect to the number of samples. As such, even with a smaller ratio, it is possible (and is observed in Figure 21) for the tiger problem to have a higher cumulative reward diference than the robot exploration problem.

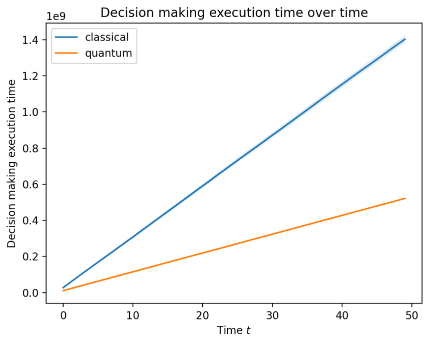
7.3.2 Execution time comparison

For the execution time comparison, results for horizon $H = 1$ are not considered, since there is no quantum advantage for this scenario as already stated and experimentally shown above.

Nevertheless, considering an horizon of $H = 2$, the quantum algorithm should start showing an execution time speedup over the classical algorithm, as it has a reduced time complexity. The results for the execution time are presented in Figure 22:



(a)



(b)

Figure 22: Mean decision making execution time over time of the quantum and classical lookahead algorithms. The algorithms were executed with an horizon $H = 2$ and using 5 classical samples for the experiments: a) the tiger problem, b) the robot exploration problem. The lower the execution time, the better the performance.

The execution time difference results clearly show that the quantum algorithm has a time complexity advantage over its classical counterpart for every experiment. Moreover, the robot exploration problem has the most noticeable quantum advantage, given it has the highest $\frac{c_l}{q_l}$ ratio, averaging 2.76. The tiger problem has a mean ratio of 1.54, leading to a more modest speedup, as can be seen by comparing the slopes of the lines of Figure 22.

7.4 Discussion

The quantum lookahead algorithm, as detailed in Chapter 6, has an advantage in terms of time complexity when compared to its classical alternative, which is dependent on the specific problem the algorithm is applied to.

In order to quantify this advantage, this Chapter compares the cumulative reward gathered by agents that use each of these algorithms across different experiments, to verify what kind of advantage the time complexity speedup of the quantum lookahead algorithm might provide to the quantum agent.

Two different aspects appear to be important for a specific problem to be able to get the full advantage of the quantum lookahead algorithm:

- A high sensitivity to varying the number of samples. That is, small variations to the number of samples used for that problem lead to high variations in the agent's cumulative reward.
- A high ratio $\frac{c_l}{q_l}$, quantities defined in Section 6.2. The higher this ratio is, the higher the number of

extra quantum samples (relative to the base number of classical samples) an agent can leverage.

The results presented in Section 7.3 show an advantage of the quantum algorithm across all experiments, clearly demonstrating that these quantum agents are better decision makers. The tiger problem quantum advantages come mainly from its high sensitivity to the number of samples, even with its low ratio. The opposite is true for the robot exploration problem, having a low sensitivity but compensating with a high ratio, allowing quantum agents to use many more samples to make their decisions.

The time complexity results further confirm the theoretical complexity analysis of Chapter 6. It was observed that the quantum algorithm has a clear speedup across all experiments, especially those with a high $\frac{c_l}{q_l}$ ratio, such as the robot exploration problem.

Chapter 8

Conclusions and future work

8.1 Conclusions

This disseration studied the interplay between Bayesian networks, reinforcement learning and quantum computing. The main aim was to verify whether or not quantum computing could have a positive impact on reinforcement learning based on Bayesian networks. It was possible to conclude that quantum computing does seem to bring advantages to this specific problem. The advantages reported in this work are comparable to those verified in Grover's algorithm, but not nearly as noticeable as landmark algorithms with exponential separation such as Shor's algorithm or QFT.

A quantum version of a lookahead algorithm for solving POMDPs is advantageous when compared to its classical counterpart in terms of time complexity, with a speedup that can be quadratic at most, as proved in Chapter 6. This naturally implies that the quantum lookahead algorithm studied in this work can reduce the decision making times of reinforcement learning agents, making them more efficient. Alternatively, as detailed in Chapter 7, this speedup can be leveraged to make more informed decisions than with the classical lookahead in the same amount of time, significantly improving the rational decision making skills of agents.

Another very interesting conclusion of this work is the fact that using quantum rejection sampling for dynamic decision networks only leads to time complexity speed-ups when they represent partially observable reinforcement learning environments. This is because this quantum sampling subroutine is only advantageous for performing a belief-update operation, which is only necessary when solving POMDPs. In contrast, all fully observable MDP distributions can be extracted using direct sampling instead, an algorithm which is more efficiently performed on a classical computer. As a consequence, fully observable Markov decision processes should not make use of quantum rejection sampling, as this would lead to poorer performance.

8.2 Prospects for future work

For future work, a very interesting approach would be the development of a fully quantum algorithm for extracting an action from the lookahead tree of the classical lookahead algorithm of Section 3.4, and comparing this method with both the classical and the quantum classical hybrid lookahead algorithms proposed in this work. A possible starting point for developing this algorithm is the encoding of every tree branch into a quantum circuit, followed by the encoding of the expected return of each branch into into the amplitude of its corresponding quantum state. Were this possible, the actions most likely to be measured from this quantum circuit would be the actions that are more likely to yield a high expected return for the agent. This approach is similar to the one applied by de Oliveira and Barbosa [2021], where the authors develop a quantum unitary operator that weights the probability amplitudes of the quantum states by their utility. This operator, however, is devised in the context of static decision making, and would have to be revised in order to be applied not to a static utility, but for a sequence of rewards, to account for the sequential decision making process applied in reinforcement learning.

Finally, this dissertation focuses on model-based approaches for solving reinforcement learning problems, as it is assumed that a model of the environment is known (in the form of a dynamic decision network). An alternative is the study of model-free methods using quantum sampling techniques. This would imply both not only using a model of the environment to choose actions, but also learning that model from the data the agent receives by interacting with its environment. This approach could even build on the work of this dissertation by using structure learning algorithms to determine a structure for the dynamic decision networks from data, and later using the quantum lookahead algorithm presented here to extract a good action.

Bibliography

Irad Ben-Gal. *Bayesian Networks*. American Cancer Society, 2008. ISBN 9780470061572. doi: https://doi.org/10.1002/9780470061572.eqr089. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/9780470061572.eqr089.

Ville Bergholm, Juha J. Vartiainen, Mikko Möttönen, and Martti M. Salomaa. Quantum circuits with uniformly controlled one-qubit gates. *Physical Review A*, 71(5), May 2005. ISSN 1094-1622. doi: 10.1103/physreva.71.052330. URL http://dx.doi.org/10.1103/PhysRevA.71.052330.

Sima E Borujeni and Saideep Nannapaneni. Modeling time-dependent systems using dynamic quantum Bayesian networks. *arXiv preprint arXiv:2107.00713*, 2021.

Sima E Borujeni, Nam H Nguyen, Saideep Nannapaneni, Elizabeth C Behrman, and James E Steck. Experimental evaluation of quantum bayesian networks on ibm qx hardware. In *2020 IEEE International Conference on Quantum Computing and Engineering (QCE)*, pages 372–378. IEEE, 2020.

Sima E Borujeni, Saideep Nannapaneni, Nam H Nguyen, Elizabeth C Behrman, and James E Steck. Quantum circuit representation of bayesian networks. *Expert Systems with Applications*, 176:114768, 2021.

Gilles Brassard, Peter Hoyer, Michele Mosca, and Alain Tapp. Quantum amplitude amplification and estimation. *Contemporary Mathematics*, 305:53–74, 2002.

Hans J Briegel, David E Browne, Wolfgang Dür, Robert Raussendorf, and Maarten Van den Nest. Measurement-based quantum computation. *Nature Physics*, 5(1):19–26, 2009.

Nicandro Cruz-Ramirez, Hector Gabriel Acosta-Mesa, Humberto Carrillo-Calvet, Luis Alonso Nava-Fernández, and Rocío Erandi Barrientos-Martínez. Diagnosis of breast cancer using bayesian networks: A case study. *Computers in Biology and Medicine*, 37(11):1553–1564, 2007.

P.W.P. D and P. Winder. *Reinforcement Learning: Industrial Applications of Intelligent Agents*. O'Reilly

Media, Incorporated, 2020. ISBN 9781098114831. URL https://books.google.pt/books?id=u87AzQEACAAJ.

Michael de Oliveira and Luis Soares Barbosa. Quantum Bayesian decision-making. *Foundations of Science*, pages 1–21, 2021.

Ashley D Edwards, Laura Downs, and James C Davidson. Forward-backward reinforcement learning. *arXiv preprint arXiv:1803.10227*, 2018.

Henry O Everitt. *Experimental aspects of quantum computing*, volume 61. Springer, 2005.

Alhussein Fawzi, Matej Balog, Aja Huang, Thomas Hubert, Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Francisco J R Ruiz, Julian Schrittwieser, Grzegorz Swirszcz, et al. Discovering faster matrix multiplication algorithms with reinforcement learning. *Nature*, 610(7930):47–53, 2022.

RP Feynman. Simulating physics with computers. *International Journal of Theoretical Physics*, 21(6), 1982.

Aleta Berk Finnila, MA Gomez, C Sebenik, Catherine Stenson, and Jimmie D Doll. Quantum annealing: A new method for minimizing multidimensional functions. *Chemical physics letters*, 219(5-6):343–348, 1994.

Mohammad Ghavamzadeh, Shie Mannor, Joelle Pineau, and Aviv Tamar. Bayesian reinforcement learning: A survey. *arXiv preprint arXiv:1609.04436*, 2016.

Lov K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, STOC '96, page 212–219, New York, NY, USA, 1996. Association for Computing Machinery. ISBN 0897917855. doi: 10.1145/237814.237866. URL https://doi.org/10.1145/237814.237866.

Haipeng Guo and William Hsu. A survey of algorithms for real-time Bayesian network inference. In *Join Workshop on Real Time Decision Support and Diagnosis Systems*, 2002.

M. Hauskrecht. Value-function Approximations for Partially Observable Markov Decision Processes. *Journal of Artificial Intelligence Research*, 13:33–94, Aug 2000. ISSN 1076-9757. doi: 10.1613/jair.678. URL http://dx.doi.org/10.1613/jair.678.

E. Herman and G. Strang. *Calculus Volume 2*. OpenStax, 2016. ISBN 9781947172821. URL https://books.google.pt/books?id=l_e1ygEACAAJ.

Hsin-Yuan Huang, Michael Broughton, Jordan Cotler, Sitan Chen, Jerry Li, Masoud Mohseni, Hartmut Neven, Ryan Babbush, Richard Kueng, John Preskill, et al. Quantum advantage in learning from experiments. *arXiv preprint arXiv:2112.00778*, 2021.

John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, et al. Highly accurate protein structure prediction with alphafold. *Nature*, 596(7873):583–589, 2021.

Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285, 1996.

Leslie Pack Kaelbling, Michael L Littman, and Anthony R Cassandra. Planning and acting in partially observable stochastic domains. *Artificial intelligence*, 101(1-2):99–134, 1998.

Arthur B Kahn. Topological sorting of large networks. *Communications of the ACM*, 5(11):558–562, 1962.

Sham Machandranath Kakade. *On the sample complexity of reinforcement learning*. University of London, University College London (United Kingdom), 2003.

B Ravi Kiran, Ibrahim Sobh, Victor Talpaert, Patrick Mannion, Ahmad A Al Sallab, Senthil Yogamani, and Patrick Pérez. Deep reinforcement learning for autonomous driving: A survey. *IEEE Transactions on Intelligent Transportation Systems*, 2021.

Jens Kober, J Andrew Bagnell, and Jan Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11):1238–1274, 2013.

Ranganath Kothamasu, Samuel H Huang, and William H VerDuin. System health monitoring and prognostics—a review of current paradigms and practices. *The International Journal of Advanced Manufacturing Technology*, 28(9-10):1012–1024, 2006.

Chenzhao Li, Sankaran Mahadevan, You Ling, Sergio Choze, and Liping Wang. Dynamic Bayesian network for aircraft wing health monitoring digital twin. *Aiaa Journal*, 55(3):930–941, 2017.

Michael L Littman. A tutorial on partially observable markov decision processes. *Journal of Mathematical Psychology*, 53(3):119–125, 2009.

Xu-Hui Liu, Zhenghai Xue, Jingcheng Pang, Shengyi Jiang, Feng Xu, and Yang Yu. Regret minimization experience replay in off-policy reinforcement learning. *Advances in Neural Information Processing Systems*, 34:17604–17615, 2021.

Guang Hao Low, Theodore J Yoder, and Isaac L Chuang. Quantum inference on Bayesian networks. *Physical Review A*, 89(6):062315, 2014.

Omid Madani, Steve Hanks, and Anne Condon. On the undecidability of probabilistic planning and related stochastic optimization problems. *Artificial Intelligence*, 147(1-2):5–34, 2003.

Thomas M Moerland, Joost Broekens, and Catholijn M Jonker. Model-based reinforcement learning: A survey. *arXiv preprint arXiv:2006.16712*, 2020.

Catarina Moreira and Andreas Wichert. Quantum-like bayesian networks for modeling decision making. *Frontiers in psychology*, page 11, 2016.

Kevin Patrick Murphy. *Dynamic Bayesian Networks: Representation, Inference and Learning*. University of California, Berkeley, 2002. AAI3082340.

Saideep Nannapaneni, Sankaran Mahadevan, and Sudarsan Rachuri. Performance evaluation of a manufacturing process under uncertainty using bayesian networks. *Journal of Cleaner Production*, 113: 947–959, 2016.

Richard Neapolitan and Xia Jiang. *Decision Analysis Fundamentals*, pages 177–228. 12 2007. ISBN 978-0-12-370477-1. doi: 10.1016/B978-012370477-1/50022-0.

Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, USA, 10th edition, 2011. ISBN 1107002176.

Giuseppe Davide Paparo, Vedran Dunjko, Adi Makmal, Miguel Angel Martin-Delgado, and Hans J Briegel. Quantum speedup for active learning agents. *Physical Review X*, 4(3):031002, 2014.

Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall Press, USA, 3rd edition, 2009. ISBN 0136042597.

Valeria Saggio, Beate E Asenbeck, Arne Hamann, Teodor Strömberg, Peter Schiansky, Vedran Dunjko, Nicolai Friis, Nicholas C Harris, Michael Hochberg, Dirk Englund, et al. Experimental quantum speed-up in reinforcement learning agents. *Nature*, 591(7849):229–233, 2021.

Robert R Schaller. Moore's law: past, present and future. *IEEE spectrum*, 34(6):52–59, 1997.

Eugene Seneta. On the history of the strong law of large numbers and boole's inequality. *Historia Mathematica*, 19(1):24–39, 1992.

Peter W Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM review*, 41(2):303–332, 1999.

Aaron Sidford, Mengdi Wang, Xian Wu, Lin F Yang, and Yinyu Ye. Near-optimal time and sample complexities for solving discounted markov decision process with a generative model. *arXiv preprint arXiv:1806.01492*, 2018.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. A Bradford Book, Cambridge, MA, USA, 2018. ISBN 0262039249.

Daochen Wang, Aarthi Sundaram, Robin Kothari, Ashish Kapoor, and Martin Roetteler. Quantum algorithms for reinforcement learning with a generative model. In *International Conference on Machine Learning*, pages 10916–10926. PMLR, 2021.

Larry Wasserman. *All of statistics : a concise course in statistical inference*. Springer, New York, 2010. ISBN 9781441923226 1441923225. URL http://www.amazon.de/All-Statistics-Statistical-Inference-Springer/dp/1441923225/ref=sr_1_2?ie=UTF8&qid=1356099149&sr=8-2.

Philippe Weber, Gabriela Medina-Oliva, Christophe Simon, and Benoît Iung. Overview on Bayesian networks applications for dependability, risk analysis and maintenance areas. *Engineering Applications of Artificial Intelligence*, 25(4):671–682, 2012.

Simon Wiedemann, Daniel Hein, Steffen Udluft, and Christian Mendl. Quantum policy iteration via amplitude estimation and grover search–towards quantum advantage for reinforcement learning. *arXiv preprint arXiv:2206.04741*, 2022.

Wim Wiegerinck, Bert Kappen, and Willem Burgers. *Bayesian Networks for Expert Systems: Theory and Practical Applications*, pages 547–578. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010. ISBN

978-3-642-11688-9. doi: 10.1007/978-3-642-11688-9_20. URL https://doi.org/10.1007/978-3-642-11688-9_20.

Colin P Williams. Quantum gates. In *Explorations in Quantum Computing*, pages 51–122. Springer, 2011.

Chao Yu, Jiming Liu, and Shamim Nemati. Reinforcement learning in healthcare: A survey. *arXiv preprint arXiv:1908.08796*, 2019.

JY Zhu and A Deshmukh. Application of Bayesian decision networks to life cycle engineering in Green design and manufacturing. *Engineering Applications of Artificial Intelligence*, 16(2):91–103, 2003.

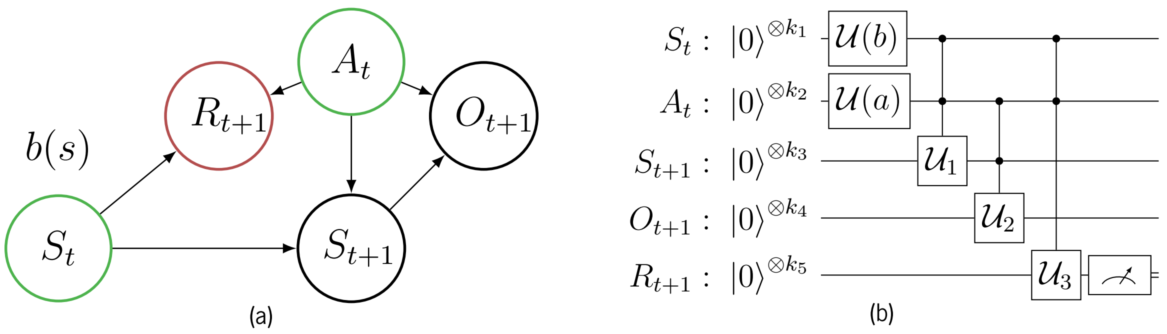
Part III
Appendices

Appendix A
Quantum reward and observation sampling

A.1 Quantum reward sampling

Once again leveraging the quantum rejection sampling algorithm, the same principles for constructing a quantum belief update can be used to perform a quantum reward sampling. This operation entails the extraction of the reward dynamics $P(r|b, a)$ of a DDN.

Consider the same circuit as in Figure 16, now with the measurement performed on the reward node and without the amplitude amplification operator, as in Figure 23:



$b(s)$
S_t
R_{t+1}
A_t
O_{t+1}
S_{t+1}
(a)
$S_t : |0\rangle^{\otimes k_1}$
$A_t : |0\rangle^{\otimes k_2}$
$S_{t+1} : |0\rangle^{\otimes k_3}$
$O_{t+1} : |0\rangle^{\otimes k_4}$
$R_{t+1} : |0\rangle^{\otimes k_5}$
$\mathcal{U}(b)$
$\mathcal{U}(a)$
$\mathcal{U}_1$
$\mathcal{U}_2$
$\mathcal{U}_3$
(b)

Figure 23: a) A DDN for performing the reward sampling operation. Green nodes are encoded and red nodes are measured in the quantum circuit on the right. b) Quantum circuit for reward sampling using the DDN on the left.

To prove that this algorithm in fact extracts the desired probability distribution, recall that it is the same quantum circuit as in the quantum belief update procedure, except the measurement is now on the reward RV and no observation needs to be amplified. As such, the final quantum state for this circuit is the same of Lemma 5.2.6:

$$|\psi_{\text{final}}(b, a)\rangle = \sum_{s\in\mathcal{S}} \sqrt{b(s)} \sum_{s'\in\mathcal{S}} \sqrt{P(s'|s, a)} \sum_{o\in\Omega} \sqrt{P(o|s', a)} \sum_{r\in\mathcal{R}} \sqrt{P(r|s', a)}\, |sas'or\rangle \tag{A.1}$$

Nonetheless, calculating the measurement probabilities now is slightly different. The same approach used to calculate the density matrix and finding its expectation is followed, but the density matrix is now constructed by executing a partial trace on every RV except for the reward one.

Using this procedure, Lemma A.1.1 shows that the quantum circuit encodes the desired probability distribution in its measurement outcomes:

Lemma A.1.1. *The measurement probabilities of the quantum circuit in figure 23 follow the reward dynamics $P(r|b,a)$ of its DDN:*

$$\langle r|\rho(b,a)|r\rangle = P(r|b,a)$$

Proof. First, for the calculation of the partial density matrix for the circuit:

$$\begin{aligned}
\rho(b,a) &= \sum_{s,a',o',s'} \langle sa's'o'|\psi_{\text{final}}(b,a)\rangle \langle \psi_{\text{final}}(b,a)|sa's'o'\rangle \\
&= \sum_{s,a',o',s'} \left(\sum_{s_\star \in \mathcal{S}} \sqrt{b(s_\star)} \sum_{s^\star \in \mathcal{S}} \sqrt{P(s^\star|s_\star,a)} \sum_{o\in\Omega} \sqrt{P(o|s^\star,a)} \sum_{r\in\mathcal{R}} \sqrt{P(r|s_\star,a)} \langle sa's'o'|s_\star a s^\star or\rangle \right) \\
&\left(\sum_{s_\star \in \mathcal{S}} \sqrt{b(s_\star)} \sum_{s^\star \in \mathcal{S}} \sqrt{P(s^\star|s_\star,a)} \sum_{o\in\Omega} \sqrt{P(o|s^\star,a)} \sum_{r\in\mathcal{R}} \sqrt{P(r|s_\star,a)} \langle s_\star a s^\star or|sa's'o'\rangle \right) \\
&= \sum_{s,a',o',s'} \left(\sum_{s_\star \in \mathcal{S}} \sqrt{b(s_\star)} \sum_{s^\star \in \mathcal{S}} \sqrt{P(s^\star|s_\star,a)} \sum_{o\in\Omega} \sqrt{P(o|s^\star,a)} \sum_{r\in\mathcal{R}} \sqrt{P(r|s_\star,a)} \delta_{ss_\star}\delta_{aa'}\delta_{s's^\star}\delta_{oo'} |r\rangle \right) \\
&\left(\sum_{s_\star \in \mathcal{S}} \sqrt{b(s_\star)} \sum_{s^\star \in \mathcal{S}} \sqrt{P(s^\star|s_\star,a)} \sum_{o\in\Omega} \sqrt{P(o|s^\star,a)} \sum_{r\in\mathcal{R}} \sqrt{P(r|s_\star,a)} \delta_{ss_\star}\delta_{aa'}\delta_{s's^\star}\delta_{oo'} \langle r| \right) \\
&= \sum_{s\in\mathcal{S}} b(s) \sum_{s'\in\mathcal{S}} P(s'|s,a) \sum_{o\in\Omega} P(o|s',a) \left(\sum_{r\in\mathcal{R}} \sqrt{P(r|s,a)}\, |r\rangle \right) \left(\sum_{r\in\mathcal{R}} \sqrt{P(r|s,a)}\, \langle r| \right)
\end{aligned}$$

Using this density matrix, the probability of measuring reward R_{t+1} with value r is given by the expression:

$$\begin{aligned}
\langle r|\rho(b,a)|r\rangle &= \sum_{s\in\mathcal{S}} b(s) \sum_{s'\in\mathcal{S}} P(s'|s,a) \sum_{o\in\Omega} P(o|s',a) \left(\sum_{r'\in\mathcal{R}} \sqrt{P(r'|s,a)}\, \langle r|r'\rangle \right) \left(\sum_{r'\in\mathcal{R}} \sqrt{P(r'|s,a)}\, \langle r'|r\rangle \right) \\
&= \sum_{s\in\mathcal{S}} b(s) \sum_{s'\in\mathcal{S}} P(s'|s,a) \sum_{o\in\Omega} P(o|s',a) \left(\sum_{r'\in\mathcal{R}} \sqrt{P(r'|s,a)}\delta_{rr'} \right) \left(\sum_{r'\in\mathcal{R}} \sqrt{P(r'|s,a)}\delta_{r'r} \right) \\
&= \sum_{s\in\mathcal{S}} b(s) \sum_{s'\in\mathcal{S}} P(s'|s,a) \sum_{o\in\Omega} P(o|s',a) P(r|s,a) \\
&= \sum_{s\in\mathcal{S}} b(s) P(r|s,a) \sum_{s'\in\mathcal{S}} P(s'|s,a) \sum_{o\in\Omega} P(o|s',a) \\
&= \sum_{s\in\mathcal{S}} b(s) P(r|s,a) \\
&= P(r|b,a)
\end{aligned}$$

□

Similarly to the belief update, Lemma A.1.1 states that the quantum circuit in Figure 23 is able to encode the environment's reward dynamics in its measurement outcomes. Therefore, by extracting $r_i \sim$

$P(r|b,a),\ i = \{1,\dots,n\}$ reward samples from this circuit, the reward dynamics can be approximated using the following equation:

$$P_n(r|b,a) = \frac{1}{n}\sum_{i=1}^{n} \delta_{rr_i} \tag{A.2}$$

Where $P_n(r|b,a)$ is an approximator for the reward dynamics $P(r|b,a)$ using n samples. Nonetheless, the usefulness of gathering the reward dynamics is to estimate the expected reward $\mathbb{E}(R|b,a)$, as this quantity shows up in every calculation of RL values. The expectation value can be easily related to the reward dynamics by the following equation:

$$\mathbb{E}(R|b,a) = \sum_{r\in\mathcal{R}} P(r|b,a)r \tag{A.3}$$

such that an estimator for the expectation value can be calculated using the estimator of the reward dynamics:

$$\mathbb{E}_n(R|b,a) = \sum_{r\in\mathcal{R}} P_n(r|b,a)r \tag{A.4}$$

Although this section shows that it is possible to extract the reward dynamics from the environment using a quantum circuit, it is disadvantageous to use this method when compared to the classical case. This is because its quantum implementation does not require amplitude amplification, which is the source of the quadratic speedup in quantum rejection sampling. Instead, this algorithm is a quantum version of direct sampling, where the DDN only needs to be encoded into a quantum circuit. However, as already seen in Section 4.4.1, quantum direct sampling is disadvantageous when compared to its classical counterpart, because of the exponential cost associated to BN encoding.

A.2 Quantum observation sampling

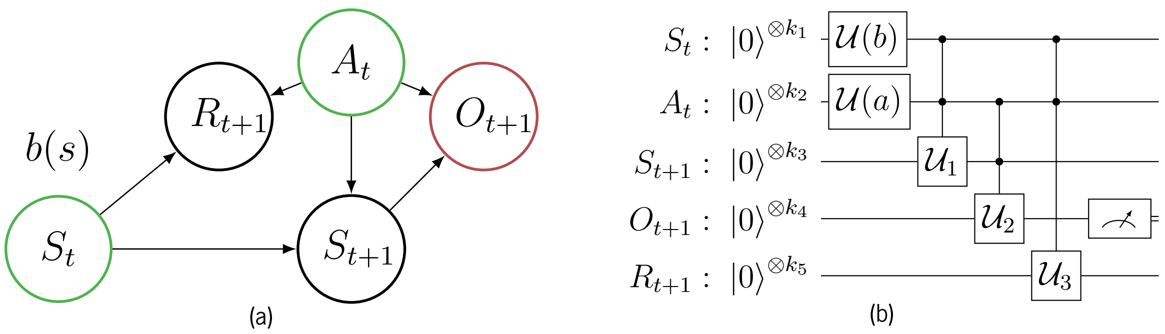
In this section, which is analogous to the previous two sections, it is shown how to extract the observation probability distribution $P(o|b,a)$ from a DDN using a quantum circuit.

The setup is analogous to the quantum reward sampling: construct a circuit that encodes the DDN and measure the observation RV (see Figure 24).

The claim is that this circuit encodes the probability distribution $P(o|b,a)$ in its measurement outcomes. To prove this, the same approach of the previous sections is followed. First, recall the final state of this circuit, analogous to quantum reward sampling:

$$|\psi_{\text{final}}(b,a)\rangle = \sum_{s\in\mathcal{S}} \sqrt{b(s)} \sum_{s'\in\mathcal{S}} \sqrt{P(s'|s,a)} \sum_{o\in\Omega} \sqrt{P(o|s',a)} \sum_{r\in\mathcal{R}} \sqrt{P(r|s',a)}\, |sas'or\rangle$$

By calculating the partial density matrix for the observation RV and its expectation value, Lemma A.2.1 proves that the initial claim is true.



$b(s)$
S_t
R_{t+1}
A_t
O_{t+1}
S_{t+1}
(a)
$S_t:\ |0\rangle^{\otimes k_1}$
$A_t:\ |0\rangle^{\otimes k_2}$
$S_{t+1}:\ |0\rangle^{\otimes k_3}$
$O_{t+1}:\ |0\rangle^{\otimes k_4}$
$R_{t+1}:\ |0\rangle^{\otimes k_5}$
$\mathcal{U}(b)$
$\mathcal{U}(a)$
$\mathcal{U}_1$
$\mathcal{U}_2$
$\mathcal{U}_3$
(b)

Figure 24: a) A DDN for performing the observation sampling operation. Green nodes are encoded and red nodes are measured in the quantum circuit on the right. b) Quantum circuit for observation sampling using the DDN in the sub-figure on the left.

Lemma A.2.1. *The measurement probabilities of the quantum circuit in Figure 24 follow the observation probability distribution $P(o|b,a)$ of its DDN:*

$$\langle o|\rho(b,a)|o\rangle = P(o|b,a)$$

Proof. First, let us start by computing the partial density matrix for the circuit:

$$\begin{aligned}
\rho(b,a) &= \sum_{s,a',s',r'} \langle sa's'r'|\psi_{\text{final}}(b,a)\rangle \langle \psi_{\text{final}}(b,a)|sa's'r'\rangle \\
&= \sum_{s,a',s',r'} \left(\sum_{s_\star\in\mathcal{S}} \sqrt{b(s_\star)} \sum_{s^\star\in\mathcal{S}} \sqrt{P(s^\star|s_\star,a)} \sum_{o\in\Omega} \sqrt{P(o|s^\star,a)} \sum_{r\in\mathcal{R}} \sqrt{P(r|s_\star,a)}\, \langle sa's'r'|s_\star as^\star or\rangle\right) \\
&\left(\sum_{s_\star\in\mathcal{S}} \sqrt{b(s_\star)} \sum_{s^\star\in\mathcal{S}} \sqrt{P(s^\star|s_\star,a)} \sum_{o\in\Omega} \sqrt{P(o|s^\star,a)} \sum_{r\in\mathcal{R}} \sqrt{P(r|s_\star,a)}\, \langle s_\star as^\star or|sa's'r'\rangle\right) \\
&= \sum_{s,a',s',r'} \left(\sum_{s_\star\in\mathcal{S}} \sqrt{b(s_\star)} \sum_{s^\star\in\mathcal{S}} \sqrt{P(s^\star|s_\star,a)} \sum_{o\in\Omega} \sqrt{P(o|s^\star,a)} \sum_{r\in\mathcal{R}} \sqrt{P(r|s_\star,a)}\delta_{ss_\star}\delta_{aa'}\delta_{s's^\star}\delta_{rr'}\, |o\rangle\right) \\
&\left(\sum_{s_\star\in\mathcal{S}} \sqrt{b(s_\star)} \sum_{s^\star\in\mathcal{S}} \sqrt{P(s^\star|s_\star,a)} \sum_{o\in\Omega} \sqrt{P(o|s^\star,a)} \sum_{r\in\mathcal{R}} \sqrt{P(r|s_\star,a)}\delta_{ss_\star}\delta_{aa'}\delta_{s's^\star}\delta_{rr'}\, \langle o|\right) \\
&= \sum_{s\in\mathcal{S}} b(s) \sum_{r\in\mathcal{R}} P(r|s,a) \sum_{s'\in\mathcal{S}} P(s'|s,a) \left(\sum_{o\in\Omega} \sqrt{P(o|s',a)}\, |o\rangle\right) \left(\sum_{o\in\Omega} \sqrt{P(o|s',a)}\, \langle o|\right)
\end{aligned}$$

Using this density matrix, the probability of measuring reward O_{t+1} with value o is given by the expression:

$$\begin{aligned}
\langle o|\rho(b,a)|o\rangle &= \sum_{s\in\mathcal{S}} b(s) \sum_{r\in\mathcal{R}} P(r|s,a) \sum_{s'\in\mathcal{S}} P(s'|s,a) \left(\sum_{o'\in\Omega} \sqrt{P(o'|s',a)}\,\langle o|o'\rangle\right)\left(\sum_{o'\in\Omega} \sqrt{P(o'|s',a)}\,\langle o'|o\rangle\right) \\
&= \sum_{s\in\mathcal{S}} b(s) \sum_{r\in\mathcal{R}} P(r|s,a) \sum_{s'\in\mathcal{S}} P(s'|s,a) \left(\sum_{o'\in\Omega} \sqrt{P(o'|s',a)}\delta_{oo'}\right)\left(\sum_{o'\in\Omega} \sqrt{P(o'|s',a)}\delta_{o'o}\right) \\
&= \sum_{s\in\mathcal{S}} b(s) \sum_{r\in\mathcal{R}} P(r|s,a) \sum_{s'\in\mathcal{S}} P(s'|s,a)P(o|s',a) \\
&= \sum_{s'\in\mathcal{S}} P(o|s',a) \sum_{s\in\mathcal{S}} P(s'|s,a)b(s) \sum_{r\in\mathcal{R}} P(r|s,a) \\
&= \sum_{s'\in\mathcal{S}} P(o|s',a) \sum_{s\in\mathcal{S}} P(s'|s,a)b(s) \\
&= \sum_{s'\in\mathcal{S}} P(o|s',a)P(s'|b,a)
\end{aligned}$$

Now, consider the following independence statement: RV O_{t+1} is independent of S_t when S_{t+1} is known[1], such that $P(o|s',a) = P(o|s',b,a)$. Therefore, the expectation value above can be re-written as:

$$\begin{aligned}
\langle o|\rho(b,a)|o\rangle &= \sum_{s'\in\mathcal{S}} P(o|s',a)P(s'|b,a) \\
&= \sum_{s'\in\mathcal{S}} P(o|s',b,a)P(s'|b,a) \\
&= \sum_{s'\in\mathcal{S}} P(o|s',b,a)\frac{P(s',b,a)}{P(b,a)} \\
&= \sum_{s'\in\mathcal{S}} \frac{P(o,s',b,a)}{P(b,a)} \\
&= \sum_{s'\in\mathcal{S}} P(o,s'|b,a) \\
&= P(o|b,a)
\end{aligned}$$

□

Given that the quantum circuit of Figure 24 does indeed capture the observation distribution $P(o|b,a)$ in its measurement outcomes, its is possible to estimate this probability distribution in a quantum device. To do this, collect n observation samples $o_i \sim P(o|b,a),\ i = \{1,\dots,n\}$ and use the following expression to construct the estimator $P_n(o|b,a)$:

$$P_n(o|b,a) = \frac{1}{n}\sum_{i=1}^{n} \delta_{oo_i} \tag{A.5}$$

[1] S_t, S_{t+1} and O_{t+1} form a chain in the DDN of Figure 24. A chain of three RVs is a very common example of independence tests in BNs: the leaf RV (O_{t+1}) is independent of the root RV S_t when the middle RV (S_{t+1}) is known.

Just as in the reward sampling case, extracting this probability distribution in a quantum device is disadvantageous relative to its classical counterpart, because it does not leverage the amplitude amplification technique.

Appendix B
Complexity analysis auxiliary results

To derive an upper bound on the lookahead algorithm's error of Definition 6.1.1, some lemmas are of use. One example is Lemma B.0.1, an inequality that relates the difference of the maxima of two vectors and the maximum of their difference:

Lemma B.0.1. *Let $\boldsymbol{U}, \boldsymbol{V} \in \mathcal{W}$ be vectors in some vector space $\mathcal{W}$. Then:*

$$\|\boldsymbol{U}\|_\infty - \|\boldsymbol{V}\|_\infty \leq \|\boldsymbol{U} - \boldsymbol{V}\|_\infty$$

Proof. Let U_i and V_i represent entries of $\boldsymbol{U}$ and $\boldsymbol{V}$, respectively. Then:

$$\begin{aligned} U_i &\leq |U_i - V_i| + V_i \\ \max_i U_i &\leq \max_i \left(|U_i - V_i| + V_i\right) \\ \max_i U_i &\leq \max_i |U_i - V_i| + \max_i V_i \\ \max_i U_i - \max_i V_i &\leq \max_i |U_i - V_i| \\ \|\boldsymbol{U}\|_\infty - \|\boldsymbol{V}\|_\infty &\leq \|\boldsymbol{U} - \boldsymbol{V}\|_\infty \end{aligned}$$

□

Another result that is used repeatedly throughout the proofs of Chapter 6 is a generalization of the triangle's inequality, stated in Lemma B.0.2:

Lemma B.0.2. *Let $\boldsymbol{U}, \boldsymbol{V} \in \mathcal{W}$ be vectors in an Euclidean space $\mathcal{W}$. Then:*

$$\|\boldsymbol{U} + \boldsymbol{V}\| \leq \|\boldsymbol{U}\| + \|\boldsymbol{V}\|$$

Proof.

$$\|\boldsymbol{U} + \boldsymbol{V}\| = \sqrt{\|\boldsymbol{U}\|^2 + \|\boldsymbol{V}\|^2 + 2\boldsymbol{U}^\top \boldsymbol{V}}$$

Using the Cauchy-Schwarz inequality, stating that $\left|\boldsymbol{U}^\top \boldsymbol{V}\right| \leq \|U\|\|V\|$, we get:

$$\begin{aligned} \|\boldsymbol{U} + \boldsymbol{V}\| &\leq \sqrt{\|\boldsymbol{U}\|^2 + \|\boldsymbol{V}\|^2 + 2\|\boldsymbol{U}\|\|\boldsymbol{V}\|} \\ \|\boldsymbol{U} + \boldsymbol{V}\| &\leq \|\boldsymbol{U}\| + \|\boldsymbol{V}\| \end{aligned}$$

□

Lemma B.0.3 simplifies a finite geometric series into a fraction:

Lemma B.0.3. *The finite geometric series can be simplified according to the following expression Herman and Strang [2016]:*

$$\sum_{i=0}^{N} z^i = \frac{z^N - 1}{z - 1}$$

Finally, a variation of the finite geometric series is also simplified in Lemma B.0.4:

Lemma B.0.4. *A variation of the finite geometric series of Lemma B.0.3 can be simplified according to the following expression Herman and Strang [2016]:*

$$\sum_{i=0}^{N} iz^i = \frac{(N-1)z^{N+1} - Nz^N + z}{(z-1)^2}$$