Universidade do Minho
Escola de Engenharia
Departamento de Informática

André Manuel Resende Sequeira

Quantum-enhanced Reinforcement Learning

May 2022

Universidade do Minho
Escola de Engenharia
Departamento de Informática

André Manuel Resende Sequeira

Quantum-enhanced Reinforcement Learning

Master dissertation
Master Degree in Engineering Physics

Dissertation supervised by
Luis Paulo Santos
Luis Soares Barbosa

May 2022

COPYRIGHTS AND TERMS OF USE

STATEMENT OF INTEGRITY

I hereby declare having conducted this academic work with integrity. I confirm that I have not used plagiarism or any form of undue use of information or falsification of results along the process leading to its elaboration. I further declare that I have fully acknowledged the Code of Ethical Conduct of the University of Minho.

ACKNOWLEDGEMENTS

This dissertation is the pinnacle of five years of study, culminating in the first real research experience in Quantum Computing, with the hope that it is not the last. There are several people I would like to thank because without them I would not get this far.

First of all, I want to thank my family, my parents, and my brother, my best friends. To my parents, not only this year but throughout my entire life, you were always there, through thick and thin. My source of inspiration, you taught me that with hard work and patience, one can do anything one sets his mind to. To my brother, thank you for all your support, all the talks, and laughs that you don't even know how refreshing they were.

Secondly, I want to thank my supervisors, professor Luis Soares Barbosa for accepting my dissertation theme suggestion and allowing me to research what I want as long as I want. Many thanks to professor Luis Paulo Santos for taking the time to guide me through this project, always providing insightful discussions and opinions, it was a pleasure.

Furthermore, I want to thank my good friends, Michael Oliveira and Daniel Carvalho for the endless support, not only for this year but for the entire five-year journey.

Lastly, my heartiest gratitude to Rita Rodrigues for all the warmth and advice that you give me. I am counting on you.

This dissertation was partially supported by the ERDF – European Regional Development Fund through the Operational Programme for Competitiveness and Internationalisation - COMPETE 2020 Programme and by National Funds through the Portuguese funding agency, FCT - Fundação para a Ciência e a Tecnologia, within project POCI-01-0145-FEDER-030947.

ABSTRACT

The field of Artificial Intelligence has lately witnessed extraordinary results. The ability to design a system capable of beating the world champion of Go, an ancient Chinese game known as the holy grail of AI, caused a spark worldwide, making people believe that something revolutionary is about to happen. A different flavor of learning called Reinforcement Learning is at the core of this revolution. In parallel, we are witnessing the emergence of a new field, that of Quantum Machine Learning which has already shown promising results in supervised/unsupervised learning. In this dissertation, we reach for the interplay between Quantum Computing and Reinforcement Learning.

This learning by interaction was made possible in the quantum setting using the concept of oraculization of task environments suggested by Dunjko in 2015. In this dissertation, we extended the oracular instances previously suggested to work in more general stochastic environments. On top of this *quantum agent-environment paradigm* we developed a novel quantum algorithm for near-optimal decision-making based on the Reinforcement Learning paradigm known as *Sparse Sampling*, obtaining a quantum speedup compared to the classical counterpart. The achievement was a quantum algorithm that exhibits a complexity independent on the number of states of the environment. This independence guarantees its suitability for dealing with large state spaces where planning may be inapplicable.

The most important open questions remain whether it is possible to improve the oracular instances of task environments to deal with even more general environments, especially the ability to represent negative rewards as a natural mechanism for negative feedback instead of some normalization of the reward and the extension of the algorithm to perform an *informed* tree-based search instead of the uninformed search proposed. Improvements on this result would allow the comparison between the algorithm and more recent classical Reinforcement Learning algorithms.

KEYWORDS Quantum Computation, Sparse Sampling, Quantum Reinforcement Learning

RESUMO

O campo da Inteligência Artificial tem tido resultados extraordinários ultimamente, a capacidade de projetar um sistema capaz de vencer o campeão mundial de Go, um antigo jogo de origem Chinesa, conhecido como o santo graal da IA, causou uma faísca em todo o mundo, fazendo as pessoas acreditarem em que algo revolucionário estará para acontecer. Um tipo diferente de aprendizagem, chamada Aprendizagem por Reforço está no cerne dessa revolução. Em paralelo surge também um novo campo, o da Aprendizagem Máquina Quântica, que já vem apresentando resultados promissores na aprendizagem supervisionada/não supervisionada. Nesta dissertação, procuramos invés a interação entre Computação Quântica e a Aprendizagem por Reforço.

Esta interação entre agente e Ambiente foi possível no cenário quântico usando o conceito de oraculização de ambientes sugerido por Dunjko em 2015. Neste trabalho, estendemos as instâncias oraculares sugeridas anteriormente para trabalhar em ambientes estocásticos generalizados. Tendo em conta este *paradigma quântico agente-ambiente*, desenvolvemos um novo algoritmo quântico para tomada de decisão aproximadamente ótima com base no paradigma da Aprendizagem por Reforço conhecido como *Amostragem Esparsa*, obtendo uma aceleração quântica em comparação com o caso clássico que possibilitou a obtenção de um algoritmo quântico que exibe uma complexidade independente do número de estados do ambiente. Esta independência garante a sua adaptação para ambientes com um grande espaço de estados em que o planeamento pode ser intratável.

As questões mais pertinentes que se colocam é se é possível melhorar as instâncias oraculares de ambientes para lidar com ambientes ainda mais gerais, especialmente a capacidade de exprimir recompensas negativas como um mecanismo natural para feedback negativo em vez de alguma normalização da recompensa. Além disso, a extensão do algoritmo para realizar uma procura em árvore *informada* ao invés da procura não informada proposta. Melhorias neste resultado permitiriam a comparação entre o algoritmo quântico e os algoritmos clássicos mais recentes da Aprendizagem por Reforço.

PALAVRAS-CHAVE Computação Quântica, Amostragem Esparsa, Aprendizagem por Reforço Quântica

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

D

DP Dynamic Programming.

M

MDP'S Markov Decision Processes.

R

RL Reiforcement Learning.

1

INTRODUCTION

Since the dawn of time, humans strove with pattern matching in data. From Kepler's to Newton's astronomical data analysis, numerous mathematical methods were developed, methods that became automatic when digital computers appeared in the mid 20^{th} century. With the increase in computer power, we are capable of analyzing more complex problems and building powerful algorithms to find movies, jobs, manage investments and even new drugs that help increase not only human life conditions but also life expectancy. In the era of information as we increase the amount of data that we leave in our digital world, the more powerful these algorithms become. In the search for the ultimate algorithm, one that is able to learn anything by itself, we're reaching a bottleneck. These algorithms are hungry of data as well hungry of computer power which is reaching a fundamental limit as Moore's Law is coming to an end. The strategy adopted so far is of course the multi-core, multi-thread computing, in order to obtain speed by parallelizing the work done by the algorithms. If parallel computing is the new paradigm, Quantum Computers are the ultimate parallel computers, that work in a conceptually and fundamentally different way compared to the classical parallel computers. These new devices, still in their infancy already proved to have a role to play in modern computing, and therefore it is imperative that we strive for their development, as this new technology may help us bounce the local optimum we are currently in.

1.1 CONTEXT

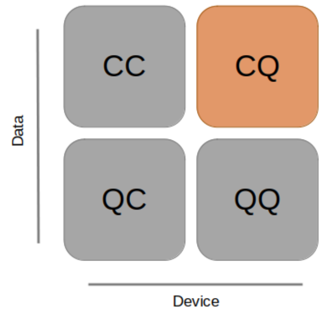


Figure 1.1.1: Four approaches to quantum machine learning, and the approach taken in this dissertation in orange

We decided to start this section with Figure 1.1.1 that summarizes the different approaches to the quantum machine learning wide landscape and state the approach taken in this dissertation. This description first appear in [6] where the authors distinguish the different approaches by whether the data generated is either classical **(C)** or quantum **(Q)** and the device that processes this data is a classical **(C)** or quantum **(Q)** device. The **CC** case refers, as one might guess, to classical machine learning, however, in this context it refers to the case where classical machine learning is enhanced by ideas taken from quantum computing. An example of this is Tensor Networks from many-body physics [14], for example in [31] the authors used Tensor Networks to devise a new formulation of Markov decision processes that provide the foundation for Reinforcement Learning. The **QC** case refers to the case whether classical machine learning could enhance quantum computations. There are several examples of this, for example the learning of quantum states and unitaries [18] and more recently, the use of reinforcement learning for the purpose of optimizing quantum error correction codes [50]. An approach towards better quantum error correction codes and fault-tolerant quantum computation in the community that's been in crescendo. The **QQ** case is one of the most interesting ones because deals with full quantum machine learning picture. This can be seen as data generated by a quantum system being fed into another quantum system, or in a more general setting, using a quantum computer to simulate the dynamics of a quantum system. The latter one is called *quantum simulation*, first suggested by Richard Feynman in [30] and could potentially lead to exponential speedups. This approach could have many applications. For example suppose nanorobots that work in the brain, perhaps restoring brain damaged structures. Due to the fact that true quantum structures called *microtubules* exist in our brains [34], and thus a kind of quantum computation may exist in our brains, the nanorobot could learn by interacting with a complex quantum environment. This is the notion of truly quantum reinforcement learning arises [27]. The **CQ** case deals with the case of quantum enhanced machine learning. Examples of this are well known by now, ranging from quantum enhanced supervised learning [57] to unsupervised learning problems like principal component analysis [45]. The quantum enhanced Reinforcement Learning algorithms developed in chapter 5 fits in this category. This can be viewed for example with a robot that interacts in the classical world, but enhanced with a quantum brain. This means that in this dissertation we aim not at the development of algorithms for quantum robots that work directly in complex quantum environments as described above, but rather to the enhancement of classical robots or more formally agents, by using a quantum simulation that we can hopefully exploit for better decision making. To make this work, we need to develop a notion of a quantum environment, i.e. the world that a quantum agent interacts with. Classically, reinforcement learning takes place in environments characterized by mathematical structures called *Markov Decision Processes* (see section 3.3), therefore we also need to have a mathematical formalism in order to turn a classical environment into a quantum one, Figure 1.1.2:

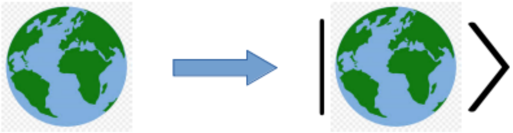


Figure 1.1.2: Conversion of classical environment into a quantum one.

Having a notion of a quantum environment, we are in position for constructing a quantum agent that using the laws of quantum mechanics instead of classical mechanics behaves in a fundamentally different way compared to classical counterpart. For example, the agent can now exploit the *superposition principle* in order to test different actions into the environment and record their results, the feedback returned by the environment, enabling the creation of parallel worlds as in Figure 1.1.3, expressed as a linear combination of multiple states, and therefore, allowing the change of both agent and world states in parallel.

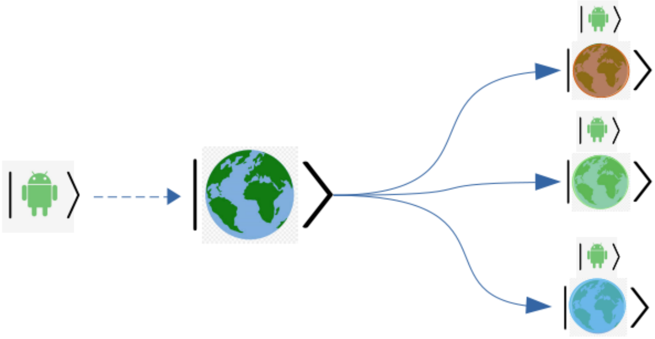


Figure 1.1.3: Parallel interaction of a quantum agent and the environment resulting in multiple agents acting in parallel

The interesting part is not the superposition itself. What good is it to have multiple superposed states if we can only read one classical state at a time once we measure the superposition? (*Holevo's Bound* [38]). Therefore in this work we aim at, given this parallel interaction, collapsing the superposition into a state that has the best possible score obtained by the agent. We will develop techniques for different cases, from the deterministic one, where an action leads always to the same state, to more general stochastic ones, where an action leads to a distribution of possible states and thereby different feedbacks or rewards.

1.2 MOTIVATION

Nowadays, Artificial Intelligence is an expression present in almost everyone's vocabulary and over time we see it more and more directly or indirectly becoming a part of us. However, in the present day, artificial intelligence is in its infancy, and most of the time it is confused with their machine learning sub-branches. Having the ability to correctly classify a new image, or having the ability to suggest a new song or a new movie based on a person's preferred genres, is far from having accomplished true intelligent systems, because seeing an object and understanding an object are completely different things. So if we want to build truly intelligent systems, the system has to be able to "understand" what it's being shown, not just map it mathematically. We need to take this into account, and not take AI for granted simply because we have these abilities. We must keep moving forward, otherwise, we will be stuck in local optima. The quest for the meaning of true intelligence is one of the most interesting problems and perhaps the most important one. This is where Reinforcement Learning enters and shines a completely different light to the design of systems that truly learn by themselves, mostly as a human does. Learning by interacting with our environment is probably the first to occur to us when we think about the nature of learning. When a child plays, waves its arms or looks around, it has no explicit teacher, but it does have a direct sensorimotor connection to its environment. Connecting with the environment produces cause and effect, about the consequences of actions, and about what to do to achieve goals. Throughout our lives, such interactions are undoubtedly a major source of knowledge about our environment and ourselves. Whether we are learning to drive a car or to hold a conversation, we are acutely aware of how our environment responds to what we do, and we seek to influence what happens through our behavior. Learning from interaction is a foundational idea underlying nearly all theories of learning and intelligence. [63]. This dissertation we want to take one step further, and see what can quantum computing offer to this picture. It was already proved to have a role to play in machine learning [57], whether to achieve truly artificial intelligence is still an open question and may well not be the case. Although it is still not proved that quantum computations occur in our brains, it may well be the case and surely will be interesting to analyze what can it offer to this new learning paradigm.

1.3 CONTRIBUTIONS

This dissertation aims at the construction of quantum algorithms for the RL paradigm. RL literature often starts by explaining concepts like *exploration/exploitation*, usually referring to the bandit problems, and from there building the mathematical structure behind RL to enable the construction of algorithms for more general frameworks. This dissertation follows roughly the same course:

- Development of an algorithm for the best arm identification that forms a central problem in bandit optimization.
- Notion of quantum *Markov Decision Processes* constructing oracular instantiations of classical environments. The notion first appeared in [27]. Our contribution relative to the original framework resides in the more general oracles, the oracles that characterize stochastic environments. Our proposal generalizes the original one by leaving the assumption on a sequence of actions that have a certain probability of being rewarded to work with the actual encoding of the rewards obtained by the agent in a sequence of actions. As we will see, this encoding will enable us to have a way of representing expected rewards which makes the decision making based on the rewards collected possible.
- Design of a quantum algorithm based on the idea of *sparse sampling* from the Reinforcement Learning literature, built on top of the oracular constructions mentioned above, obtaining a speed up compared to the classical counterpart. Given the stochasticity of the environments, these algorithms use sampling to form a distribution over the possible actions that the agent can make to this way reach the optimal action to take.
- A novel strategy to derive the optimal number of samples, based on the sample complexity of the algorithm.

1.4 OUTLINE

There are two kinds of audience this dissertation is aimed at, those who come from the field of Quantum Computing and want to know how can quantum algorithms enhance machine learning in general, and those who come from the field of Artificial Intelligence and have been recently introduced to the possibility of quantum algorithms enhancing machine learning. In either case, this dissertation starts with the respective background theory. In chapter 2, a brief review of the theory that leads to quantum computing as well as important quantum algorithms and quantum subroutines are presented. In chapter 3, a review of the theory of Reinforcement Learning is given, as well as some important algorithms in the literature to compare with the quantum algorithms developed that forms the core of this dissertation.

In chapter 4, a review of some known quantum enhancements for machine learning, in general, is given and a discussion of how these previous results can be generalized to the field of quantum reinforcement learning.

In chapter 5 two novel quantum algorithms that have a computational advantage relative to classical *sparse sampling* methods are presented as well as the complexity analysis of both algorithms and the conditions imposed in order to gain advantage.

Last but not least, chapter 6 is devoted not only to the summary of the results and open problems but also to enumerate a series of questions for further research that came along the work done in this dissertation.

2

QUANTUM INFORMATION AND QUANTUM ALGORITHMS

In this chapter, we will go over the theory behind quantum computation starting by introducing the formalism of quantum mechanics generalized by the use of the Dirac Notation [68], which is the standard in quantum computing. Then we will turn to the postulates of quantum information, introducing the *qubit* as the basic unit of quantum information and the fundamental operations used to manipulate it. Thereby, section 2.1 and section 2.2 serve as a reference for the notation used in the remainder of the text. In the sections ahead we cover some important quantum circuits and algorithms in order to provide the reader with some examples. We do this because the quantum algorithms and the techniques that we introduce here will play an important role in the algorithms developed in chapter 5 and hence a thorough understanding of the ideas presented here is central to fully grasp the ideas behind the algorithms that form the core of this dissertation.

2.1 QUANTUM MECHANICS POSTULATES

The basic principles of quantum mechanics can be summarized in four statements, which are commonly referred to as the postulates of quantum mechanics, providing a connection between the physical world and the mathematical formalism of quantum mechanics, and forming the framework on top of which the theory of quantum computing is built.

2.1.1 *State Space*

The first postulate, called *State Space*, sets the playground where quantum mechanics takes place.

Postulate 2.1.1: State Space

A quantum state is described by a unit vector living in a separable complex *Hilbert Space* $\mathcal{H}$.

We consider the discrete finite-dimensional Hilbert space over the complexes which is isomorphic to $\mathbb{C}^N$. Therefore a quantum state is a complex vector, known as a *complex amplitude vector* $\alpha = (\alpha_1, ..., \alpha_N)^T \in \mathbb{C}^N$, denoted by the "ket", $|\psi\rangle$. This leads to the radical idea that the *superposition* of the states belonging to $\mathcal{H}$ is again a state of the system. Thus, if we have $|\psi_1\rangle$ and $|\psi_2\rangle$ as possible states of the system:

$$|\psi\rangle = \alpha_1|\psi_1\rangle + \alpha_2|\psi_2\rangle \tag{2.1.1}$$

is also a possible state of the system. In general a state has the form:

$$|\psi\rangle = \sum_n \alpha_n|\psi_n\rangle \tag{2.1.2}$$

where if $|\psi\rangle$ is normalized and the basis states $|\psi_n\rangle$ are orthonormal then,

$$\sum_n |\alpha_n|^2 = 1 \tag{2.1.3}$$

and we can interpret $|\alpha_n|^2$ as the probability that a suitable measurement, also known as the collapse of the wavefunction, will find the system in state $|\psi_n\rangle$. The complex conjugate of the state vector is denoted by a "bra", $\langle\psi|$. The norm of a vector is defined in terms of the inner product, which is represented as the "bracket" $\langle\psi|\psi\rangle$. Furthermore, the norm of a vector $|\psi\rangle$ is:

$$||\psi|| = \sqrt{\langle\psi|\psi\rangle} \tag{2.1.4}$$

The inner product between two different quantum states, is a measure of the *"closeness"* between the two states, given that the more different the states are, the closer the inner product will be to zero and the similar the states are, the greater the value of their inner product.

The outer product is constructed as $|\psi\rangle\langle\varphi|$. For every vector $|\psi\rangle \in \mathcal{H}$ there is a complete orthonormal basis $\{|e_i\rangle\}, \forall i \in N$, meaning that the inner product of the basis elements are a delta function $\langle e_i|e_j\rangle = \delta_{ij}$ and:

$$|e_i\rangle\langle e_i| = \begin{pmatrix} e^1 \\ \vdots \\ e^N \end{pmatrix} \begin{pmatrix} e^1 \dots e^N \end{pmatrix} = \begin{pmatrix} e^1e^1 & e^1e^2 & \dots & e^1e^N \\ e^2e^1 & e^2e^2 & \dots & e^2e^n \\ \vdots & \vdots & \ddots & \vdots \\ e^Ne^1 & e^Ne^2 & \dots & e^Ne^N \end{pmatrix} \tag{2.1.5}$$

forms a *projection operator*, which when multiplied by some vector, projects the vector onto the subspace of the basis vector. This notion is important for the measurement of a quantum state. Projection operators can be used to change basis, expressing some state $|\psi\rangle$ in the basis $\{|e_i\rangle\}$:

$$|\psi\rangle = \sum_i |e_i\rangle\langle e_i|\psi\rangle = \sum_i \langle e_i|\psi\rangle|e_i\rangle \tag{2.1.6}$$

2.1.2 *Observables*

In physics, an observable is some quantity that can be measured like position or momentum. In quantum mechanics, it is an operator that acts on a quantum state, in general changing it.

Postulate 2.1.2: Observables

Every observable is described by an operator acting on the state that describes the system.

By convention, an operator $\mathcal{A}$ acts on a quantum state $|\psi\rangle$ as follows:

$$\mathcal{A}|\psi\rangle = |\psi'\rangle \tag{2.1.7}$$

For every operator, there are states that are not changed by the action of an operator, but scaled by some constant:

$$\mathcal{A}|\psi\rangle = a|\psi\rangle \tag{2.1.8}$$

These states are called *eigenstates* and the multiplication constant α is called an *eigenvalue* of the operator. If the state is an arbitrary superposition, then the operator acts linearly as:

$$\mathcal{A}(\alpha_1|\psi_1\rangle + \alpha_2|\psi_2\rangle) = \alpha_1\mathcal{A}|\psi_1\rangle + \alpha_2\mathcal{A}|\psi_2\rangle = a_1|\psi_1\rangle + a_2|\psi_2\rangle \tag{2.1.9}$$

2.1.3 *Time Evolution*

Postulate 2.1.3: Time-Evolution

The time evolution of the state of a quantum system is described by some unitary operator U

The evolution of a quantum mechanical system is described by the well known Schrödinger equation:

$$i\hbar\frac{d}{dt}|\psi\rangle = H|\psi\rangle \tag{2.1.10}$$

where H denotes the Hamiltonian of the system and $\hbar$ the Plank's constant. In the case of the Hamiltonians being time-independent, the solutions of the equation starting in an initial state $|\psi_0\rangle$ are given by:

$$|\psi(t)\rangle = U(t)|\psi_0\rangle \tag{2.1.11}$$

where:

$$U(t) = e^{-i\frac{t}{\hbar}H} \tag{2.1.12}$$

is the unitary time-evolution operator. Cleraly, $U(t)$ corresponds to a unitary matrix.

2.1.4 *Measurement*

The only possible result of the measurement of an observable $\mathcal{A}$ is one of the eigenvalues of the corresponding operator and since we measure only real values, the eigenvalues correponding to observables are also real, therefore operators are *Hermitian*.

Postulate 2.1.4: Measurement

When a measurement is made on a generic state $|\psi\rangle$, the probability of obtaining an eigenstate $|\psi_i\rangle$ is given by the square of the inner product of $|\psi\rangle$ and the eigenstate, $|\langle\psi_i|\psi\rangle|^2$.

The Measurement postulate is also known as the *Born Rule*. If we have some arbitrary state of the form:

$$|\psi\rangle = \alpha_1|\psi_1\rangle + \alpha_2|\psi_2\rangle \tag{2.1.13}$$

then we will measure each eigenstate with probability:

$$|\langle\psi_n|\psi\rangle|^2 = \begin{cases} {\alpha_1}^2 & if \quad |\psi_n\rangle = |\psi_1\rangle \\ {\alpha_2}^2 & if \quad |\psi_n\rangle = |\psi_2\rangle \end{cases} \tag{2.1.14}$$

Moreover, if we have an operator $\mathcal{A}$ of the form:

$$\mathcal{A} = \sum_n \rho_n|\psi_n\rangle\langle\psi_n| = \rho_1|\psi_1\rangle\langle\psi_1| + \rho_2|\psi_2\rangle\langle\psi_2| \tag{2.1.15}$$

acting on $|\psi\rangle$, we measure the system with probability:

$$|\langle\psi_n|\mathcal{A}|\psi\rangle|^2 = \begin{cases} (\alpha_1\rho_1)^2 & if \quad |\psi_n\rangle = |\psi_1\rangle \\ (\alpha_2\rho_2)^2 & if \quad |\psi_n\rangle = |\psi_2\rangle \end{cases} \tag{2.1.16}$$

2.1.5 *Composite Systems*

This subsection does not discuss a postulate per se, but rather an important notion to have on the composition of quantum systems. Suppose that we have a quantum system represented by the Hilbert space $\mathcal{H}$, that is composed by two subsystems $\mathcal{H}_i, i = 1, 2$, with dimension $\dim(\mathcal{H}_i) = N_i$, each one spanned by orthonormal basis set $\{|e^i_j\rangle\}, j \in N_i$. Then, the total system is represented by the tensor product of the two subsystems:

$$\mathcal{H} = \mathcal{H}_1 \otimes \mathcal{H}_2 \tag{2.1.17}$$

of dimension $\dim(\mathcal{H}) = N_1 N_2$. The states in this joint Hilbert space can be written as:

$$\sum_j \sum_k \alpha_{jk} |e^1_j\rangle \otimes |e^2_k\rangle = \sum_j \sum_k \alpha_{jk} |e^1_j\rangle |e^2_k\rangle \tag{2.1.18}$$

Meaning that if we have a third quantum system represented by an Hilbert space $\mathcal{H}_3$ with an orthonormal basis set $\{|b_j\rangle\}, j \in \mathbb{N}$, we can extend the original Hilbert space, by the tensor product, generating a new Hilbert space:

$$\mathcal{H} \otimes \mathcal{H}_3 = \sum_j \sum_k \sum_l \alpha_{jkl} |e^1_j\rangle \otimes |e^2_k\rangle \otimes |b_l\rangle \tag{2.1.19}$$

Sometimes, when having a composite system of the form of $\mathcal{H}$, we are interested in operators $\mathcal{A}$ that act only on one of two subsystems. For this we need to construct a new operator:

$$\mathcal{A}' = \mathcal{A} \otimes \mathbb{1} \tag{2.1.20}$$

such that $\mathcal{A}$ operate on the first subsystem and $\mathbb{1}$ (the identity operator) leaves the second subsystem intact. An important notion to have is a quantum feauture called *entanglement*. Suppose that we have a composite system of two isolated physical systems. Then, we say that the isolated physical systems are *entangled* when we cannot write the state of the composite system as a simple tensor product:

$$|\psi\rangle = |\psi_1\rangle \otimes |\psi_2\rangle \tag{2.1.21}$$

i.e. we cannot have a description of each individual part of system, but only a description of the overall system. For a better understanding, we will revisit this feature in next section, when introducing qubits.

2.2 QUBIT SYSTEMS

There are various computational models that formalize the idea of quantum computation, however most of them are based on the concept of a qubit, the quantum analog of a bit. The claim for the power of a qubit is that the quantum system can be in a linear combination of 0 and 1. This is not entirely true, because a random bit also has this property to an extent. The major difference stems from the fact that the amplitudes in a qubit can be complex numbers and certain evolutions can make the amplitudes *interfere* with each other. Together with other subtleties, like *entanglement* leads to statistics that are fundamentally different from what can be reproduced classically [57].

A qubit is a quantum mechanical system with two levels that can be measured in one of the two possible states, $|0\rangle$ and $|1\rangle$. These vectors form an orthonormal basis of an Hilbert space isomorphic to $\mathbb{C}^2$ called the *computational basis*. Therefore, a qubit can be expressed as a linear combination of both basis states:

$$|\psi\rangle = \alpha_0|0\rangle + \alpha_1|1\rangle \tag{2.2.1}$$

where $\alpha_0, \alpha_1 \in \mathbb{C}$ and the state vector is normalized, thus, $|\alpha_0|^2 + |\alpha_1|^2 = 1$. The Hermitian conjugate is the state:

$$\langle\psi| = \alpha_0^*\langle 0| + \alpha_1^*\langle 1| \tag{2.2.2}$$

The quantum state can be represented as a vector of amplitudes:

$$|\psi\rangle = \begin{pmatrix} \alpha_0 \\ \alpha_1 \end{pmatrix} \in \mathbb{C}^2 \tag{2.2.3}$$

representing the standard basis states as the vectors:

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \in \mathbb{C}^2 \qquad\qquad |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \in \mathbb{C}^2$$

We will mostly use the Dirac notation throughout the text, however sometimes we will resort to the usual vector notation, especially in the representation of unitary matrices as quantum operators. Another useful representation of a qubit, stems from a geometrical representation, known as the *Bloch Sphere* (Figure 2.2.1). A generic qubit can be in fact parametrized as:

$$|\psi\rangle = e^{i\phi}\left(cos\frac{\theta}{2}|0\rangle + e^{i\varphi}sin\frac{\theta}{2}|1\rangle\right) \tag{2.2.4}$$

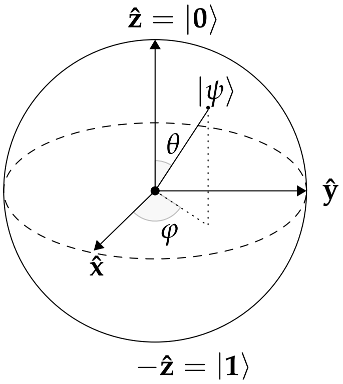

$\hat{\mathbf{z}} = |\mathbf{0}\rangle$
$|\psi\rangle$
θ
$\hat{\mathbf{y}}$
φ
$\hat{\mathbf{x}}$
$-\hat{\mathbf{z}} = |\mathbf{1}\rangle$

Figure 2.2.1: Bloch Sphere

The global phase of ϕ has no observable effect, therefore can be ommited in the Bloch sphere representation. However, it will have its own role to play later on. The angles $0 \leq \theta \leq \pi$ and $0 \leq \varphi \leq 2\pi$ have the interpretation of spherical coordinates, so that $|\psi\rangle$ can be visualized as a vector $(sin\theta cos\varphi, sin\theta sin\varphi, cos\varphi)$ pointing from the origin to the surface of the sphere.

As we have seen in subsection 2.1.3, the evolution of a quantum system is done via quantum operators, that are represented as unitary matrices, also called as *quantum gates* in the *circuit model of quantum computation*. This is a well-known model in the community and the model that we follow in this dissertation. The model has constraints such as:

- A set of qubits initialized in the ground state $|0\rangle$
- A sequence of unitary transformations transforms the initial ground state.
- Computational basis measurements performed to retrieve the result of some computation.

A qubit is in $\mathbb{C}^2$, therefore, single-qubit gates are formally described by $2x2$ unitary transformations, being the well known *Pauli matrices*, some useful examples:

$$\sigma_x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \qquad \sigma_y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} \qquad \sigma_z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

One can see that the σ_x unitary acts as a *NOT* gate, inverting the amplitude of the quantum state:

$$\sigma_x|0\rangle = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix} = |1\rangle \tag{2.2.5}$$

The Pauli-z gate, σ_z acts as a phase gate, inverting the phase of a quantum state that is in state $|1\rangle$:

$$\sigma_z|1\rangle = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ -1 \end{pmatrix} = -|1\rangle \tag{2.2.6}$$

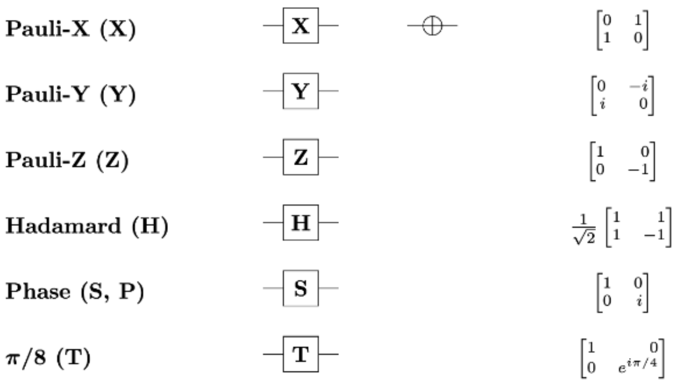
This is one example showing that the global phase is irrelevant, that is, it has no observable effect, because if we measure the state, we measure $|1\rangle$ with the square of the amplitude, therefore, it doesn't affect the state. However, it doesn't mean that this gate is useless, as we will see in section 2.5, inverting the phase will serve as a mechanism for amplifying the amplitude of a state.
One of the most important gates used in quantum computation, is the *Hadamard gate*:

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \tag{2.2.7}$$

This gate is responsible for creating uniform superpositions, by acting on basis states:

$$H|0\rangle = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \frac{1}{\sqrt{2}} \left(|0\rangle + |1\rangle\right) \tag{2.2.8}$$

$$H|1\rangle = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \frac{1}{\sqrt{2}} \left(|0\rangle - |1\rangle\right) \tag{2.2.9}$$



Pauli-X (X)	X	⊕	$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$
Pauli-Y (Y)	Y		$\begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}$
Pauli-Z (Z)	Z		$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$
Hadamard (H)	H		$\frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$
Phase (S, P)	S		$\begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix}$
$\pi/8$ (T)	T		$\begin{bmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{bmatrix}$

Figure 2.2.2: Single qubit gates and their matrix representation

What if we don't want to create a uniform superposition state, but rather an arbitrary superposition:

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle \tag{2.2.10}$$

It is easy to see that if we have a qubit initialized in the ground state, and rotate it in the y-direction Figure 2.2.3, we can induce an amplitude change.

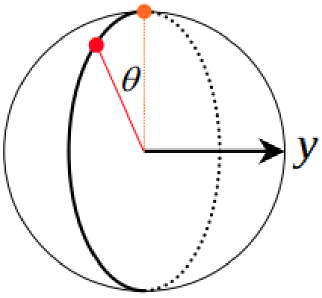


Figure 2.2.3: y-rotation of an angle θ on a qubit. Image from School on Quantum Computing, Keio University

$$R_y(\theta) \equiv e^{i\frac{\theta}{2}\sigma_y} = cos\frac{\theta}{2}\mathbb{1} - isin\frac{\theta}{2}\sigma_y = \begin{pmatrix} cos\frac{\theta}{2} & -sin\frac{\theta}{2} \\ sin\frac{\theta}{2} & cos\frac{\theta}{2} \end{pmatrix} \tag{2.2.11}$$

Therefore if we act with this unitary matrix on the ground state, we prepare the superposition:

$$R_y(\theta)|0\rangle = \begin{pmatrix} cos\frac{\theta}{2} & -sin\frac{\theta}{2} \\ sin\frac{\theta}{2} & cos\frac{\theta}{2} \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = cos\frac{\theta}{2}|0\rangle + sin\frac{\theta}{2}|1\rangle \tag{2.2.12}$$

At this point it is a matter of simple arithmetic to derive the right angle to prepare the desired superposition. As an example, suppose that we want to prepare the superposition, such that, when the state is measured, we read the basis state $|1\rangle$ $\frac{3}{4}$ of the time, then the angle becomes:

$$sin^2\frac{\theta}{2} = \frac{3}{4} \quad \rightarrow \quad \theta = 2arcsin\sqrt{\frac{3}{4}} \tag{2.2.13}$$

Of course, one qubit is interesting but only one qubit will get us so far, thus the goal is to create networks of qubits. As we have seen in subsection 2.1.5, we can form a composite system by the tensor product of sub systems, therefore we can create the tensor product of n qubits:

$$|\psi\rangle = |q_1\rangle \otimes \cdots \otimes |q_n\rangle \tag{2.2.14}$$

If we apply the Hadamard gate to each one the n qubits, $H^{\otimes n}$, we get a superposition state composed by the 2^n basis states:

$$|\psi\rangle = \alpha_1|0\ldots0\rangle + \alpha_2|0\ldots01\rangle + \cdots + \alpha_{2^n}|1\ldots1\rangle \tag{2.2.15}$$

with $\alpha_i \in \mathbb{C}$ and $\sum_{i=1}^{2^n} |\alpha_i|^2 = 1$.

The fundamental two-qubit gate is known as the *CNOT* gate, in other words, the *controlled-not* gate. The state of a qubit is changed (target qubit), based on the value of another qubit (control qubit):

$$CNOT : |a\rangle \otimes |b\rangle \mapsto |a\rangle \otimes |b \oplus a\rangle \tag{2.2.16}$$

The gate can be constructed using the Dirac notation:

$$CNOT \equiv |0\rangle\langle 0| \otimes \mathbb{1} + |1\rangle\langle 1| \otimes \sigma_x \tag{2.2.17}$$

$$CNOT = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

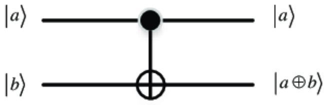


Figure 2.2.4: CNOT gate circuit representation

meaning that if the first qubit is in the state zero, it applies the identity matrix to the second qubit, otherwise it applies the not gate, inverting the amplitude of the second qubit. The CNOT gate is a special case of more general controlled gate, in fact, replacing σ_x in Equation 2.2.17 by an arbitrary single qubit gate, we have a prescription for constructing arbitrary controlled gates.

$$cU \equiv |0\rangle\langle 0| \otimes \mathbb{1} + |1\rangle\langle 1| \otimes U \tag{2.2.18}$$

Using the CNOT gate simultaneously with an Hadamard gate we can construct an *entangling* circuit, thus, creating entanglement between two quantum bits

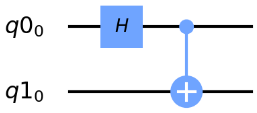


Figure 2.2.5: Entangling circuit: Both qubits start in the ground state, and the evolution is made from left to right

$$\begin{aligned} CNOT(H \otimes \mathbb{1})(|0\rangle \otimes |0\rangle) &= CNOT(\frac{1}{\sqrt{2}}|0\rangle \otimes |0\rangle + \frac{1}{\sqrt{2}}|1\rangle \otimes |0\rangle) \\ &= \frac{1}{\sqrt{2}}(|0\rangle \otimes |0\rangle + |1\rangle \otimes |1\rangle) \end{aligned} \tag{2.2.19}$$

In subsection 2.1.5 we said that two quantum systems are entangled if we cannot write the state of the overall system as a tensor product expression. In fact, Equation 2.2.19 cannot be expressed as one such expression. As an example, consider the uniform superposition state over two qubits:

$$H^{\otimes 2}|00\rangle = \frac{1}{2}|00\rangle + \frac{1}{2}|01\rangle + \frac{1}{2}|10\rangle + \frac{1}{2}|11\rangle \tag{2.2.20}$$

The superposition state can be expressed as a single tensor product:

$$\left(\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle\right) \otimes \left(\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle\right) \tag{2.2.21}$$

therefore it is not an entangled state, proving that the circuit of Figure 2.2.5 is in fact an entangling circuit.

At last, we just need to refer another important gate, which is the CCNOT gate, also known as the *Toffoli* gate. It is the quantum version of the AND gate, that acts on three qubits: it has two control qubits, and it flips the third qubit if and only if both control qubits are in state $|1\rangle$.

$$CCNOT : |a\rangle \otimes |b\rangle \otimes |c\rangle \mapsto |a\rangle \otimes |b\rangle \otimes |c \oplus (a.b)\rangle \tag{2.2.22}$$

$$CCNOT = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

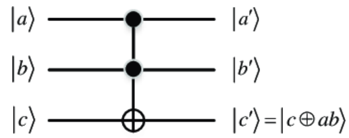


Figure 2.2.6: CNOT gate circuit representation.

The Toffoli gate is a multi qubit gate, thus needs to be decomposed into a sequence of elementary gates as follows:

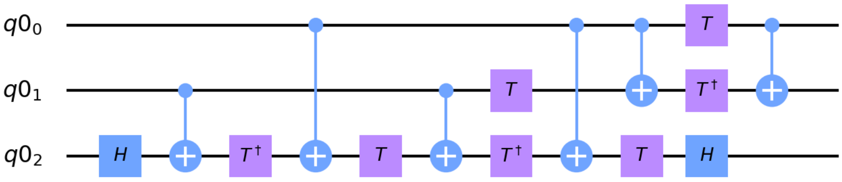


Figure 2.2.7: Toffoli gate decomposed with elementary gate complexity of 16

2.3 QUANTUM STATE PREPARATION

As we said in chapter 1, the focus of this dissertation is in the **CQ** approach which deals with classical data encoded into a quantum state. Thus we need to have efficient ways of performing *state preparation*. In fact, state preparation may not be of great interest for quantum computing in general, for example in Grover's algorithm section 2.5 the state preparation routine is simply the encoding of an n-qubit uniform superposition, which is efficiently done by the tensor product of n Hadamard gates. However, in the context of machine learning, including big data applications, it is of great importance to consider the encoding procedure, because the encoding itself is part of the algorithm and may account for a crucial part of its complexity [57]. Due to the fact that most quantum algorithms are probabilistic by nature, we'll need to repeat the algorithm several times, thus to encode the same classical data several times. Below we discuss some known state preparation routines.

2.3.1 *Basis Encoding*

Suppose we are given a dataset $\mathcal{D}$ with N input data points. We need to encode N datapoints as basis states. For the sake of simplicity let us consider the dataset being in fact binary, however in contrary we could neglect the cost of converting the dataset into binary. We want to prepare the state:

$$|\mathcal{D}\rangle = \frac{1}{\sqrt{N}} \sum_{i=0}^{N-1} |x_i\rangle \tag{2.3.1}$$

An important notion to have is that in the case of datapoints with $\mathcal{O}(n)$ bits, we need $\mathcal{O}(Nn)$ classical bits to store the data. However, in the quantum setting, we just need $\mathcal{O}(n)$ qubits due to the fact that we're preparing the uniform superposition of the datapoints. It is the most straightforward encoding scheme, and one of the most important ones because we can prepare unitaries that represent some machine learning model $\mathcal{M}$, and compute the result of acting on the N input datapoints in parallel:

$$\frac{1}{\sqrt{N}} \sum_{i=0}^{N-1} |x_i\rangle|0\rangle^{\otimes n} \mapsto \frac{1}{\sqrt{N}} \sum_{i=0}^{N-1} |x_i\rangle|\mathcal{M}(x_i)\rangle \tag{2.3.2}$$

Several approaches to obtain $|\mathcal{D}\rangle$ have been proposed by now. The most promising one is based on the notion of a *quantum random access memory* [32], whose the purpose is to read the entire dataset in parallel. This devices query index positions of memory $|i\rangle$ in parallel, loading the i^{th} data point basis encoded:

$$\frac{1}{\sqrt{N}} \sum_{i=0}^{N-1} |i\rangle|0\rangle \mapsto \frac{1}{\sqrt{N}} \sum_{i=0}^{N-1} |i\rangle|x_i\rangle \tag{2.3.3}$$

This approach promises query access in logarithmic time in terms of the size of the dataset, however, a practical implementation of these devices is still lacking, so we're left with sequential approaches, i.e, preparing each datapoint at a time.
One interesting sequential approach is the branching technique [71] leading to a linear time state preparation $\mathcal{O}(N)$. As we don't use this encoding strategy in the algorithms developed in this dissertation, it will not be explained here. However, we refer the curious reader to [57] chapter 5.1 for a thorough explanation of the algorithm. In the next section, we present an encoding strategy that generalizes this one.

2.3.2 *From Amplitude encoding to QSamples*

Several quantum machine learning algorithms, instead of encoding a dataset in basis states, encode rather in the amplitudes of quantum states, for example, given the data vector $x = (x_0, x_1, x_2, ..., x_N)^T$, we prepare the state:

$$|x\rangle = \sum_{i=0}^{N-1} x_i |i\rangle \tag{2.3.4}$$

constraining the encoded amplitudes to a normalization constant such that $\sum_i^{N-1} |x_i|^2 = 1$.
The major advantage of this approach is that we can encode the N-dimensional datapoints using only $n \in log(N)$ qubits. Thus any polynomial circuit applied to the n-qubit register encoding the data constitutes only a polylogarithmic computation relative to the size of the data-vector, and this is at the basis of all possible exponential improvements with quantum computing [26]. The disadvantage of this approach relies on the fact that to be able to encode a dataset into the amplitudes of a quantum state we need to normalize the data first. Furthermore, caution is necessary when measuring the quantum state if the result of some computation is in the amplitude because the number of measurements needed to retrieve it, scale with the number of amplitudes.
One common technique to encode data in to the amplitudes of quantum states is based on the routine devised in [49] where the authors considered the opposite of the problem, instead of starting with some ground state and apply unitary operations to encode data, they considered the problem of how to map some final state $|x\rangle$ back to the ground state. **For the purpose of state preparation, all that one has to do is to invert all gates used and apply them in the reverse order**. This is done with *multi control Ry , Rz gates* on a target qubit. Composed of all possible controlled rotations on said qubit, this can be used to address individual elements of the state vector. The algorithm is based on 2 *cascades* of rotation operations, one cascade of R_z gates that equalizes the phases of the state vector, and one cascade of R_y rotation gates in order to rotate the now real state vector into the direction of the ground state. If one wants to encode real amplitude vectors, then the cascade of R_z gates

can be ignored and the general circuit for the preparation becomes just a sequence of R_y gates:

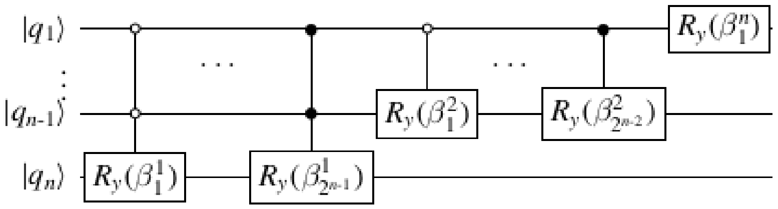


Figure 2.3.1: Real amplitude vector preparation, based on controlled R_y gates. Image from [57]

.

where the angles β are defined in terms of the original components of the amplitude vector:

$$\beta_j^s = 2arcsin\left(\frac{\sqrt{\sum_{l=1}^{2^{s-1}} |x_{(2j-1)2^{s-1}+l}|^2}}{\sqrt{\sum_{l=1}^{2^s} |x_{(j-1)2^s+l}|^2}}\right) \tag{2.3.5}$$

This routine scales linearly with the size of the input vector. However, there's a problem with this approach: it relies on the use of multi control rotation gates, which need to be decomposed into elementary gates, which becomes exponentially heavier with the increase of the control qubits. To see that, we need to take into account that a n-qubit controlled unitary U (in our case U is a R_y or R_z gate) can be decomposed in the following v-shaped sequence of gates:

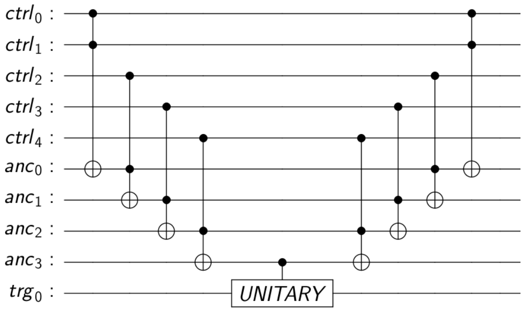


Figure 2.3.2: n=5 qubit controlled unitary decomposed into Toffoli gates and single control unitary using n-1 ancillas.

and each Toffoli gate is decomposed with elementary gates such that,

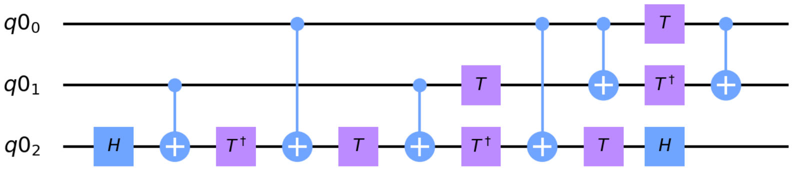


Figure 2.3.3: Toffoli gate decomposed with elementary gate complexity of 16

Gates that act on three or more qubits are prohibitively difficult to implement directly. Thus, implementing a quantum computation as a sequence of two-qubit gates is of crucial importance. In [59] the authors followed that intuition improving the previous technique by developing a new recursive decomposition of multiple controlled rotation gates, based only on CNOT gates and single control rotation gates, proving that an arbitrary n-qubit quantum state can be prepared by a circuit containing no more than $2^{n+1} - 2n$ CNOT [1] gates.

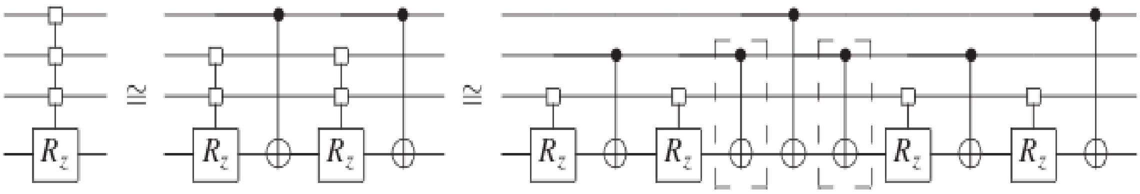


Figure 2.3.4: Recursive decomposition of multi control rotation gates [59] where the white squares represent any control state

This encoding strategy is widely used in many quantum computing platforms, including IBM Qiskit [1] platform in which the algorithms developed in chapter 5 were simulated.

In Machine Learning we're often interested in sampling from a probability distribution due to the fact that sampling provides a flexible way to approximate many sums and integrals at reduced cost. In some cases, our learning algorithm requires us to approximate an intractable sum or integral and in many other cases, sampling is actually our goal, in the sense that we want to train a model that can sample from the training distribution [39], so we need to be able to encode probability distributions in the quantum framework as well.

We can encode a classical discrete probability distribution over binary strings given by the vector $(p_1, ..., p_N)^T$ as a real amplitude vector $(\sqrt{p_1}, ..., \sqrt{p_N})^T$ represented by the following quantum state:

$$|\psi\rangle = \sum_{i=1}^{N} \sqrt{p_i}|i\rangle \tag{2.3.6}$$

[1] implementations of CNOT gates are orders of magnitude more error-prone than implementations of single-qubit gates and have longer durations, thus the cost of a quantum circuit can be realistically calculated by counting the number of CNOT gates [59]

in the community referred as a *qsample* [46], and the encoding of discrete probability distribuitions referred as qsample encoding. As one might have guessed, state preparation works the same way as amplitude encoding. The only difference is that the amplitude vector is composed with a probability distribuition instead, so there's no need to worry about the normalization of the data, because the amplitudes are already normalized. Thus we can also use the strategy documented in [49] as well. A major advantage in encoding probability distributions is that we can prepare joint distributions "for free". Suppose that we prepared the following qsamples:

$$|\phi\rangle = \sum_{i=0}^{N} \sqrt{\phi_i}|i\rangle \tag{2.3.7}$$

$$|\rho\rangle = \sum_{j=0}^{N} \sqrt{\rho_j}|j\rangle \tag{2.3.8}$$

Tensoring both qsamples, yields the joint distribution:

$$|\psi\rangle = |\phi\rangle \otimes |\rho\rangle = \sum_{i,j}^{N} \sqrt{\phi_i\rho_j|i\rangle}|j\rangle \tag{2.3.9}$$

2.4 AMPLITUDE AMPLIFICATION

Suppose that we have a quantum algorithm $\mathcal{A}$ applied to a n qubit ground state such that:

$$\mathcal{A}|0\rangle^{\otimes n} \mapsto |\psi\rangle = sin(\theta)|\psi_{good}\rangle + cos(\theta)||\psi_{bad}\rangle, \quad \theta \in (0, \frac{\pi}{2}) \tag{2.4.1}$$

and we want to construct the state spanned by the good state $|\psi_{good}\rangle$. If we measure $|\psi\rangle$ we read a good state with probability:

$$sin^2\theta = a = |\langle\psi_{good}|\psi\rangle|^2 \tag{2.4.2}$$

Depending on the value of θ we may or may not have a good probability of reading a good state. The amplitude amplification algorithm [20] promises to construct a good state after an expected number of applications of $\mathcal{A}$ and its inverse $\mathcal{A}^{-1}$ which is proportional to $\frac{1}{\sqrt{a}}$ assuming the algorithm A makes no measurements. This algorithm is a subroutine in a variety of quantum algorithms as well as the generalization of the well-known *Grover's Algorithm* [33]. So it makes sense to be reviewed first because it will help to understand better the quantum search algorithms in section 2.5 and the quantum optimization algorithm in section 2.6.

The purpose of the algorithm is to amplify the probability associated with a quantum state so that when measuring the quantum state we have a higher chance of reading a good state. Suppose the quantum operator $\mathcal{R}_{good}$ that inverts the phase associated with good states:

$$\mathcal{R}_{good}|x\rangle = \begin{cases} -|x\rangle & if \quad |x\rangle = |\psi_{good}\rangle \\ |x\rangle & if \quad |x\rangle = |\psi_{bad}\rangle \end{cases}$$

$$\mathcal{R}_{good}|\psi\rangle \mapsto -sin(\theta)|\psi_{good}\rangle + cos(\theta)|\psi_{bad}\rangle, \quad \theta \in (0, \frac{\pi}{2}) \tag{2.4.3}$$

Notice that $|\psi_{good}\rangle$ and $|\psi_{bad}\rangle$ form an orthonormal basis. So we can visualize the effect of amplitude amplification graphically, by representing $|\psi\rangle$ in the unit circle in $\mathbb{R}^2$:

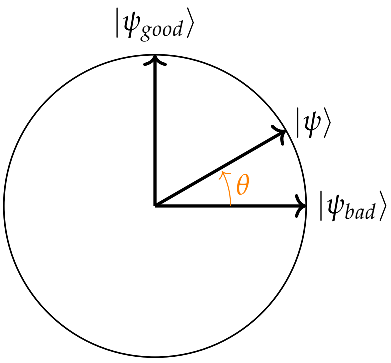


Figure 2.4.1: Arbitrary state generated by $\mathcal{A}$

We can see that the $\mathcal{R}_{good}$ operator that inverts the phase of good states, acts as a reflection operator, by shifting the phase of $|\psi_{good}\rangle$, we're reflecting $|\psi\rangle$ around $|\psi_{bad}\rangle$ generating a new state $|\psi'\rangle$. After the first reflection, if one reflects around $|\psi\rangle$ instead, we generate a new quantum state with an angle of 3θ. This is explained by noticing that the reflection around $|\psi\rangle$ in fact induces a rotation of 2θ as depicted in Figure 2.4.2.

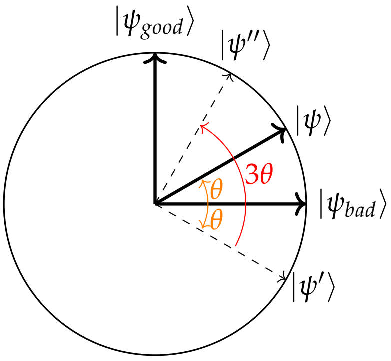


Figure 2.4.2: Applying $\mathcal{R}_{good}$ followed by a reflection around $|\psi\rangle$

By interleaving these two types of reflection operators, the reflection around the bad states, $\mathcal{R}_{good}$ and the reflection around $|\psi\rangle$ that we now call $\mathcal{R}_{\psi}$ we can amplify the amplitude of the good states.

$$\mathcal{G} = \mathcal{R}_{\psi}\mathcal{R}_{good} \tag{2.4.4}$$

At this point one may ask: *"$\mathcal{R}_{good}$ is trivially implemented but how can we construct $\mathcal{R}_{\psi}$?"*
To that end, consider the reflection around the all-zero state, $\mathcal{R}_0$:

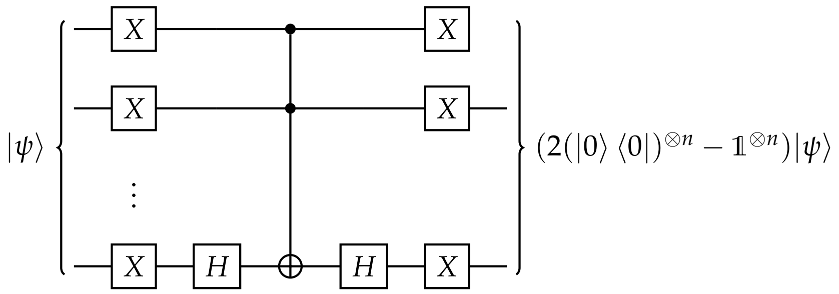

$|\psi\rangle$
X
X
X
H
H
X
X
X
$(2(|0\rangle\langle 0|)^{\otimes n} - \mathbb{1}^{\otimes n})|\psi\rangle$

Figure 2.4.3: Reflection around the all-zero state circuit

Recall that the circuit $\mathcal{A}$ prepared $|\psi\rangle$. Now that we have a circuit for the reflection around the all zero state, it turns out that we can use it to construct the reflection around $|\psi\rangle$:

$$\mathcal{R}_{\psi} = \mathcal{A}\mathcal{R}_0\mathcal{A}^{-1} = 2|\psi\rangle\langle\psi| - \mathbb{1}^{\otimes n} \tag{2.4.5}$$

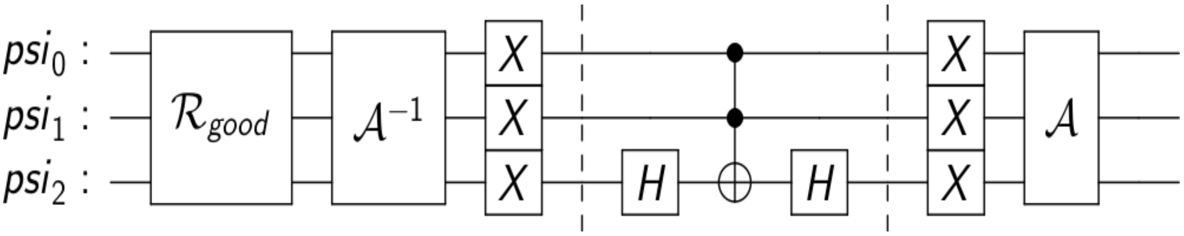


Figure 2.4.4: Quantum circuit for the $\mathcal{G}$ operator acting on a 3-qubit state

One iteration of the operator $\mathcal{G}|\psi\rangle$ gives the state:

$$\mathcal{G}|\psi\rangle = sin(3\theta)|\psi_{good}\rangle + cos(3\theta)|\psi_{bad}\rangle \tag{2.4.6}$$

So we need to apply $\mathcal{G}$, a number of times j such that the probability of reading a good state becomes close to 1:

$$\mathcal{G}^j|\psi\rangle = sin((2j+1)\theta)|\psi_{good}\rangle + cos((2j+1)\theta)|\psi_{bad}\rangle \tag{2.4.7}$$

$$sin^2((2j+1)\theta) \approx 1 \tag{2.4.8}$$

$$j = \lfloor \frac{\pi}{4\theta} \rfloor \tag{2.4.9}$$

provided that we know the value of θ, i.e. we know the initial probability distribuition prepared by the circuit $\mathcal{A}$. In section 2.5 we will explore the case when we don't know the initial probability distribuition, in the context of quantum searching.

2.5 QUANTUM SEARCH

Suppose that we have an array $\mathcal{A}$ with N elements, $\mathcal{A}[0, ..., N-1]$ and we want to find an element x in $\mathcal{A}$. More precisely, we want to find the index i such that $\mathcal{A}[i] = x$. Classically, if the array was sorted, we could find i in $\mathcal{O}(log(N))$. However, if we have an unstructured array, no classical algorithm can find i with more than $\frac{1}{2}$ probability without looking at least at half of the entries of the array.

The well-known *Grover's Algorithm* is a quantum algorithm that solves this problem in expected time $\mathcal{O}(\sqrt{N})$, i.e. provides a quadratic speed-up compared to any classical algorithm. However, Grover's original algorithm [33], deals with the case that the element x is unique in $\mathcal{A}$, hence we know that the search problem has exactly one solution. In the field of quantum information in general we're often interested in problems that may have multiple solutions or even problems for which the number of solutions is unknown. In the context of quantum machine learning, we're interested in encoding a dataset $\mathcal{D}$ into a quantum state $|\mathcal{D}\rangle$ represented by the superposition of the input data points in $\mathcal{D}$ and then use Grover's algorithm to search through the dataset. Suppose that $\mathcal{D}$ is basis encoded (subsection 2.3.1):

$$|\mathcal{D}\rangle = \frac{1}{\sqrt{|\mathcal{D}|}} \sum_{i \in |\mathcal{D}|} |x_i\rangle \tag{2.5.1}$$

Grover search rotates this data superposition by the average. However, if $\mathcal{D}$ is a sparse dataset, Grover's algorithm will rotate the terms that have zero amplitude over the average, thus assigning a non-zero amplitude to the points that are not in $\mathcal{D}$ decreasing the probability of measuring the desired result significantly. [71].

The original algorithm assumes an initial uniform amplitude distribution. In the field of quantum machine learning we're also interested, as in the classical case, on the encoding of probability distributions into quantum states. Suppose the discrete probability distribution $\mathcal{P}$ as a *qsample* subsection 2.3.2:

$$|\mathcal{P}\rangle = \sum_i \sqrt{\mathcal{P}_i} |i\rangle \tag{2.5.2}$$

Qsamples will represent arbitrary probability distributions, so clearly it is of great importance to know the behaviour of the Grover's algorithm in these cases. In this section, we're going from the original Grover's algorithm to the variants that generalizes the algorithm in order to deal with the above mentioned problems.
As we said before, *Grover's Algorithm* is a first step towards the amplitude amplification technique, the only differences are that the $\mathcal{R}_{good}$ operator is viewed as a *phase oracle* U_f that implements a function f that gives the result $f(x) = 0$ for unwanted states and $f(x) = 1$ for the wanted state, thus inverting the phase of the element x that we want to find:

$$U_f|x\rangle = (-1)^{f(x)}|x\rangle = \begin{cases} -|x\rangle & if \quad f(x) = 1 \\ |x\rangle & if \quad f(x) = 0 \end{cases}$$

and the quantum circuit that prepares the initial state $\mathcal{A}$ in fact prepares an uniform superposition over the search space. Recall the operator $\mathcal{G}$ (section 2.4) to be refered in the sequel as the *Grover Iterate*:

$$\mathcal{G} = \mathcal{R}_{\psi}\mathcal{R}_{good} = \mathcal{R}_{\psi}U_f = (2|\psi\rangle\langle\psi| - \mathbb{I}^{\otimes n})U_f \tag{2.5.3}$$

This algorithm has a nice geometrical interpretation in terms of the two reflection operators, that can visualized in the unit circle in $\mathbb{R}^2$ as we did in section 2.4. Let's consider the case of $N = 4$ and suppose that we want to find the state $|x\rangle = |10\rangle$. The only two special states we need to consider are $|x\rangle$ and the uniform superposition $|\psi\rangle$. These two vectors span a two-dimensional plane in the vector space. These vectors are not actually perpendicular because $|x\rangle$ occurs in the superposition with amplitude $\frac{1}{N}$ as well, but, in fact, we can introduce an additional state $|\psi'\rangle$ that is in the span of these two vectors, which is perpendicular to $|x\rangle$ simply by removing $|x\rangle$ on $|\psi\rangle$ and rescaling.

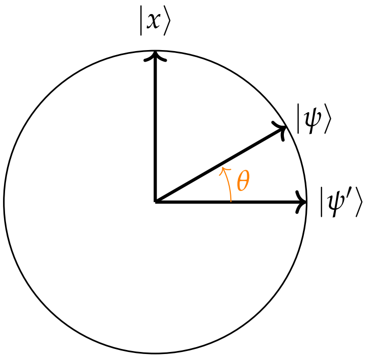


Figure 2.5.1: Representation on the unit circle.

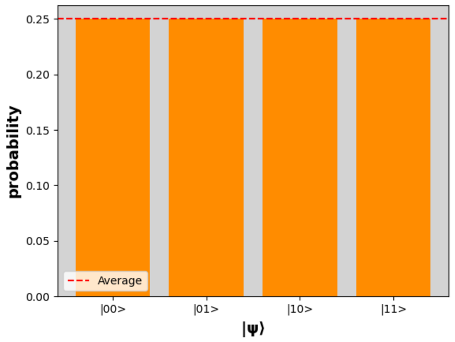


Figure 2.5.2: Uniform superposition and U_f for the state $|10\rangle$.

Now, if we apply $\mathcal{G}$, notice that first we invert the phase of $|x\rangle$ and then we reflect over $|\psi\rangle$. The latter reflection works as an inversion about the average amplitude.

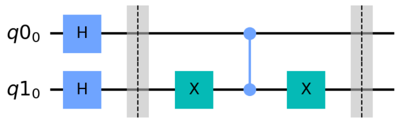


Figure 2.5.3: Example of a circuit for the Uniform superposition and the oracle that marks the state $|10\rangle$ on 2-qubit state system.

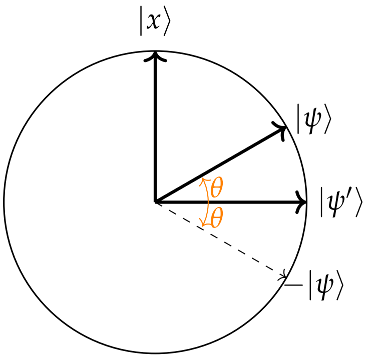


Figure 2.5.4: Action of U_f.

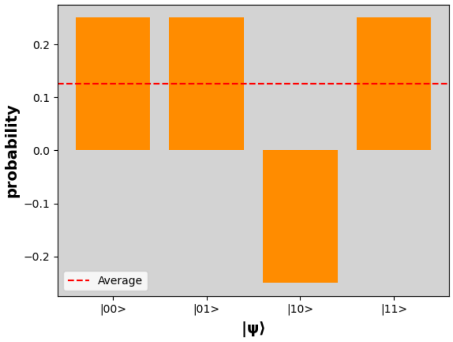


Figure 2.5.5: state $|10\rangle$ reflected.

Since the average amplitude has been lowered by the first reflection U_f, $\mathcal{R}_\psi$ boosts the negative amplitude of $|x\rangle$ to roughly three times its original value, while it decreases the other amplitudes. The algorithm needs $\mathcal{O}(\sqrt{N})$ applications of the Grover Iterate $\mathcal{G}$ which can be explained by the fact that the initial state can be expressed as:

$$|\psi\rangle = sin(\theta)|x\rangle + cos(\theta)|\psi'\rangle \tag{2.5.4}$$

where $\theta = arcsin\langle\psi|x\rangle = arcsin\frac{1}{\sqrt{N}}$ and, as derived in section 2.4, we need $\lfloor\frac{\pi}{4\theta}\rfloor\mathcal{G}$ iterations:

$$\lfloor\frac{\pi}{4\theta}\rfloor = \frac{\pi}{4arcsin\frac{1}{\sqrt{N}}} \approx \frac{\pi}{4}\sqrt{N} = \mathcal{O}(\sqrt{N}) \tag{2.5.5}$$

For the current example of N=4 we just need 1 iteration of $\mathcal{G}$ [52]

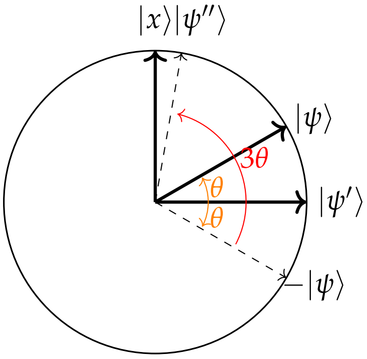


Figure 2.5.6: Action of $\mathcal{R}_{\psi}$.

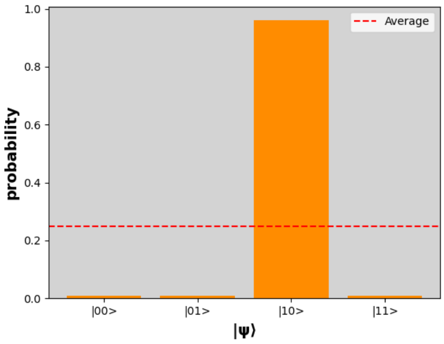


Figure 2.5.7: state $|10\rangle$ amplified.

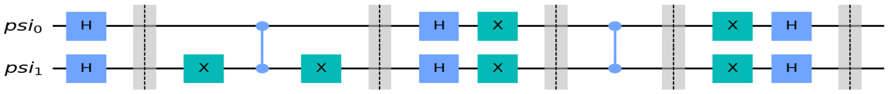


Figure 2.5.8: Final Grover's Algorithm circuit for 2-qubit with target state $|10\rangle$.

What about if instead of a single solution we have **multiple solutions**? Suppose that we know that our search problem has exactly n solutions in a search space with N elements. We can easily adapt Grover's Algorithm to deal with these cases, the only difference being the phase oracle U_f that needs to be able to mark the n solutions. We can derive a bound on the run time complexity of the algorithm as well. In this case, the probability of measuring a marked state is not $\frac{1}{N}$ as in the previous case, but instead $\frac{n}{N}$, so we can derive the number of times we need to apply $\mathcal{G}$ in order to increase the probability:

$$sin^2(\theta) = \frac{n}{N} \Leftrightarrow \theta = arcsin(\sqrt{\frac{n}{N}}) \tag{2.5.6}$$

thus, the number of iterations is given by:

$$\lfloor \frac{\pi}{4\theta} \rfloor = \frac{\pi}{4}\sqrt{\frac{N}{n}} = \mathcal{O}(\sqrt{\frac{N}{n}}) \tag{2.5.7}$$

As one may have noticed, this comes with a little caveat, that is, the algorithm will not work for the case where we have $n = \frac{N}{2}$.

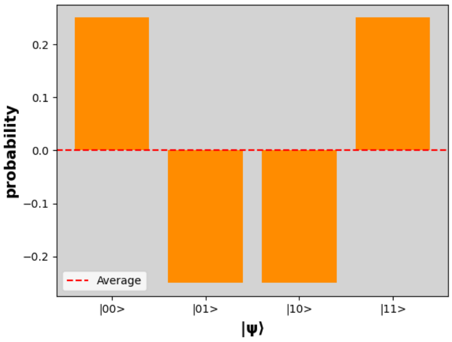


Figure 2.5.9: $N = 4$ and $n = \frac{N}{2}$ marked states

This is because we started with an uniform superposition state and marked half of the states, so as one may see in Figure 2.5.9, the average becomes null, thus the reflection about the average will not work, and all of the states will remain with the same amplitude. However there is a trick. In these cases if we extend the original Hilbert space, i.e. add more states in the overall superposition state, we will end up doing more iterations but still get a quadratic speedup. As a matter of fact, the algorithm will not work when $n > \frac{N}{2}$, and that's because in these cases the average will become negative, thus amplitude amplification will work in an opposite way, amplifying the non-marked states instead. Moreover, in this case we will read a marked state with probability $p = \frac{n}{N} > \frac{1}{2}$, thus amplitude amplification is not necessary. A more interesting case comes when $n = \frac{N}{4}$, the generalization of the case we have been following as an example. In this case $sin^2(\theta) = \frac{t}{N} = 1/4$ and $\theta = \frac{\pi}{6}$, so, a solution is found with certainty after a single iteration. Furthermore, the quantum algorithm becomes exponentially better than any possible classical algorithm if we compare worst-case performances, taking the worst possible coin flips in the case of a probabilistic algorithm [19].

A much more interesting case comes when we don't even know the number of solutions. Clearly iterate $\frac{\pi}{4}\sqrt{N}$ times won't do the job, given that this may lead to a vanishingly small probability of reading a marked state. We could use the approximate quantum counting algorithm [4] in a first step to knowing the number of solutions to the problem and then use Grover's Algorithm to find one of them. However, this technique entails $\mathcal{O}(\frac{1}{\epsilon}\sqrt{\frac{N}{n}})$ additional work. We can avoid this step if we perform an exponential search instead. The notion of exponential search comes way back in computer science [13]. The idea is to start with a sub-list of size 1. Compare the last element of the list with the target element, then try size 2, then 4 and so on until we find the solution. In the quantum setting the idea of exponential search already exists [19] and it is relatively similar to the classical counterpart. The idea is to start the Grover search with one iteration and then exponentially increase the number of iterations until either we reach a solution or the number of iterations corresponds to the maximum which is $\sqrt{N}$. The authors proved that this algorithm still converges in time $\mathcal{O}(\sqrt{\frac{N}{n}})$.

Algorithm 1: QSearch

```
$m \leftarrow 1$ , $max\_it \leftarrow \sqrt{N}$ ,$\lambda = \frac{6}{5}$ , $A[N]$;
while $m \leq max\_it$ do
    $it \leftarrow random(1, m)$;
    Prepare uniform superposition $|\psi\rangle = \frac{1}{\sqrt{N}} \sum_i |i\rangle$;
    Apply $it$ iterations of Grover's algorithm;
    Measure $|\psi\rangle$ , $i \leftarrow |i\rangle$;
    if $A[i] == x$ then
        Return $i$;
    else
        $m \leftarrow min(\lambda m, max\_it)$;
    end
end
Return Null;
```

At this point, the curious reader might think what happens if we relax the idea of having a uniform superposition state, to an arbitrary superposition state instead? What guarantees can Grover's algorithm give in this case?

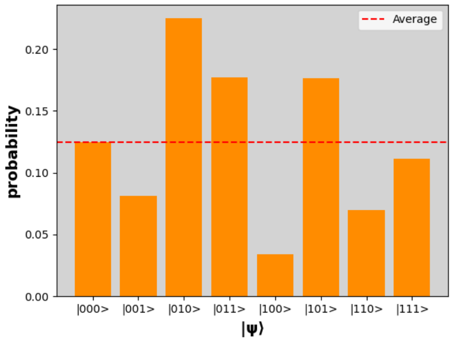


Figure 2.5.10: Non-uniform intial distribution.

When we relax the idea of an initial uniform superposition state, we get a more fundamental idea about Grover's Algorithm, first presented in [16], where the key observation is that the entire dynamics dictated by Grover's algorithm can be described entirely by the time-dependence of the *averages* of the amplitudes of marked/unmarked states. Defining the time-dependent averages of marked/unmarked states to be:

$$\textit{marked} : \bar{k}(t) = \frac{1}{n}\sum_{j=1}^{n} k_j(t) \tag{2.5.8}$$

$$\textit{unmarked} : \bar{l}(t) = \frac{1}{N-n}\sum_{j=n+1}^{N} l_j(t) \tag{2.5.9}$$

provided that one knows the initial average of marked/unmarked $[\bar{k}(0)/\bar{l}(0)]$ **states and the number of marked states**, the authors proved in [16] Theorem 3, that the number of iterations needed are:

$$-\frac{1}{2}\frac{\bar{k}(0)}{\bar{l}(0)} + \frac{\pi}{4}\sqrt{\frac{N}{n}} + \frac{\pi}{24}\sqrt{\frac{n}{N}} + \mathcal{O}(\frac{n}{N}) = \mathcal{O}(\sqrt{\frac{N}{n}}) \tag{2.5.10}$$

confirming the quadratic speed-up for arbitrary distributions. The advantage of an initial amplitude distribution with a relatively high average of the marked states is manifested in the constant offset $-\frac{1}{2}\frac{\bar{k}(0)}{\bar{l}(0)}$, which may reduce the number of iterations substantially. In the case of **unknown** averages and variance of the initial amplitude distribution but different runs of the algorithm use initial amplitudes drawn from the same distribution, the same authors showed in [17] that one needs twice as many iterations in order to have half of the probabilty of success compared to the previous case, thus leading to a slowdown with at most a factor of 4.

2.6 QUANTUM MAXIMUM FINDING

A common approach in quantum computing for optimization is precisely the use of Grover's search algorithm and amplitude amplification. This comes to no surprise due to the fact that these algorithms promise quadratic speedup compared to their classical counterparts [57]. In this section we're going through [29] quantum minimum finding algorithm which uses Grover's search as a subroutine for optimization. Consider an unsorted array $\mathcal{A}$ with N distinct elements. The algorithm finds the index i such that $A[i]$ is the minimum in time $\mathcal{O}(\sqrt{N})$. Although the algorithm was initially devised for this task, it can actually be generalized not only to find the minimum of an arbitrary function but also in the context of quantum unsupervised learning, for findind clusters of data [7], speeding-up the search of closest data points.

The algorithm starts by constructing the uniform superposition state $\frac{1}{\sqrt{N}}\sum_x |x\rangle|y\rangle$ where y an index chosen at random $y \in [0, N-1]$. Then a *phase oracle*, as in Grover's Algorithm, is used to mark the states, but this time we want to mark all the states $|x\rangle$

where $A[x] < A[y]$. This can be made using magnitude comparator circuits. The goal is, when measuring the final state, to read a new index that is smaller than y. We dont't know how many solutions we have, i.e. we don't know how many states $|x\rangle$ are smaller than y, so we need to run the quantum exponential search algorithm in (algorithm 1)instead of original Grover's algorithm. When measuring the final state if we read a state smaller than y we replace the current value of y by the measured value. The routine is repeated until the oracle runs empty, at which point $|x\rangle$ is the desired minimum.

Algorithm 2: QMF - Quantum Minimum Finding

```
$A[N]$ , $y \leftarrow random(0, N-1)$, $it \leftarrow 0$;
while $it \leq \sqrt{N}$ do
    Prepare uniform superposition $|\psi\rangle = \frac{1}{\sqrt{N}} \sum_x |x\rangle |y\rangle$;
    Apply the magnitude comparator Oracle $M$;
    Apply the QSearch algorithm 1;
    Measure $|x\rangle$ , $x' \leftarrow |x\rangle$;
    if $A[x'] < A[y]$ then
        $y \leftarrow x'$;
    end
end
Return y;
```

This algorithm is trivially generalized to a quantum algorithm that instead of finding the minimum, finds the maximum. This is done simply by inverting the magnitude comparator oracle M, marking states $|x\rangle$ such that $A[x] > A[y]$.

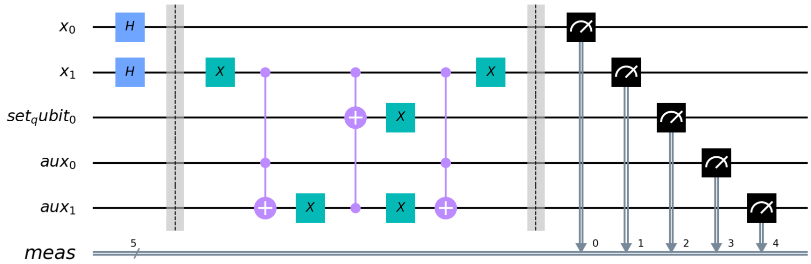


Figure 2.6.1: Example of a magnitude comparator oracle that marks all states greater than 1 in an uniform superposition state with 2 qubits setting the set_qubit equal to 1 when that is the case

An important remark to have is the fact that this is again a probailistic algorithm and the sucess probabilty of the algorithm is in fact $\frac{1}{2}$. So running the algorithm many times will improve the probability of sucess. In [5], the authors improve the upper bound on the computational complexity of the algorithm to $6.8k\mathcal{O}(\sqrt{N})$ with k being the number of repetitions. With $k = 2$ we have increased the sucess probability up to 75%

2.7 QUANTUM TREE SEARCH

In artificial intelligence there's been for a long time the interest in developing agents that can efficiently represent knowledge, reasoning and master problem solving. Intelligent agents are supposed to maximize their performance measure. Achieving this is sometimes simplified if the agent can adopt a **goal** and pursue it. [55] Pursuing that goal can be pictured as finding a fixed sequence of actions that lead to that goal or in the general case, where the future actions may depend on future percepts. The latter will be discussed in chapter 3. One way of pursuing that goal is *tree-search*. Tree search algorithms play an important role in many applications, ranging from game playing systems [23] to robotic control systems [44].

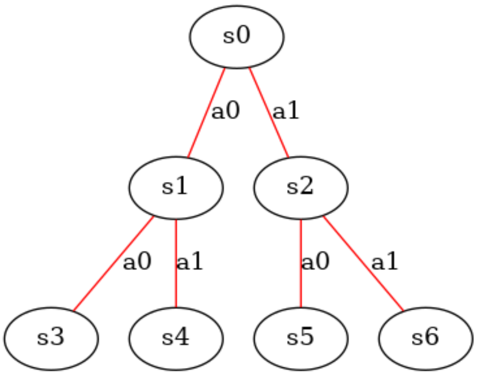


Figure 2.7.1: Binary Tree with depth $d = 2$

There are two different strategies in these types of algorithms, **Uninformed tree-search**, when we can only distinguish between goal states and non-goal states, and thus we need to expand every possible node in the tree until we reach the goal, and **Informed tree search**, when we have information about the search problem and can employ some heuristic function to choose the most promising tree node to expand. For more information relative to classic tree-search algorithms we refer the reader to [55] chapter 3.

Algorithm	Strategy	Run-Time
Depth-First [65]	Uninformed	$\mathcal{O}(b^m)$
Breadth-First [48]	Uninformed	$\mathcal{O}(b^{d+1})$
Iterative-deepening [42]	Uninformed	$\mathcal{O}(b^d)$
Greedy [51]	Informed	$\mathcal{O}(b^m)$
A* [36]	Informed	$\mathcal{O}(b^d)$
Best-First [43]	Informed	$\mathcal{O}(b^d)$

Table 1: Tree search algorithms (b - branching factor ; d - depth; m - maximum depth).

We're interested in using Grover's Algorithm and its variations (section 2.5), to develop tree search algorithms in the quantum setting that hopefully can show quadratic speedups as well. As one might think there are a few things to consider, like the depth of the tree, the branching factor associated (whether it is constant or non-constant), the number of goal states and whether there's only one correct sequence of actions or multiple ones. A trivial case of a quantum tree search algorithm can be pictured for the case of a fixed depth d and a **constant** branching factor b, with a single sequence of correct actions that lead to the goal. This is exactly Grover's Algorithm. We just need to prepare the initial superposition state $|\psi\rangle$ of the possible actions at each level of the tree. Then, an Oracle O that marks the correct sequence of actions and Grover's algorithm takes care of the rest.

$$|a\rangle = \frac{1}{\sqrt{b}} \sum_i |a_i\rangle \tag{2.7.1}$$

$$|\psi\rangle = |a\rangle^{\otimes d} \tag{2.7.2}$$

$$\tag{2.7.3}$$

$$O|\psi\rangle = \begin{cases} -|a_1 a_2 a_3 ... a_d\rangle & if \quad |a_1 a_2 a_3 ... a_d\rangle \mapsto goalstate \\ |a_1 a_2 a_3 ... a_d\rangle & otherwise \end{cases}$$

This algorithm was first suggested by [66] where the authors showed that the complexity of the algorithm is $\mathcal{O}(\sqrt{b^d})$. However, if we consider the problem of non-constant branching factors, the story changes. Suppose that we have a tree with action space $A = \{a0, a1, a2, a3, a4, a5\}$ and an average branching factor $b_{avg} = \frac{2+1+3+2+1+3+1}{7} \approx 2$ Figure 2.7.2:

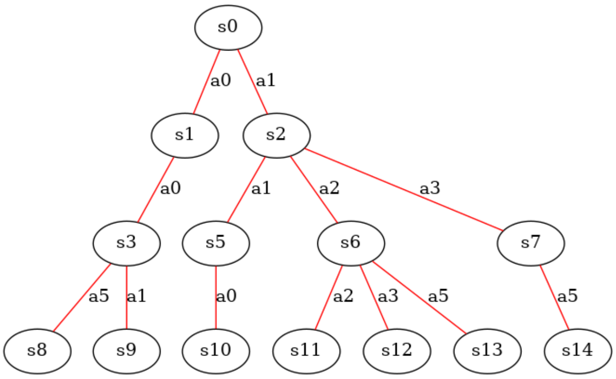


Figure 2.7.2: Non-constant branching factor tree with depth $d = 3$

Although in these cases one can classically show that the effective branching factor converges to the average branching factor, in the quantum setting one must use the maximum branching factor to prepare the superposition states (at least in this formulation of the Hilbert spaces) due to the fact that we don't know how to differentiate what actions are possible at each level of the tree. In these cases, Grover's algorithm may fail to produce speedups compared with classical tree-search because the quantum algorithm may have a larger state space to search through. In [66] the authors showed that as the average branching factor b_{avg} grows closer to the maximum branching factor b_{max}, Grover's algorithm will grow optimally compared to classical algorithms, and for $b_{avg} > 2^{\frac{log_2|A|}{2}}$ the quantum algorithm will yield a speedup compared to the classical algorithm. As the b_{avg} grows distant to the maximum branching factor however, the quantum algorithm looses its quadratic speed-up and for $b_{avg} < 2^{\frac{log_2|A|}{2}}$, the classical search evaluates less nodes than the quantum search algorithm, as depicted in Figure 2.7.3. The same authors in another paper [67] provided a quantum version of the iterative deepening algorithm and showed that the algorithm runs in time $\mathcal{O}(\sqrt{b^d})$. In section 5.2, we propose a new model that generalizes the tree search algorithm for the case of non-constant branching factors that can also be used as a new version of the iterative deepening algorithm of [67].

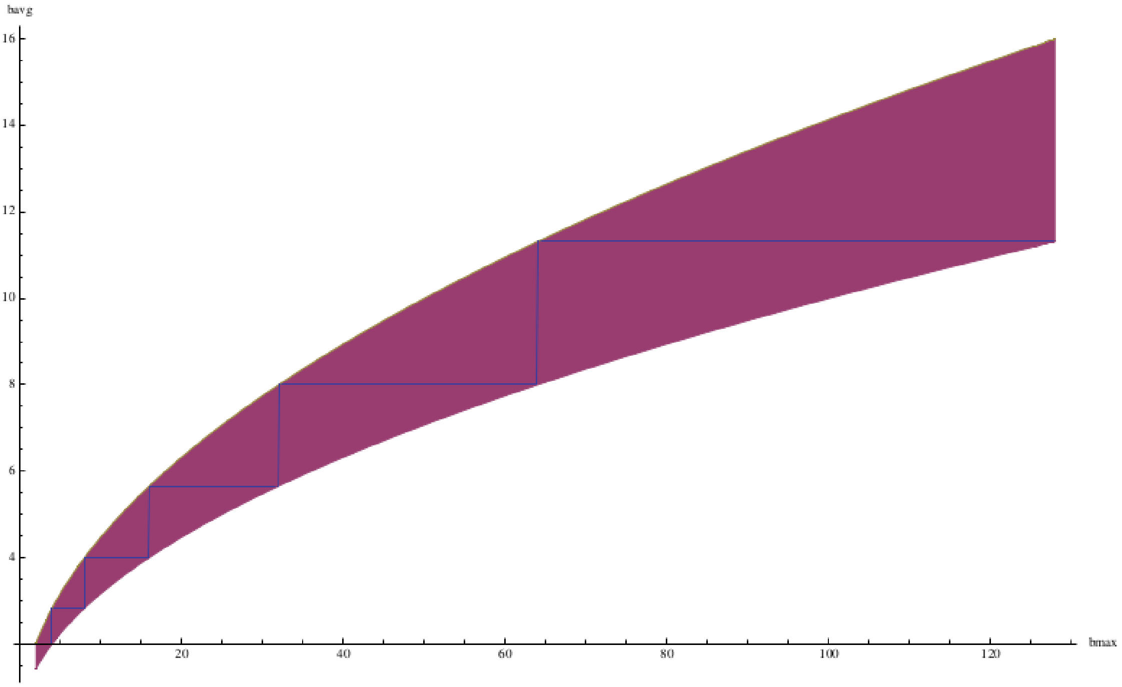


Figure 2.7.3: Growth separation between b_{avg} and b_{max}, being the rose shaded area, the area where the quantum search algorithm still performs faster than the classical search. Image from [66]

3

REINFORCEMENT LEARNING

3.1 INTRODUCTION

RL is different from supervised learning, where learning is done from a training set of labeled examples provided by a knowledgeable external supervisor. Each example is a description of a situation together with a specification, i.e. the label of the correct action the system should take when encountering that situation, which is often to identify a category to which the situation belongs. The objective of this kind of learning is to extrapolate, or generalize, the system responses so that it acts correctly in situations not present in the training set. This is an important kind of learning, but alone it is not adequate for learning from interaction [63]. An agent must be able to learn from its own experience, taking an action upon the world, and interpret the ”reward” that the environment gives back.

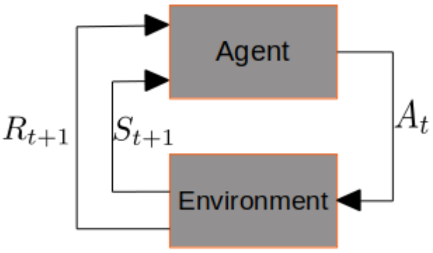


Figure 3.1.1: Agent-Environment Paradigm of RL: The agent performs action A over the environment at time t, and the environment produces a new state S_{t+1} for the agent and a respective reward R_{t+1}.

RL is also different from unsupervised learning, which typically resorts to finding structure hidden in collections of unlabeled data. At first sight, one may be tempted to think about RL as unsupervised learning, however, it is radically different in the sense that it seeks to maximize a reward signal returned by the environment instead of trying to find an hidden structure. When trying to maximize the reward signal, a challenge arises in RL and not in other kinds of learning, the so-called *exploration and exploitation dilemma*. This dilemma occurs in our everyday lives, suppose that one’s favorite restaurant is right around the corner.

Going to that restaurant we are pretty sure that we are going to leave the restaurant with a smile on our faces. Now, do we try the new restaurant that recently opened?

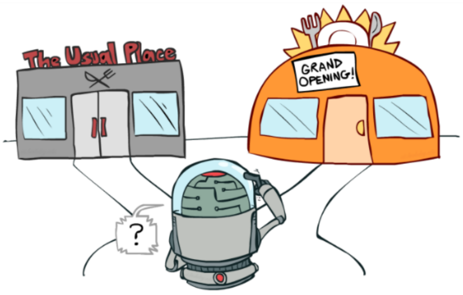


Figure 3.1.2: Exploration-Exploitation dilemma - Image from UC Berkeley AI course.

If we try the new place, we may enjoy it or we may get disappointed. How do we proceed? Always vouching for new places, very likely will leave us with unpleasant meals from time to time. So we need to gather enough information to make the best overall decision while keeping the risk under control. The best long-term strategy may involve short-term sacrifices, like once in a while try a new place. As a short introduction to RL we will consider the *Bandit Problems* which demonstrate this dilemma:

The original bandit problem has the form the *k-armed bandit problem*, named as an analogy to k slot machines, like the ones we found in casinos. The objective is to select arms at every time step and by repeated action selections try to maximize the total payoff concentrating our actions in the best arms. Another case is the *clinical trial* example. In this case we have k different medical treatments with unknown efficacy and by selecting a treatment we try to save as most patients as we can.

Figure 3.1.3: k=4 slot machines with unknown reward distributions

Consider the original problem, if we have a k=4 multi-armed bandit an unknown payoff distribution depicted in Figure 3.1.3, what should be the strategy to maximize payoff? One can try an arm successively, and by the law of large numbers, eventually will figure the true distribution of the arms payoff. However, this is quite wasteful and does not guarantee the best long term payoff. A better alternative is the balancing between exploration and exploitation, for example, the *ϵ-greedy* algorithm selects the best action known at a given time, i.e. the action with the highest accumulated payoff, but play a random arm occasionally.

We estimate the value of a given arm according to the past experience by averaging the payoffs, p, observed so far:

$$Q(arm) = \frac{1}{N_T(arm)} \sum_{t=0}^{T} p_t$$

where $N_T(arm)$ is how many times the *arm* has been selected so far. The *ϵ-greedy* algorithm then chooses a random arm with probability ϵ and selects the best arm so far with probability $1 - \epsilon$. By starting the algorithm with a high value for ϵ, we are exploring most of the time, which makes sense because we don't know anything about the machines. Over time decreasing the value of ϵ assures that we begin to take the best arms more often, exploiting the information we gained over time. The next section makes a brief review of Decision Theory in order to give a formal definition of RL.

3.2 DECISION THEORY

Decision theory is the study of an agent's choices, among possible actions based on the desirability of their immediate outcomes. [55]. The desirability of some outcome is quantified by an ***utility function U***. The existence of a real-valued measure of utility emerges from a set of assumptions about preferences.

- $A \succ B$ if we prefer A over B .
- $A \sim B$ if we are indifferent between A and B .
- $A \succeq B$ if we prefer A over B or are indifferent.

From this utility function, it is possible to define what it means to make rational decisions, but first, we need to impose some constraints on preferences in order to understand rational decision making from a computational perspective, the *Von-Neumann and Morgenstern axioms* [35]:

- *Completeness* - Exactly one of the following hold: $A \succeq B, B \succeq A, or A \sim B$
- *Transivity* - If $A \succeq B$ and $B \succeq C$ then $A \succeq C$
- *Continuity* - If $A \succeq C \succeq B$ then there exists a probability p s.t $[A : p; B : 1 - p] \sim C$
- *Independence* - If $A \succ B$ then for any C and probability p , $[A : p; C : 1 - p] \succ [B : p; C : 1 - p]$

From this axioms on rational preferences, we can say that there exists a **utility function U** such that:

- $U(A) > U(B)$ iff $A \succ B$

- $U(A) = U(B)$ iff $A \sim B$

We are interested in rational decision making agents, perhaps with imperfect knowledge of the state of the world. In either case suppose that we have a probabilistic model $P(s'|s,a)$ that represents the probability of the state of the world becoming s' when taking action a in state s. Consider also a utility function $U(s)$ that represents the preferences over the space of states (outcomes). The imperfect knowledge of the agent may arise from the fact that the state transition probabilities incorrectly characterize the environment. The **expected utility** of taking an action a at state s is given by:

$$\mathbb{E}U(a|s) = \sum_{s'} P(s'|s,a)U(s')$$

At this point, the rational decision making agent chooses the action that maximizes the expected utility a^*, i.e, respects the **maximum expected utility principle** as discussed in [35]:

$$a^* = \text{argmax}_a \, \mathbb{E}U(a|s)$$

As an example, consider the graph in Figure 3.2.1 representing the probabilistic model of the environment, and Equation 3.2.1 representing the **utility function U** associated to each of the outcomes. The decision making agent starts in state $s0$ and has to decide between actions $a0$ and $a1$.

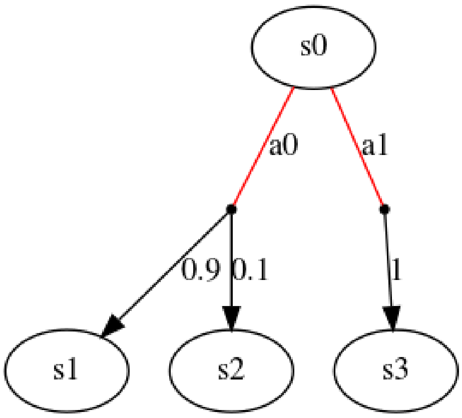


Figure 3.2.1: probabilistic model.

$$U(s) = \begin{cases} 10 & \text{if } s = s1 \\ 100 & \text{if } s = s2 \\ 20 & \text{if } s = s3 \end{cases} \tag{3.2.1}$$

At first glance, looking at the utility function in Equation 3.2.1, one might think that the appropriate action to choose is action $a0$ because it leads to state $s2$ with the highest utility $U(s2) = 100$. That would be an inappropriate selection method, because taking action $a0$ in state $s0$ leads to state $s2$ only 10% of the time. That's why the agent must choose the action that respects the maximum expected utility principle. Calculating the expected utility of each action:

$$\mathbb{E}U(a_0|s) = \sum_{s'} P(s'|s, a_0)U(s') = 0.9 \times 10 + 0.1 \times 100 = 19$$
$$\mathbb{E}U(a_1|s) = \sum_{s'} P(s'|s, a_1)U(s') = 20$$

we find that the agent must choose action $a1$, the action that maximizes the expected utility.

Until now we just considered *one-shot* decision making agents, agents that are concerned only with immediate outcomes. In a real world scenario, humans often make decisions with a delayed *reward* in mind, i.e. having an ultimate goal, one has to make a series of decisions until reaching it. *Markov Decision Processes (MDP's)* section 3.3 are a way of describing these types of **sequential decision making** problems, in which the utility depends on a sequence of decisions, thus they are an extension of the decision theory principles presented so far.

3.3 From Markov Chains to Markov Decision Processes

At the beginning of this chapter, we talked about the agent-environment paradigm of RL. Consider Figure 3.1.1 which depicts an *Agent* and the *Environment*. The agent makes sequential decisions and the environment returns to the agent a reward associated with the action taken, and a new observable state. The goal of the agent is to choose actions that maximize the expected reward in the long run.

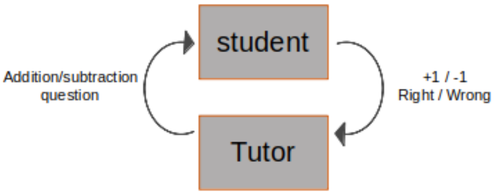


Figure 3.3.1: Adaptative Tutoring system that uses RL to teach a student how to do addition/subtraction mathematical problems.

An example of a real-world application of the agent-environment paradigm and consequently RL is the tutoring system in Figure 3.3.1. The goal of the tutoring agent is to teach a student to solve addition/subtraction problems. The agent outputs a problem and receives a reward of $+1$ if the student answers the question correctly and a negative reward of -1 otherwise. Therefore, maximizing the expected reward, the student will in the long run learn how to solve this type of mathematical operation. However, there is a caveat here, namely, how do the tutoring agent choose what questions to pose to the student? If all he receives is the binary reward based on the answer of the student, then, the goal may be achieved for the tutoring agent, but on the other hand, it can be misleading for the student. For example, most people find it easier to learn addition rather than subtraction. If the student answers correctly to addition questions and fails at subtraction, the tutoring will survey more and more addition questions. This way the agent does his duty, but the student gets deceived. This problem reflects the importance of defining the proper reward function.

This two-party system forms the basis of RL, however, the paradigm itself doesn't say much about the *agent* and the *environment*. There is a lot to say about the structure of the agent, but for us, the only concern is to define the agent mathematically as a function that maps every possible *percept*[1] sequence to an action. Here, we focus more on the aspect of the structure of the environment.

Figure 3.3.2: Chess board - Image from Google Deepmind

The environment can be *Fully Observable*, meaning that the agent directly observes the environmental state, which can be formally described by MDP's. An example of this is, for example, the game of chess Figure 3.3.2. In this board game, the agent has always full knowledge about the state of the game. On the other hand, the environment can be *Partially Observable*, meaning that the agent indirectly observes the environment, formally described by *Partially Observable Markov Decision Processes (POMDP's).* The well-known game of poker is an example of this, as the player only observes the cards in the table. The focus of this work is in the *Fully observable* case, so we will not go into the details of POMDP's.

In the fully observable case, the current state of the agent characterizes completely the process. Given that the state captures all relevant information from history, we say that the state is a sufficient statistic of the future, that is, the state is *Markov*, i.e. obeys the Markov property:

[1] A percept is the input that an intelligent agent perceives at a given moment. For example, suppose a robot with a camera. The recorded frames are the percepts of the agent. Then the agent plans to act according to this percept or a sequence of percepts.

Definition 3.3.1: Markov Property

A State S_t is *Markov* iff:

$$\mathbb{P}[S_{t+1}|S_t] = \mathbb{P}[S_{t+1}|S_1, ..., S_t] \tag{3.3.1}$$

"The future is independent of the past given the present"

From a set of states S following the Markov property, if we have a state transition probability between each Markov state s and a successor state s':

$$\mathcal{P}_{ss'} = \mathbb{P}[S_{t+1} = s'|S_t = s] \tag{3.3.2}$$

we can construct a state transition matrix $\mathcal{P}$, which is called a *Markov Chain* or a *Markov Process*.

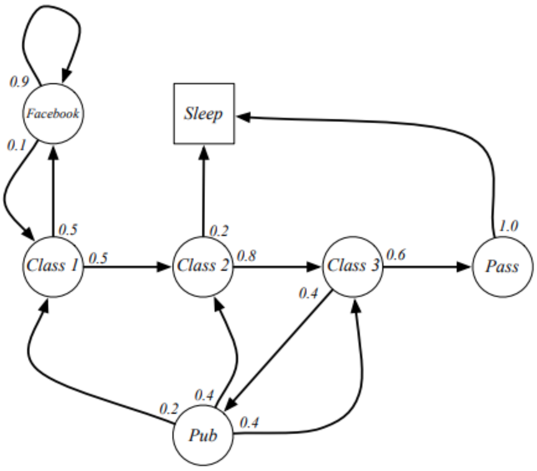


Figure 3.3.3: Student Markov Chain - Image from David Silver course on RL

Definition 3.3.2: Markov Chain(Markov Process)

A Markov Process is a tuple $\langle S, \mathcal{P} \rangle$ where:

- S is a finite set of states
- $\mathcal{P}$ is a state transition probability matrix

$$\mathcal{P} = \begin{pmatrix} \mathcal{P}_{11} & \dots & \mathcal{P}_{1n} \\ \vdots & & \\ \mathcal{P}_{n1} & \dots & \mathcal{P}_{nn} \end{pmatrix} \tag{3.3.3}$$

where $\sum_{j=1}^{n} \mathcal{P}_{ij} = 1 \quad, \forall i \in (1, n)$

In the context of RL, it makes sense to add some value judgement associated to the states of the chain, i.e. adding information on how good it is to be in a certain state. The value judgement is the so called *reward*. Moreover, we need a way of representing the accumulated reward throughout a (possibly infinite) trajectory in the chain, G_t, i.e. the *return* of some sequence of rewards. Of course, we could simply add the rewards together as in:

$$G_t = R_{t+1} + R_{t+2} + ... + R_T \tag{3.3.4}$$

However, this formulation can be problematic, for example for *continuing tasks*[2], because in that case we have $T = \infty$, and the reward could be infinite as well. To overcome this problem, we need to add a *discount factor*:

$$G_t = R_{t+1} + \gamma R_{t+2} + ... = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{3.3.5}$$

The discount factor shall be $\gamma \in [0, 1]$, otherwise it will not work as intended. At this point we can reach for the meaning of the discount extreme cases. When we have $\gamma \to 1$ we say that all reward accumulated is equally important for the agent and when we have $\gamma \to 0$ we converge towards a ”myopic” approach, meaning that we are only concerned with the immediate reward, reflecting interestingly enough, human or animal behavior, due to the fact that we tend to prefer immediate reward. Said that, we are ready to define a *Markov Reward Process (MRP)*:

Definition 3.3.3: Markov Reward Process (MRP)

A Markov Reward Process is a tuple $\langle S, \mathcal{P}, \mathcal{R}, \gamma \rangle$ where:

- S is a finite set of states
- $\mathcal{P}$ is a state transition probability matrix
- $\mathcal{R}$ is a reward function. $\mathcal{R}_s = \mathbb{E}[R_t | S_t = s]$
- γ is a discount factor, $\gamma \in [0, 1]$

[2] A continuing task differ from episodic task in a sense that here the interaction does not naturally break into episodes but continues without limit. One example of this is a cleaning robot. The machine tries to clean the place for as long as we want. Even when the robots battery is dead, he move into a ”recharge” state, but, once the battery is charged he continues its task.

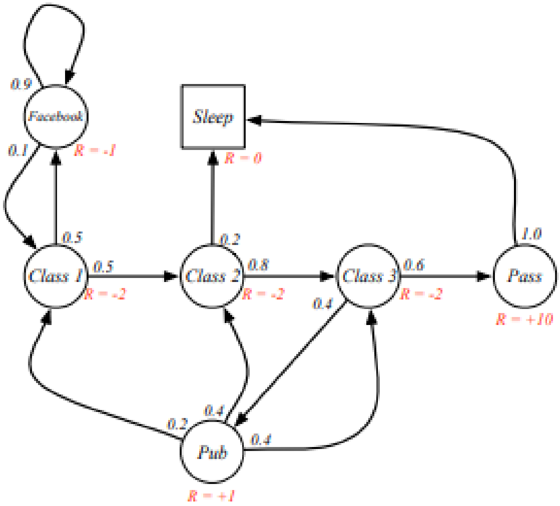


Figure 3.3.4: Student Markov Chain of Figure 3.3.3 converted into a MRP.

At this point, we can use the *Return* to define the long-term value of a given state. Starting from state s, the *value function v(s)* of an MRP represents the expected return.

$$v(s) = \mathbb{E}[G_t | S_t = s] \tag{3.3.6}$$

A fundamental property of value functions used throughout RL and dynamic programming (section 3.4) is that they satisfy recursive relationships [63]. For any state s, the following consistency condition holds between the value of s and the value of its possible successor states:

$$\begin{aligned} v(s) &= \mathbb{E}[G_t | S_t = s] \\ &= \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... | S_t = s] \\ &= \mathbb{E}[R_{t+1} + \gamma (R_{t+2} + \gamma R_{t+3} + ...) | S_t = s] \\ &= \mathbb{E}[R_{t+1} + \gamma G_{t+1} | S_t = s] \\ &= \mathbb{E}[R_{t+1} + \gamma v(s_{t+1}) | S_t = s] \end{aligned} \tag{3.3.7}$$

where Equation 3.3.7 is the so called *Bellman Equation* [12]. It expresses the relationship between the value of a state and the values of its successor states. We can think of it as looking ahead from a state to its possible successor states. Starting from state s_0, the environment could respond with one of several next states and a reward r, depending on the dynamics. The expectation $\mathbb{E}$ reflects the stochasticity of the environment, so the Bellman equation averages over all the possibilities, weighting each by its probability of occurring.

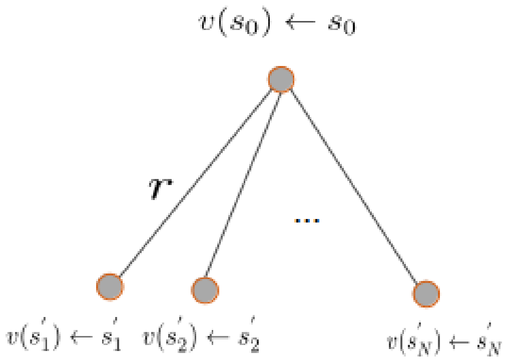

$v(s_0) \leftarrow s_0$
r
...
$v(s_1^{'}) \leftarrow s_1^{'}$
$v(s_2^{'}) \leftarrow s_2^{'}$
$v(s_N^{'}) \leftarrow s_N^{'}$

Figure 3.3.5: Backup Diagram. One step lookahead tree, from a starting state s_0

Definition 3.3.4: Bellman Equation

value function of a state s averaging over its possible sucessor states:

$$v(s) = R_s + \gamma \sum_{s' \in S} \mathcal{P}_{ss'} v(s') \tag{3.3.8}$$

All that we are able to do with the help of MRP's is to randomly sample transitions and average the rewards in order to define the value of states. However, there is no agency involved, in a sense that we want to be able to make decisions. So the natural step here, is to extend the above MRP with the set of actions that the agent can take. This extension leads us to the definition of a *Markov Decision Process (MDP)*. The difference is the state tansition probability and the reward obtained that is now dependent on the action taken by the agent.

Definition 3.3.5: Markov Decision Process (MDP)

A Markov Decision Process is a tuple $\langle S, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$:

- S is a finite set of states
- $\mathcal{P}$ is a state transition probability matrix

$$\mathcal{P}^a_{ss'} = \mathbb{P}[S_{t+1} = s' | S_t = s, A_t = a]$$

- $\mathcal{A}$ is a finite set of actions
- $\mathcal{R}$ is a reward function. $\mathcal{R}^a_s = \mathbb{E}[R_t | S_t = s, A_t = a]$
- γ is a discount factor, $\gamma \in [0, 1]$

Almost all RL algorithms are based on the estimation of the *value-function* that tell us how good it is to be in a certain state. The notion of the goodness of a given state depends on the reward accumulated which itself depends on the actions of the agent. Thereby, the value function depends on the agents choices, i.e. the *policy* π employed by the agent. We can formally define a policy as simply a mapping from states to actions $\pi : S \mapsto A$. Formally a *policy* is a distribution over actions given states $\pi(a|s)$:

Definition 3.3.6: Policy

Distribution over actions given states

$$\pi(a|s) = \mathbb{P}[A_t = a|S_t = s] \tag{3.3.9}$$

Given a policy π, we can define the value function of a given state s, v_π, as well as the value of another particularly interesting quantity, the value of taking an action in a given state and only then following π, $q_\pi(s,a)$, called the *action-value function*. This way we don't only estimate the value of a given state, but also define the value of state action pairs, also known as the *Bellman Expectation equations*.

Definition 3.3.7: Bellman Expectation of $v_\pi(s)$

Expected return starting from state s and following π

$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(R_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_\pi(s') \right) \tag{3.3.10}$$

Definition 3.3.8: Bellman Expectation of $q_\pi(s,a)$

Expected return starting from state s, taking action a, and then following policy π

$$q_\pi(s,a) = R_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a \sum_{a' \in \mathcal{A}} \pi(a'|s') q_\pi(s',a') \tag{3.3.11}$$

The above equations are the *expectation* version of the Bellman equations because we're starting from a state s and averaging over all the possible outcomes following policy π as we can see from Figure 3.3.6:

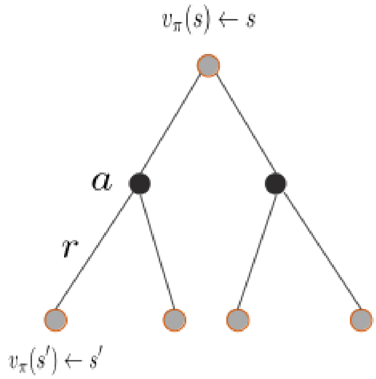


Figure 3.3.6: One step lookahead tree. Th value of state s is computed by averaging the value of all possible outcomes following policy π

Furthermore, RL methods specifies the change in the agents policy with experience, the goal being to reach the optimal policy π_*, i.e. the policy that leads to the highest expected accumulated reward, thereby, the optimal *value function / action-value function*

Definition 3.3.9: Optimality

Optimal value function $v_*(s)$ / action-value function $q_*(s,a)$ is the maximum over all possible policies

$$v_*(s) = \max_\pi v_\pi(s) \tag{3.3.12}$$

$$q_*(s,a) = \max_\pi q_\pi(s,a) \tag{3.3.13}$$

From here, we can extract the optimal policy as $\pi_*(s) = \operatorname{argmax}_a q_*(s,a)$. In the next section we will discuss algorithms to solve for the optimal policy.

3.4 PLANNING BY DYNAMIC PROGRAMMING

Dynamic Programming (DP) is widely used in optimization. Wherever we see a recursive solution that has repeated calls for same inputs, we can optimize it, by breaking the problem into sub-problems and store the result of this smaller problems, so that we do not have to recompute them again later. In RL, the recursive structure is the Bellman equation (3.3.7) and we can use DP to solve the equation for the optimal policy, provided that we have **complete knowledge of the environment**, i.e. we have access to a perfect model of the environment. We usually assume that the environment is described by a finite MDP. Due to the fact that we have access to the MDP of the environment, this approach is commonly known as **Model-Based RL**. There are two main model-based algorithms used: *Policy Iteration* and *Value Iteration*. In the policy iteration algorithm we start with a random policy, associating to every state of the MDP a respective random action. Then we loop into two different stages of the algorithm, the *evaluation* where we evaluate this random policy creating a respective value function and the *improvement* where we update the policy using a greedy action selection based on the value function computed. We loop until the convergence of the policy. Again, this is only possible because we have a perfect description of the environment, both dynamics and reward function. Therefore we can use the Bellman equation for the evaluation procedure:

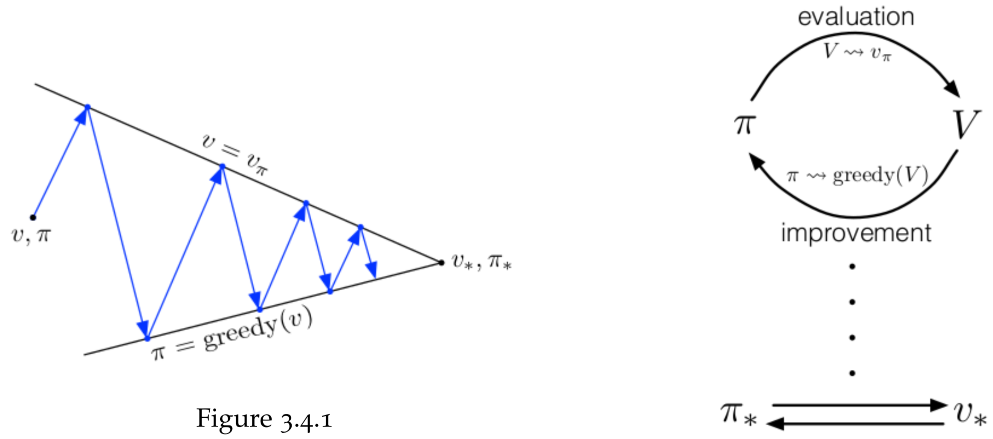


Figure 3.4.1

Figure 3.4.2

Figure 3.4.3: Figure 3.4.1 - convergence to the optimal policy. Figure 3.4.2 - Policy Evaluation + Policy Improvement loop. Images from [63]

$$V(s) \leftarrow \sum_{s',r} p(s', r|s, \pi(s))[r + \gamma V(s')] \tag{3.4.1}$$

and then the new computed value function in order to improve the current policy choosing greedily the action that lead to the highest expected reward as:

$$\pi(s) \leftarrow \underset{a}{\text{argmax}} \sum_{s',r} p(s', r|s, \pi(s))[r + \gamma V(s')] \tag{3.4.2}$$

Algorithm 3: Policy Iteration

```
stochastic policy $\pi$;
initialize $V(s) \in \mathbb{R}, \forall s \in S, \quad \epsilon$(precision of the estimation);

(1) Policy Evaluation:
  $\Delta \leftarrow 0$;
  while $\Delta \geq \epsilon$ do
    for each  $s \in S$ do
      $v \leftarrow V(s)$;
      $V(s) \leftarrow \sum_{s',r} p(s', r|s, \pi(s))[r + \gamma V(s')]$;
      $\Delta \leftarrow \max(\Delta, |v - V(s)|)$;
    end
end

(2) Policy Improvement:
  policy-stable $\leftarrow$ true;
  for each  $s \in S$ do
      old-action $\leftarrow \pi(s)$;
      $\pi(s) \leftarrow \text{argmax}_a \sum_{s',r} p(s', r|s, \pi(s))[r + \gamma V(s')]$;
    if old-action $\neq \pi(s)$ then
      policy-stable $\leftarrow$ false;
    end
end
  if policy-stable then
    Return $V \approx v_* \quad , \quad \pi \approx \pi_*$;
end
  else
    goto (2)
end
;
```

The algorithm spends to much computation time in the evaluation component. In fact, the algorithm loops through all states of the MDP until the value function stabilizes in some value $\Delta(s) \leq \epsilon \quad , \quad \forall s \in S$. The *Value Iteration* algorithm tackles this problem by instead of initializing a policy at random and then alternating between *evaluation* and *improvement*, ininitializing the value function arbitrarily and removing the *improvement* step, by maximizing over all the possible action at each step of the evaluation, i.e. choosing greedly the best value function for each state taking into account every possible action in that state:

$$V(s) \leftarrow \max_a \sum_{s',r} p(s',r|s,a)[r + \gamma V(s')] \tag{3.4.3}$$

This routine is repeated until the value function for all states converges. At that point, the algorithm converts the optimal value function into the optimal policy:

$$\pi(s) \leftarrow \operatorname*{argmax}_a \sum_{s',r} p(s',r|s,a)[r + \gamma V(s')] \tag{3.4.4}$$

Both algorithms theoretically converge to the optimal policy, however, the difference in the update, makes the Value Iteration algorithm converge faster to the optimal policy. The major drawback in these algorithms though is that they operate over the entire state space of the MDP, which for many real world applications become prohibitively expensive. For example, the backgammon game has 10^{20} states. A branch of research takes this challenge through *asynchronous* Dynamic Programming, in which updates are made not on the entire state space but in states in any order whatsoever.

3.5 MODEL-FREE PREDICTION AND CONTROL

In this section, as the name suggests, we will talk about a branch of RL which omits model of the environment, i.e. an MDP that represents the dynamics of the world, hence the name of Model-Free RL. This is in a sense the true RL because learning happens only by the interaction of the agent with its surrounding environment. By receiving a sequence of percepts and returns followed by a sequence of actions taken by the agent, one can expect the agent to learn via trial and error, sampling experience from either the real world or a simulated environment. Monte-Carlo methods sample and average return for each state-action pair much like the bandit problems discussed in section 3.1 sample and average rewards for each action. The difference is that one have now multiple states, each acting like in a bandit problem with the different bandits interrelated [63]. Moreover, the sampling must be made efficiently: we do not want just random sampling, but we want to be smart in the way we do sampling. In the beginning, the agent knows nothing about the environment. Therefore we attribute maximum criterion to exploration, which is crucial to gain experience. However, over time, the agent will record information about the dynamics of the environment and so, continuously performing randomly will not generate learning. Thus we need ways of balancing between the exploration that the agent does to exploit the information gained over time, the so-called **exploration/exploitation dillema**. The model-free problem is divided into two sub-problems as in model-based planning, the *prediction* problem, where we need to attribute value to a policy by test it into the environment, and the *control* problem, where based on this value we can improve our policies for the agent to converge to the optimal behavior over time.

The key difference in the *prediction* problem relative to model-based RL, is the way we estimate the value of our policy. In this case, we use Monte-Carlo methods to sample state-action-reward sequences derived from a policy π, called *episodes* and update the value of states not based on the Bellman equation as before, because we don't have a model of the dynamics, but rather use the episodes

$$Ep_{\pi} \equiv (S_0, A_0, R_0); (S_1, A_1, R_1); \ldots; (S_{h-1}, A_{h-1}, R_{h-1}); \tag{3.5.1}$$

to derive averages of the rewards received. Of course, in an episode, a single state may occur multiple times. Therefore, we can update the value of a state considering only a single appearance of the state (*first visit monte-carlo*), or, taking into account every appearance of said state (*every visit monte-carlo*). Theoretically, as the number of episodes grows to infinity, by the law of large numbers, both methods converge to the optimal value. An interesting fact about Monte-Carlo methods is that estimating the value of a single state of the MDP is independent of the number of states of the MDP. This is to say that we can start every episode from a state of interest and derive its actual value over time, interestingly enough for the case where we just need to derive the value of a subset of the state space. There is a clear subtlety compared to the model-based policy evaluation because here we are not under the same rules as before. With a model of the environment, state values alone are sufficient to determine a policy, using the one-step lookahead and choose whichever action leads to the best combination of reward and next state. In this case, however, this cannot be done In turn, it makes sense to use instead state-action values, given that we need to estimate the value of each action to properly suggest a policy. Therefore we are interested in determining the optimal action-value function, $q_*(s,a) \quad , \quad \forall s \in S, \forall a \in A$.

An issue is generating experience from a policy π. If this policy is deterministic, then following π will observe returns for only one action and this will enforce that some state action pairs will never be visited, which will make improvement over time impossible. Therefore we need to consider a stochastic policy to be able to estimate the value of all actions reachable from each state. It makes sense to start with a policy as an uniform distribution over the possible actions to take at every state, and sample an action from that distribution.

$$\pi(a|s) = \frac{1}{|A(s)|} \quad , \quad \forall s \in S \quad \mapsto \quad a_t \sim \pi(a|s)$$

Thus, before constructing the monte-carlo policy evaluation algorithm, one should take into account that the action-value function is characterized by:

$$\begin{aligned} Q^{\pi}(s,a) &= \mathbb{E}_{\pi}[G_t|S_t = s, A_t = a] \\ &= \mathbb{E}_{\pi}[R_1 + \gamma R_2 + \cdots + \gamma^{h-1} R_h|S_t = s, A_t = a] \\ &= \mathbb{E}_{\pi}[R_1 + \gamma G_{t+1}|S_t = s, A_t = a] \\ &= \mathbb{E}_{\pi}[R_1 + \gamma Q^{\pi}(s_{t+1}, a_{t+1})|S_t = s, A_t = a] \end{aligned} \tag{3.5.2}$$

Algorithm 4: Monte-Carlo Policy Evaluation

```
stochastic policy $\pi$;
initialize $N(s,a) = 0, Q^{\pi}(s,a) = 0, \forall s \in S, \forall a \in A(s), h$;
episode Ep=0 , EPISODES;
while $Ep < EPISODES$ do
    $Ep_{\pi} \equiv (S_0, A_0, R_0); (S_1, A_1, R_1); \ldots; (S_{h-1}, A_{h-1}, R_{h-1})$;
    $G = 0$;
    for $t = h-1, h-2, \ldots, 0$ in $Ep$ do
        $G = \gamma G + R_{t+1}$;
        $N(s_t, a_t) = N(s_t, a_t) + 1$;
        $Q^{\pi}(s_t, a_t) = \frac{Q^{\pi}(s_t,a_t)+G}{N(s_t,a_t)}$
    end
end
Return $Q^{\pi}$;
```

Note: The algorithm works for the *every visit* Monte-Carlo method. For the *first visit*, we need a separate counter to ensure that the update of the action value function is made only if the state was not yet visited in the current episode.

The Monte-Carlo *control* problem bears the same overall idea than model-based RL, meaning that the aim is to improve the policy based on the previous policy estimation. Therefore it makes sense to include an improvement step at the end of the policy evaluation as before, to converge to the optimal one. However, we cannot follow the same pattern as in model-based RL, given that we cannot maximize over the action-value function

$$\pi(s) = \operatorname*{argmax}_a Q^{\pi}(s,a)$$

because here we are doing sampling, and by maximizing we are converting the stochastic policy directly into a deterministic policy, and by doing that we might be depriving ourselves of further improvement because we might not have gathered enough information to properly decide. Therefore the strategy for policy improvement is to overtime converge to a deterministic policy, but with the gained experience. For that, we need to assure that we have a proper balance between exploration and exploitation. With *ϵ-greedy* policies (described in section 5.1), we assure that over time we select more often the current best action to take at a particular state, but still, sometimes selecting a random action to keep exploring.

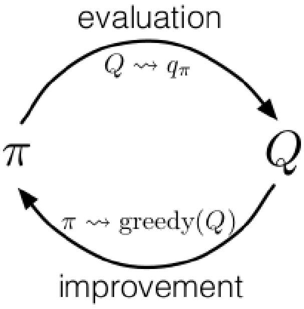

evaluation
$Q \rightsquigarrow q_\pi$
π
Q
$\pi \rightsquigarrow \text{greedy}(Q)$
improvement

Figure 3.5.1: Monte-Carlo policy evaluation + improvement loop taking into account action-value functions.

Algorithm 5: Monte-Carlo Policy control

```
stochastic policy $\pi$;
initialize $N(s,a) = 0, Q^\pi(s,a) = 0, \forall s \in S, \forall a \in A(s)$;
episode Ep=0 , EPISODES;
while $Ep < EPISODES$ do
    $Ep_\pi \equiv (S_0, A_0, R_0); (S_1, A_1, R_1); \ldots; (S_h, A_h, R_h)$;
    $G = 0$;
    for $t = h-1, h-2, \ldots, 0$ in $Ep$ do
        $G = \gamma G + R_{t+1}$;
        $N(s,a) = N(s,a) + 1$;
        $Q^\pi(s,a) = \frac{Q^\pi(s,a)+G}{N(s,a)}$;
        $a^* = \text{argmax}_a Q^\pi(s_t, a)$;
        $\forall a \in A \quad , \quad \pi(a|s_t) = \begin{cases} 1 - \epsilon + \frac{\epsilon}{|A(s_t)|} & if \quad a = a* \\ \frac{\epsilon}{|A(s_t)|} & if \quad a \neq a* \end{cases}$
    end
end
Return $Q^\pi$;
```

If we adjust ϵ over time, decaying ϵ with the increase of the number of episodes, in the limit we will converge to the optimal deterministic policy. This is called *GLIE (Greedy in the Limit with Infinite Exploration)*, which is a *on-policy* method, meaning that we are using the same policy to generate experience to further improve the policy. As a note, we are missing a lot of details about model-free RL, for example *off-policy* methods, which don't follow a policy for generating experience, and *Temporal difference*(TD) methods, which compared to Monte-Carlo, don't need that full episodes terminate to update the values of state-action pairs. These methods are a symbiosis between the dynamic programming methods and the Monte-Carlo methods to update "on the fly" the values associated to state-action pairs, a strategy called *bootstrapping*. This is a critical consideration for some applications that have considerably long episodes in which suspending all the learning until the episode ends substantially slows down the process. The disadvantage compared to Monte-Carlo methods is convergence. Actually is still an open question whether TD methods converge in all cases, however, in practice, they have been achieving better results than the Monte-Carlo methods. For the rest of the chapter we will stick to a Monte-Carlo approach. For more information about the TD learning methods, we refer the reader to [63] for a in-depth explanation of these methods.

3.6 SPARSE SAMPLING

The methods explained above both, *model-based* and *model-free* RL are called *tabular methods*, because they construct a policy, i.e, a table which has a direct mapping from every state of the MDP to an action. In model-based, the policy is constructed via *planning*, because we have a description of the environment, whereas model-free use sampling methods are used. Both methods have excellent results in practice, however with the increase of the size of the MDP, storing a policy becomes intractable. This is where the *sparse sampling* idea takes the problem from a different angle. These algorithms implement the policy itself, sampling from a model, at each state to compute an action at that particular state. The algorithm is given a single state of the MDP as input and then builds a tree with the state at the root. By constructing the lookahead tree, the algorithm can return a near-optimal action to take at that particular state. The next step is to perform the action that was returned and repeat the process. How can we assure that the returned action is optimal? How do we build such a tree that we can extract optimal actions from it? Clearly for large state space MDP's, building a tree that accurately approximates the transition model becomes again intractable. In the infinite horizon, γ-discounted setting, we can define a cutoff time for the horizon as $H_\epsilon = log(\frac{1}{\epsilon})\frac{1}{1-\gamma}$. If we have a deterministic MDP with an action space $\mathcal{A}$, we can build a lookahead tree by performing every action $a \in \mathcal{A}$ on the initial state generating $|A|$ states and then repeat the process for every newly generated state until reaching the cutoff time. In the end, we can use dynamic programming to reach the optimal decision. In the most general case, for stochastic environments, this will not suffice because taking only one sample of a given action will not give a good approximation of the dynamics of the MDP. Said that for these cases we need to build the tree taking as root the initial state given, sampling each possible action m times and repeating the process for the $m|A|$ new states generated as in Figure 3.6.1

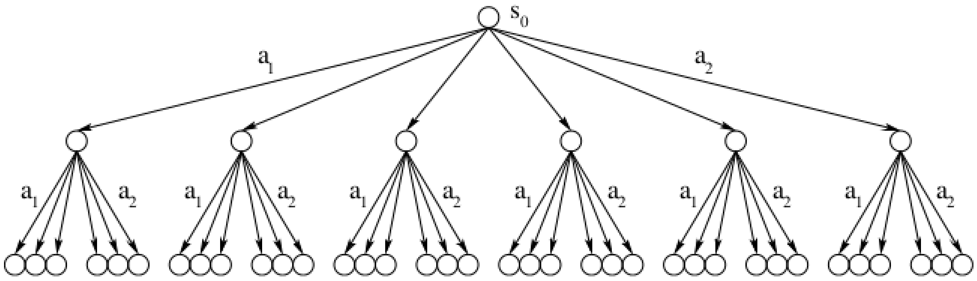


Figure 3.6.1: Lookahed tree for a binary action MDP, with $m = 3$ and horizon $H = 2$. Figure from [41]

Therefore, for a tree depth correspondent to the horizon, the algorithm needs to call the model

$$\mathcal{O}((m|A|)^{H_\epsilon}) \tag{3.6.1}$$

The authors originally in [41] proved that to have an ϵ-approximation of the optimal action to take at the current state, we need to take m samples of each action at every state generated such that the complexity of the algorithm becomes

$$\left(\frac{|A|H_\epsilon}{\epsilon}\right)^{\mathcal{O}(H_\epsilon log\frac{H_\epsilon}{\epsilon})} \tag{3.6.2}$$

This algorithm performs exponentially with the horizon, however, interestingly, it is independent of the number of states of the MDP, which is an important result, making the algorithm suitable for dealing with large state space MDP's. Sparse sampling in itself is not enough and we shall see why. However, it was a great idea that paved the way for more sophisticated algorithms and shaped the way of thinking within the Reinforcement Learning community. Sparse Sampling does not focus search on highly valued nodes, i.e. nodes with high values, and returns. This means that the algorithm does not prioritize over the nodes that one should expand: instead it spends the same amount of computation time in every node, independently of its current status. This is where Monte-Carlo Tree Search (MCTS) excels with a new method for extending the ”game tree” based on promising nodes. Given that this is not a subject of study of this dissertation, we will not go over the algorithm, however, we want to point out its existence and refer the reader to [63] for a review of the algorithm.

4

QUANTUM ENHANCEMENTS FOR MACHINE LEARNING

Before digging straight into the design of quantum algorithms for RL, it is convenient to know what are the possible advantages that quantum computing can offer to machine learning. Often, research in quantum computing refers only to advantages related to *computational complexity*. This is reminiscent to the asymptotic computational complexity of algorithms wherein the quantum setting proved relevant in several examples like the quadratic speed-up in Grover's Algorithm or the exponential speed-up in the celebrated Shor's factoring algorithm [60]. However, in the context of machine learning, there's more to take into account than simply the computational complexity. For example, we're interested in the *sample complexity* of algorithms, coming from *statistical learning theory* [56] In section 4.1, we provide the reader with some background theory on what constitutes learning and review known results on a recent theoretical branch of quantum machine learning responsible for developing a quantum theory of learning. It is important to refer that statistical learning theory was originally developed for the case of Supervised Learning. However, it was already adapted to the Reinforcement Learning case [40] and we will gradually picture the original theory in the latter case. To the best of our knowledge, there's still not a connection between quantum sample complexity and the field of quantum reinforcement learning since the latter is still in its infancy. The algorithm developed in section 5.4 maybe the first attempt to connect them.

When developing learning algorithms we're also interested in another metric called *model complexity* focused in studying if quantum computing can offer new types of models to machine learning. Section 4.2 exhibits some quantum models that can possibly enhance machine learning with an emphasis in enhancing Reinforcement Learning.

In section 4.3 we will review known results on Quantum Reinforcement Learning and outline the connections with the work reported in the following chapter.

4.1 SAMPLE COMPLEXITY

Sample complexity deals with the problem of how large a training set is required to be in order to learn a good approximation of the target **concept** [1]. The most used model for dealing with this problem comes from statistical learning theory, namely *Probably approximately correct* learning (PAC-learning) [70]. PAC-learning gives a precise complexity-theoretic definition of what it means for a concept to be efficiently learnable. We will briefly sketch the concept with an example in binary classification of supervised learning and then translate it into a different learning paradigm, namely Reinforcement Learning. Suppose we're given images of people being sad and images of people being happy.

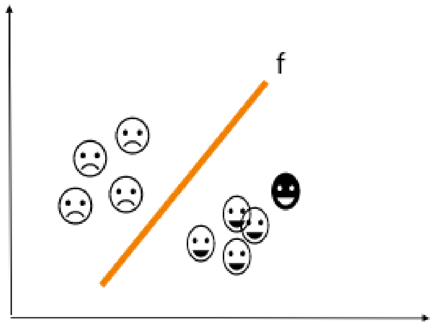


Figure 4.1.1: characteristic function f that correctly separates the two subsets of images, where white faces represent the training set, and the blackface represents a new face that f must be able to classify.

This can be formulated as N examples of tuples $(x, f(x))$ with x corresponding to the image and $f(x) \in \{sad, happy\} \mapsto \{0,1\}$ being the associated label. The objective is given N examples to output an hypothesis $h : \{sad, happy\} \mapsto \{0,1\}$ as the "best guess" for the actual concept f. In PAC-learning the quality of the hypothesis given the N data points is measured by the total error ϵ compared with f with probability δ:

$$\mathbb{P}[\sum_x |h(x) - f(x)| \leq \epsilon] \geq 1 - \delta$$

There can be many functions that respect such requirements of PAC learnability so we will define the sample complexity of learning a concept as the minimal integer that satisfies the requirements. It turns out that to produce an ϵ-optimal hypothesis with probability δ we need N examples such that:

$$N \geq \mathcal{O}\left(\frac{1}{\epsilon} log(\frac{|\mathcal{H}|}{\delta})\right)$$

[1] A concept is the rule f that divides the input space into subsets of the two-class labels 0 and 1, in other words, it is the law that we want to recover within a model.

where $\mathcal{H}$ is a finite hypothesis class, i.e. $|\mathcal{H}|$ is the number of possible different classifiers, also called the **VC-dimension**.[56]. At this point, we may ask where the examples come from in the first place? In this setting we're assuming that either the N examples are given, which can be viewed as the examples first labeled by humans and then fed into the learning machine, or more broadly, sampled from an arbitrary distribution. In either case we assume that the learner has access to the labeling function which is perfect, i.e. given an input point it produces an output y deterministically. By doing this we're not allowing the model to have noise associated or being a stochastic model which in practice occurs frequently. Also, we're restricted to the requirement that the learner can always output a hypothesis h that is present in the hypothesis class $\mathcal{H}$, where in practice this may not be the case. If we relax these requirements we have a generalization of PAC-learning called **agnostic learning**, where we know that

$$N \geq \theta(\frac{|\mathcal{H}| + log(1/\delta)}{\epsilon^2})$$

examples are necessary and sufficient to output a hypothesis h whose error is at most ϵ worse than the error of the best concept. At this point, one may ask: What does this even have to do with Reinforcement Learning? Is it really important? Well, in chapter 3 we discussed two different types of doing Reinforcement Learning In one setting the agent has **complete knowledge of its environment**, dealing with the problem of finding a good policy in a fully known environment. This is perhaps the best-studied problem in reinforcement learning. For large scale applications, where we know the dynamics of the world, it is hard to store a table of transition probabilities and rewards in terms of memory consumption, so it uses some form of compact model (like in Bayesian networks) to reduce the size of the problem. However, it is still computationally expensive to perform operations like computing expectations in these compact models On the other hand, it is efficient to obtain Monte-Carlo samples and perform estimations. So by using Monte-Carlo, sample complexity refers to how much experience must be simulated by the model to find a good policy.

In an alternative setting, the agent has **complete ignorance about the environment**. This is the purest reinforcement learning setting, an agent is placed in an environment, and simply by interacting with it, one tries to reach the optimal policy. Now, sample complexity in this case may have different meanings, depending on how the agent achieves the optimal policy. If the agent wants to represent the model of the environment and then use planning to determine the policy, sample complexity means how much experience do we need to construct an ϵ-approximation of the environment.

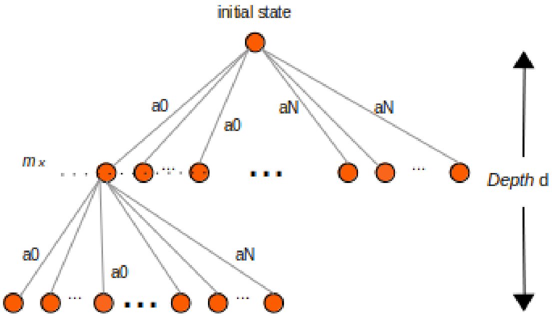


Figure 4.1.2: Sparse-sampling algorithm of section 3.6, each action is sampled m times

If the agent tries a direct policy search approach as in sparse-sampling Figure 4.1.2, sample complexity means how much experience we need to gather to have an ϵ-approximation of the "goodness" of some action. Thus we need a way of estimating how many times do we need to *roll* the same action until a reasonable estimate of its expected outcome is reached. Now that we highlighted the importance of sample complexity in the context of reinforcement learning, it is time to establish a connection between *agnostic* PAC-learning and reinforcement learning, in order to sketch how one can analyze the sample complexity of these learning algorithms. Agnostic learning is characterized by **(1)** no assumption is made about the input distribution, which can be either deterministic or stochastic. **(2)** No assumption is made about the true concept being contained in the hypothesis set $\mathcal{H}$. Let's see:
In RL we typically search for a policy π within a policy class Π. In the model-based variant we have complete knowledge of the environment, thus we know the dynamics and the reward function, thus it does not make sense to consider sample complexity in this case, because the problem is purely computational . On the other hand, in the model-free variant, we don't make any assumption about the distribution, so agnostic learning fits. We can define the sample complexity of a model-free RL algorithm $\mathcal{A}$ that takes m samples by comparing the value function implemented by the algorithm with the optimal value function:

$$V^{\mathcal{A}}(s) - V^{*}(s) \leq \epsilon, \quad \forall s \in S$$

Thus we can the say that $\mathcal{A}$ is **PAC-MDP** if for any ϵ and δ, the sample complexity is polynomial in the relevant quantities $(S, A, 1/\epsilon, 1/\delta, H)$ with probability δ as in [62].

PAC-learning asks how many examples one needs from the original concept in the worst case to train a model so that the probability of error ϵ is smaller than δ. In the quantum framework, we can define quantum example oracles that work as a parallelized version of classical sample generators [57] like the qsample oracle of Figure 4.1.3.

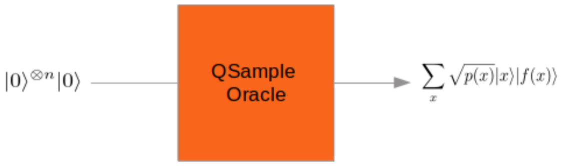

$|0\rangle^{\otimes n}|0\rangle$
QSample Oracle
$\sum_x \sqrt{p(x)}|x\rangle|f(x)\rangle$

Figure 4.1.3: Qsample oracle that provides examples sampling from an unknown distribution $p(x)$

In the quantum setting the hope is that computing with amplitudes will make possible to extract more information compared with the classical counterpart. The first contribution was made in [22] by showing that disjunctive normal form expressions are efficiently learnable by a qsample oracle that samples by interfering the amplitudes of the qsample with a Quantum Fourier Transform. However, this was done considering only the uniform distribution, whereas learning under any distribution is needed. Unfortunately, it turns out that we cannot expect any exponential speed-up from quantum sample complexity, first in [58] the authors suggested a polynomial separation between quantum and classical sample complexity, showing that if any class C of Boolean functions is learnable with Q evaluations of the qsample oracle, then it is learnable with $\mathcal{O}(nQ)$ classical evaluations. More recently in [9] it was proved that classical and quantum sample complexity are in fact equal up to polynomial factors, the results holding for both PAC-learning and agnostic PAC-learning. Despite this discouraging result, nothing is lost, for example in the case where noise is associated with the samples. Also in [22] the authors show that this condition renders the classical problem unlearnable, whereas in the quantum setting, with more examples it is still possible to learn.

4.2 MODEL COMPLEXITY

Model complexity captures how complicated functions the learner can learn: the more complicated the model, the higher chance of "overfitting" and consequently, the weaker the guarantees on the generalization outside the training set. Intuitively, we want to use quantum mechanics to develop new models that can be better at capturing patterns and correlations in data and discover which problems are suitable for quantum computing. There are numerous paths along which quantum computing may offer something different to the game of machine learning. For example, Neural Networks are one of the most revolutionaries ideas in machine learning that show extremely good results in practice, with applications ranging from supervised learning, as in recommendation systems, to help designing an agent capable of beating the world champion in the game of Go [47]. In the quantum setting there is still a lot of work to do in developing quantum neural networks. However, the simplest neural network to quantize is the Boltzmann Machine [37], consisting of bits with tunable interactions.

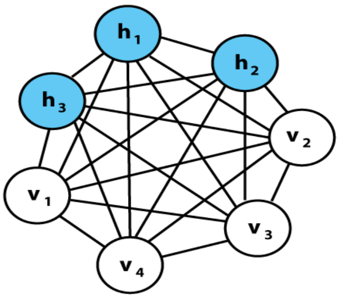


Figure 4.2.1: Example of a Boltzmann Machine with 3 hidden units and 4 visible units with each edge of the graph representing a dependence

The Boltzmann machine is thus trained by adjusting those interactions so that the thermal statistics of the bits, described by a Boltzmann distribution reproduces the statistics of data. Now, quantum methods can make the system thermalize quadratically faster than its classical counterpart and accelerate training by improving ways of sampling [15]. In [24] , the authors used a quantum annealer as implemented by D-Wave systems, to design a reinforcement learning algorithm in which the set of visible nodes representing the states and actions of an optimal policy are the first and last layers of a deep network, showing that it outperforms classical reinforcement learning with Boltzmann machines.

Another interesting quantum model that can be a fruitful way of connecting machine learning with quantum computing, is quantum walks, more precisely the quantization of Markov chains, as for example *Szegedy quantum walk* [64]. The algorithm shows a quadratic speed-up in the hitting-time of an ergodic Markov chain with symmetric transition matrix. However, it was not clear if this quadratic speed-up could be generalized to arbitrary Markov Chains until recently [8], proposed an algorithm that finds any set of marked vertices in an arbitrary graph quadratically faster than the classical counterpart. In [21], the authors used a quantum walk in the *projective simulation* method of reinforcement learning, which consists of an agent that has a graph of memory clips of its interaction with the environment, and selects an action based on a random walk through such memory clips. Replacing the random walk by a quantum walk the authors showed a quadratic speed up in the traversing time of the graph.

4.3 QUANTUM REINFORCEMENT LEARNING

In chapter 1 we mentioned that when we talk about AI, we talk about a more general learning framework, of agents which interact with their environments. In our view, we consider the agent-environment paradigm one of the most important aspects of any intelligent agent. Given that for intelligent agents, like humans, learning is done by reinforcement and environments are often complex and susceptible to changes dependent on the agent's actions, we believe that RL forms the basis of possible improvements in the design of artificially intelligent agents. Therefore, we are interested in all possible quantum enhancements in the context of RL. In this section, we want to review some known quantum enhancements ranging from new action selection methods to the generalization of the agent-environment paradigm, where both agent and environment are treated according to quantum theory.

One the most used action selection methods in RL that balances *exploration* and *eploitation* is the *ϵ-greedy* algorithm, that selects a random action with probability ϵ and the action with the highest expected reward with probability $1 - \epsilon$:

$$\pi(a|s) = \begin{cases} argmax_{a \in \mathcal{A}} Q(s,a) & with \quad probability \quad 1-\epsilon \\ random(\mathcal{A}) & with \quad probability \quad \epsilon \end{cases} \tag{4.3.1}$$

However, this action selection mechanism has some problems, namely the fact that one has to set a value for ϵ and decide whether to decrease the value or not or when to decrease the value in order to exploit more often. Another problem is the fact that the algorithm doesn't prioritize actions. By selecting randomly an action, it may select the worst possible action or even the best known one so far. In [25] the first step into quantum RL was made, trying precisely to overcome this problem The authors designed a novel action selection method based on the collapse postulate of quantum mechanics and the well known Grover iteration. The idea is that initially, the current environment state puts the agent in a uniform superposition over the set of possible actions $\mathcal{A}$:

$$|\psi_A\rangle = \frac{1}{\sqrt{|\mathcal{A}|}} \sum_{i \in |\mathcal{A}|} |a_i\rangle \tag{4.3.2}$$

Then, if we measure $|\psi_A\rangle$, we will read an action with equal probability, which is a good technique for the exploration policy. Next, we execute the collapsed action into a **predefined classical environment** that returns a reward r and a new state s associated with the action. Provided that we have classical data structures to accumulate the rewards and the action taken in the state s, data structures that create the value function $V(s), \quad \forall s \in S$ with the experience, we can update the value of some state based on the reward and the new state received by the interaction with the classical environment, using the $TD(0)$ update rule [63]

$$V(s) \leftarrow V(s) + \alpha(r + \gamma V(s') - V(s)) \tag{4.3.3}$$

To update the value of some state every time that we collapse the quantum action register. Next, to produce learning, at each interaction, we reconstruct the uniform superposition, but now, instead of collapsing the state, we amplify the amplitude of the action previous taken in the state we're in, relatively to the its value function and the reward received, thus we apply L Grover Iterations such that:

$$L = int(k(r + V(s'))) \tag{4.3.4}$$

In fact, as discusssed in section 2.4, the amplitude of an action given L Grover iterations becomes:

$$sin(\theta) \xrightarrow{\mathcal{G}^L} sin((2L+1)\theta) \tag{4.3.5}$$

and depending on the value of L, the probability of reading an action, may become very small (if L increases). So we select L as:

$$L = min\{int(k(r + V(s'))), int(\frac{\pi}{4\theta} - \frac{1}{2})\} \tag{4.3.6}$$

Then, we simply repeat the process until $\Delta V(s) \leq \epsilon, \quad \forall s \in S$. There are quite a few problems with this approach though. For example, the authors suggested that this algorithm performs better than ϵ-greedy action selection, however, this depends heavily on an appropriate choice of the value k in L. Moreover, consider the case of an action set with 4 possible actions, common in all gridworld environments as depicted in Figure 4.3.1 with $\mathcal{A} = \{up, down, left, right\}$.

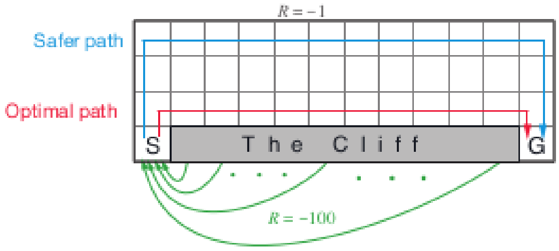


Figure 4.3.1: Gridworld environment example - Cliff walking [63].

In this cases, the superposition over the action set is the 2-qubit state

$$|\psi_A\rangle = \frac{1}{2}[|00\rangle + |01\rangle + |10\rangle + |11\rangle] \tag{4.3.7}$$

which is a particular superposition state as we need $\frac{\pi}{4\theta}$ iterations to amplify a given superposition term with certainty, for this case, one Grover iteration suffices. This is to say that, for these problems, L will be a binary variable, $L \in [0, 1]$ and due to the normalization in Equation 4.3.6, the agent will often take random actions because, the Grover iteration will not be able to distinguish one from another. Thus the algorithm does not guarantee convergence to the optimal policy as expected.

Furthermore, the agent-environment interaction is made via a classical environment. Thus the algorithm is rather an action selection algorithm than a truly quantum reinforcement learning algorithm, in the sense that no definition is made about the dynamics or the reward function in the quantum setting. Efforts have been made to generalize MDP's to quantum MDPs (qMDPs) [73] and POMDPs to quantum POMDPs (QOMDPs) [10]. Both frameworks use superoperators to express actions and observations. Superoperators are defined by a set of k Kraus matrices [2] K of dimension d, that acting on a density matrix ρ, generate one of the k possible states:

$$\rho'_i \leftarrow \frac{K_i \rho K_i^\dagger}{Tr(K_i \rho K_i^\dagger)} \tag{4.3.8}$$

with probability $Tr(K_i \rho K_i^\dagger)$ [3], if the i^{th} Kraus matrix was applied.

Definition 4.3.1: qMDP [73]

A qMDP is a 4-tuple $\langle H, \mathcal{A}, \mathcal{M}, \mathcal{Q} \rangle$ where:

- H is a Hilbert Space representing a set of possible states as density matrices of pure/mixed states.
- $\mathcal{A}$ is a finite set of actions where each $a \in \mathcal{A}$ is a superoperator responsible for evolving the system
- $\mathcal{M}$ is a finite set of measurements
- $\mathcal{Q} : \mathcal{A} \cup \mathcal{M} \rightarrow 2^{\mathcal{A} \cup \mathcal{M}}$ is the set of possible actions and measurements available after performing a certain action and a measurement.

In a qMDP, a series of measurements is introduced to infer the state and help select the next action, based on their outcomes. The algorithm that utilizes the obtained measurement information for decision making is called a scheduler. A scheduler selects the next action based on the outcomes of previously performed measurements and actions. A key difference between MDPs and qMDPs is that with MDPs, the scheduler can use the exact state of the system because the state of the classical world is known in the MDP. In a quantum environment, the true world state is not known and must be inferred from partial measurements of the system that give information about the system but do not collapse the superposition. When measurements are needed to infer the environment state, the environment is known to be partially observable Figure 3.3. Partial observability of the true underlying state of a quantum system in superposition may suggest POMDPs as a potentially better model of quantum systems than MDPs

[2] A set of matrices of dimension d is a set of Kraus matrices if $\sum_{i=1}^{k} K_i^\dagger K_i = \mathbb{I}_d$.

[3] The trace operator is the sum of the diagonal elements in a given quantum operator. This operators act linearly in a vector space, therefore the diagonal elements represent the probability of basis states being measured.

Definition 4.3.2: QOMDP [10]

A QOMDP is a tuple $\langle S, \Omega, \mathcal{A}, \mathcal{R}, \gamma, \rho_o \rangle$ where:

- S is Hilbert Space representing a set of possible states represented as density matrices of pure/mixed states.
- $\Omega = \{o_1, ..., o_{|\Omega|}\}$ is a set of possible observations.
- $\mathcal{A} = \{A^1, ..., A^{|\mathcal{A}|}\}$ is a set of superoperators. Each superoperator $\mathcal{A}^a$ represents the possible observations of taking action a. If o_i is observed then the next state is:

$$\rho_i' \leftarrow \frac{A_i^a \rho A_i^{a\dagger}}{Tr(A_i^a \rho A_i^{a\dagger})}$$

- $\mathcal{R} = \{R_1, ..., R_{|\mathcal{A}|}\}$ is a set of operators representing the reward associated to taking the action a at a given state ρ.
- $\gamma \in [0, 1)$ is a discount factor.
- $\rho_0 \in S$ is an initial state.

Quantum superoperators formalize an agent's ability to take an action in a quantum environment and receive a percept. Both formulations use superoperators with partial measurements in a feedback loop for evolving and gaining information about the system.

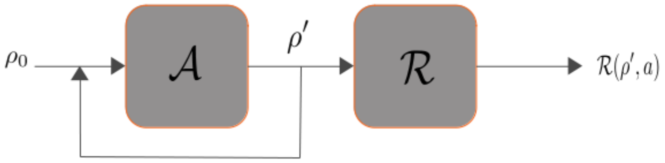


Figure 4.3.2: Feedback loop of a QOMDP.

In the QOMDP, acting and sensing in the environment are coupled via superoperators. At each time step, the agent chooses a superoperator to apply from its set of superoperators. Observation of the system may be done, often via indirect measurement methodologies, and the agent receives an observation according to the laws of quantum mechanics and a reward given by $\mathcal{R}$ based on the state value after the superoperator is applied. As with MDPs, a policy for a QOMDP is a function $\pi : S \rightarrow \mathcal{A}$ mapping states at time t to actions. The value of the policy over horizon h, starting from state ρ_0, can be computed by the following equation representing the Bellman equation in the quantum setting:

$$V^{\pi}(\rho_0) = \sum_{t=0}^{h} \mathbb{E}[\gamma^t \mathcal{R}(\rho_t, \pi(\rho_t))|\pi] \tag{4.3.9}$$

This formulation enables the existence of a clear complexity separation between the quantum and the classical case for qMDPs [73] and QOMDPs [10]. In both papers, the authors proved that the *Goal State Reachability Problem* is **Undecidable** in the quantum case, yet decidable in the classical one. On the other hand the complexity of deciding policy existence for finite horizons is the same for the quantum and the classical case.

Despite the interesting computability results in quantum MDP's, there are still missing algorithms that use this framework for the context of RL. In [27], a different schema for doing RL in the quantum setting was suggested, the generalization of the Agent-Environment paradigm enabled the authors to achieve a quadratic improvement for certain RL tasks and exponential improvement in terms of the success rate. The idea is based on the oracular construction of task environments.

*"An approach to quantum improvements in reinforcement learning – This brings us to our schema for improving RL agents. First, given a classical environment E, we define **fair unitary oracular equivalents** E^q. Here, fair is meant in the same sense as quantum oracles of boolean functions are fair analogs of classical boolean functions. E^q should not provide more information than E under classical access, which is guaranteed, e.g., when E^q is realizable from a reversible version of E."*

Based on the oraculization of the task environment the authors suggested a quantum oracle for deterministic environments with a single reward, that depends only on the actions that the agent has taken. An example of this is a maze with an initial state and goal state. The only state with reward $R = 1$ is the goal state and every other state has $R = 0$. So in these cases we are to learn a correct sequence of actions and the quantum oracle E^q is dramatically simplified as a phase-flip oracle for a sequence of M actions, denoting M interactions with the quantum environment.

$$|a_0, .., a_M\rangle \xrightarrow{E^q} (-1)^{\Lambda(a_1,...,a_M)}|a_0, .., a_M\rangle \tag{4.3.10}$$

where Λ is the reward function. This type of oracles points towards the use of Grover Search, which can return with certainty the correct sequence of actions. For deterministic environments with multiple rewards, counting oracles U_{count} can also be built for a sequence of actions and states $\bar{s}$

$$|a\rangle \otimes |\bar{s}\rangle \otimes |y\rangle \xrightarrow{U_{count}} |a\rangle \otimes |\bar{s}\rangle \otimes |\Sigma\lambda \oplus y\rangle \tag{4.3.11}$$

where the total reward appearing in the sequence $\bar{s}$, $\Sigma\lambda$ is appended to the counting register $|y\rangle$, with $\oplus$ representing the addition in the appropriate group. From here, using phase-kick back again, we reflect over the sequence of actions $|a\rangle$ that satisfies the property of the reward being above a certain chosen value. The authors also considered stochastic environments by constructing an oracle for the case of a certain sequence of actions having a probability of being rewarded:

$$|a\rangle \otimes |0\rangle \otimes |0\rangle \xrightarrow{S_E} |a\rangle \otimes \sum_s \sqrt{p(a)}|s\rangle \otimes |\Lambda(s,a)\rangle \tag{4.3.12}$$

$$|a\rangle \otimes |0\rangle \otimes |0\rangle \xrightarrow{S_E} |a\rangle \otimes (\sum_{s,\Lambda(s,a)=0} \sqrt{p(a)}|s\rangle \otimes |0\rangle + \sum_{s,\Lambda(s,a)=1} \sqrt{1-p(a)}|s\rangle \otimes |1\rangle) \tag{4.3.13}$$

From this, one can use amplitude amplification to amplify the terms spanned by reward register being in the state $|\Lambda(s,a)\rangle = |1\rangle$. This oracle instance of a stochastic environment is rather a narrow formulation, in the sense that it lacks the ability to capture stochastic state-transition dynamics and just expresses the probability of a sequence of actions being rewarded, missing the capacity to express more general reward functions in the quantum setting.

Within the framework of quantum accessible RL environments, the goal is to build general-purpose oracle instances of classical task environments. It turns out that obtaining generalization is a hard task and we're left to construct oracle instances of rather specific environments. However, in [28] the authors demonstrate that quantum agents can in fact achieve exponential improvements in learning efficiency, provided that one can build task environments that encode well-known oracle problems like in the Simon's Problem [61] and Recursive Fourier sampling.
More recently, [54] provided quantum algorithms to solve dynamic programming problems. The authors considered RL from the *linear programming* perspective and propose specific oracles to build a quantum version of a linear program. Separations and lower bounds for the learning of optimal policies were given, provided quantum oracular construction of transition functions in Markov decision processes. Up to polylogarithmic factors, the quantum algorithms proved to be quadratically faster in terms of the number of states $|S|$ and the number of actions $|A|$, when dealing with deterministic transition kernels. For stochastic transition kernels the quantum advantage is also quadratic in terms of the number of actions, however less than quadratic (from $|S|^2$ to $|S|^{\frac{3}{2}}$) in terms of the number of states relative to the classical counterpart.

5 QUANTUM-ENHANCED REINFORCEMENT LEARNING

Most introductory books or courses on Reinforcement Learning often start with a problem that manifests the importance of RL. Typically, the *multi armed bandit problems*, are the chosen ones and that's because these types of problems give a clear demonstration of the *Explore-Exploit dillema*. Here, we want to take the same approach, starting with a quantum algorithm for the best-arm identification of the *multi-armed bandit* problems discussed in section 3.1 and then gradually move into more general ideas for performing quantum RL.
To go from bandits to more general ideas we need to have the notion of MDP's in the quantum framework. As we have seen in section 4.3, there are two ways of constructing MDP's in the quantum setting, using superoperators [11, 73] or considering quantum oracular instances of the classical MDP [27]. Here we consider the latter approach to construct oracles for both deterministic and stochastic environments.
In section 5.2 we propose a route for the generalization of the quantum tree search algorithm developed by Tarrataca et al presented in section 2.7 to work with arbitrary non-constant branching factor search trees because this will play an important role in the upcoming algorithms. We will see that using the generalized tree search algorithm with some extensions, namely the notion of quantum MDP's we can develop a quantum algorithm for the deterministic MDP (section 5.3) that provides a quadratic speed-up for the action selection mechanism of a RL agent.
It is known that the critical issue for the application of MDP's to realistic problems is the scaling complexity of planning with the increase of the size of the MDP. Actually, traditional planning for stochastic environments with large state spaces may be inapplicable due to the linear dependence on the number of states. Although the quadratic speedup provided by [54] in planning, it is still hard to solve large state-space problems. Thus, instead of performing planning on the entire MDP, in section 5.4 we considered the *sparse sampling* approach section 3.6, to sample a look-ahead tree in order to compute near-optimal actions for any state of the MDP. Thus, we present a novel algorithm for stochastic MDP's that works as a quantum version of the sparse sampling algorithm [41], providing a speed-up compared with the original classical algorithm.
In the last chapter, we point out to the importance of sample complexity in quantum learning

algorithms and despite that the bounds obtained by [9], showing that in fact quantum and classical sample complexity are equal up to constant factors, it is still of great importance, for developing quantum learning algorithms, study their sample complexity. Moreover, metrics like computational and model complexity, can possibly enhance learning algorithms.
All previous results in quantum reinforcement learning (section 4.3), deal only with the problem of computational complexity. To the best of our knowledge, there's still not a connection between sample complexity and quantum reinforcement learning algorithms. In this chapter, we introduce novel quantum algorithms that will serve as a new route for enhancing reinforcement learning in general.
In section 5.5 we provide the reader with an analysis of the complexity of such algorithms. At last, in section 5.6 we conclude the chapter by delving into the details of the proposed quantum algorithms in order to achieve real speed-up.

5.1 QUANTUM BANDITS

As we have seen in section 3.1, several real-world applications can be formulated as a multi-armed bandit problem, like the *clinical trial* example. We want now to consider the original bandit problem of a k-armed bandit consisting of a gambler with k slot machines, trying to decide the best machine to play. Pulling any one of the k arms gives you a stochastic reward of either $R = +1$ for success, or $R = 0$ for failure. This is also called a *Bernoulli Bandit*. In the language of RL, the objective is to find the optimal policy, the optimal arm, the arm that maximizes the reward obtained in the long run. One will notice that pulling the optimal arm will not always beat any other arm on a given pull since the rewards are stochastic. However, it is in the long-term reward average that you will find the optimal arm to dominate.

In the quantum setting, we can treat the problem in a fundamentally different way. Due to the principle of superposition of quantum mechanics, we can interact with every k arms in parallel and record the information about the reward. There are two subtleties in the quantum algorithm, namely **(1)** the encoding mechanism of the k armed bandits stochastic reward distribution and **(2)** the strategy to reach the optimal arm. We will clarify each subtlety next. For simplicity, we will consider the case of a *Bernoulli Bandit* with $k = 2$ arms, however, this can be easily generalized to arbitrary k arms.

(1) Encoding strategy: We need a way of encoding the arm with which we will interact and a way of encoding the stochastic reward distribution of playing the respective arm.

The arm can easily be encoded into a basis state, i.e. given that we're working in the $k = 2$ case, $arm = \{0, 1\}$,we just need one qubit to represent the arm, which we will call $|a\rangle$.

$$|a\rangle = \begin{cases} |0\rangle & \Longleftarrow arm = 0 \\ |1\rangle & \Longleftarrow arm = 1 \end{cases}$$

For the stochastic reward distribution, we can use the *angle encoding* scheme and, given that in this case we're working with binary rewards, we just need to use a single y-rotation gate to induce the appropriate amplitude change relative to each arm in a single qubit representation of the reward, initialized in the ground state, $|r\rangle = |0\rangle$:

$$|\phi\rangle = R_y(\theta)|r\rangle = R_y(\theta)|0\rangle = cos(\theta)|0\rangle + sin(\theta)|1\rangle$$

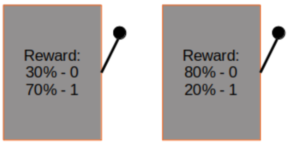


Figure 5.1.1: $k = 2$ classical arms with stochastic reward distribution

To explain the encoding of the reward distribution, suppose the following two arm reward distribution as represented in Figure 5.1.8. Of course the distribution will be unknown to the agent, we're just displaying the distribution for the illustration of the encoding. When playing the first arm , 70% of the time we will receive a reward and 30% of the time we will receive null reward. Now, we want to discover the angle θ on the y-rotation gate that respect this distribution. When measuring the state $|\phi\rangle$ above, by the Born rule (2.1.4) we measure the respective basis state with probability:

$$|\langle a|\phi\rangle|^2 = \begin{cases} cos^2(\theta) & if \quad a = 0 \\ sin^2(\theta) & if \quad a = 1 \end{cases}$$

So for having 70% probability of receiving a reward, we need θ to be

$$sin^2(\theta) = 0.7 \to \theta = \arcsin(\sqrt{0.7}) \approx 56.7 \approx 0.989\, rad$$

and doing the same for the second arm we derive that the rotation angle theta must be

$$sin^2(\theta) = 0.2 \to \theta = \arcsin(\sqrt{0.2}) \approx 26.5 \approx 0.463\, rad$$

In fact, the rotation gate must be applied to the reward qubit depending on the state of arm, i.e. we will have a controlled unitary $\mathcal{A}$ representing the effect of interacting in superposition with both arms, as shown in Figure 5.1.2. Therefore, $\mathcal{A}$ will be a quantum operator of the form:

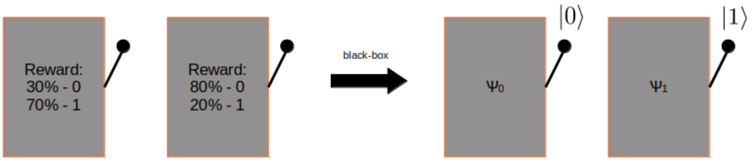


Figure 5.1.2: Conversion of an arbitrary k=2 armed bandit into quantum black boxes where action $|0\rangle$ acts on bandit ψ_0 and action $|1\rangle$ acts on bandit ψ_1

$$\mathcal{A} = |0\rangle\langle 0| \otimes R_y(0.989) + |1\rangle\langle 1| \otimes R_y(0.463) \tag{5.1.1}$$

$$\mathcal{A} = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \otimes \begin{pmatrix} cos(0.989) & -sin(0.989) \\ sin(0.989) & cos(0.989) \end{pmatrix} + \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix} \otimes \begin{pmatrix} cos(0.463) & -sin(0.463) \\ sin(0.463) & cos(0.463) \end{pmatrix}$$

$$\mathcal{A} = \begin{pmatrix} cos(0.989) & -sin(0.989) & 0 & 0 \\ sin(0.989) & cos(0.989) & 0 & 0 \\ 0 & 0 & cos(0.463) & -sin(0.463) \\ 0 & 0 & sin(0.463) & cos(0.463) \end{pmatrix}$$

The full quantum operator that prepares the superposition and interacts with both arms in parallel is of the form:

$$\mathcal{B} = \mathcal{A}(H \otimes \mathbb{1}) \tag{5.1.2}$$

where H creates the superposition of the arms and $\mathcal{A}$ acts linearly on both branches of the superposition. Assuming an initial state $|b\rangle$ to be the tensor product of both Hilbert spaces representing the arm and the reward, the action of $\mathcal{B}$ prepares the following non-uniform superposition state:

$$\mathcal{B}|b\rangle = \mathcal{A}(H \otimes \mathbb{1})(|a\rangle \otimes |r\rangle) \tag{5.1.3}$$

$$= \frac{1}{\sqrt{2}}\mathcal{A}[|00\rangle + |10\rangle] \tag{5.1.4}$$

$$= \frac{1}{\sqrt{2}}[cos(0.989)|00\rangle + sin(0.989)|01\rangle] + \frac{1}{\sqrt{2}}[cos(0.463)|10\rangle + sin(0.463)|11\rangle] \tag{5.1.5}$$

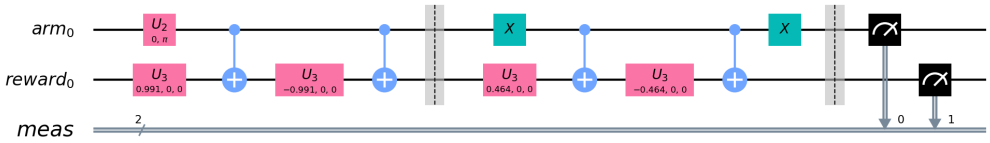


Figure 5.1.3: Quantum circuit that prepares the superposition equivalent to the k=2 armed bandit and the encoding of the stochastic reward distribution

If we sample the quantum circuit of Figure 5.1.3 a sufficient number of times, we will obtain a distribution that is ϵ-close to the stochastic reward distribution depicted in Figure 5.1.5

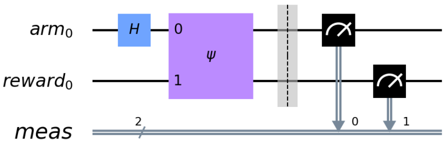


Figure 5.1.4: Abstraction of the k=2 armed bandit circuit of Figure 5.1.3

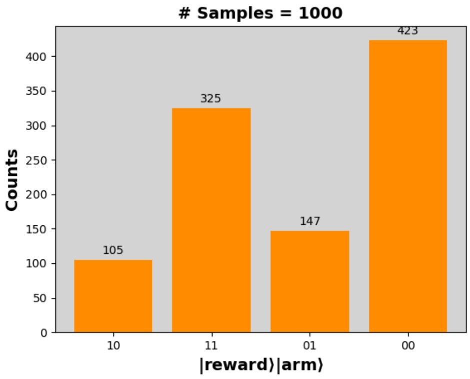


Figure 5.1.5: Distribution obtained by measuring the quantum bandit circuit with 1000 samples taken

However, this does not give us what we have been looking for, i.e the best arm. This is where the second subtlety mentioned above, i.e. a strategy to reach the optimal arm, has to be considered.

(2) Reaching the optimal arm: Instead of measuring the state, we could amplify the amplitude of the states with reward state $|r\rangle = |1\rangle$. How many Grover iterations do we need? In general, the stochastic reward distribution will be unknown to us, which can be viewed as the stochastic reward being given as black boxes that we can only interact with. Thus, we don't know how many iterations we need a priori, because we don't know the initial amplitude distribution. For that reason we need to run the exponential search algorithm 1 that will choose the number of iterations in an exponentially larger set until it finds a state with a larger amplitude such that $|r\rangle = |1\rangle$. By doing this we decrease the amplitude of the states that have reward 0 and amplify the ones that have reward 1.However, this does entail a collapse of the state, corresponding to a measure of a state that might not be the optimal arm, because of the probabilistic nature of the algorithm. Most of the time the initial distribution will be far from uniform and the exponential algorithm will amplify more than one superposition term. So, we will not collapse the state in the optimal arm with "certainty"[1] like in the original Grover's search problem. However, we can run this algorithm for a certain number of times m sampling the amplitude amplified state that will enable us to achieve a distribution that will tell us what is the state with a higher amplitude, thus, the optimal arm. Hence, with this strategy we hope to reduce the number of operations needed to identify the best arm, and if possible reduce the number of samples needed to achieve an optimal degree of confidence on the choice of a certain arm.

Algorithm 6: QBandits - Quantum Bandits

```
input sample size $m$ , it=0, $arm = \{0,1\}$;
$dist_a$ = Null;
while $it < m$ do
    create initial ground state $|b\rangle = |a\rangle \otimes |r\rangle = |00\rangle$;
    prepare uniform superposition in the arms $(H \otimes \mathbb{1})|b\rangle = \frac{1}{\sqrt{2}} \sum_{i=0}^{2} |a_i\rangle|r\rangle = |b_1\rangle$;
    Apply stochastic reward operators $\mathcal{A}$ to create parallel interaction $|\phi\rangle = \mathcal{A}|b_1\rangle$;
    $s \leftarrow$ QSearch algorithm 1 on $|\phi\rangle$;
    Append $s$ to $dist_a$;
end
Return $\text{argmax}_a\, dist_a$
```

Following the amplitude amplification technique as we did in section 2.4, the amplitude will be amplified depending on the average amplitude of marked states. Thus, we can deduce the probability of each state after one iteration of the Grover Iterate $\mathcal{G}$, which is optimal for the above example, by using the inversion about the average operator:

$$\forall a \in arm, \quad |\langle a|\mathcal{G}|\phi\rangle|^2 = \begin{cases} -|a\rangle + 2|\phi\rangle\langle\phi| & if \quad |r\rangle = |0\rangle \\ |a\rangle + 2|\phi\rangle\langle\phi| & if \quad |r\rangle = |1\rangle \end{cases}$$

where the average is given in function of the average of marked states:

$$\frac{\frac{0.3}{2} - \frac{0.7}{2} + \frac{0.8}{2} - \frac{0.2}{2}}{4} = 0.025$$

[1] Certainty in a sense that in the original Grover Search problem: we start with a uniform superposition state and we amplify the amplitude of a good state. The amplification procedure makes it possible to reach almost unity probability in measuring the marked state, which here is not the case because we cannot know for sure the number of Grover iterations that the exponential search performs and it could well be not the optimal number of iterations, given that if the algorithm returns a good state, the search is over, even if it has some probability of not being the optimal state

thus we will read each state with the following probability:

$$|\langle a|\mathcal{G}^j|\phi\rangle|^2 = \begin{cases} -\frac{0.3}{2} + 2 * 0.025 \approx 0.1 & if \quad |a\rangle = |0\rangle, |r\rangle = |0\rangle \\ \frac{0.7}{2} + 2 * 0.025 \approx 0.4 & if \quad |a\rangle = |0\rangle, |r\rangle = |1\rangle \\ -\frac{0.8}{2} + 2 * 0.025 \approx 0.35 & if \quad |a\rangle = |1\rangle, |r\rangle = |0\rangle \\ \frac{0.2}{2} + 2 * 0.025 \approx 0.15 & if \quad |a\rangle = |1\rangle, |r\rangle = |1\rangle \end{cases}$$

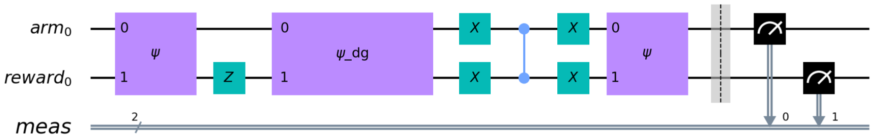


Figure 5.1.6: Optimal circuit obtained by QSearch leading to 2 Grover iterations

So, by applying one Grover iteration we will read the optimal arm with probability 0.4. Now, running algorithm 6, the exponential search algorithm finds the optimal arm with one Grover iteration as we can see in Figure 5.1.6. Taking $m = 10000$ samples we obtain the following distribution:

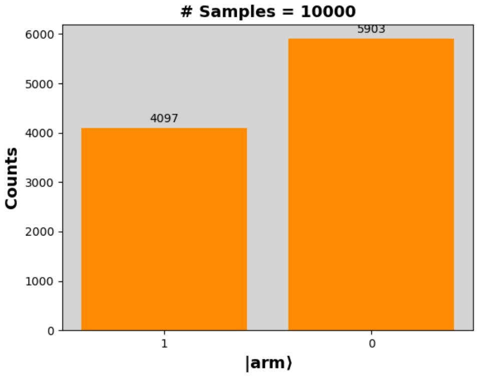


Figure 5.1.7: Distribution for one Grover iteration with m=10000 samples

showing that the algorithm returns the optimal arm. The number of samples taken was chosen at random, however in section 5.5 we will derive a bound for the number of samples that we need to take. There is one "if" to this algorithm, that is it depends on the initial unknown distribution and there is one case where the algorithm may fail to return the optimal arm, i.e. when both arms have exactly opposite stochastic reward distribution.

Reward:
30% - 0
70% - 1
Reward:
70% - 0
30% - 1

Figure 5.1.8: $k = 2$ classical arms with opposite stochastic reward distribution

Recall the original Grover's algorithm when we have n marked elements. When $n = \frac{N}{2}$ the amplification will not work since the inversion about the average fails as the average is null. Now, in the quantum bandits when we have exactly opposite stochastic reward distribution, we encounter the same problem. Moreover, we could have the case of an initial distribution for which marking half of the elements leads to a negative average. In that case the algorithm may fail as well.

5.2 Generalized Quantum Tree Search

In [66] the authors proposed a quantum algorithm for performing tree search. However for non-constant branching factors, the quantum algorithm still needs to use the maximum branching factor, in certain cases resulting in a slowdown compared to the classical counterpart. Our aim is to generalize the algorithm to deal with arbitrary non-constant branching factors. The idea is to rather than considering only the Hilbert space represented by the superposition of the actions at each level of the tree, we could have separate basis states representing the actions and the states we're in, i.e. a node in the tree. Suppose that we have a search tree with action space A. We know that for a depth d even with a non-constant branching factor b, there will be at most $|A|^d$ leaf nodes. So, we can encode a node in a quantum state with $log_2|A|^d$ qubits. We can represent an action initially in the ground state and a node that will be initially the root of the tree, with the following initial quantum state $|\psi_0\rangle$ represented by the tensor product of both basis states:

$$|s\rangle = |0\rangle^{\otimes log_2|A|^d}, \quad |a\rangle = |0\rangle^{\otimes log_2|A|} \tag{5.2.1}$$

$$|\psi_0\rangle = |s\rangle \otimes |a\rangle \tag{5.2.2}$$

Now, we can construct an unitary operator $\mathcal{A}$ that prepares the superposition of the actions controlled by the state of the node. By doing this, we prepare the superposition of admissable actions at a given node.

$$\mathcal{A} : |s\rangle \otimes |0\rangle^{\otimes log_2|A|} \mapsto |s\rangle \otimes \frac{1}{\sqrt{|A_s|}} \sum_{i \in |A_s|} |a_i\rangle \tag{5.2.3}$$

We can actually build this unitary operator,because we can in fact maintain zero amplitude on the superposition terms that don't represent an admissable action [71]. We also need to construct another unitary operator $\mathcal{T}$ that will be responsible for state transitions, i.e. the evolution operator to deal with the traversal within the tree.

$$\mathcal{T} : |s\rangle \otimes |a\rangle \mapsto |s'\rangle \otimes |a\rangle \tag{5.2.4}$$

We are going to interleave these two unitary operators for a predefined depth d, at each iteration extending the initial Hilbert space with new basis states for representing a new action and a new state for depth d. To reduce the complexity associated to the application of these operators, we may consider only neighbouring states. Applying the operators d times, can be interpreted as computing the tree in superposition for a depth d:

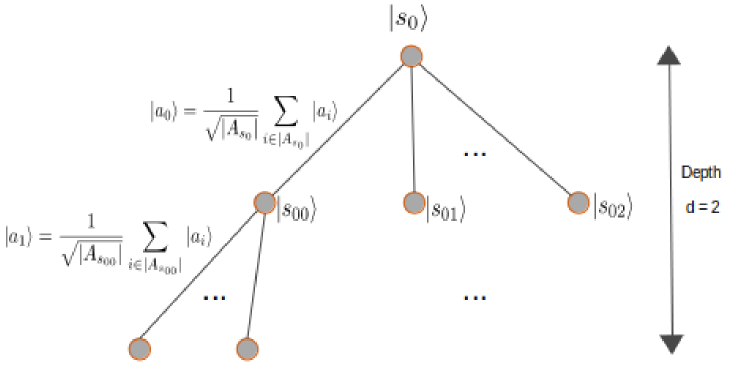

$|s_0\rangle$
$|a_0\rangle = \frac{1}{\sqrt{|A_{s_0}|}} \sum_{i\in|A_{s_0}|} |a_i\rangle$
...
$|s_{00}\rangle$
$|s_{01}\rangle$
$|s_{02}\rangle$
Depth
d = 2
$|a_1\rangle = \frac{1}{\sqrt{|A_{s_{00}}|}} \sum_{i\in|A_{s_{00}}|} |a_i\rangle$
...
...

Figure 5.2.1: Superposition of a tree with arbitrary branching factor, result of the quantum operator $(\mathcal{T}\mathcal{A})^d$

At this point we just need to construct a Grover Oracle O that reckons an arbitrary state as a goal state, inverting the associated phase:

$$O|\psi\rangle = \begin{cases} -|s\rangle|a_0a_1...a_d\rangle & if \quad |s\rangle \mapsto goalstate \\ |s\rangle|a_0a_1...a_d\rangle & otherwise \end{cases}$$

Now, the Grover iterate will amplify the correct sequence of actions that lead to the goal state. With this formulation we end up consuming more memory, due to the binary representation of the node. However, providing that the unitary operators $\mathcal{A}$ and $\mathcal{T}$ have an efficient representation, i.e. can be constructed in polynomial time, then the complexity of the algorithm will be dominated by the dimension of the search space, thus Grover's algorithm still guarantees a quadratic speedup. This formulation of the Hilbert spaces provides a way of dealing with arbitrary search trees with non-constant branching factors because Grover's algorithm will have always the correct search space associated. This algorithm can also work as an iterative deepening version of [67].

5.3 A quantum algorithm for the deterministic MDP

As we said in the introduction to this chapter, in order to deal with more complex problems in RL we need to consider MDP's, which form the basis for sequential decision making, where actions influence not just immediate but also subsequent states and, through those, future rewards. The *multi-armed bandit* problem that we solved in section 5.1, is formalized by an MDP as well, in fact, it is an MDP with one state. The agent has a set of actions that can perform, which is interpreted by which arm he decides to pull. By playing one of the arms the agent moves deterministically for the same state, with some reward returned by an arbitrary stochastic reward function. We want to take one step further and consider the case where we have multiple states. For that we need to construct MDP's in the quantum framework and resorting to the idea of oraculization of the task environment.

Before proceeding to the algorithmic design, it is worth clarifying some points about both the quantum agent and the environment. What does it even mean to construct oracular instances of the classical MDP? Why do we call it Oracles? Well, to answer that question, we need to look again at classical RL. If one recalls, there are essentially two different approaches to RL, namely the *model-based* case, when we have access to the dynamics of the world around us, and the *model-free* case, when we don't know anything about the environment. We consider the model-free approach as the *true* RL, the most general framework, where the agent learns simply by the interaction with the environment. Thus, we can picture the environment as sort of a black-box over which the agent performs some actions and in return, it places the agent into a new state and reward the agent for the action taken. In the quantum setting, if one want to be able to construct model-free RL algorithms to enhance classical agents, the quantum agent still does not have any information about the environment. Therefore the natural way to go is to construct quantum oracles that represent the dynamics of some classical environment. But how do we separate the quantum environment from the quantum agent? Again, we need to recall the definition of a classical agent. As we said back in section 3.3, the only concern is to define the agent mathematically as a function that maps every possible percept to an action. Thus, the agent has access to states and actions (and the rewards returned by the environment of course) and so is essentially an entity that maps states to actions, with the purpose of getting to the optimal mapping over time. Formally, we consider the quantum agent q as:

$$q : \quad |s\rangle \mapsto |a\rangle \quad \forall s \in S \tag{5.3.1}$$

and the quantum environment as an entity that has encoded the dynamics of the classical environment, $\hat{\mathbf{T}}$, for transition kernel and the rules for attributting the reward of the action taken by the agent, $\hat{\mathbf{R}}$ (Figure 5.3.1).

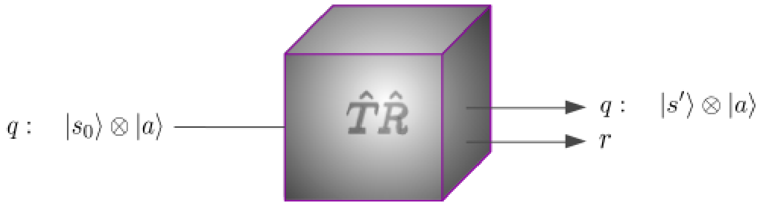

$q: \quad |s_0\rangle \otimes |a\rangle$
$\hat{T}\hat{R}$
$q: \quad |s'\rangle \otimes |a\rangle$
r

Figure 5.3.1: Representation of the interaction of the quantum agent with the quantum environment

with the difference that the quantum agent has a fundamentally different property compared to the classical agent, that is, the ability to perform actions in superposition, i.e. the quantum agent will interact in superposition with the quantum oracle that represents the environment.

$$|a\rangle = \sum_{i \in |A|} \alpha_i |a_i\rangle \tag{5.3.2}$$

such that $\sum_{i \in |A|} |\alpha_i|^2 = 1$. In fact, given that the agent will not know the proper action to take, the action register will be the uniform superposition state:

$$|a\rangle = \frac{1}{\sqrt{|A|}} \sum_{i \in |A|} |a_i\rangle \tag{5.3.3}$$

Now, the goal of the agent is to find a way to collapse the superposition in the action register into the action that leads to the highest accumulated reward.

If one understood the generalized quantum tree search algorithm of the last section, then one is one step closer to understand the algorithm of this section. The generalized tree search algorithm has almost all ingredients to be used in deterministic MDP's. Actually, we already constructed operators for representing the traversal of the tree, which we called $\mathcal{T}$:

$$\mathcal{T} : |s\rangle \otimes |a\rangle \mapsto |s'\rangle \otimes |a\rangle$$

This operator can be used as a quantum oracle for a classical MDP transition kernel. We know that we are working in a deterministic setting, therefore the encoding of the state may be irrelevant because what matters is a sequence of actions. However, the oracle will need to have some internal representation of the state to identify the sequence of actions and return to the agent the respective reward, so we prefer to encode the state in a quantum register because this will help building more general MDP's in the quantum framework. Moreover in the original $\mathcal{T}$ the state register is updated at each step. Thus, we will modify the oracle to be able to store the action taken in a given state as:

$$\mathcal{T} : |s\rangle \otimes |a\rangle \otimes |0\rangle^{\otimes n_s} \mapsto |s\rangle \otimes |a\rangle \otimes |s'\rangle \tag{5.3.4}$$

For now, we refer to the number of qubits used to represent a state as n_s but we will comeback to that later. Also in the tree search algorithm, we constructed an operator that creates the superposition of the admissable actions in a given node, $\mathcal{A}$:

$$\mathcal{A} : |s\rangle \otimes |0\rangle^{\otimes log_2|A|} \mapsto |s\rangle \otimes \frac{1}{\sqrt{|A_s|}} \sum_{i \in |A_s|} |a_i\rangle$$

It makes sense to consider this operator in the context of tree search because using this formulation enables us to consider trees with arbitrary branching factors, but does this relate to RL problems? Yes, almost all of the toy problems in RL work with the assumption that having MDP $\mathcal{M}$ with a finite set of states $\mathcal{S}$ and a finite set of actions $\mathcal{A}$, then any action $a \in \mathcal{A}$ is admissible in any state. However, in real-world problems, that is not the case, for example in robotics, we might have to control the joints of some robot, which may have different degrees of freedom. Using this formulation we can work on this set of problems. Using $\mathcal{A}$ and $\mathcal{T}$ for a given *horizon, h*. We can evolve through the MDP in superposition, however, to fully represent an MDP, we still need to rephrase the notion of a reward function in the quantum setting. There are numerous reward functions with various forms of representing rewards. We will consider two different reward functions: **(1)**: as a mapping from states to a real number, $R_s : S \mapsto \mathbb{R}$, which represent rewards that are only dependent on the agent state, and **(2)**: a mapping from state-action pairs to a real number, $R_{sa} : S \times A \mapsto \mathbb{R}$, representing rewards that depend on the transition itself, a more general representation.

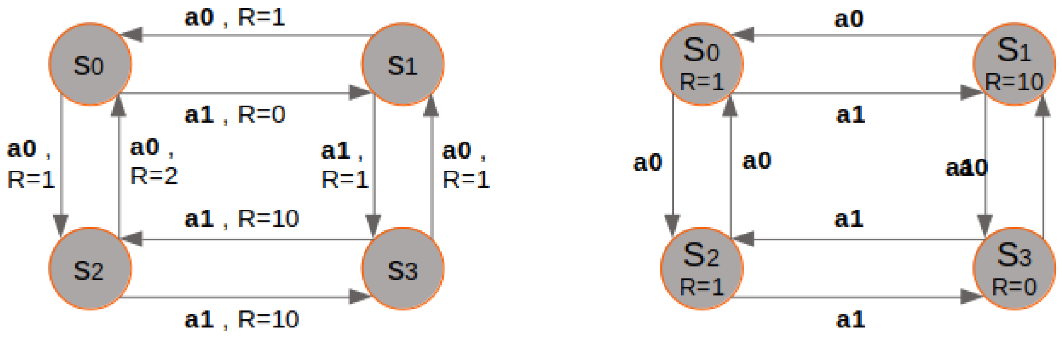


Figure 5.3.2: The same MDP represented with rewards depending only on the state *left image* and rewards depending on the action that the agent has taken *right image*

In the quantum framework, we will have a quantum register with the appropriate size (we will get to that later) and represent reward functions as oracles acting on the corresponding basis states. Thus, we can count the rewards for the sequence of actions such that:

$$R_s : |s\rangle \otimes |r\rangle \mapsto |s\rangle \otimes |r \oplus \mathcal{R}_s\rangle \tag{5.3.5}$$

for the case of state-dependent rewards and

$$R_{sa} : |s\rangle \otimes |a\rangle \otimes |r\rangle \mapsto |s\rangle \otimes |a\rangle \otimes |r \oplus \mathcal{R}_{sa}\rangle \tag{5.3.6}$$

for the case of rewards depending on the transition, where the symbol $\oplus$ represents the sum in the appropriate group. Thus, the quantum MDP is then composed by an Oracle itself composed by two sub-oracles, one responsible for the transition dynamics, $\mathcal{T}$, and one representing the reward function, either R_s or R_{sa}. At each transition, the agent applies $\mathcal{A}$ to the action register of the current time step responsible for creating the desired superposition of the admissable actions in the current state $\mathcal{A}$. Applying this set of operators for a given *horizon h*, is the same as computing the entire tree of possible trajectories, to a depth corresponding to the horizon, holding in an auxiliary quantum register the superposition of all possible received rewards. The careful reader, at this point hopefully, is thinking that we forgot to define the *discount factor* γ, present in a classical MDP. However, such is not the case, given that we're interested in finite horizon tasks, then γ can be ignored, at the same time that if the horizon is not infinite, but really large or, if we want to prioritize immediate rewards over delayed ones, then it could be done by adding the discounted reward into the reward quantum register at each oracle call t.

$$R_{sa} : |s\rangle \otimes |a\rangle \otimes |r\rangle \mapsto |s\rangle \otimes |a\rangle \otimes |r \oplus \lfloor \gamma^t \mathcal{R}_{sa} \rfloor\rangle \tag{5.3.7}$$

Again, we need the discount factor to be $\gamma \in [0, 1]$.

A note on the Space Complexity:

In classical computing not always is important to consider the space used by some program and, in fact, computer architectures and programming languages are so evolved nowadays that it is irrelevant to even talk about the number of bits used. However, when the size of the problem grows, especially in machine learning or data science applications, memory optimization becomes a crucial part of the algorithms. In quantum computing, we have until now, a slightly harder problem. It is believed that a quantum computer with 50-100 qubits, may outperform any classical computer, because its computational power is beyond what can be simulated by brute force using the most powerful existing supercomputers. However, despite that we already have 50 qubit machines nowadays, we still have imperfect control over those qubits and noise will place serious limitations on what quantum devices can achieve in the near term [53]. Thus, when designing quantum applications in the NISQ (Noisy Intermediate Scale Quantum) era, it is of great importance to consider the number of qubits used. The quantum algorithm section 5.3 is not an exception.

In the quantum algorithm for deterministic MDP's we started by formalizing the oraculization of the task environment, with the state transition oracle T:

$$T : |s\rangle \otimes |a\rangle \otimes |0\rangle^{\otimes n_s} \mapsto |s\rangle \otimes |a\rangle \otimes |s'\rangle$$

and the oracle R_s representing the reward oracle for environments with reward function dependent on only the current state:

$$R_s : |s\rangle \otimes |r\rangle \mapsto |s\rangle \otimes |r \oplus \mathcal{R}_s\rangle$$

or, the oracle R_{sa} for environments where the reward function is dependent on the transition itself:

$$R_{sa} : |s\rangle \otimes |a\rangle \otimes |r\rangle \mapsto |s\rangle \otimes |a\rangle \otimes |r \oplus \mathcal{R}_{sa}\rangle$$

Therefore, we need to analyse how many qubits do we need to represent states, actions and rewards. At each call to the transition oracle, we need to add to the full quantum state a new register initialized in the ground state, that is able to represent some state $s' \in S$. So, a natural question arises: How many qubits do we need to represent a state? We have two options that we will call **(1) static** and **(2) adaptive**.

(1) static: The simplest way is to consider all the possible states that we can represent. For example, if we have a MDP with a state space S, then we have $|S|$ states, thereby we need

$$n_s = \lceil log_2 |S| \rceil$$

qubits to represent a state. Moreover, we know that the algorithm computes a lookahead tree for a given *horizon* h in superposition, with the purpose of computing the optimal action for the given tree depth. This tell us that we need S_{static} qubits to represent states such that:

$$S_{static} = h \lceil log_2 |S| \rceil$$

(2) adaptive: Instead of having a quantum register that can allocate every possible state of the MDP at each iteration, i.e. each call to the transition oracle, we can exploit the fact that we're working with deterministic MDP's. Thus the number of possible states generated by each call will be dependent on the number of actions, $|A|$ as depicted in Figure 5.3.3.

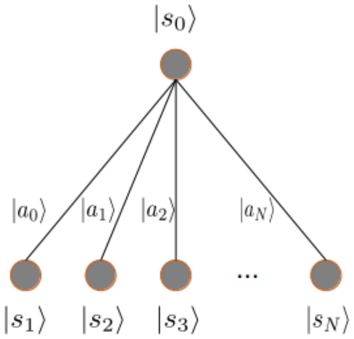


Figure 5.3.3: One step lookahead tree from N possible actions

so we just need to add to the full quantum state a quantum register with:

$$n_s = \lceil log_2 |A| \rceil$$

qubits. Furthermore, for an *horizon* h we have

$$S_{adaptive} = h\lceil log_2 |A| \rceil$$

The difference for both strategies can be significant. For example, in gridworld environments we have the action space $A = \{up, down, left, right\}$, $|A| = 4$. For small gridworlds, like 4×4 we have that

$$\frac{S_{static}}{S_{adaptive}} = \frac{h\lceil log_2 |S| \rceil}{h\lceil log_2 |A| \rceil} = \frac{log_2 |S|}{log_2 |A|} = \frac{log_2 16}{log_2 4} = 2$$

thereby, doubling the number of qubits used if we choose the static approach. Of course, the quotient increases when we have larger state space environments. On the other hand we can also have MDP's in which we could use both strategies with the same complexity for example the MDP of Figure 5.3.4 that with N different action leads to N different terminal states.

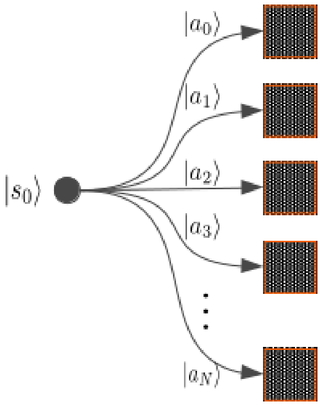


Figure 5.3.4: MDP with N possible actions that with one transition leads to N terminal states

This is to say that with the adaptive approach, will not always be the best approach. We may have MDP's with loops as in Figure 5.3.5, and in this case, constructing the oracle in the adaptive way will lead generally to complex oracles with non-trivial implementation if possible at all, because, in general, the oracle will need some fancy way of interpreting the state and do the correct operations based on that.

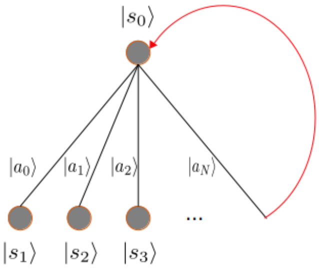


Figure 5.3.5: One step lookahead tree with a loop

The more complex the oracle is, the greater the impact will be in the run time complexity of the algorithm, so, in general, the decision will be based on a tradeoff between the computational complexity of the algorithm and the number of qubits available. Since the algorithm was applied to small MDP's use cases and given the non-trivial construction of the adaptive oracles, in practice we resorted to the static approach.

Beyond the qubits used to represent states, we still have to analyze the qubits used to represent actions and rewards. For actions we follow almost the same reasoning as before, given that the algorithm tests every possible action in superposition for a given horizon h, then we have

$$h\lceil log_2|A|\rceil$$

qubits used. For rewards, the algorithm has a single quantum register initialized in the ground state and continuously adds to the same register the reward accumulated throughout the trajectory. Therefore, as said before, we create the quantum register with a number of qubits sufficiently large, but how large? We have a reference value for the maximum reward the agent can collect in a single step, R, so for a given horizon h, the maximum reward that the agent can accumulate is hR, thereby we create the quantum register with:

$$\lfloor log_2 hR \rfloor + 1$$

qubits. Of course, this is not the end of the story, if one wants to fully characterize the space used by the algorithm, one has to think about the auxiliary qubits used for quantum subroutines responsible for reaching the optimal action sequence, the decomposition of multi control gates, etc. This is a part of the technology itself used, therefore we are not going into the details, but should keep this in mind.

To a clear understanding, let's work in an small example. Suppose the deterministic MDP of Figure 5.3.6 with state space $S = \{S_0, S_1\}$ and action space $A = \{a_0, a_1\}$. The MDP has rewards defined per state. The agent starts in state S_0, taking action a_1 moves to state S_1 and has a reward of 1. On the other hand, if the agent takes action a_0, it remains in the same state with zero reward.

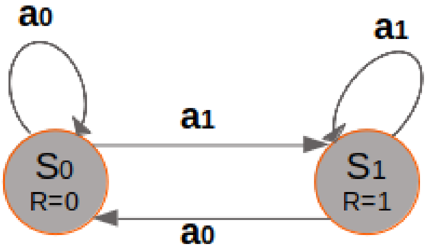


Figure 5.3.6: Example of a deterministic MDP with 2 states

Given that we have a two state MDP, in order to represent a state in the **static** approach, we need $log_2|S| = log_2 2 = 1$ qubit, and given that the initial state is S_0 we initialize the state register in the ground state. The same reasoning applies to the action register, resulting again in 1 qubit, given that $|A| = 2$. Both actions are admissable in both states of the MDP, so the agent will prepare the action register in an uniform superposition of both actions. Therefore for this example $\mathcal{A}$ is simply the Hadamard gate.

$$|s_0\rangle = |0\rangle \quad , \quad |a_0\rangle = \frac{1}{\sqrt{|A|}} \sum_{i \in |A|} |a_i\rangle = \frac{1}{\sqrt{2}}[|0\rangle + |1\rangle]$$

For representing the reward register we need to consider also the horizon. One interesting fact about this MDP is that it suffices to consider the horizon $h = 2$. If we will be using actions in superposition, with two single steps, we already have seen the entire MDP in superposition, so the maximum reward that the agent can receive in a single step is $R = 1$ of state S_1, thereby, for the horizon $h = 2$ we have that the maximum reward is 2 then we need $\lfloor log_2 2 \rfloor + 1 = 2$ qubits to represent the maximum reward.

$$|r\rangle = |00\rangle$$

The initial quantum state is represented by the tensor product of the three quantum registers:

$$|\psi_0\rangle = |s_0\rangle \otimes |a_0\rangle \otimes |r\rangle = |0\rangle \otimes \frac{1}{\sqrt{2}}[|0\rangle + |1\rangle] \otimes |00\rangle = \frac{1}{\sqrt{2}}|0000\rangle + \frac{1}{\sqrt{2}}|0100\rangle \tag{5.3.8}$$

In order to traverse in superposition thorough the MDP with $\mathcal{T}$, we need to add to the initial quantum state, the register encoding the state in which the agent is.

$$|s_1\rangle = |0\rangle \mapsto |\psi_0\rangle \otimes |s_1\rangle$$

Applying oracle $\mathcal{T}$, we achieve the following state:

$$\begin{aligned} |\psi_0'\rangle = \mathcal{T}(|\psi_0\rangle \otimes |s_1\rangle) &= \mathcal{T}\left((\frac{1}{\sqrt{2}}|0000\rangle + \frac{1}{\sqrt{2}}|0100\rangle) \otimes |s_1\rangle \right) \\ &= \frac{1}{\sqrt{2}}|0000\rangle \otimes |0\rangle + \frac{1}{\sqrt{2}}|0100\rangle \otimes |1\rangle \end{aligned} \tag{5.3.9}$$

Now, given that the reward function is defined in terms of the state, we apply oracle R_s to the respective quantum registers for the current time step to obtain:

$$\begin{aligned} |\psi_1\rangle = R_s|\psi_0'\rangle &= \frac{1}{\sqrt{2}}|00\rangle \otimes |0 \oplus \mathcal{R}_{s_0}\rangle \otimes |0\rangle + \frac{1}{\sqrt{2}}|01\rangle \otimes |0 \oplus \mathcal{R}_{s_1}\rangle \otimes |1\rangle \\ &= \frac{1}{\sqrt{2}}|0000\rangle \otimes |0\rangle + \frac{1}{\sqrt{2}}|0101\rangle \otimes |1\rangle \end{aligned} \tag{5.3.10}$$

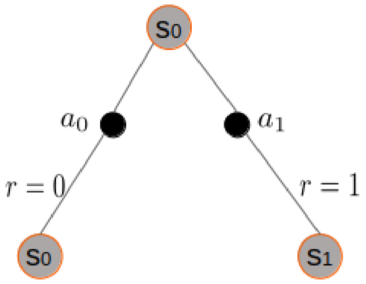


Figure 5.3.7: One-step lookahead tree computed by the superposition state $|\psi_1\rangle$.

The state $|\psi_1\rangle$ corresponds to computing the one-step lookahead tree in superposition as we can see in Figure 5.3.7. For the next transition we need to add to the state $|\psi_1\rangle$ new registers to represent the state and the action to take at the current state that the agent is in:

$$|s_2\rangle = |0\rangle \quad , \quad |a_1\rangle = \frac{1}{\sqrt{2}}[|0\rangle + |1\rangle] \mapsto |\psi_1\rangle \otimes |a_1\rangle \otimes |s_2\rangle \tag{5.3.11}$$

Now, we repeat the process, applying first the oracle $\mathcal{T}$ to the respective quantum registers for the current time step to reach the state:

$$|\psi_1'\rangle = \mathcal{T}\left((\frac{1}{\sqrt{2}}|0000\rangle \otimes |0\rangle + \frac{1}{\sqrt{2}}|0101\rangle \otimes |1\rangle) \otimes \frac{1}{\sqrt{2}}[|0\rangle + |1\rangle] \otimes |0\rangle \right) \tag{5.3.12}$$

The new action register added, breaks the current superposition state into more two terms, in which generating in total 4 superposition terms, corresponding to the four possible states in which the agent can be in the MDP when we take two steps in superposition. Breaking $|\psi_1\rangle$ into the respective superposition terms results in:

$$\begin{aligned} |\psi_1'\rangle = &\frac{1}{2}|0000\rangle \otimes |0\rangle \otimes |0\rangle \otimes |0\rangle + \frac{1}{2}|0000\rangle \otimes |0\rangle \otimes |1\rangle \otimes |1\rangle \\ &+ \frac{1}{2}|0101\rangle \otimes |1\rangle \otimes |0\rangle \otimes |0\rangle + \frac{1}{2}|0101\rangle \otimes |1\rangle \otimes |1\rangle \otimes |1\rangle \end{aligned} \tag{5.3.13}$$

and the respective reward accumulated through the four possible paths is given by applying the oracle R_s. By linearity we achieve the following quantum state that resembles the computation of the two step lookahead tree in superposition like in Figure 5.3.8:

$$\begin{aligned} |\psi_2\rangle = R_s|\psi_1'\rangle = &\frac{1}{2}|0000\rangle \otimes |0\rangle \otimes |0\rangle \otimes |0\rangle + \frac{1}{2}|0001\rangle \otimes |0\rangle \otimes |1\rangle \otimes |1\rangle \\ &+ \frac{1}{2}|0101\rangle \otimes |1\rangle \otimes |0\rangle \otimes |0\rangle + \frac{1}{2}|0110\rangle \otimes |1\rangle \otimes |1\rangle \otimes |1\rangle \end{aligned} \tag{5.3.14}$$

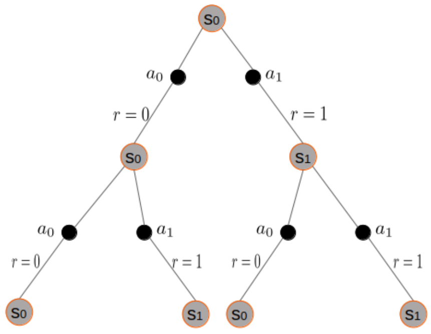


Figure 5.3.8: Two-step lookahed tree generated by the superposition state $|\psi_2\rangle$

All that we have been able to do until now, is to construct a quantum oracular version of a classical MDP and to design a quantum agent that can interact with the quantum environment in superposition, computing every possible path and the respective reward received for a certain horizon. We still have to collapse the superposition state to read an action, but if we collapse the state we will read the optimal action only with probability $\frac{1}{4}$. So we need to amplify the amplitude of the state that has the optimal action, which is the state that has collected the maximum reward. At this point we use the quantum maximum finding algorithm of section 2.6 as a subroutine, taking state $|\psi_2\rangle$ as the initial state.

Algorithm 7: quantum action selection for deterministic MDP's

```
$horizon \quad h, \quad R_{max}, \quad i \leftarrow 0;$
$|s_0\rangle \leftarrow |0\rangle^{\otimes log_2|S|}, |r\rangle \leftarrow |0\rangle^{\otimes \lfloor log_2 h R_{max} \rfloor + 1};$
$|\psi_0\rangle \leftarrow |s_0\rangle \otimes |r\rangle;$
while $i < h$ do
    $|a_i\rangle \leftarrow |0\rangle^{\otimes log_2|A|}, \quad |s_{i+1}\rangle \leftarrow |0\rangle^{\otimes log_2|S|};$
    $|\psi_i^{'}\rangle \leftarrow |\psi_i\rangle \otimes |a_i\rangle \otimes |s_{i+1}\rangle;$
    $|\psi_i^{''}\rangle \leftarrow \mathcal{T}\left(|s_i\rangle \otimes |a_i\rangle \otimes |s_{i+1}\rangle\right);$
    $|\psi_{i+1}\rangle \leftarrow R_s\left(|s_{i+1}\rangle \otimes |r\rangle\right);$
    $i \leftarrow i + 1;$
end
Apply quantum maximum finding (QMF) of section 2.6 on initial state $|\psi_h\rangle$;
$|\psi_{max}\rangle \leftarrow QMF(|\psi_h\rangle);$
Return $|\psi_{max}\rangle;$
```

Note: The algorithm uses R_s as the reward function. Of course, any other reward function could be used.

An interesting fact about this algorithm is that we are not trying to come up with the optimal policy for the MDP. Thus, we are not trying to define an action for every state of the MDP, just the actions for the states that the agent goes through in order to maximize the collected reward. This makes sense, at least from our point of view, because if we are dealing with deterministic MDP's, it is useless to define the action in states the agent will never reach if the goal of the agent is merely maximizing the reward in the long run. Moreover, the goal of the algorithm is rather to devise a sequence of actions that the agent may take. In certain cases however, the algorithm will also be able to devise the optimal policy, dependently on the MDP. This is the case for the above two states MDP. We have defined that with a horizon of $h = 2$ the agent is able to see the entire MDP in superposition, therefore the sequence of actions the agent may follow will simply be two actions that maximize the reward. For the MDP in question, the sequence of action that the algorithm return is $[a_0, a_1]$, meaning that the agent takes action a_0 and moves to state s_1 to collect the reward $R = 1$, then the agent takes action a_1 to remain in the same state, continuously keeping the same reward. In this case, we see that the algorithm can return the optimal policy, but, in general, it will not be the case.

In the next section, we will generalize deterministic MDP to stochastic ones. We will need not only to construct a new set of oracles able to deal with these types of problems, but also to construct new techniques in order to reach optimal decision-making.

5.4 QUANTUM SPARSE SAMPLING

In the last section we dealt with deterministic MDP's, which are a special case where the state transition probability matrix for each state-action pair has only one resulting state with probability one:

$$\mathcal{P}^a_{ss'} = \mathbb{P}[S_{t+1} = s' | S_t = s, A_t = a] = 1 \tag{5.4.1}$$

Stochastic MDP's, allows us to deal with more general environments. In a sense, it can capture more rigorously the nature around us, given that nature has always some uncertainty involved. Of course we're not saying that the deterministic setting is useless. In fact, there are tons of examples in which the environment is deterministic, like the well-known game of chess, or even the game of Go, that for many people, is known as the holy grail of AI. However, stochastic environments capture more realistic problems and, in a sense, deterministic MDP's can be thought of as a subset of stochastic MDP's, therefore it makes sense to consider them in the quantum framework as well. As we said before when we're dealing with large state space MDP's, or even dependently on the stochasticity of the problem, it may not be feasible to come up with a policy, i.e. a mapping from every state of the MDP to a corresponding action. Quantum MDP's lives under the same problem, thus, the quantum algorithm developed here is based on a different idea, the idea of *sparse sampling* (section 3.6). Instead of computing a policy, we will sample from a look-ahead tree, this way covering a fraction of the full look-ahead tree, which suffices to compute near-optimal actions from every state of the MDP. The process starts again by modeling an MDP as a simulated quantum environment that a quantum agent can interact in parallel for a given horizon h, generating a distribution, where one can extract the near-optimal action within the horizon. We will start by defining the quantum environment for stochastic environments.

In the deterministic case we start by constructing $\mathcal{T}$ oracle, the state transition oracle:

$$\mathcal{T} : |s\rangle \otimes |a\rangle \otimes |0\rangle^{\otimes log_2|S|} \mapsto |s\rangle \otimes |a\rangle \otimes |s'\rangle$$

It is not difficult to adapt this oracle into a stochastic setting. Given a state-action pair, we just need to add the probability associated to move into a certain state, $P^a_{ss'}$:

$$\mathcal{T} : |s\rangle \otimes |a\rangle \otimes |0\rangle^{\otimes log_2|S|} \mapsto |s\rangle \otimes |a\rangle \otimes \sum_{s' \in S} \sqrt{P^a_{ss'}} |s'\rangle \tag{5.4.2}$$

Notice that we amplitude encoded the square root of the probability, because, in the quantum world, we read a state with the square of the amplitude associated. We can simply just use the same algorithm as in the determinist setting (algorithm 7) but with the stochastic transition oracle instead. That's not so trivial, however. In the deterministic setting, to represent the rewards the agent get through the sequence of actions it takes into the environment, we considered that either we know the maximum reward the agent can achieve in a single step and for a given horizon we create a quantum register for the agent accumulated reward, or, we create the quantum register sufficiently large for the problem at hands. For purposes of making a more rigorous analysis we considered the first case and derived the necessary number of qubits:

$$\lfloor log_2 h R_{max} \rfloor + 1$$

In either case, the important point is that we're computing in basis encoding the collected reward by the agent, and this has its repercussions. To see that, we need to consider an example. In the deterministic setting, we considered two types of reward functions, state-dependent, R_s, or dependent on the transition itself, R_{sa}. We are going to consider in the example the transition dependent reward function:

$$R_{sa} : |s\rangle \otimes |a\rangle \otimes |r\rangle \mapsto |s\rangle \otimes |a\rangle \otimes |r \oplus \mathcal{R}_{sa}\rangle$$

Now, consider the stochastic MDP of Figure 5.4.1 with state-space $|S| = 3$ and action space $|A| = \{a_0, a_1\}$. The agent starts in state s_0 and, taking one of the possible actions, move probabilistically into the same state, receive a reward, continues to take actions, or moves into a terminal state and ends the game. In order to discover the action that leads to the highest reward we need to take into account the stochasticity of the environment. Therefore, we need to refer back to the **maximum expected utility principle** discussed in section 3.2. We need to take the expectation of the collected reward.

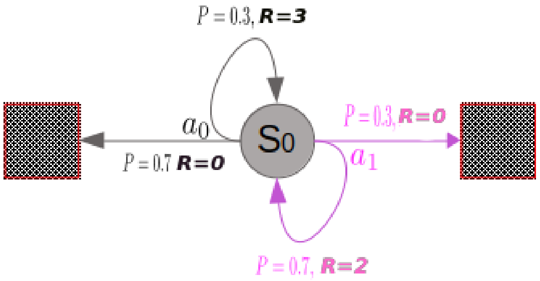


Figure 5.4.1: Stochastic MDP with 3 states, in which two of them are terminal states. Taking one of the two possible actions, the agent either moves to a terminal state or remains in the same state with some probability.

$$a^* = \underset{a}{\operatorname{argmax}}\, \mathbb{E}U(a|s) \tag{5.4.3}$$

where the utility function U is in fact the reward function, R_{sa}. Therefore, we have that the expected reward of some action is given by:

$$\mathbb{E}R(a|s) = P(s'|s,a)R(s,a) \tag{5.4.4}$$

Moreover, we have that the expected reward of each action is:

$$\begin{aligned} \mathbb{E}R(a_0|s_0) &= 0.7 \times 0 + 0.3 \times 3 = 0.9 \\ \mathbb{E}R(a_1|s_0) &= 0.7 \times 2 + 0.3 \times 0 = 1.4 \end{aligned} \tag{5.4.5}$$

Then we conclude that the optimal action is action a_1 because it leads to the highest expected reward.

In the quantum setting, if we consider the horizon as $h = 1$, because we know that the maximum reward in a single step is $R_{max} = 3$, then we know that we need to create a register with

$$\lfloor log_2 h R_{max} \rfloor + 1 = 2$$

qubits to represent the maximum reward achievable. Having quantum registers for representing the state of the agent with $log_2|S| = log_2 3 = 2$ qubits, with the initial state represented as the ground state of a two qubit system, as $|00\rangle$ and the action registers represented as a single qubit in superposition, we can achieve the propagation through the stochastic MDP by applying the state transition oracle $\mathcal{T}$ to the respective quantum registers, generating the following superposition state:

$$\begin{aligned}\mathcal{T}(|00\rangle \otimes |a\rangle \otimes |00\rangle) &= \mathcal{T}(|00\rangle \otimes \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \otimes |00\rangle) \\ &= \mathcal{T}(\frac{1}{\sqrt{2}}|00\rangle \otimes |0\rangle \otimes |00\rangle + \frac{1}{\sqrt{2}}|00\rangle \otimes |1\rangle \otimes |00\rangle) \\ &= \frac{1}{\sqrt{2}}|00\rangle \otimes |0\rangle \otimes \sum_{s' \in S} \sqrt{P^a_{ss'}}|s'\rangle + \frac{1}{\sqrt{2}}|00\rangle \otimes |1\rangle \otimes \sum_{s' \in S} \sqrt{P^a_{ss'}}|s'\rangle \\ &= \frac{1}{\sqrt{2}}|00\rangle \otimes |0\rangle \otimes \left[\sqrt{0.7}|01\rangle + \sqrt{0.3}|00\rangle\right] + \frac{1}{\sqrt{2}}|00\rangle \otimes |1\rangle \otimes [\sqrt{0.7}|00\rangle + \\ &+ \sqrt{0.3}|10\rangle] \end{aligned} \tag{5.4.6}$$

which gives the following non-uniform superposition state with four superposition terms:

$$\sqrt{\frac{0.7}{2}}|00\rangle \otimes |0\rangle \otimes |01\rangle + \sqrt{\frac{0.3}{2}}|00\rangle \otimes |0\rangle \otimes |00\rangle + \sqrt{\frac{0.7}{2}}|00\rangle \otimes |1\rangle \otimes |00\rangle + \sqrt{\frac{0.3}{2}}|00\rangle \otimes |1\rangle \otimes |10\rangle \tag{5.4.7}$$

Now, if we append to the above quantum state, the quantum register responsible for representing the reward associated to the transition, initialized in the ground state and apply the reward Oracle, R_{sa}, we get the state:

$$R_{sa}\left[\sqrt{\frac{0.7}{2}}|00001\rangle \otimes |00\rangle + \sqrt{\frac{0.3}{2}}|00000\rangle \otimes |00\rangle + \sqrt{\frac{0.7}{2}}|00100\rangle \otimes |00\rangle + \sqrt{\frac{0.3}{2}}|00110\rangle \otimes |00\rangle\right]$$

$$|\psi\rangle = \sqrt{\frac{0.7}{2}}|00001\rangle \otimes |00\rangle + \sqrt{\frac{0.3}{2}}|00000\rangle \otimes |11\rangle + \sqrt{\frac{0.7}{2}}|00100\rangle \otimes |10\rangle + \sqrt{\frac{0.3}{2}}|00110\rangle \otimes |00\rangle \tag{5.4.8}$$

If we run the quantum maximum finding algorithm 2 with $|\psi\rangle$ as the initial state in order to collapse the superposition into the state that has the highest reward, the algorithm will return the superposition term with reward $R = 3$. Therefore the algorithm concludes that the action a_0 is the best action to take, which is incorrect. The algorithm instead of amplifying the state that corresponds the highest **expected** reward, amplifies the state that has the **absolute** maximum reward, proving that this technique works well for the deterministic case, but for the stochastic, failed miserably. For the example at hands, we're considering horizon $h = 1$ because we just needed to make a single action. If we basis encode the reward and use the quantum maximum finding as a subroutine for reaching the optimal action to take, the algorithm will fail because it does not respect the **maximum expected utility principle**. In general, if we consider a sequence of actions instead of a single action, the algorithm will also fails, in this case because it does not return the action that has the highest **expected cumulative reward**.

At this point, we need new ways of encoding the rewards that the agent gets throughout the exploration of the MDP, and new ways of making optimal decisions based on them. Recall the Bloch Sphere Figure 2.2.1, representing a qubit in an arbitrary state:

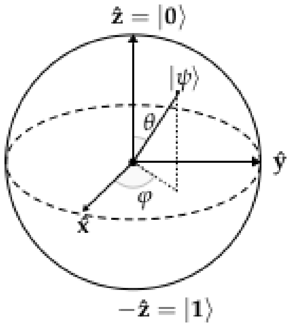


Figure 5.4.2: Bloch Sphere, representing a qubit in an arbitrary state

$$|\psi\rangle = e^{i\phi}\left(cos\frac{\theta}{2}|0\rangle + e^{i\varphi}sin\frac{\theta}{2}|1\rangle\right)$$

What if, instead of encoding the reward in the basis of some quantum register, we rather encode the rewards in the phase of a qubit ? If we set $\theta = \pi$ and $\varphi = 0$ we have the state:

$$|\psi\rangle = e^{i\phi}|1\rangle$$

that has a global phase associated. We can achieve the latter state by using rotations on the z-axis. If the goal of representing the reward in basis states is to have the power of performing addition using arithmetic circuits, we can do the same using phase encoding without the use of arithmetic circuits. Notice the z-rotation gate:

$$R_z(\phi) = \begin{pmatrix} e^{-i\phi} & 0 \\ 0 & e^{i\phi} \end{pmatrix} \tag{5.4.9}$$

If we have a qubit initialized in $|1\rangle$ state, and perform two rotations of θ_1 and θ_2, we get the following state:

$$R_z(\phi_2)R_z(\phi_1)|1\rangle = \begin{pmatrix} e^{-i\phi_2} & 0 \\ 0 & e^{i\phi_2} \end{pmatrix} e^{i\phi_1}|1\rangle = e^{i\phi_2}e^{i\phi_1}|1\rangle = e^{i(\phi_1+\phi_2)}|1\rangle \tag{5.4.10}$$

By using rotations around the z-axis we can perform addition essentially "for free" and without the need of having a quantum register sufficiently large to record the reward accumulation. Furthermore, we can also consider easily the discounted finite horizon case as a sequence of z-rotation gates:

$$\prod_{t=0}^{h} R_z(\gamma^t R_z(R_t))|1\rangle = exp(i(R_0 + \gamma R_1 + \gamma^2 R_2 + \cdots + \gamma^h R_h))|1\rangle \tag{5.4.11}$$

In order for this to work in the oracularization of task environment paradigm that we have been proposing, we need to consider the z-rotation gates as a part of the reward oracles. For example, for MDP's in which the reward depends on the transition, we will have the oracle:

$$R_{sa} : |s\rangle \otimes |a\rangle \otimes |r\rangle \mapsto |s\rangle \otimes |a\rangle \otimes e^{jR_{sa}}|r\rangle \tag{5.4.12}$$

An important observation is that the rotation gate is applied accordingly to the state and action registers, meaning that the gate is in fact a controlled rotation gate. Where does this leads us ? The point of encoding the rewards into the phase of some qubit is to try to have acess to the expected values. At this point, given that we already have the oracles for stochastic environments, consider an high level representation for one oracle call to both $\mathcal{T}$ and R_{sa}, having quantum registers representing some initial state:

$$\mathcal{T}\left(|s\rangle \otimes \frac{1}{\sqrt{|A|}} \sum_{i\in|A|} |a_i\rangle \otimes |0\rangle^{\otimes n_s}\right) = |s_0\rangle \otimes \frac{1}{\sqrt{|A|}} \sum_{i\in|A|} |a_i\rangle \otimes \sum_{s'\in S} \sqrt{P^a_{ss'}}|s'\rangle \tag{5.4.13}$$

If we append to the state the reward register and apply the reward oracle we get the overall state:

$$|s_0\rangle \otimes \frac{1}{\sqrt{|A|}} \sum_{i\in|A|} |a_i\rangle \otimes \sum_{s'\in S} \sqrt{P^a_{ss'}}|s'\rangle \otimes \sum_{s,a} e^{jR_{sa}}|r\rangle \tag{5.4.14}$$

If we unravel the superposition terms, we reckon that the amplitude associated to each term will have the shape:

$$\frac{1}{\sqrt{|A|}} e^{jR_{sa}} \sqrt{P^a_{ss'}}$$

which looks almost like computing the expected reward and store the result into the amplitude of the quantum state. But it is not exactly the same. Moreover, recall that the quantum register for the reward is initialized into the basis state $|r\rangle = |1\rangle$, taking the z-basis as the computational basis. This means that the phase gained representing the accumulation of the reward, has no observable effect because it represents rotations in the z-axis and measuring in the computational basis, reveals nothing, in fact, when measuring the quantum state, we read an arbitrary superposition term with probability:

$$\frac{1}{|A|} P^a_{ss'}$$

We could use quantum phase estimation [52] in order to get an estimation of the cummulative reward, but the subroutine needs to use powers of the controlled unitaries that prepare the phase. Therefore, we would need to use powers of the controlled versions of the reward oracle, which increase exponentially with the precision required. Moreover, if we want to estimate the phase up to n bits, with a sucess probabilty of at least $1 - \epsilon$, we will we need

$$n + \lceil log(2 + \frac{1}{2\epsilon}) \rceil \tag{5.4.15}$$

qubits. Furthermore, even though we are able to estimate the reward, we still do not know how to proceed from there. We could run quantum maximum finding into the estimated reward. However, we are again running into the same problem, as we would be amplifying the amplitude of the state with the highest absolute reward and not the expected, because the phase estimation subroutine estimates the reward encoded in the phase, not taking into account the probability encoded in the amplitude. Well, that was unfortunate, but, it revealed a way of encoding the rewards which is meaningful when we collapse the superposition. The problem with the approach above is that using z-rotations, the reward is properly accumulated, but, we run into problems for extracting the cumulative reward because it was in the quantum state global phase which has no physical meaning. Thus instead of encoding the reward as a global phase, we could encode it into the amplitude. For that instead of using rotations around the z-axis, we should use rotations around the y-axis, known as *angle encoding*. Recall the y-rotation gate acting on a qubit in the ground state

$$R_y(\theta)|0\rangle = \begin{pmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} cos\theta \\ sin\theta \end{pmatrix} = cos\theta|0\rangle + sin\theta|1\rangle \tag{5.4.16}$$

If the value of θ was the value of the reward we could encode the reward into the amplitude of quantum state. If we perform two rotations θ_1 and θ_2 into a qubit initialized in the ground state, we generate the following quantum state

$$\begin{aligned} R_y(\theta_2)R_y(\theta_1)|0\rangle &= \begin{pmatrix} cos\theta_2 & -sin\theta_2 \\ sin\theta_2 & cos\theta_2 \end{pmatrix} \begin{pmatrix} cos\theta \\ sin\theta \end{pmatrix} \\ &= \begin{pmatrix} cos\theta_2 cos\theta_1 - sin\theta_2 sin\theta_1 \\ sin\theta_2 cos\theta_1 + cos\theta_2 sin\theta_1 \end{pmatrix} = \begin{pmatrix} cos(\theta_2 + \theta_1) \\ sin(\theta_2 + \theta_1) \end{pmatrix} \\ &= cos(\theta_2 + \theta_1)|0\rangle + sin(\theta_2 + \theta_1)|1\rangle \end{aligned} \tag{5.4.17}$$

With this, we can encode a sequence of h rewards, h being the horizon associated to the problem, within a sequence of h y-rotation gates

$$\prod_{t=0}^{h} R_y(\gamma^t R_t)|0\rangle = cos\left(\sum_{t=0}^{h}\gamma^t R_t\right)|0\rangle + sin\left(\sum_{t=0}^{h}\gamma^t R_t\right)|1\rangle \tag{5.4.18}$$

Using this formulation for the reward function, we notice that for horizon $h = 1$, with one oracle call to $\mathcal{T}$ and R_{sa}, we get the the following superposition state:

$$|s_0\rangle \otimes \frac{1}{\sqrt{|A|}} \sum_{i\in|A|} |a_i\rangle \otimes \sum_{s'\in S} \sqrt{P^a_{ss'}}|s'\rangle \otimes \sum_{s,a}[cosR_{sa}|0\rangle + sinR_{sa}|1\rangle] \tag{5.4.19}$$

meaning that if we unravel the superposition, each term will have an associated amplitude

$$\begin{cases} \frac{1}{\sqrt{|A|}}cosR_{sa}\sqrt{P^a_{ss'}} & ,|r\rangle = |0\rangle \\ \frac{1}{\sqrt{|A|}}sinR_{sa}\sqrt{P^a_{ss'}} & ,|r\rangle = |1\rangle \end{cases} \tag{5.4.20}$$

that for a sequence of actions, horizon h, translates to

$$\begin{cases} \frac{h}{\sqrt{|A|}}cos\left(\sum_{t=0}^{h}\gamma^t R_t\right)\sqrt{P^a_{s_0s_1}P^a_{s_1s_2}\dots P^a_{s_{h-1}s_h}} & ,|r\rangle = |0\rangle \\ \frac{h}{\sqrt{|A|}}sin\left(\sum_{t=0}^{h}\gamma^t R_t\right)\sqrt{P^a_{s_0s_1}P^a_{s_1s_2}\dots P^a_{s_{h-1}s_h}} & ,|r\rangle = |1\rangle \end{cases} \tag{5.4.21}$$

Ignoring the normalization factor created by the superposition in the action registers $\frac{h}{\sqrt{|A|}}$, we see that if we measure the quantum state, we read each superposition term with probability:

$$\begin{cases} cos^2\left(\sum_{t=0}^{h}\gamma^t R_t\right)P^a_{s_0s_1}P^a_{s_1s_2}\dots P^a_{s_{h-1}s_h} & ,|r\rangle = |0\rangle \\ sin^2\left(\sum_{t=0}^{h}\gamma^t R_t\right)P^a_{s_0s_1}P^a_{s_1s_2}\dots P^a_{s_{h-1}s_h} & ,|r\rangle = |1\rangle \end{cases} \tag{5.4.22}$$

which closely resembles the expected cumulative reward, with the computation made in the amplitude of quantum states. Thus *in principle* the superposition term with the highest amplitude associated is the state with the highest expected cumulative reward. Therefore, we can turn this into an optimal decision making algorithm. Before discussing the decision making component, we need to check a couple things more.

Notice that the reward is being added via rotations around the y-axis, into the amplitude of a quantum state that is initially in the ground state, meaning that we need to take care of how much reward we can add, which is in fact when the overall reward is equivalent to a full rotation of $\frac{\pi}{2}$ Figure 5.4.3. Otherwise, we will loose the information about the true reward received by the agent.

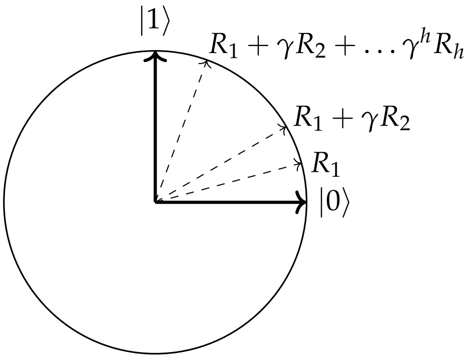


Figure 5.4.3: Reward qubit evolution with the addition of the rewards

Furthermore, we need to normalize the reward such that we know for sure that in any possible transition the reward, is at most $\frac{\pi}{2}$. For that, we will consider the case where we have a reference value for the absolute maximum reward that the agent could get in a single transition step, R_{max}. We know that:

$$\sum_{t=0}^{\infty} \frac{\gamma^t R_t}{R_{max}} \leq \sum_{t=0}^{\infty} \frac{\gamma^t R_{max}}{R_{max}} \leq \sum_{t=0}^{\infty} \gamma^t = \frac{1}{1-\gamma} \quad , \quad \gamma \in [0,1) \tag{5.4.23}$$

so if we set the reward equal to:

$$\frac{\pi}{2}(1-\gamma) \sum_{t=0}^{\infty} \frac{\gamma^t R_t}{R_{max}} \leq \frac{\pi}{2} \tag{5.4.24}$$

the requirements are met. However, this is not entirely true, because, the convergence of the series is in the infinity. Thus the value of the reward will converge to $\frac{\pi}{2}$ in the infinity. The problem with this is that we are working with finite horizon problems, therefore, if we consider convergence in the infinity, the ampltitude change due to the reward will be almost indistinguishable for every path because it will be very small for almost every sequence of rewards. Furthermore we need to take into account the horizon h:

$$\sum_{t=0}^{h-1} \frac{\gamma^t R_t}{R_{max}} \leq \sum_{t=0}^{h-1} \gamma^t \leq \frac{\gamma^h - 1}{\gamma - 1} \quad , \quad \gamma \in [0,1) \tag{5.4.25}$$

Moreover, setting the reward equal to:

$$\frac{\pi(\gamma-1)}{2(\gamma^h-1)} \sum_{t=0}^{h-1} \frac{\gamma^t R_t}{R_{max}} \leq \sum_{t=0}^{h-1} \gamma^t \leq \frac{\pi}{2} \tag{5.4.26}$$

the requirements are met, meaning that each rotation applied by the reward oracle will be

$$R_y(\eta \gamma^t \frac{R_t}{R_{max}}) \tag{5.4.27}$$

with η denoting the normalization factor:

$$\eta = \frac{\pi(\gamma - 1)}{2(\gamma^{h+1} - 1)} \tag{5.4.28}$$

We close the discussion on the requirements for the rewards that the agent get by constructing the final versions for the reward oracles that will be used in stochastic environments, for both state dependent reward environments and transition dependent reward environments:

$$R_s : |s\rangle \otimes |r\rangle \mapsto |s\rangle \otimes R_y(\eta\gamma^t \frac{R_s}{R_{max}})|r\rangle \tag{5.4.29}$$

$$R_{sa} : |s\rangle \otimes |a\rangle \otimes |r\rangle \mapsto |s\rangle \otimes |a\rangle \otimes R_y(\eta\gamma^t \frac{R_{sa}}{R_{max}})|r\rangle \tag{5.4.30}$$

We turn now our attention to the optimal decision making component: how can the agent, based on the information gained by interacting in superposition with the environment, decide which action to take. Notice that if the agent interacts with the quantum environment in superposition, taking every possible action at each transition step, for a given horizon h, it will have an internal representation of the expected cumulative reward calculated for every possible path until the horizon is met. Moreover, the agent will be in a non-uniform superposition state of every possible path weighted by the probability of each transition effectively occuring when taking an action and will have represented in the amplitude of each superposition term, the respective reward obtained by following that path. Essentially, given an initial state representing the starting point in a given MDP, if we create all the necessary quantum registers and call the quantum environment oracles for a given horizon, we will be computing in superposition the tree with every possible path, as in the deterministic case. The branching factor will not be just the number of actions alone, because of the stochastic environment. Therefore, the branching factor will depend on state-action pair transition probabilities. Furthermore, the reward is applied by a rotation around the y-direction onto the reward register:

$$R_y(\eta\gamma^t R_t)|r\rangle = cos\left(\eta\gamma^t R_t\right)\right) |0\rangle + sin\left(\eta\gamma^t R_t\right)\right) |1\rangle \tag{5.4.31}$$

meaning that each oracle call will break the superposition state into two other states due to the cosine and sine terms as illustrated in Figure 5.4.4

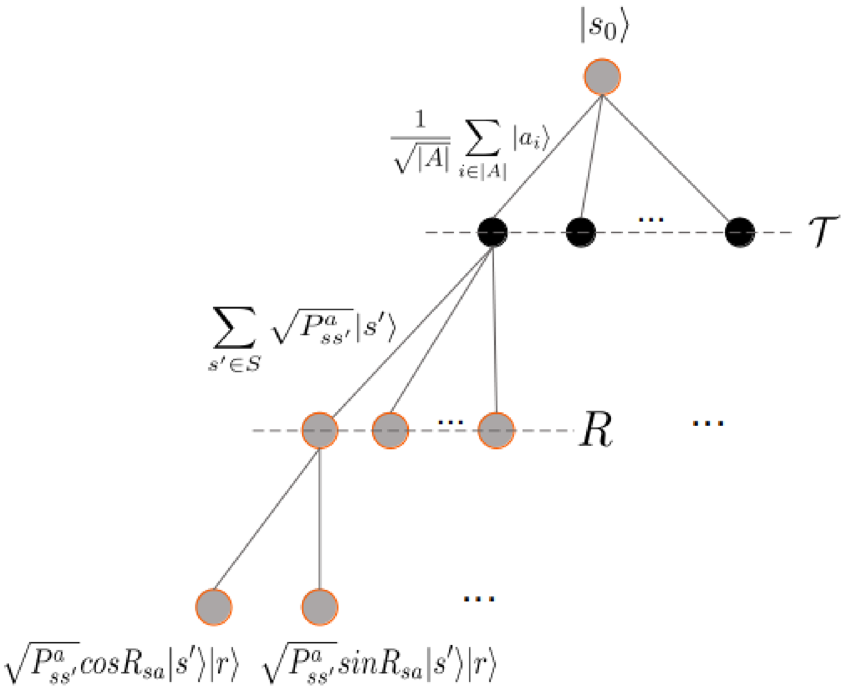

$|s_0\rangle$
$\frac{1}{\sqrt{|A|}}\sum_{i\in|A|}|a_i\rangle$
...
$\mathcal{T}$
$\sum_{s'\in S}\sqrt{P^a_{ss'}}|s'\rangle$
...
R
...
...
$\sqrt{P^a_{ss'}}cosR_{sa}|s'\rangle|r\rangle$
$\sqrt{P^a_{ss'}}sinR_{sa}|s'\rangle|r\rangle$

Figure 5.4.4: One step lookahead tree computed in superposition, created by the oracle calls of $\mathcal{T}$ and R

We said before that this way, the superposition term with the highest amplitude will be the term of interest *in principle* because we could have MDP's with a sequence of actions that lead to small rewards or even null rewards, meaning that in these cases the cosine term in the accumulated reward will be larger than other modest rewards created by other sequence of actions, or in the case of null reward sequence , the cosine term will be maximized. Along with proper state transition probabilities this may raise superposition terms whose amplitude does not correspond to the true highest expected reward. The logical step to take at this point, is to amplify the amplitude of good states, i.e. the states that have the sine term associated with, which reflects the true value of the reward accumulated throughout the sequence of actions taken by the agent. As discussed in section 2.4, in order to amplify good states, we need two reflection operators, also known as the *Grover operator* $\mathcal{G}$, the reflection over the space of the good states, $\mathcal{R}_{good}$ and the reflection over the overall quantum state, $\mathcal{R}_{\psi}$:

$$\mathcal{G} = \mathcal{R}_{\psi}\mathcal{R}_{good} = (\mathcal{A}\mathcal{R}_0\mathcal{A}^{-1})\mathcal{R}_{good}$$

The quantum operator $\mathcal{A}$ is the initial quantum state preparation operator. In our case, this operator will be dictated by the oracles which the agent interacts with, for a given horizon h, namely the state transition oracle $\mathcal{T}$ and the reward oracle R. Given the parallel interaction, we need to consider the Hadamard gates applied to the action register as well. The reflection around the good states can be easily made by applying a phase gate in the reward qubit $\hat{Z}_r$, thus inverting the phase of the states that have the reward qubit equal to $|1\rangle$, the states that have the sine term associated reflecting the true value of the reward accumulated throughout the sequence of actions taken by the agent.

$$\hat{Z}_r|r\rangle = \hat{Z}_r \left[cos\left(\sum_{t=0}^{h}\gamma^t R_t\right)|0\rangle + sin\left(\sum_{t=0}^{h}\gamma^t R_t\right)|1\rangle\right] = cos\left(\sum_{t=0}^{h}\gamma^t R_t\right)|0\rangle - sin\left(\sum_{t=0}^{h}\gamma^t R_t\right)|1\rangle \tag{5.4.32}$$

Therefore, the Grover operator responsible for one round of amplitude amplification will be:

$$\mathcal{G} = \mathcal{R}_\psi \mathcal{R}_{good} = \mathcal{A}\mathcal{R}_0\mathcal{A}^{-1}\hat{Z}_r = \mathcal{A}(-\mathbb{1} + 2|0\rangle\langle 0|)\mathcal{A}^{-1}\hat{Z}_r \tag{5.4.33}$$

How many Grover iterations do we need? After the agent interacts with the quantum environment created by the state transition and reward oracles, the overall quantum state of the agent will be a non-uniform superposition state and we don't know a priori the distribution of the amplitudes. Moreover, the reward oracle will break each superposition term into two branches, so unless we have a problem with a single reward, we will have multiple marked states. Therefore we cannot compute for sure the optimal number of iterations that we need to read the optimal superposition term, i.e. the superposition term that corresponds the highest expected reward. We need to perform an exponential search like in algorithm 1. If we call the exponential search algorithm we will read a "good state", i.e. a state with the reward qubit equal to $|1\rangle$, but this does not guarantee that we read the state that corresponds to the highest expected cumulative reward. However, if we call the exponential search algorithm a sufficient number of times (more on that later) we will be able to generate a distribution over the set of possible actions $\mathcal{A}$. Therefore, from the m samples taken we generate statistics from which the optimal action can be extracted.

$$a^* = \underset{a}{\operatorname{argmax}}\, \mathcal{A}$$

At this point we already have all the ingredients for constructing an algorithm for decision making in stochastic environments, however, we need to construct a modified version of the exponential search algorithm (algorithm 1) because in our case we are not searching for an element in a list of elements, but for a marked state such that the reward qubit is equal to one, meaning that the state collapsed in a true reward sequence.

Algorithm 8: QSS - Quantum Sparse Sampling

```
$horizon \quad h, \quad R_{max}, \quad samples \quad S;$
$m \leftarrow 0, \quad \mathcal{A} \leftarrow Null;$
**while** $m < S$ **do**
    $i \leftarrow 0;$
    $|s_i\rangle \leftarrow |0\rangle^{\otimes log_2|S|}, |r\rangle \leftarrow |0\rangle;$
    $|\psi_i\rangle \leftarrow |s_i\rangle \otimes |r\rangle;$
    **while** $i < h$ **do**
        $|a_i\rangle \leftarrow |0\rangle^{\otimes log_2|A|}, \quad |s_{i+1}\rangle \leftarrow |0\rangle^{\otimes log_2|S|};$
        $|\psi_i^{'}\rangle \leftarrow |\psi_i\rangle \otimes |a_i\rangle \otimes |s_{i+1}\rangle;$
        $|\psi_i^{''}\rangle \leftarrow \mathcal{T}\left(|s_i\rangle \otimes |a_i\rangle \otimes |s_{i+1}\rangle\right);$
        $|\psi_{i+1}\rangle \leftarrow R_s\left(|s_{i+1}\rangle \otimes |r\rangle\right);$
        $i \leftarrow i + 1;$
    **end**
    Apply modified exponential search (MQSearch) algorithm 9 on initial state $|\psi_h\rangle$;
    $s \leftarrow MQSearch(|\psi_h\rangle);$
    Append $s$ to $\mathcal{A}$
**end**
**Return** $\text{argmax}_a \mathcal{A};$
```

Note: The algorithm works with any reward function. R_s was used just for illustration purposes.

Algorithm 9: MQSearch - Modified Exponential Search

```
$m \leftarrow 1$ , $\lambda = \frac{6}{5}$ , $horizon \quad h$;
$s\_qubits \leftarrow hlog|S|$ , $a\_qubits \leftarrow hlog|A|$;
$max\_it \leftarrow 2^{s\_qubits} + 2^{a\_qubits}$;
$initial\ state\ |\psi_h\rangle$;
while $m \leq max\_it$ do
    $it \leftarrow random(1, m)$;
    $|\psi_{copy}\rangle \leftarrow |\psi_h\rangle$;
    Apply $it$ iterations of Grover's algorithm;
    Measure $|\psi_{copy}\rangle$ , $r \leftarrow |r\rangle, a_0 \leftarrow |a_0\rangle$;
    if $r == 1$ then
        Return $a_0$;
    else
        $m \leftarrow min(\lambda m, max\_it)$;
    end
end
Return Null;
```

As an example, let's recall the probabilistic model of section 3.2 because it is an MDP with a single transition step with horizon equal to 1. Thus, we can see the entire MDP in superposition and select the action that respects the maximum expected utility principle.

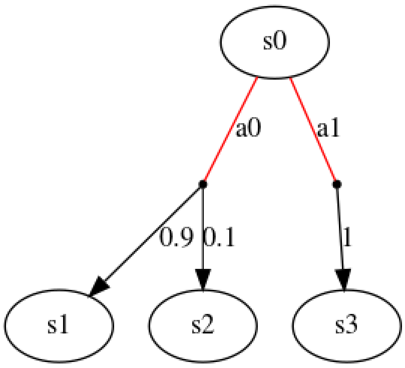


Figure 5.4.5: probabilistic model

The reward function plays the role of the utility function and it is designed to depend on the state only.

$$R(s) = \begin{cases} 10 & \text{if } s = s1 \\ 50 & \text{if } s = s2 \\ 20 & \text{if } s = s3 \end{cases} \quad (5.4.34)$$

The system is composed by a state space with four states $S = \{s_0, s_1, s_2, s_3\}$ and an action space with two possible actions $A = \{a_0, a_1\}$. The goal is to find the optimal action to take at the initial state s_0. We need to define quantum registers for both states, actions and the reward:

$$|s_0\rangle = |0\rangle^{\otimes log|S|} = |00\rangle \quad , \quad |s'\rangle = |00\rangle \quad , \quad |a\rangle = |0\rangle^{\otimes log|A|} = |0\rangle \quad , \quad |r\rangle = |0\rangle$$

where the action register will be in an uniform superposition:

$$|a\rangle = H|a\rangle = \frac{1}{\sqrt{|A|}} \sum_{i \in |A|} |a_i\rangle = \frac{1}{\sqrt{2}} \left[|a_0\rangle + |a_1\rangle \right]$$

$$\begin{aligned} |\psi_0\rangle &= |s_0\rangle \otimes |a\rangle \otimes |s'\rangle \otimes |r\rangle \\ &= \frac{1}{\sqrt{2}} |s_0\rangle|a_0\rangle|s'\rangle|r\rangle + \frac{1}{\sqrt{2}} |s_0\rangle|a_1\rangle|s'\rangle|r\rangle \end{aligned} \tag{5.4.35}$$

The state transition oracle will act linearly on both branches of the uniform superposition state resulting in the following non-uniform superposition state:

$$\begin{aligned} \mathcal{T}|\psi_0\rangle &= \frac{1}{\sqrt{2}} \mathcal{T} \left[|s_0\rangle|a_0\rangle|s'\rangle \right] |r\rangle + \frac{1}{\sqrt{2}} \mathcal{T} \left[|s_0\rangle|a_1\rangle|s'\rangle \right] |r\rangle \\ &= \frac{1}{\sqrt{2}} \sqrt{0.9} |s_0\rangle|a_0\rangle|s_1\rangle|r\rangle + \frac{1}{\sqrt{2}} \sqrt{0.1} |s_0\rangle|a_0\rangle|s_2\rangle|r\rangle + \frac{1}{\sqrt{2}} |s_0\rangle|a_1\rangle|s_3\rangle|r\rangle = |\psi_0'\rangle \end{aligned} \tag{5.4.36}$$

If we apply the state dependent reward oracle to the respective transition step registers, we generate the non uniform superposition state that contain the parallel interaction of the agent with the quantum environment. The reward is applied via controlled y-rotation of the reward qubit as:

$$R_y(\eta \gamma^t \frac{R_s}{R_{max}})$$

In this case we're dealing with a single transition step MDP, with horizon equal to 1, therefore the term γ vanishes and setting $\gamma = 0.9$, η in the equation becomes:

$$\eta = \frac{\pi(\gamma - 1)}{2(\gamma^h - 1)} = \frac{\pi}{2}$$

$$\begin{aligned}
R_s|\psi_0'\rangle &= \sqrt{\frac{9}{20}}|s_0\rangle|a_0\rangle R_s\left[|s_1\rangle|r\rangle\right] + \sqrt{\frac{1}{20}}|s_0\rangle|a_0\rangle R_s\left[|s_2\rangle|r\rangle\right] + \frac{1}{\sqrt{2}}|s_0\rangle|a_1\rangle R_s\left[|s_3\rangle|r\rangle\right] \\
&= \sqrt{\frac{9}{20}}cos\left(\frac{\pi}{10}\right)|s_0\rangle|a_0\rangle|s_1\rangle|r\rangle + \sqrt{\frac{9}{20}}sin\left(\frac{\pi}{10}\right)|s_0\rangle|a_0\rangle|s_1\rangle|r\rangle + \\
&+ \sqrt{\frac{1}{20}}cos\left(\frac{\pi}{2}\right)|s_0\rangle|a_0\rangle|s_2\rangle|r\rangle + \sqrt{\frac{1}{20}}sin\left(\frac{\pi}{2}\right)|s_0\rangle|a_0\rangle|s_2\rangle|r\rangle + \\
&+ \frac{1}{\sqrt{2}}cos\left(\frac{\pi}{5}\right)|s_0\rangle|a_1\rangle|s_3\rangle|r\rangle + \frac{1}{\sqrt{2}}sin\left(\frac{\pi}{5}\right)|s_0\rangle|a_1\rangle|s_3\rangle|r\rangle \\
&= |\psi_1\rangle
\end{aligned} \tag{5.4.37}$$

At this point, prior to the amplification step governed by the modified exponential search algorithm (algorithm 9), if we measure $|\psi_1\rangle$ multiple times, assuming that we can prepare exactly the same state as before, we obtain a distribution over the actions and the corresponding expected reward as shown in Figure 5.4.6:

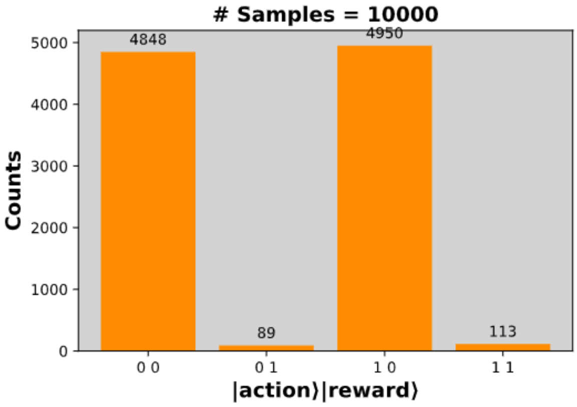


Figure 5.4.6: Statistics generated by measuring the state $|\psi_1\rangle$, taking 10000 samples

.

As we can see in the distribution,action a_1 as the highest rewarded action, which is in fact the optimal action to take. The reward qubit however, is in the state $|0\rangle$, meaning that it's not the true reward value. Although, without the amplification step one could retrieve the optimal action, in general, such is not the case, reflecting the importance of the amplitude amplification step. Calling the exponential search algorithm on state $|\psi_1\rangle$, we will amplify the amplitude of the states that have reward qubit $|1\rangle$. Repeating the process a sufficient number of times we will be able to retrieve the optimal action from the distribution generated as in Figure 5.4.7:

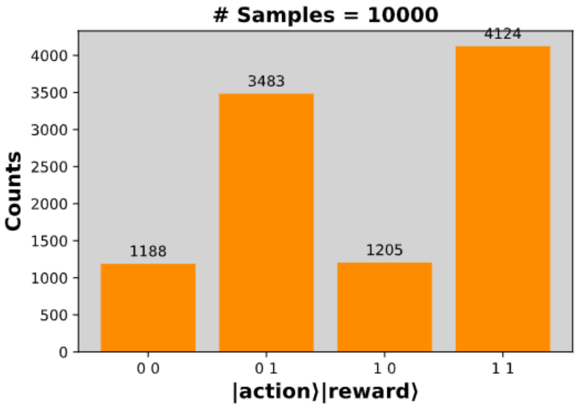


Figure 5.4.7: Distribution resulted from the application of the exponential search algorithm, gererating experimentally m=10000 samples

At this point executing argmax$_a$, will retireve the optimal action, that is action a_1. In general, we need to apply the algorithm to sequential decision making problems, MDP's with both more states and a larger transition kernel to study its performance. Actually, the above example deals only with a problem with an immediate reward, and a single decision. In general, we want to make optimal decision taking into account a delayed reward in mind. For that we are going to extend the above example with a couple of extra states and possible transitions, as in Figure 5.4.8.

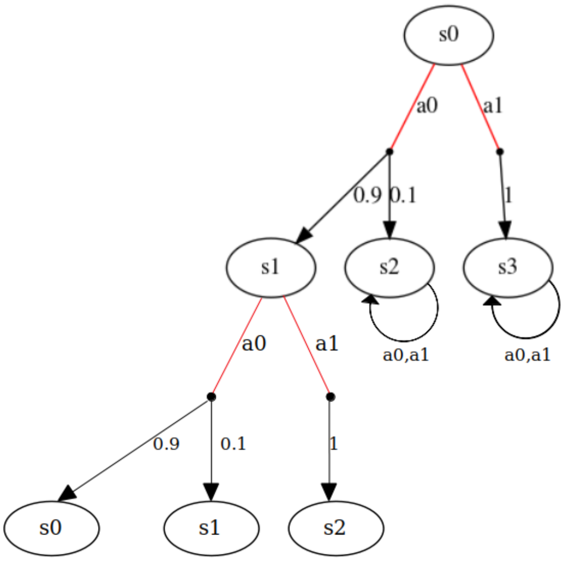


Figure 5.4.8: Extended probabilistic model

with a new reward function:

$$R(s) = \begin{cases} 50 & \text{if } s = s0 \\ 10 & \text{if } s = s1 \\ 50 & \text{if } s = s2 \\ 20 & \text{if } s = s3 \end{cases} \tag{5.4.38}$$

In this case we don't need to repeat every single step all over again, starting from the initial state of the agent in the beginning, because we already know the superposition state that the agent is in after the first parallel interaction with the quantum environment:

$$|\psi_0'\rangle = \sqrt{\frac{9}{20}}|s_0\rangle|a_0\rangle|s_1\rangle|r\rangle + \sqrt{\frac{1}{20}}|s_0\rangle|a_0\rangle|s_2\rangle|r\rangle + \frac{1}{\sqrt{2}}|s_0\rangle|a_1\rangle|s_3\rangle|r\rangle$$

The difference is that the reward to be applied now has a new normalization factor, due to the horizon being different. Therefore for horizon $h = 2$ we have

$$\eta = \frac{\pi}{2}\frac{\gamma - 1}{\gamma^h - 1} \approx \frac{\pi}{4}$$

Interacting with the reward oracle, the state of the quantum agent at the end of the first parallel interaction becomes

$$\begin{aligned} R_s|\psi_0'\rangle &= \sqrt{\frac{9}{20}}|s_0\rangle|a_0\rangle R_s\left[|s_1\rangle|r\rangle\right] + \sqrt{\frac{1}{20}}|s_0\rangle|a_0\rangle R_s\left[|s_2\rangle|r\rangle\right] + \frac{1}{\sqrt{2}}|s_0\rangle|a_1\rangle R_s\left[|s_3\rangle|r\rangle\right] \\ &= \sqrt{\frac{9}{20}}cos\left(\frac{\pi}{20}\right)|s_0\rangle|a_0\rangle|s_1\rangle|r\rangle + \sqrt{\frac{9}{20}}sin\left(\frac{\pi}{20}\right)|s_0\rangle|a_0\rangle|s_1\rangle|r\rangle + \\ &+ \sqrt{\frac{1}{20}}cos\left(\frac{\pi}{4}\right)|s_0\rangle|a_0\rangle|s_2\rangle|r\rangle + \sqrt{\frac{1}{20}}sin\left(\frac{\pi}{4}\right)|s_0\rangle|a_0\rangle|s_2\rangle|r\rangle + \\ &+ \frac{1}{\sqrt{2}}cos\left(\frac{\pi}{10}\right)|s_0\rangle|a_1\rangle|s_3\rangle|r\rangle + \frac{1}{\sqrt{2}}sin\left(\frac{\pi}{10}\right)|s_0\rangle|a_1\rangle|s_3\rangle|r\rangle \\ &= |\psi_1\rangle \end{aligned} \tag{5.4.39}$$

To perform the second interaction with the quantum environment we need to extend the state $|\psi_1\rangle$ with the respective quantum registers for representing the next parallel interaction, which can be done by tensoring the quantum state and the new quantum registers:

$$|s'\rangle = |0\rangle^{\otimes log|S|} \quad , \quad |a'\rangle = H^{\otimes log|A|}|0\rangle^{\otimes log|A|} = \frac{1}{\sqrt{2}}\left[|a_0'\rangle + |a_1'\rangle\right]$$

$$|\psi_1'\rangle = |\psi_1\rangle \otimes |a'\rangle \otimes |s'\rangle$$

For convenience, if we apply swap gates between the the reward quantum register and the new state and action registers, we will obtain the following quantum state, ready for the new parallel interaction, created by state transition oracle:

$$\begin{aligned}
\mathcal{T}|\psi_1'\rangle = &\sqrt{\frac{9}{20}}cos\left(\frac{\pi}{20}\right)|s_0\rangle|a_0\rangle\mathcal{T}\left[|s_1\rangle|a'\rangle|s'\rangle\right]|r\rangle + \sqrt{\frac{9}{20}}sin\left(\frac{\pi}{20}\right)|s_0\rangle|a_0\rangle\mathcal{T}\left[|s_1\rangle|a'\rangle|s'\rangle\right]|r\rangle + \\
&+ \sqrt{\frac{1}{20}}cos\left(\frac{\pi}{4}\right)|s_0\rangle|a_0\rangle\mathcal{T}\left[|s_2\rangle|a'\rangle|s'\rangle\right]|r\rangle + \sqrt{\frac{1}{20}}sin\left(\frac{\pi}{4}\right)|s_0\rangle|a_0\rangle\mathcal{T}\left[|s_2\rangle|a'\rangle|s'\rangle\right]|r\rangle + \\
&+ \frac{1}{\sqrt{2}}cos\left(\frac{\pi}{10}\right)|s_0\rangle|a_1\rangle\mathcal{T}\left[|s_3\rangle|a'\rangle|s'\rangle\right]|r\rangle + \frac{1}{\sqrt{2}}sin\left(\frac{\pi}{10}\right)|s_0\rangle|a_1\rangle\mathcal{T}\left[|s_3\rangle|a'\rangle|s'\rangle\right]|r\rangle \\
&= |\psi_1'\rangle
\end{aligned}$$

$$\begin{aligned}
|\psi_1'\rangle = &\frac{9}{20}cos\left(\frac{\pi}{20}\right)|s_0\rangle|a_0\rangle|s_1\rangle|a_0'\rangle|s_0\rangle|r\rangle + \sqrt{\frac{9}{40}}cos\left(\frac{\pi}{20}\right)|s_0\rangle|a_0\rangle|s_1\rangle|a_0'\rangle|s_1\rangle|r\rangle + \\
&+ \sqrt{\frac{9}{20}}cos\left(\frac{\pi}{20}\right)|s_0\rangle|a_0\rangle|s_1\rangle|a_1'\rangle|s_1\rangle|r\rangle + \\
&+ \frac{9}{20}sin\left(\frac{\pi}{20}\right)|s_0\rangle|a_0\rangle|s_1\rangle|a_0'\rangle|s_0\rangle|r\rangle + \sqrt{\frac{9}{40}}sin\left(\frac{\pi}{20}\right)|s_0\rangle|a_0\rangle|s_1\rangle|a_0'\rangle|s_1\rangle|r\rangle + \\
&+ \sqrt{\frac{9}{20}}sin\left(\frac{\pi}{20}\right)|s_0\rangle|a_0\rangle|s_1\rangle|a_1'\rangle|s_1\rangle|r\rangle + \\
&+ \sqrt{\frac{1}{20}}cos\left(\frac{\pi}{4}\right)|s_0\rangle|a_0\rangle|s_2\rangle|a_0'\rangle|s_2\rangle|r\rangle + \sqrt{\frac{1}{20}}cos\left(\frac{\pi}{4}\right)|s_0\rangle|a_0\rangle|s_2\rangle|a_1'\rangle|s_2\rangle|r\rangle + \\
&+ \sqrt{\frac{1}{20}}sin\left(\frac{\pi}{4}\right)|s_0\rangle|a_0\rangle|s_2\rangle|a_0'\rangle|s_2\rangle|r\rangle + \sqrt{\frac{1}{20}}sin\left(\frac{\pi}{4}\right)|s_0\rangle|a_0\rangle|s_2\rangle|a_1'\rangle|s_2\rangle|r\rangle + \\
&+ \frac{1}{\sqrt{2}}cos\left(\frac{\pi}{10}\right)|s_0\rangle|a_1\rangle|s_3\rangle|a_0'\rangle|s_3\rangle|r\rangle + \frac{1}{\sqrt{2}}cos\left(\frac{\pi}{10}\right)|s_0\rangle|a_1\rangle|s_3\rangle|a_1'\rangle|s_3\rangle|r\rangle + \\
&+ \frac{1}{\sqrt{2}}sin\left(\frac{\pi}{10}\right)|s_0\rangle|a_1\rangle|s_3\rangle|a_0'\rangle|s_3\rangle|r\rangle + \frac{1}{\sqrt{2}}sin\left(\frac{\pi}{10}\right)|s_0\rangle|a_1\rangle|s_3\rangle|a_1'\rangle|s_3\rangle|r\rangle
\end{aligned}$$

Applying the reward oracle to $|\psi_1'\rangle$, we get the superposition state at the end of the second parallel interaction, with the new discounted reward added to the to the amplitude of each superposition term corresponding to the reward gained at every transition, discounted by the factor γ.

$$\begin{aligned}R_s|\psi_1'\rangle = {} & \frac{9}{20}cos\left(\frac{\pi}{20}+\gamma\frac{\pi}{4}\right)|s_0\rangle|a_0\rangle|s_1\rangle|a_0'\rangle|s_0\rangle|r\rangle+\sqrt{\frac{9}{40}}cos\left(\frac{\pi}{20}+\gamma\frac{\pi}{20}\right)|s_0\rangle|a_0\rangle|s_1\rangle|a_0'\rangle|s_1\rangle|r\rangle+\\
&+\sqrt{\frac{9}{20}}cos\left(\frac{\pi}{20}+\gamma\frac{\pi}{20}\right)|s_0\rangle|a_0\rangle|s_1\rangle|a_1'\rangle|s_1\rangle|r\rangle+\\
&+\frac{9}{20}sin\left(\frac{\pi}{20}+\gamma\frac{\pi}{4}\right)|s_0\rangle|a_0\rangle|s_1\rangle|a_0'\rangle|s_0\rangle|r\rangle+\sqrt{\frac{9}{40}}sin\left(\frac{\pi}{20}+\gamma\frac{\pi}{20}\right)|s_0\rangle|a_0\rangle|s_1\rangle|a_0'\rangle|s_1\rangle|r\rangle+\\
&+\sqrt{\frac{9}{20}}sin\left(\frac{\pi}{20}+\gamma\frac{\pi}{20}\right)|s_0\rangle|a_0\rangle|s_1\rangle|a_1'\rangle|s_1\rangle|r\rangle+\\
&+\sqrt{\frac{1}{20}}cos\left(\frac{\pi}{4}+\gamma\frac{\pi}{4}\right)|s_0\rangle|a_0\rangle|s_2\rangle|a_0'\rangle|s_2\rangle|r\rangle+\sqrt{\frac{1}{20}}cos\left(\frac{\pi}{4}+\gamma\frac{\pi}{4}\right)|s_0\rangle|a_0\rangle|s_2\rangle|a_1'\rangle|s_2\rangle|r\rangle+\\
&+\sqrt{\frac{1}{20}}sin\left(\frac{\pi}{4}+\gamma\frac{\pi}{4}\right)|s_0\rangle|a_0\rangle|s_2\rangle|a_0'\rangle|s_2\rangle|r\rangle+\sqrt{\frac{1}{20}}sin\left(\frac{\pi}{4}+\gamma\frac{\pi}{4}\right)|s_0\rangle|a_0\rangle|s_2\rangle|a_1'\rangle|s_2\rangle|r\rangle+\\
&+\frac{1}{\sqrt{2}}cos\left(\frac{\pi}{10}+\gamma\frac{\pi}{10}\right)|s_0\rangle|a_1\rangle|s_3\rangle|a_0'\rangle|s_3\rangle|r\rangle+\frac{1}{\sqrt{2}}cos\left(\frac{\pi}{10}+\gamma\frac{\pi}{10}\right)|s_0\rangle|a_1\rangle|s_3\rangle|a_1'\rangle|s_3\rangle|r\rangle+\\
&+\frac{1}{\sqrt{2}}sin\left(\frac{\pi}{10}+\gamma\frac{\pi}{10}\right)|s_0\rangle|a_1\rangle|s_3\rangle|a_0'\rangle|s_3\rangle|r\rangle+\frac{1}{\sqrt{2}}sin\left(\frac{\pi}{10}+\gamma\frac{\pi}{10}\right)|s_0\rangle|a_1\rangle|s_3\rangle|a_1'\rangle|s_3\rangle|r\rangle\\
&=|\psi_2\rangle\end{aligned}$$

If we measure the state, we will run into the same problem as before unable to get the optimal action, which in this case is action a_0.

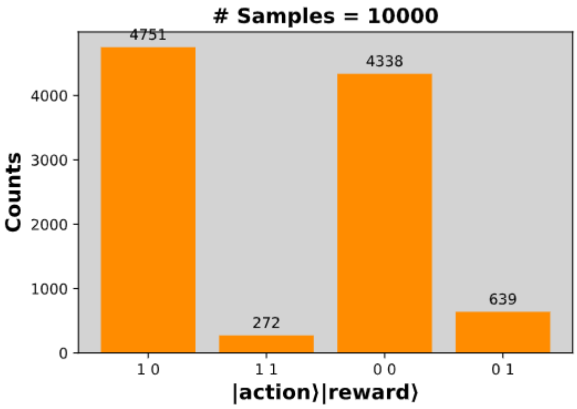


Figure 5.4.9: Statistics generated by measuring the state $|\psi_1\rangle$, taking 10000 samples

However, if we subsequently perform now the exponential search over state $|\psi_1\rangle$ we reach a distribution from where we can extract the optimal action as shown in Figure 5.4.10

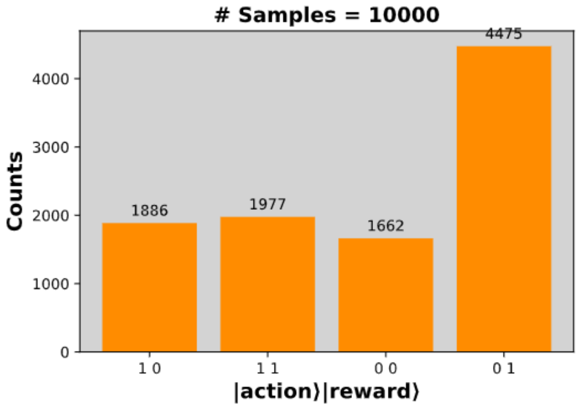


Figure 5.4.10: Distribution resulted from the application of the exponential search algorithm, gererating experimentally m=10000 samples

As we can see, the algorithm is able to deal with both a larger state space MDP and horizon problems, as expected. For implementations, and a quantum environment that one can play implemented using the IBM Qiskit Platform, we refer the reader to https://github.com/andre-sequeira10/Quantum-Reinforcement-Learning. Now, there is a crucial point that we have not yet explained , being how many samples do we need in order to have an ϵ-approximation of the optimal action? Is it feasible, in the first place? We are going to deal with this question in the next section, through the analysis of the complexity of the algorithm.

5.5 COMPLEXITY ANALYSIS

As explained in the last chapter, when we analyze the complexity of machine learning algorithms, we consider two distinct metrics, namely *computational complexity*, the number of operations needed, and *sample complexity*, the number of examples that we need, in order to compute an approximation of the target function. While improvements in the computational complexity of quantum machine learning algorithms have been reported, sample complexity has received less attention. The quantum algorithm developed in section 5.3 computes the optimal sequence of actions that the agent has to take to maximize the reward, for a given horizon h. The algorithm works in a sense as a deterministic pathfinding algorithm. Therefore, there is no need to study the sample complexity given that the problem is purely a computational one. On the other hand, the algorithms of section 5.1 and section 5.4 are stochastic, so it makes sense to analyze their sample complexity. Given that the sample complexity of both algorithms is based on the same technique, we are going to analyze first the complexity of the deterministic algorithm, and then the stochastic cases.

5.5.1 *Quantum algorithm for deterministic MDP*

The goal of the algorithm is not to define the optimal action to take in every state of the MDP, reaching the optimal policy, but rather reaching the actions for the states that the agent goes through in order to maximize the collected reward. The deterministic MDP is represented as a graph, but the algorithm can be interpreted as producing a tree from which one can derive a path while exploring the state space. The state-space will be explored dependently on the action space of the MDP and the horizon given as input to the algorithm. The much farther away we look, the more confident we are about the actions we need to take.

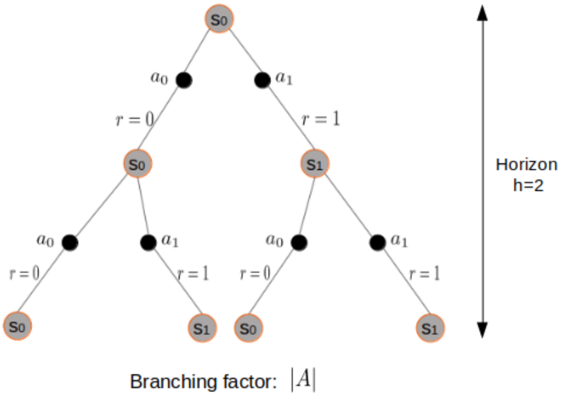


Figure 5.5.1: Two-step lookahead tree generated in superposition, with branching factor equivalent to the action space

Figure 5.5.1 represents the tree computed by the parallel interaction of the agent with the environment, for horizon $h = 2$, in the MDP that we already have seen as an example in Figure 5.3.8. For this example, we see that every state has the same branching factor, equivalent to the action space, given that at every state, the agent tries every possible action. This is not the case for MDP's where the admissible actions differ from state to state. The worst-case scenario occurs when the branching factor is the action space itself, $|A|$. However, we know that the computed tree will have depth h, thereby, we will have $|A|^h$ superposition terms, corresponding to every possible path until the horizon is met. The superposition containing every path is uniform, corresponding to the initial state of the quantum maximum finding subroutine, which is responsible for collapsing the superposition in the sequence of actions that leads to the maximum collected reward. We know that the subroutine in a search space with N elements, returns the maximum of them in time $\mathcal{O}(\sqrt{N})$, and we also know that in this case, the search space is $N = |A|^h$. Therefore we conclude that the worst-case computational complexity of the algorithm is:

$$\mathcal{O}(\sqrt{|A|^h}) \tag{5.5.1}$$

Of course, the algorithm is exponential in the horizon. However, note that we are not using it to do planning in the entire state-space, but just trying to make optimal decisions in a quantum simulated environment. This approach leads to an algorithm that is **independent of the number of states of MDP**, which is a crucial point to deal with large state space MDP's, providing, of course, that the oracles responsible for the quantum representation of a classical MDP, can be implemented efficiently. In that case, the complexity of the algorithm is entirely dominated by the size of the search space.

Before going into the analysis of the stochastic algorithms discussed in the previous section, we want to consider a slightly different problem. Suppose that we have access to a black-box that prepares a single qubit in an arbitrary state. If we measure the qubit, we will retrieve one the possible basis states $\{|0\rangle, |1\rangle\}$ with unknown probability. Now, provided that we have an available *"reset operation"*, meaning that we can always reset the state of the qubit to the initial arbitrary state and that the black-box always prepares the same density matrix ρ, thus no error is associated to the state preparation. How can we estimate the probability of reading one of the basis states[2]?

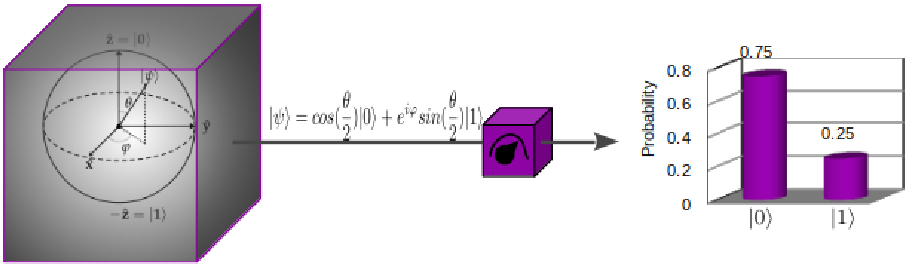


Figure 5.5.2: Black-box that prepares a single qubit system in an arbitrary state. By measuring this state successively, we generate a distribution over the basis states.

If we repeatedly measure the qubit provided by the black-box in the computational basis, we achieve a distribution over the basis states. How many times do we need to measure the state in order to have a reliable prediction? When measuring the single qubit, we read one of two possible outcomes, thus, qubit measurement can be viewed as sampling from a *Bernoulli distribution*. Now there are several methods for estimating the probability, but, the black-box output qubit can be in arbitrary state. For example, if we sample the qubit 100 times and read the state $|1\rangle$ everytime, what does this tell us ? That the qubit is in a classical state, or that we have not sampled enough times ? Probabilities could be close to either 0

[2] At first sight one may think that we're talking about *quantum state tomography*, where we want to reconstruct a quantum state based on measurements, which is known to grow exponentially with the number of qubits [2]. However, we're only interested here in the diagonal elements of the density matrix and not the full coherence terms.

or 1, therefore, in general we want to consider methods that account for these cases. One method is the Wilson Interval [72].

$$p = \frac{1}{1+\frac{\delta^2}{S}}\left(\overline{p} + \frac{\delta^2}{2S}\right) \tag{5.5.2}$$

where S is the sample size, $\overline{p}$ is the average of samples and δ is the confidence value of the samples, a tabulated value, for example $\delta = 2.58$ corresponds to 99% confidence in their samples. The Wilson interval has an error associated given by:

$$\epsilon = \frac{\delta}{1+\frac{\delta^2}{S}}\sqrt{\frac{\overline{p}(1-\overline{p})}{S} + \frac{\delta^2}{4S^2}} \tag{5.5.3}$$

We say that the error is maximized when we have $\overline{p} = \frac{1}{2}$, corresponding to a perfect superposition state, achieving maximum uncertanty in the samples:

$$\epsilon = \sqrt{\delta^2 \frac{S+\delta^2}{4S^2}} \tag{5.5.4}$$

Furthermore, we can solve Equation 5.5.4 for the number of samples needed in order to get an ϵ-approximation to the probability:

$$S \leq \mathcal{O}\left(\frac{\delta^2}{8\epsilon^2}(\sqrt{16\epsilon^2+1}+1)\right) \tag{5.5.5}$$

Now that we have seen how to estimate the probability of reading a basis when sampling from a single qubit, we're ready to analyse the complexity of the algorithms described in the previous section.

5.5.2 *Quantum Bandits*

Before the amplification procedure, the QBandits algorithm has a behavior different from the original Grover's Algorithm, because the latter starts with a uniform superposition state and QBandits starts with an arbitrary initial amplitude distribution prepared by the stochastic reward function. In these cases, the optimal measurement time will depend on the average initial amplitudes and the number of marked states. It is known that the optimal measurement probability is $\mathcal{O}(\sqrt{N/r})$ [16] where N is the number of states in the search space and r the number of marked states. Based on this result, we can derive the complexity of the QBandits algorithm. In our case N will be the search space spanned by the superposition of the k arms, however, due to the y-rotation that prepares the stochastic reward distribution branching each arm into two sub-branches,

$$|\phi\rangle = R_y(\theta)|r\rangle = R_y(\theta)|0\rangle = cos(\theta)|0\rangle + sin(\theta)|1\rangle$$

the search space will be $N = 2k$. Assuming that the stochastic reward function is, indeed, stochastic for every arm and doesn't act deterministically in a single-arm, which makes a term vanish in the y-rotation for the stochastic reward, then every Grover iteration marks half of the states, states that have reward qubit equal to 1. Due to the probabilistic nature of the algorithm and because it is practically impossible to read the optimal arm with probability close to 1, sampling is needed to reach the optimal arm. The number of samples needed, m, can again be estimated using the Wilson Interval Equation 5.5.5. In the case of a binary arm, we have the exact setting as sampling from a single qubit, thus the number of samples needed are:

$$m \leq \mathcal{O}\left(\frac{\delta^2}{8\epsilon^2}(\sqrt{16\epsilon^2+1}+1)\right) \tag{5.5.6}$$

In the more general case, we will have k arms, represented by $log(k)$ qubits. Thereby we can approximate the multinomial distribution using the binomial for each of the qubits. Caution is needed because in this case, the confidence level of our samples is not δ but $\delta^{log(k)}$ as we're doing multiple confidence intervals from the same sample:

$$m \leq \mathcal{O}\left(\frac{\delta^2 log(k)}{8\epsilon^2}(\sqrt{16\epsilon^2+1}+1)\right) \tag{5.5.7}$$

Therefore, for a bandit problem with k arms, we can say that the complexity of the algorithm is $\mathcal{O}(m\sqrt{k})$:

$$\mathcal{O}(m\sqrt{k}) = \mathcal{O}\left(\frac{\delta^2 log(k)}{8\epsilon^2}(\sqrt{16\epsilon^2+1}+1)\sqrt{k}\right) \tag{5.5.8}$$

We can derive the number of samples needed for the Bernoulli bandit example, with k=2 arms. With an error of $\epsilon = 0.05$ and defining $\delta = 2.58$ (99% confidence in the drawn samples), we only need approximately 672 samples to reach the optimal arm.

5.5.3 *Quantum Sparse Sampling*

As referred in the section 5.4, the algorithm does not compute a policy per se, but rather near-optimal actions for every state of a given MDP. The algorithm is based on the notion of *Sparse Sampling*, meaning that the quantum algorithm computes a lookahead tree for a given predefined horizon, and based on that horizon computes the action the agent should take to maximize the expected reward. The algorithm starts by given a quantum construction of a MDP, an oracular version of a classical task environment, putting a quantum agent in a predefined initial state, and then interacts with the quantum environment in parallel, due to the superposition created around the action registers. Figure 5.5.3 represents the one-step lookahead tree created by this interaction, with both state transition, $\mathcal{T}$ and reward, R oracles.

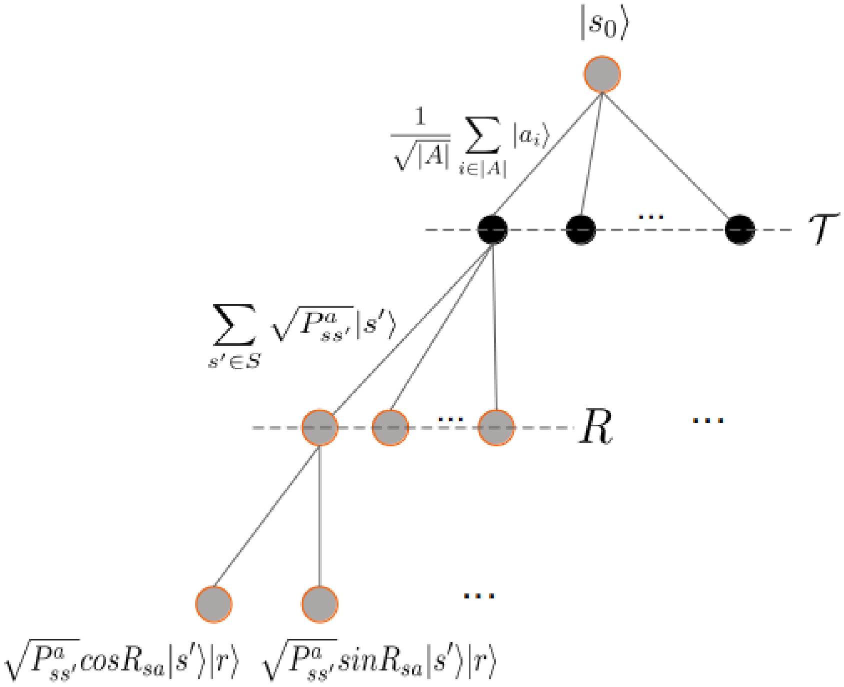

$|s_0\rangle$
$\frac{1}{\sqrt{|A|}}\sum_{i\in|A|}|a_i\rangle$
$\mathcal{T}$
$\sum_{s'\in S}\sqrt{P^a_{ss'}}|s'\rangle$
R
$\sqrt{P^a_{ss'}}cosR_{sa}|s'\rangle|r\rangle$
$\sqrt{P^a_{ss'}}sinR_{sa}|s'\rangle|r\rangle$

Figure 5.5.3: One step lookahead tree computed in superposition, created by the oracle calls of $\mathcal{T}$ and R

After the interaction is made for a given horizon, an exponential search is performed in order to amplify the amplitude of good states, i.e. the states that have the reward qubit equal to 1, because these hold the true value of the reward accumulated throughout the transition. We then measure the state, measuring an action. Assuming that we can always reconstruct the same state as before, we take m samples and for sufficiently large m, we can retrieve the optimal/near-optimal action to take at the initial state, as we have seen in section 5.4. The computational complexity of the algorithm will be essentially measured by the performance of the exponential search, assuming that the oracles are efficiently constructed. As we have seen back in algorithm 1, the quantum exponential search will find a good state in time complexity $\mathcal{O}(\sqrt{\frac{N}{r}})$, N standing for the search space and r for the number of marked states. We will take m samples of actions, making the running time of the quantum sparse sampling:

$$\mathcal{O}(m\sqrt{\frac{N}{r}}) \tag{5.5.9}$$

The search space will be dictated by the dynamics of the MDP created by the state transition oracle and the reward oracle. In the deterministic case, discussed in section 5.3, we know that the branching factor associated with each transition step will be the number of actions that the agent can take, and so, for a given tree depth we have at most $|A|^h$ superposition terms. In the stochastic case this is not true, given that the branching factor will be a function of the number of actions that the agent may take and the state transition probabilities as can be seen in Figure 5.5.3. These state transition probabilities will vary according to the problem, therefore we will bound the dependence on the state transition probabilities by a variable k, reflecting the maximum number of alternative outcomes for any given action, as given by the stochastic transition model. Said that we will have at most $k|A|^h$ superposition terms. Moreover, in the worst case, the exponential search algorithm will mark a single state, because this will increase the number of grover iterations that we need to perform, however, given that we want to be able to distinguish different actions, the worst case will be the case where the exponential search marks half of the states, case in which we will need more samples in order to reach the optimal action. Due to search space doubling by the reward oracle, given as a y-rotation as

$$R_y(\theta)|r\rangle = cos(\theta)|0\rangle + sin(\theta)|1\rangle$$

We reach the running time of the algorithm to be

$$\mathcal{O}(m\sqrt{k|A|^h}) \tag{5.5.10}$$

The important question now is defining the optimal number of samples required for an ϵ-approximation to the action to take in order to maximize the reward. For that we will use once again the *Wilson Score interval* to get an estimate of the number of samples that we need from an arbitrary qubit to get an ϵ-approximation to the probability of measuring the said qubit in any of the basis states $\{|0\rangle, |1\rangle\}$. Following Equation 5.5.5, we know that the number of samples necessary for a single qubit are:

$$m \leq \mathcal{O}\left(\frac{\delta^2}{8\epsilon^2}(\sqrt{16\epsilon^2+1}+1)\right)$$

However, we want now to find the action that leads to the highest expected reward, which typically is encoded in more than a single qubit, in general, in $log|A|$ qubits. Again, caution is needed because we are using a binomial distribution method for a multinomial distribution, thereby doing multiple confidence intervals from the same sample, thus we need to scale the confidence level of our samples, δ, to $\delta^{log|A|}$. Said that we can now bound the number of samples needed as:

$$m \leq \mathcal{O}\left(\frac{\delta^2 log|A|}{8\epsilon^2}(\sqrt{16\epsilon^2+1}+1)\right) \tag{5.5.11}$$

At this point, we conclude that the complexity of the quantum sparse sampling algorithm is:

$$\mathcal{O}(m\sqrt{k|A|^h}) = \mathcal{O}\left(\frac{\delta^2 log|A|}{8\epsilon^2}(\sqrt{16\epsilon^2+1}+1)\sqrt{k|A|^h}\right) \tag{5.5.12}$$

The term k will depend on the MDP itself, and we don't know for sure how to bound this "branching factor" without further assumptions on the MDP.

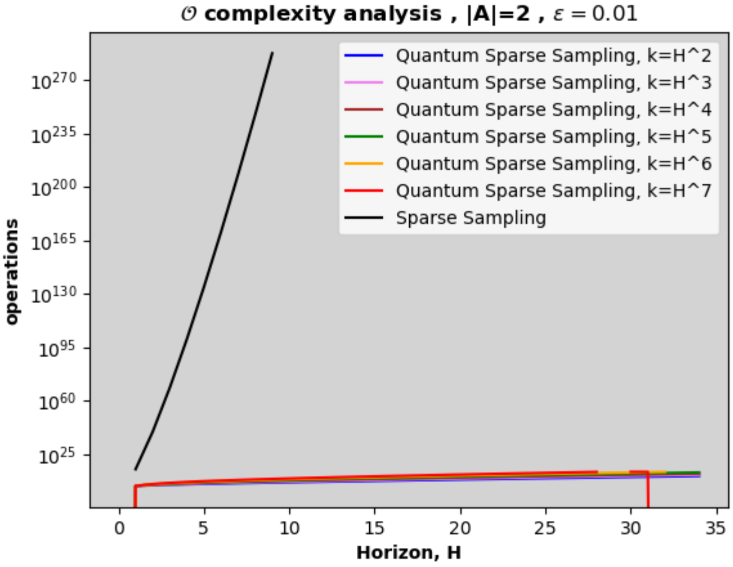


Figure 5.5.4: Complexity separation for a binary action MDP, varying the term k exponentially with the horizon in a logarithmic scale on the number of operations

In Figure 5.5.4, we see the complexity separation between the classical algorithm from section 3.6 and the quantum version, for the case of a binary action MDP, varying the k term exponentially with the horizon as well.

5.6 Read the fine print

We decided to call this section *"Read the fine print"* after Scott Aaronson's seminal paper [3]. This work is crucial not only for a quantum machine learning scientist but for the entire quantum computing community, as it encourages us to read the fine print on our quantum algorithms, meaning that speed ups obtained so far usually come with certain caveats that sometimes are not properly addressed. Here we want to follow this recommendation and go through the caveats behind the algorithms developed throughout this dissertation.

(a) Efficient State Preparation

In subsection 2.3.2 we described an algorithm that works for the encoding of classical information in the amplitude of quantum states. The same routine works as well for the encoding of probability distributions, also known as *qsamples*. Now, in our vision of oracular instances of classical environments, we need to use this algorithm in order to construct the quantum circuit associated to the interaction of the quantum agent with these oracles. In deterministic environments, we don't have much to say, but in stochastic environments, we actually need to prepare these *qsamples*

$$\mathcal{T} : |s\rangle \otimes |a\rangle \otimes |0\rangle^{\otimes n_s} \mapsto |s\rangle \otimes |a\rangle \otimes \sum_{s' \in S} \sqrt{p^a_{ss'}} |s'\rangle$$

Moreover, we want the state preparation to be efficiently realizable. It turns out that if $D = \{p_x\}_x$ is an efficiently samplable distribution, for the associative quantum state

$$|\psi_D\rangle = \sum_{x \in \{0,1\}^n} \sqrt{p_x} |x\rangle \tag{5.6.1}$$

is still unclear that can be realized by a polynomial-size quantum circuit in general. Therefore this could lead to distributions that are samplable but otherwise not qsamplable [69]. Moreover, notice that these oracles are not built based only on a classical distribution as in Equation 5.6.1, but also conditioned on the state and action registers, which makes this construction potentially harder, since we need to use multi-control Toffoli gates, which need to be decomposed into elementary gates. This can have a major impact in performance, given the exponential decomposition. That is to say that when analyzing the complexity of the algorithm, we **did not account for the complexity of both construction of the associated oracles as well their running time which can impact the runtime of the algorithm**.

(b) Error Correction

In the present NISQ (*Nearly Intermediate Scale Quantum devices*) era, not only the number of qubits necessary to apply the technology to large scale industrial problems is lacking, but also an even more important limiting factor is present, which is their engineering is a really hard problem with decoherence, as well as faulty gates, that affects the output of programs. Without fully fault-tolerant quantum computing, applications will be largely affected leaving almost of results to be theoretical only.

Now, in the stochastic algorithms developed in section 5.1 and section 5.4, we needed to sample the quantum state in order to reach the optimal action to take. The optimal number of samples that we need to take was based on the *Wilson Score Interval* that establishes a clear bound to assure a ϵ-approximation to the probability of reading a quantum bit in any of the basis states $\{|0\rangle, |1\rangle\}$. As we said in section 5.5, this method considers the sampling from a qubit equivalent as sampling from a *Bernoulli distribution*. The caveat here is the assumption that the device that recreates the quantum state of the qubit, can prepare always the same state, which is an idealized assumption. As mentioned before, current quantum devices have faulty gates as well as a short coherence time. Therefore in real NISQ devices, the output of measurements, at least for the time being, will have an error accumulated that may result in a misleading outcome.

6 CONCLUSION

6.1 CONCLUDING

Quantum Computing is a fascinating computing paradigm. The ability to construct devices that exploit the laws of quantum mechanics for information processing is game-changing potentially in ways that we can't even predict. The technology is still in the beginning but already proved fruitful in many research areas like cryptography, computing, sensing, and metrology. This dissertation routes towards a different vision, a vision of interconnection between quantum computing and artificial intelligence. There are already some results empowering this connection, however more in the supervised/unsupervised learning side, thus, closer to data science rather than to artificial intelligence per se. We believe that truly intelligent systems lie beyond simply data analysis, being the ultimate goal of AI the design of an artificial agent that thrives in unknown environments, having the capacity to tackle all problems. We believe that Reinforcement Learning brings us closer to that reality. We know that RL alone is not enough, an interplay between machine learning techniques and this learning by interaction, yet may not be sufficient because we clearly have not yet established what intelligence really is, let alone consciousness. However, the key point is that RL leaves us one step closer to achieve this. We know that a connection between RL and neuroscience, specifically by dopamine, a chemical involved in reward processing, suggests that this reward-based learning occurs in the brains of animals. The evidence of quantum structures in our brains lead the idea of this dissertation, the study of RL from a quantum computing perspective.

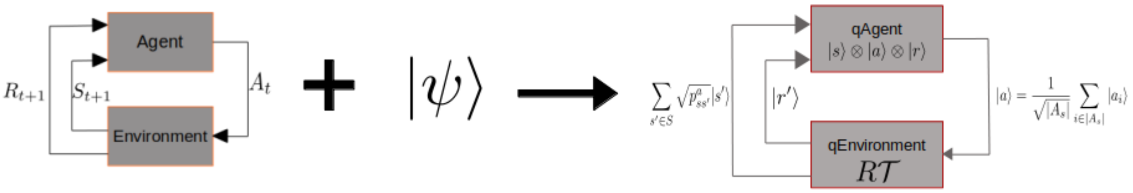

Agent
Environment
R_{t+1}
S_{t+1}
A_t
+
$|\psi\rangle$
qAgent
$|s\rangle \otimes |a\rangle \otimes |r\rangle$
qEnvironment
$R\mathcal{T}$
$\sum_{s' \in S} \sqrt{p^a_{ss'}} |s'\rangle$
$|r'\rangle$
$|a\rangle = \frac{1}{\sqrt{|A_s|}} \sum_{i \in |A_s|} |a_i\rangle$

Figure 6.1.1: The Quantum Agent-Environment paradigm. Interaction of quantum agent with its environment by performing a superposition of actions and the environment returns the agent in a superposition of new states with the respective rewards

This ignited the pursuit of ways of quantizing this agent-environment interaction, which leads to a quantum agent-environment paradigm realized by constructing oracular versions of classical environments as depicted in Figure 6.1.1. Different modes of expression were developed dependently on the nature of the environment itself. Both deterministic and stochastic environments enable a truly parallel interaction of a quantum agent with its environment. From this interaction, quantum algorithms were developed to address the problem of decision making within the RL paradigm. The goal of an RL agent, based on the agent-environment paradigm, is the maximization of the expected reward obtained, and with this, the achievement of an optimal policy such that the agent is capable of recognizing each state of its environment and map it directly into an action. This notion leads to various types of classical algorithms to solve the problem for the optimal policy as discussed in chapter 3. However, the drawback is the scaling of the complexity with the increase of the size of the problem, i.e. the dependence on the number of states present in the environment. In the quantum setting, even with the use of superposition, we suffer from the same problem. Therefore another way of converting the quantum agent-environment interaction into proper decision-making algorithms was suggested using a different paradigm, known as *sparse sampling*, presented in section 3.6 which says that given the state the agent currently is, sampling a lookahead tree from there, is sufficient to return a near-optimal action at that particular state. This type of algorithm achieves near-optimal decision making independently of the number of states. In the quantum setting, we followed the same idea. We start from a given state of the environment and compute a lookahead tree, being this tree computed in superposition. Up to a given horizon, we can use quantum searching techniques like maximum finding (section 2.6) for deterministic environments and a variant of the quantum exponential search algorithm (algorithm 1) for stochastic environments. In these cases, due to the probabilistic nature of the environment, we need to repeat the execution of the algorithm multiple times in order to achieve a distribution from where to infer the optimal action to take (section 5.4). The work done made possible the construction of more general oracular instances compared to previous work and the construction of algorithms that benefit from a quantum speedup relatively to their classical counterparts.

Besides the open questions enumerated in section 5.6, where we pointed out to the need of efficient quantum state preparation as well as fault-tolerant quantum computation in order these algorithms can be applied in large scale problems, we want now to guide the reader into a brief analysis of generalizations of the algorithms proposed. We will mention what is still missing and highlight some of the ideas that came along throughout this dissertation, such that it can hopefully ignite the spirit of the reader into further research.

6.2 FUTURE WORK

We would like to start this brief discussion with the analysis of the encoding mechanisms used in the oraculization of task environments. Recall that for deterministic environments, the reward oracle behaves as:

$$R_s : |s\rangle \otimes |r\rangle \mapsto |s\rangle \otimes |r \oplus \mathcal{R}_s\rangle$$

For a given state the agent is in, a reward found is via basis encoding, added to the respective reward quantum register. How many qubits do we need in order to have the capacity to add rewards until a given horizon is met? For that we used a reference value R_{max}, that represents, for a given horizon, the maximum reward that the agent is able to collect. In stochastic environments, we used angle encoding to accumulate the rewards the agent may receive throughout a sequence of actions:

$$R_s : |s\rangle \otimes |r\rangle \mapsto |s\rangle \otimes R_y(\eta\gamma^t \frac{R_s}{R_{max}})|r\rangle$$

For these interactions we needed to normalize the rewards collected such that for a given horizon, the total reward accumulated is not more than rotation equivalent to $\frac{\pi}{2}$, otherwise we would not be able to retrieve the action that trully leads to the maximum expected reward. For the normalization we again needed to refer to a threshold value R_{max} as in the deterministic setting. This is to say that both oracles made use of a predefined value in order to properly encode the rewards. Perhaps this is not an actual problem due to the fact that this information can be regarded as part of the oracle itself. However it would be interesting to remove this dependence on R_{max}, because this would make it possible to apply these oracles to different problems without the need to actually reconstruct the oracle as the problem changes.

On the same line of thought, it may not be noticed, but both deterministic and stochastic encodings lack the ability to express negative rewards, i.e. lacking the ability of true negative feedback which is important for the agent to learn. In the deterministic setting it may actually be done, if one dedicate enough time in adapting binary representation of signed

numbers to the quantum setting, extending quantum arithmetic to be able to use signed integers representations such as two's complement.
In the stochastic setting, we may be tempted to use negative rotations in order to represent negative feedback, however more thought must be put to actually guarantee that it behaves the way we want, because negative rotations will enable negative amplitude terms as well as positive amplitude terms and it is not clear that this balancing between the averages of amplitudes will result in a favorable distribution.

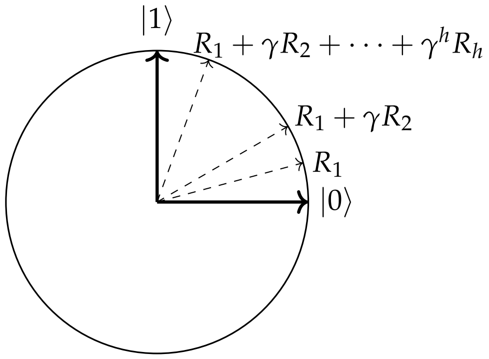


Figure 6.2.1: Reward qubit evolution with the addition of the rewards

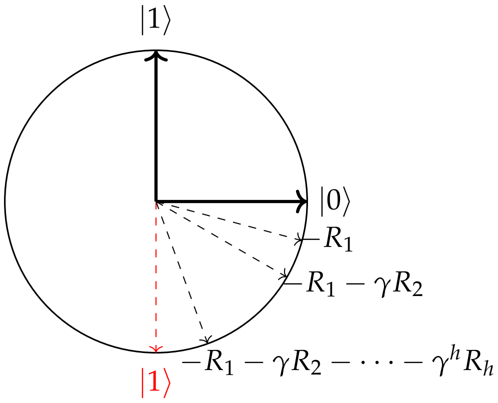


Figure 6.2.2: Reward qubit evolution with the addition of the rewards

We defined the reward accumulation using y-rotations on a qubit initialized in the ground state. As the agent follows a path derived from positive rewarded actions, the reward qubit will be closer to state $|1\rangle$ as in Figure 6.2.1. However if the agent follows a negative rewarded sequence of actions, the reward qubit will be closer to state $|1\rangle$ as well, but with inverted amplitude an in Figure 6.2.2:

$$\begin{aligned} R_y(-\theta)|0\rangle &= cos(-\theta)|0\rangle + sin(-\theta)|1\rangle \\ &= cos(\theta)|0\rangle - sin(\theta)|1\rangle \quad , \quad \forall\theta \in [-\frac{\pi}{2}, 0] \end{aligned} \tag{6.2.1}$$

When marking the states with reward qubit equal to $|1\rangle$ in order to amplify the amplitude of good states, it is not clear this will result in the desired outcome, because we may get trajectories with full positive rewards collected as well as full negative reward collected and this variation makes the amplification procedure rather uncertain.

From a more algorithmic complexity-oriented perspective, consider the following reasoning: Both deterministic and stochastic algorithms developed in this dissertation allow for the encoding of environments where the set of actions depend on the agent state, rather than on a set of actions, all of them, admissable to any state. For purposes of simplicity in the analysis of the algorithms, this was put aside, considering all actions admissible to any state, therefore creating a uniform superposition over the possible actions. However, it could be done by applying $\mathcal{A}$ before the interaction with the state transition oracle, to create a superposition over the admissable actions at the agent's state.

$$\mathcal{A} : |s\rangle \otimes |a\rangle \mapsto |s\rangle \otimes \frac{1}{\sqrt{|A_s|}} \sum_{i \in |A_s|} |a_i\rangle$$

In either case, a more important question comes to mind: We have always created a uniform superposition over the set of actions, either $|A|$ or $|A_s|$, expanding the computed tree in an *uninformed way*. With this we are expanding the search space regardless of the importance of states, wasting computation time on states that are not worth exploring. This calls for *informed search* techniques, i.e. the development of some kind of heuristic to help in *pruning* worthless branches of the tree, and focus the search in more promising nodes, somewhat reminiscent to methods like *Thompson Sampling* or *Monte-Carlo Tree Search*.

As a final remark, we want to point out to the importance of memory consumption in Reinforcement Learning applications, which translates directly to the quantum setting. As we know by now, the proposed algorithms have space complexity (the number of qubits used) that depends logarithmically on the number of states as well as on the number of actions of a given environment, multiplied by the horizon, the further we look into the future to make a decision. This dependence becomes heavier with the increase of the size of the problem, especially in the NISQ era. For that reason, we may still be limited in what concerns the application of the technology to larger-scale problems. An interesting approach for the years to come, resides in a different reinforcement learning paradigm, that we don't have covered here in this dissertation, called *Function Approximation*[63]. This paradigm has recently achieved great results, using Neural Networks as natural function approximators, and for that reason, it's been used for computing value functions, action-value functions beyond the tabular paradigm as a method that rather parametrizes values instead of saving a table of values. In the quantum setting, *variational quantum circuits* [57] emerge as a promise in quantum neural network research, with outstanding benefits from a linear layer construction, and therefore it would be interesting to know if we could develop new algorithms for reinforcement learning based on the variational principle as well.

BIBLIOGRAPHY

[1] *Learn Quantum Computation Using Qiskit*. 2020. URL http://community.qiskit.org/textbook.

[2] Scott Aaronson. The learnability of quantum states. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 2007. ISSN 14712946. doi: 10.1098/rspa.2007.0113.

[3] Scott Aaronson. Read the fine print, 2015. ISSN 17452481.

[4] Scott Aaronson and Patrick Rall. Quantum Approximate Counting, Simplified. In *Symposium on Simplicity in Algorithms*. 2020. doi: 10.1137/1.9781611976014.5.

[5] Ashish Ahuja and Sanjiv Kapoor. A Quantum Algorithm for finding the Maximum. (July):1–2, 1999. URL http://arxiv.org/abs/quant-ph/9911082.

[6] Esma Aïmeur, Gilles Brassard, and Sébastien Gambs. Machine learning in a quantum world. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2006. ISBN 3540346287. doi: 10.1007/11766247_37.

[7] Esma Aïmeur, Gilles Brassard, and Sébastien Gambs. Quantum speed-up for unsupervised learning. *Machine Learning*, 2013. ISSN 08856125. doi: 10.1007/s10994-012-5316-5.

[8] Andris Ambainis, András Gilyén, Stacey Jeffery, and Martins Kokainis. Quadratic speedup for finding marked vertices by quantum walks. 2020. ISBN 9781450369794. doi: 10.1145/3357713.3384252.

[9] Srinivasan Arunachalam and Ronald De Wolf. Optimal quantum sample complexity of learning algorithms. *Journal of Machine Learning Research*, 2018. ISSN 15337928.

[10] Jennifer Barry, Daniel T. Barry, and Scott Aaronson. Quantum partially observable Markov decision processes. *Physical Review A - Atomic, Molecular, and Optical Physics*, 2014. ISSN 10941622. doi: 10.1103/PhysRevA.90.032311.

[11] Jenny Barry. 6 . 845 Final Project : Quantum POMDPs Partially Observable Markov Decision Processes (POMDPs). pages 1–23, 2012.

[12] Richard Bellman. Dynamic programming. *Science*, 1966. ISSN 00368075. doi: 10.1126/science.153.3731.34.

[13] Jon Louis Bentley and Andrew Chi Chih Yao. An almost optimal algorithm for unbounded searching. *Information Processing Letters*, 1976. ISSN 00200190. doi: 10.1016/0020-0190(76)90071-5.

[14] Jacob Biamonte. Lectures on quantum tensor networks, 2019.

[15] Jacob Biamonte, Peter Wittek, Nicola Pancotti, Patrick Rebentrost, Nathan Wiebe, and Seth Lloyd. Quantum machine learning, 2017. ISSN 14764687.

[16] Eli Biham, Ofer Biham, David Biron, Markus Grassl, and Daniel A. Lidar. Grover's quantum search algorithm for an arbitrary initial amplitude distribution. *Physical Review A - Atomic, Molecular, and Optical Physics*, 1999. ISSN 10941622. doi: 10.1103/PhysRevA.60.2742.

[17] Eli Biham, Ofer Biham, David Biron, Markus Grassl, Daniel A. Lidar, and Daniel Shapira. Analysis of generalized Grover quantum search algorithms using recursion equations. *Physical Review A - Atomic, Molecular, and Optical Physics*, 2001. ISSN 10502947. doi: 10.1103/PhysRevA.63.012310.

[18] Alessandro Bisio, Giulio Chiribella, Giacomo Mauro D'Ariano, Stefano Facchini, and Paolo Perinotti. Optimal quantum learning of a unitary transformation. *Physical Review A*, 81(3), Mar 2010. ISSN 1094-1622. doi: 10.1103/physreva.81.032324. URL http://dx.doi.org/10.1103/PhysRevA.81.032324.

[19] Michel Boyer, Gilles Brassard, Peter HØyer, and Alain Tapp. Tight bounds on quantum searching. *Fortschritte der Physik*, 46(4-5):493–505, 1998. ISSN 00158208. doi: 10.1002/(SICI)1521-3978(199806)46:4/5⟨493::AID-PROP493⟩3.0.CO;2-P.

[20] Gilles Brassard, Peter Høyer, Michele Mosca, and Alain Tapp. Quantum amplitude amplification and estimation. 2002. doi: 10.1090/conm/305/05215.

[21] Hans J. Briegel and Gemma De Las Cuevas. Projective simulation for artificial intelligence. *Scientific Reports*, 2012. ISSN 20452322. doi: 10.1038/srep00400.

[22] Nader H. Bshouty and Jeffrey C. Jackson. Learning dnf over the uniform distribution using a quantum example oracle. *SIAM J. Comput.*, 28(3):1136–1153, February 1999. ISSN 0097-5397. doi: 10.1137/S0097539795293123. URL https://doi.org/10.1137/S0097539795293123.

[23] Murray Campbell, A. Joseph Hoane, and Feng Hsiung Hsu. Deep Blue. *Artificial Intelligence*, 2002. ISSN 00043702. doi: 10.1016/S0004-3702(01)00129-1.

[24] Daniel Crawford, Anna Levit, Navid Ghadermarzy, Jaspreet S. Oberoi, and Pooya Ronagh. Reinforcement learning using quantum boltzmann machines. *Quantum Information and Computation*, 18(1-2):51–74, 2018. ISSN 15337146.

[25] Daoyi Dong, Chunlin Chen, Hanxiong Li, and Txyh Jong Tarn. Quantum reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 38(5): 1207–1220, 2008. ISSN 10834419. doi: 10.1109/TSMCB.2008.925743.

[26] Vedran Dunjko and Hans J. Briegel. Machine learning & artificial intelligence in the quantum domain: A review of recent progress. *Reports on Progress in Physics*, 2018. ISSN 00344885. doi: 10.1088/1361-6633/aab406.

[27] Vedran Dunjko, Jacob M. Taylor, and Hans J. Briegel. Quantum-Enhanced Machine Learning. *Physical Review Letters*, 117(13):1–19, 2016. ISSN 10797114. doi: 10.1103/PhysRevLett.117.130501.

[28] Vedran Dunjko, Yi-Kai Liu, Xingyao Wu, and Jacob M. Taylor. Exponential improvements for quantum-accessible reinforcement learning. 2017. URL http://arxiv.org/abs/1710.11160.

[29] Christoph Durr and Peter Hoyer. A quantum algorithm for finding the minimum, 1996.

[30] Richard P. Feynman. Simulating physics with computers. *International Journal of Theoretical Physics*, 1982. ISSN 00207748. doi: 10.1007/BF02650179.

[31] Edward Gillman, Dominic C. Rose, and Juan P. Garrahan. A Tensor Network Approach to Finite Markov Decision Processes. 2020. URL http://arxiv.org/abs/2002.05185.

[32] Vittorio Giovannetti, Seth Lloyd, and Lorenzo MacCone. Quantum random access memory. *Physical Review Letters*, 2008. ISSN 00319007. doi: 10.1103/PhysRevLett.100.160501.

[33] Lov K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the Annual ACM Symposium on Theory of Computing*, 1996. ISBN 0897917855. doi: 10.1145/237814.237866.

[34] Stuart Hameroff. Quantum computation in brain microtubules? The Penrose-Hameroff 'Orch OR' model of consciousness. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 1998. ISSN 1364503X. doi: 10.1098/rsta.1998.0254.

[35] Robert W. Harrison, John von Neumann, and Oskar Morgenstern. The Theory of Games and Economic Behavior. *Journal of Farm Economics*, 1945. ISSN 10711031. doi: 10.2307/1232672.

[36] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics*, 1968. ISSN 21682887. doi: 10.1109/TSSC.1968.300136.

[37] Geoffrey Hinton. Boltzmann Machines. In *Encyclopedia of Machine Learning and Data Mining*. 2017. doi: 10.1007/978-1-4899-7687-1_31.

[38] A. S. Holevo. Bounds for the Quantity of Information Transmitted by a Quantum Communication Channel. *Probl. Peredachi Inf.*, 1973.

[39] Aaron Courville Ian Goodfellow, Yoshua Bengio. Deep Learning Book. *Deep Learning*, 2015. ISSN 1437-7780. doi: 10.1016/B978-0-12-391420-0.09987-X.

[40] Sham Machandranath Kakade. On the Sample Complexity of Reinforcement Learning. In *International Conference on Machine Learning*, 2003. ISBN 978-1-4503-1285-1. doi: 10.1.1.164.7844.

[41] Michael Kearns, Yishay Mansour, and Andrew Y. Ng. A sparse sampling algorithm for near-optimal planning in large Markov decision processes. In *IJCAI International Joint Conference on Artificial Intelligence*, 1999.

[42] Richard E. Korf. Depth-first iterative-deepening: An optimal admissible tree search. *Artif. Intell.*, 27(1):97–109, September 1985. ISSN 0004-3702. doi: 10.1016/0004-3702(85)90084-0. URL https://doi.org/10.1016/0004-3702(85)90084-0.

[43] Richard E. Korf. Linear-space best-first search. *Artificial Intelligence*, 1993. ISSN 00043702. doi: 10.1016/0004-3702(93)90045-D.

[44] Steven M. LaValle. *Planning algorithms*. 2006. ISBN 9780511546877. doi: 10.1017/CBO9780511546877.

[45] Seth Lloyd, Masoud Mohseni, and Patrick Rebentrost. Quantum principal component analysis. *Nature Physics*, 10(9):631–633, Jul 2014. ISSN 1745-2481. doi: 10.1038/nphys3029. URL http://dx.doi.org/10.1038/nphys3029.

[46] Guang Hao Low, Theodore J. Yoder, and Isaac L. Chuang. Quantum inference on Bayesian networks. *Physical Review A - Atomic, Molecular, and Optical Physics*, 89(6), 2014. ISSN 10941622. doi: 10.1103/PhysRevA.89.062315.

[47] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015. ISSN 14764687. doi: 10.1038/nature14236.

[48] Edward F Moore. The shortest path through a maze. *Proceedings of an International Symposium on the Theory of Switching (Cambridge, Massachusetts, 2–5 April 1957). Cambridge: Harvard University Press*, 1959. ISSN 09593543, 00000000. doi: 10.1177/0959354304043641.

[49] Mikko Möttönen, Juha J. Vartiainen, Ville Bergholm, and Martti M. Salomaa. Transformation of quantum states using uniformly controlled rotations. *Quantum Information and Computation*, 2005. ISSN 15337146.

[50] Hendrik Poulsen Nautrup, Nicolas Delfosse, Vedran Dunjko, Hans J. Briegel, and Nicolai Friis. Optimizing quantum error correction codes with reinforcement learning. *Quantum*, 3:215, Dec 2019. ISSN 2521-327X. doi: 10.22331/q-2019-12-16-215. URL http://dx.doi.org/10.22331/q-2019-12-16-215.

[51] Allen Newell. The search for generality.

[52] Isaac L. Nielsen, Michael A. Chuang. Quantum Computation and Quantum Information. 1995.

[53] John Preskill. Quantum Computing in the NISQ era and beyond. *Quantum*, 2018. ISSN 2521-327X. doi: 10.22331/q-2018-08-06-79.

[54] Pooya Ronagh. Quantum Algorithms for Solving Dynamic Programming Problems. pages 1–30, 2019. URL http://arxiv.org/abs/1906.02229.

[55] Stuart Russel and Peter Norvig. *Artificial intelligence—a modern approach 3rd Edition*. 2012. ISBN 0136042597. doi: 10.1017/S0269888900007724.

[56] Stephan R. Sain and V. N. Vapnik. The Nature of Statistical Learning Theory. *Technometrics*, 1996. ISSN 00401706. doi: 10.2307/1271324.

[57] Maria Schuld, Ville Bergholm, Christian Gogolin, Josh Izaac, and Nathan Killoran. Evaluating analytic gradients on quantum hardware, 2018. ISSN 23318422.

[58] Rocco A. Servedio and Steven J. Gortler. Equivalences and Separations Between Quantum and Classical Learnability. *SIAM Journal on Computing*, 33(5):1067–1092, jan 2004. ISSN 0097-5397. doi: 10.1137/S0097539704412910. URL http://epubs.siam.org/doi/10.1137/S0097539704412910.

[59] Vivek V. Shende, Stephen S. Bullock, and Igor L. Markov. Synthesis of quantum-logic circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2006. ISSN 02780070. doi: 10.1109/TCAD.2005.855930.

[60] P.W. Shor. Algorithms for quantum computation: discrete logarithms and factoring. 2002. doi: 10.1109/sfcs.1994.365700.

[61] Daniel R. Simon. On the power of quantum computation. *SIAM Journal on Computing*, 26(5):1474–1483, 1997. doi: 10.1137/S0097539796298637. URL https://doi.org/10.1137/S0097539796298637.

[62] Alexander L. Strehl, Li Lihong, Eric Wiewiora, John Langford, and Michael L. Littman. PAC model-free reinforcement learning. In *ACM International Conference Proceeding Series*, 2006. ISBN 1595933832. doi: 10.1145/1143844.1143955.

[63] Sutton. *Reinforcement Learning Book*. 2018. ISBN 9780262039246.

[64] Mario Szegedy. Quantum speed-up of Markov Chain based algorithms. In *Proceedings - Annual IEEE Symposium on Foundations of Computer Science, FOCS*, 2004. doi: 10.1109/focs.2004.53.

[65] TARJAN R. Depth- first search and linear graph algorithms. 1971. doi: 10.1109/swat.1971.10.

[66] Luís Tarrataca and Andreas Wichert. Tree search and quantum computation. *Quantum Information Processing*, 2011. ISSN 15700755. doi: 10.1007/s11128-010-0212-z.

[67] Luís Tarrataca and Andreas Wichert. Quantum Iterative Deepening with an Application to the Halting Problem. *PLoS ONE*, 2013. ISSN 19326203. doi: 10.1371/journal.pone.0057309.

[68] G. Temple and P. A. M. Dirac. The Principles of Quantum Mechanics. *The Mathematical Gazette*, 1935. ISSN 00255572. doi: 10.2307/3606137.

[69] S. Vadhan and Kunal Talwar. Lecture notes for the 26th mcgill invitational workshop on computational complexity. 2016.

[70] L. G. Valiant. A theory of the learnable. In *Proceedings of the Annual ACM Symposium on Theory of Computing*, 1984. ISBN 0897911334. doi: 10.1145/800057.808710.

[71] Dan Ventura and Tony Martinez. Quantum associative memory. *Information sciences*, 2000. ISSN 00200255. doi: 10.1016/S0020-0255(99)00101-2.

[72] Edwin B. Wilson. Probable Inference, the Law of Succession, and Statistical Inference. *Journal of the American Statistical Association*, 1927. ISSN 1537274X. doi: 10.1080/01621459.1927.10502953.

[73] Shenggang Ying and Mingsheng Ying. Reachability analysis of quantum Markov decision processes. *Information and Computation*, 2018. ISSN 10902651. doi: 10.1016/j.ic.2018.09.001.