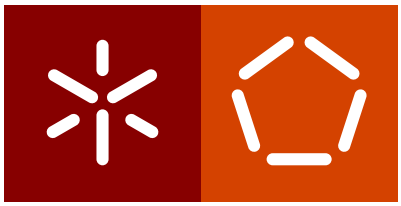
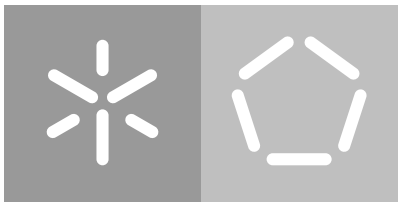


Universidade do Minho
Escola de Engenharia
Departamento de Informática

Renato Alberto Soares de Brito

Quantum Reinforcement Learning
A heuristic approach to solve deterministic MDPs

February 2022



Universidade do Minho
Escola de Engenharia
Departamento de Informática

Renato Alberto Soares de Brito

Quantum Reinforcement Learning
A heuristic approach to solve deterministic MDPs

Master Dissertation
Integrated Master's in Physics Engineering

Dissertation supervised by
Luís Paulo Santos

February 2022

DIREITOS DE AUTOR E CONDIÇÕES DE UTILIZAÇÃO DO TRABALHO POR TERCEIROS

ACKNOWLEDGEMENTS

This dissertation is the result of years of study and dedication, despite being a subject that I have not that much experience or practice I think it is a welcoming challenge.
First of all, I would like to thank, first to my parents, for providing me this opportunity and love; second, to all the scientific community, a special thanks to those who came before me because the knowledge they shared is the foundation of this work and without them, this thesis would not be possible. Thank you to all the professors I encountered throughout my academic life; to my supervisor, professor Luís Paulo Peixoto Santos, and my colleague, André Sequeira, for guiding me through this journey.

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.

ABSTRACT

This thesis works on Reinforcement Learning tree search and attempts to find the best possible sequence of actions the agent needs to execute to get the most reward while using less computational effort than by just applying a quantum maximum finding algorithm.
To achieve this we will use the property that makes it possible to limit our search space to the elements that were marked by the oracle in Grover's Algorithm, by marking a fourth of the search space and following it with a quantum maximum finding subroutine. From this, one of the marked elements is obtained and the information encoded in it is used to update a probabilistic distribution stored in a classical memory.
The goal is to encounter the minimum amount of iterations of this process and compare the results, i.e., percentage of success which is measured as the number of times the algorithm produces a solution (element with maximum reward) and the number of queries used - with a traditional quantum maximum finding procedure. If this is observed, it is also hypothesized that the algorithm could be used to observe a step further into the future compared to the traditional procedure, i.e., use the same or fewer queries to evaluate a larger number of sequences fruit of increasing the horizon of the episodes. The last hypothesis tests the depth of the circuits, more specifically the number of gates used. If the algorithm evaluates shallower circuits than the quantum maximum finding, the approach can be applied on the current quantum machines (NISQ) because the shallower circuits produces more error-proof measurements.
The results show that the proposed algorithm has no advantages compared to a traditional quantum maximum finding procedure due to using more queries to achieve the same rate of success which, consequently, invalidates the first and second hypothesis. For the third hypothesis, the gate complexity was not directly measured. Instead, was opted to measure the number of queries used by circuit which might not be sufficient to conclude that the algorithm uses shallower circuits.

Keywords: Amplitude Amplification, Heuristic, Model-Free, Maximum Finding, Query Complexity

RESUMO

Esta tese trabalha com a busca em árvores utilizando a Aprendizagem por Reforço para encontrar a melhor sequência possível de ações que o agente terá de executar de forma a obter o maior prémio possível, isto enquanto usa menos esforço computacional em comparação com utilizar apenas um algoritmo de procura quãntica pelo máximo. Para atingir estes objectivos, usaremos a propriedade que possibilita limitar o espaço de procura para os elementos marcados pelo oráclo no Algoritmo de Grover, marcando exatamente um quarto do espaço de procura, procedendo com uma procura quântica. Disto resulta um dos elementos marcado e a informação codificada nele será usada para atualizar uma distribuição probabilistica guardada em memória clássica.
O objectivo é encontrar o minímo de iterações deste processo necessária para obter uma percentagem de sucesso - número de vezes que o algoritmo retornou um elemento que é solução do problema e o número de queries usado - e comparar estes resultados com um procedimento tradicional de procura quântica. Caso isto se obseve, é colocada a hipótese de se usar este algoritmo como forma de observar ações futuras em comparação com os algortimos tradicionais, isto é, usar o mesmo ou menos queries para avaliar um maior número de sequências fruto do aumento do horizonte dos episódios a avaliar. A última hipótese testa se a profundidade dos circuitos, mais concretamente o número de gates usadas. Caso o algoritmo proposto utilize circuitos menos profundos que o algoritmo quantum maximum finding, este poderá ser utilizado nas máquinas quânticas atuais pois estes circuitos produzem medições mais resistentes a erros.
Os resultados mostraram que o algoritmo proposto não possui qualquer vantagem comparado ao quantum maximum finding por usar mais queries para atingir a mesma percentagem de sucesso o que, consequentemente, invalida a primeira e segunda hipótese. Quanto à terceira hipótese, o número de gates usadas por circuito não foi medido diretamente. Em vez disso, optou-se por medir o número de queris por circuito o que poderá não ser suficiente para obter conclusões quanto à profundidade dos circuitos medidos.

Palavras-Chave: Amplificação de Amplitude, Complexidade de Query, Heurística, Modelo Livre, Procura pelo Máximo

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1

INTRODUCTION

Learning methods are studied in a wide range of fields, from psychology to mathematics and computer science, in a way to understand how to efficiently transmit information through various agents and how to acquire new information.
Why do we need to teach a machine how to solve a problem if we can program it?
Some problems are too complex or lack a predetermined and easy-to-recognize algorithm to solve them. Artificial intelligence algorithms process huge quantities of information and produce an answer that is the best statistical solution to the problem in a way to circumvent this.
These algorithms require a huge amount of data and computing power, which is solved today with parallelization techniques. Quantum computers make use of quantum principles to push this parallelization paradigm even further, which justifies the recent efforts on developing these machines and algorithms that are built on quantum properties.
In this thesis, we explore a branch of the machine learning field of study, Reinforcement Learning. In reinforcement learning, because the agent has no information before the exploration of the environment, the learning process requires a balance between exploration and exploitation. This way the agent can obtain a complete view of the environment and project the best action to take at a given time.
The environment is typically represented as a Markov decision process (MDP), either deterministic or stochastic. MDPs are usually solved using algorithms based on dynamic programming, which assumes the knowledge of the environment or, model-free techniques in case the agent has no a priori access to information.

1.1 MOTIVATION

In Reinforcement Learning, in the presence of a deterministic environment, the agent's goal reduces to find the optimal sequence of actions that either lead to a goal state or that maximizes reward. In this work, we consider the problem of finding the optimal sequence of actions that maximize the reward in an episodic setting, i.e., the sequence of actions is computed for a fixed number of time steps, also known as the horizon, h.

Classically, it reduces to find the path in the search tree, leading to the maximum reward. If the number of actions available at each state is constant, $|A|$, then the complexity of finding the optimal path is $O(|A|^h)$. The problem can even be more general than that, assuming a possibly different number of actions for each state, also known as the non-constant branching factor. $O(|A|^h)$ is a loose upper bound in these cases. A groverized solution is often employed in the quantum setting, reducing the complexity to $O(\sqrt{|A|^h})$. However, for non-constant branching factor trees, the gain in complexity could be lost since the trivial Hadamard superposition allows every action to be represented even though not used.

1.2 CONTRIBUTIONS

The paper [4] proposes an heuristic aproach to quantum search in tree-like structures. In the paper, the authors propose an oracle that marks one fourth of the states "closer" to the goal. The oracle will look like

$$U\left|a_1 a_2 \cdots a_d\right\rangle = \begin{cases} -\left|a_1 a_2 \cdots a_d\right\rangle \text{ if } f(a_1, a_2, \cdots, a_d) \leq F^{-1}(0.25) \\ \left|a_1, a_2, \cdots, a_d\right\rangle \textit{ otherwise} \end{cases} \tag{1}$$

Where $F(p)$ is a cumulative distribution function and $F^{-1}(x)$ is the quantile function of the cumulative function, i.e., if $F(x) = p$ represents the cumulative probability of the element x being p, $F^{-1}(p) = x$ represents the element with cumulative probability equal to p. By selecting one-fourth of the state space guarantees that one of the marked states will be measured since under these conditions Grover's algorithm exhibits a success probability equal to 1 for a single application of Grover's operator.

Another interesting approach that shares some similarities with what is presented here is *partial quantum search* proposed in [5], where the authors propose a separation of the search space in K blocks, followed by an inversion about the mean in every K block after marking the desired state. Next, the target state is marked again and a last inversion about the global average is made. The result is the concentration of the amplitude in the block where the target state is present.

In this thesis, we will have the probability distribution of the states - sequence of actions from the start state to an arbitrary horizon - stored in classical memory with the "weight" of each sequence of actions, and from it, we will select a fourth of the possible set of actions, which will be lately marked by the oracle present in the Grover's Algorithm. Then, a Quantum Maximum Finding (QMF) procedure will be executed, hopefully, returning the sequence with the highest reward present in the superposition of the marked states. The weight will be increased in the just mentioned probability distribution over the possible state. The algorithm will iterate over these steps until a given termination criterion is satisfied.

Hopefully, the state (sequence of actions) with higher reward will have been assigned a higher weight. The experiments were run to find how many iterations of this algorithm are needed to get a similar success rate compared to running the same experiments with its absence.
As is explained in Sections 2.4.1 and 2.4.2, the QMF algorithm will apply eventually $\sqrt{N}$ applications of the exponential search algorithm (Qsearch, see algorithm 1), which in turn can potentially apply a maximum of $\sqrt{N}$ Grover operators (queries). By reducing the initial search space to $\frac{N}{4}$ it, hopefully, reduces the previous numbers to $\sqrt{\frac{N}{4}}$.
Due to this algorithm being iterative, the goal is to find if reducing the number of queries needed balances the number of iterations of the algorithm, resulting in a lower total number of queries than by just applying the QMF algorithm on the entire state space at once while converging to the maximum rewarded action sequence.
The hypothesis is that the query complexity of the algorithm acting on 25% of the actions sequences is less than the query complexity of the algorithm searching in the full action sequence domain. Moreover, it is hypothesized that this query complexity gain is maintained even though the proposed algorithm has to iterate multiple times. Should the hypothesis be validated, the algorithm could explore larger lookaheads at a similar query complexity as full tree search in smaller horizons. The algorithm that explores larger lookaheads can perform better decisions.

1.3 STRUCTUTRE

We aim at two kinds of readers, those who are in the field of classical Reinforcement/Machine Learning and are curious about what quantum computation has to offer to their field, and, those who come from the Quantum Computing field and want to know how quantum algorithms enhance machine learning.
To reach both, we will start with the respective background theory such as Quantum Mechanics concepts (2.1), an introduction to quantum computing and the algorithms that will be used in this work (2.2) and Reinforcement Learning techniques like dynamic programming and Markov Decision Processes (3).
Next, we present the algorithm proposed, the problems to which we applied the technique, a discussion of the results and potential future work.
We will compare the number of queries in the circuit and the percentage of success, i.e., the number of times the algorithms return the maximum reward present in the environment between both methods to find if by sampling the search space we get better results.
Lastly, we discuss some ways of upgrading this algorithm, from complexity improvements to the way the probability distribution can be updated.

2

BRIEF INTRODUCTION TO QUANTUM MECHANICS/INFORMATION

In this chapter we will go over the theory behind quantum computation/information, introducing some quantum mechanics concepts and how they are applied in computations. Then, we give an overview of some relevant quantum gates, circuit examples and algorithms that will be used as subroutines.

2.1 QUANTUM MECHANICS

The contents of this section follow the introduction of [6] and are stated here for completion. Quantum mechanics describes systems at the subatomic/atomic scales. At this size, classical theories can not describe with certainty said systems. The key differences are the quantities of a bound system which are restricted to discrete values (quantization). Objects behave as particles and waves (wave-particle duality) and there are limits to how accurately the value of a physical quantity can be predicted before its measurement. Here is introduced the Dirac notation where a *ket* $|\psi\rangle$, which denotes a quantum state, is a complex vector and a *bra* $\langle\psi|$, its complex conjugate. The norm of a vector is defined as the inner product, represented as a *bracket*, $||\psi|| = \sqrt{\langle\psi|\psi\rangle}$.

2.1.1 *Superposition*

The state of a quantum mechanical system is a vector, ψ that belongs to a complex Hilbert space.
A quantum state can be an eigenvector of an observable, and if so, we call it an eigenstate, and the associated eigenvalue corresponds to the value of the observable in that eigenstate. Thus, a quantum state can be a linear combination of the eigenstates, a *quantum superposition*:

$$|\psi\rangle = \alpha_1 |\psi_1\rangle + \cdots + \alpha_N |\psi_N\rangle \tag{2}$$

where $\{\alpha_1 \cdots \alpha_N\}$ is a complex amplitude vector. $|\psi_1\rangle \cdots |\psi_N\rangle$ and N is the total number of states in superposition. In general we can write:

$$|\psi\rangle = \sum_i \alpha_i |\psi_i\rangle \tag{3}$$

where $|\psi\rangle$ is normalized and the eigenstates are orthonormal:

$$\sum_i |\alpha_i|^2 = 1 \tag{4}$$

Thus $|\alpha_i|^2$ is the probability of a measure collapsing the state $|\psi\rangle$ to $|\psi_i\rangle$. The outer product is constructed as $|\psi\rangle \langle\psi|$. For every vector there is an orthonormal basis $e_i, \forall i \in N$ such as the inner product $\langle e_i | e_j \rangle = \delta_{ij}$ and:

$$|e_i\rangle \langle e_i| = \begin{pmatrix} e_1 \\ \vdots \\ e_N \end{pmatrix} (e_1 \cdots e_N) = \begin{pmatrix} e_1 e_1 & e_1 e_2 & \cdots & e_1 e_N \\ \vdots & \vdots & \ddots & \vdots \\ e_N e_1 & e_N e_2 & \cdots & e_N e_N \end{pmatrix} \tag{5}$$

forms a *projection operator* that, when multiplied by some vector, projects the vector onto the subspace of the basis vector. The projection can be used to change basis, expressing some state (ψ) in the basis e_i:

$$P |\psi\rangle = \sum_i |e_i\rangle \langle e_i|\psi\rangle = \sum_i \langle e_i|\psi\rangle |e_i\rangle \tag{6}$$

This operator is used in the process of measuring a quantum state.

2.1.2 *Measurements and Operators*

Other important entity are the *operators*, which in physics correspond to the observables like momentum and position, that act on a quantum state. Operators act linearly by left multiplication on a superposition state.
A *measurement* on a state $|\psi\rangle$ is given by the Born Rule; lets say we have the state

$$|\psi\rangle = \alpha_1 |\psi_1\rangle + \alpha_2 |\psi_2\rangle \tag{7}$$

the probability of measuring each eigenstate is

$$|\langle \psi_i|\psi\rangle|^2 = \begin{cases} \alpha_1^2, & \text{if } |\psi_i\rangle = |\psi_1\rangle \\ \alpha_2^2, & \text{if } |\psi_i\rangle = |\psi_2\rangle \end{cases} \tag{8}$$

If we have an operator ρ, acting on $|\psi\rangle$, of the form

$$\rho = \sum_i \rho_i |\psi_i\rangle \langle\psi_i| = \rho_1 |\psi_1\rangle \langle\psi_1| + \rho_2 |\psi_2\rangle \langle\psi_2| \tag{9}$$

this operator is also called the density operator and ρ_i describes the probability of measuring a pure state. This one describes a density matrix we will measure the system with probability:

$$|\langle\psi_i| \rho |\psi\rangle|^2 = \begin{cases} \alpha_1^2 \rho_1^2, & \text{if } |\psi_i\rangle = |\psi_1\rangle \\ \alpha_2^2 \rho_2^2, & \text{if } |\psi_i\rangle = |\psi_2\rangle \end{cases} \tag{10}$$

2.1.3 *Time evolution and Unitary Operators*

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

$$i\hbar \frac{d}{dt} |\psi\rangle = \hat{H} |\psi\rangle \tag{11}$$

where $\hat{\mathrm{H}}$ denotes the Hamiltonian of the system.

The solution to this equation is the unitary operator $\psi(t) = \mathcal{U}(t) |\psi_0\rangle$; here ψ_0 is the initial state, $\hbar$ is the Planck constant and $\mathcal{U}(t)$ is the unitary time-evolution operator given by

$$\mathcal{U}(t) = e^{-i\frac{t}{\hbar}\hat{H}} \tag{12}$$

Unitarity means that the operator preserves the inner product between the vectors in an Hilbert space. The Hamiltonian is hermitian ($H = H^\dagger$), which means that the possible measured energies (eigenvalues of the Hamiltonian), are always real numbers.
The *Born rule* states that if an observable corresponding to an operator A is measured in a system with normalized wave function $|\psi\rangle$, then:

- the measured result will be one of the eigenstates of A
- the probability of measuring a given eigenvalue will equal $\langle\psi| P_i |\psi\rangle$, where P_i is the projection onto the eigenspace of A corresponding to ψ_i.

Since the complex number $\langle\psi_i|\psi\rangle$ is known as the probability amplitude that the state vector $|\psi\rangle$ assigns to the eigenvector $|\psi_i\rangle$, the Born rule generally says that this probability is equal to the amplitude-square $|\langle\psi_i|\psi\rangle|^2$.
The Schrödinger equation is a linear differential equation meaning that if we have two wave functions that are solutions then a linear combination of both is also a solution allowing for the superposition principle that was mentioned before.

2.1.4 *Composite systems and Entanglement*

Suppose there are two quantum systems, A and B, with the respective Hilbert spaces $\mathcal{H}_A$ and $\mathcal{H}_B$. These can be combined with the tensor product resulting in the system spanned over the Hilbert space $\mathcal{H}$, which is given by

$$\mathcal{H} = \mathcal{H}_A \otimes \mathcal{H}_B \tag{13}$$

Following this, we can introduce the concept of *entanglement*. Let us fix a basis $\{|i\rangle_A\}$ for $\mathcal{H}_A$ and $\{|j\rangle_B\}$ for $\mathcal{H}_B$. The general state of $\mathcal{H}$ is of the form

$$|\psi\rangle_{AB} = \sum_{i,j} c_{ij} |i\rangle_A \otimes |j\rangle_B \tag{14}$$

This state is separable if exists the vectors c_i^A, c_j^B so that $c_{ij} = c_i^A c_j^B$, yielding $|\psi\rangle_A = \sum_i c_i^A |i\rangle_A$ and $|\psi\rangle_B = \sum_j c_j^B |j\rangle_B$. If the state $|\psi\rangle_{AB}$ is entangled, we can not describe each component of the subsystems A and B meaning, that for at least one pair of coordinates (c_i^A, c_j^B), $c_{ij} \neq c_i^A c_j^B$. Instead, a density matrix is defined which describes the statistics that can be obtained by making measurements on either component system alone.
If a quantum system was perfectly isolated it would remain *coherent* but would be impossible to make any measurement on. During a measurement, occurs a loss of information to the environment (just as a heat bath [7]) resulting in coherence being lost over time. The dynamics between the state and the environment can be viewed as an entangled state, sharing quantum information.

2.2 QUANTUM COMPUTATION

There are several quantum computing models, however, only a subset of them can offer universal quantum computation. Some examples are: *quantum circuit model*, *quantum Turing machine*, *adiabatic quantum computer*, *blind quantum computing* and others [8].
In terms of computability theory, quantum computers, offer no advantages over classical computers, meaning, that quantum computers and classical computers offer solutions to the same problems [9]. However, quantum computation, in several applications has a lower time complexity over the classical ones. Quantum computers are believed to quickly solve certain problems that no classical computer can solve in feasible time - also known as *quantum supremacy*. The advantages comes from exploiting quantum mechanics properties such as *superposition*, *entanglement* and *interference*. In the quantum circuit model, generally, the computation is composed of three steps:

- Initialization: usually this step involves transforming the ground state into a superposition of the sub-states that encode the information
- Intended Computation: unitary transformations on the input state
- Measurement: collapsing the final state into one of the substates present prior to the measurement

2.2.1 *Qubit*

Any two-level quantum-mechanical system can be used as a qubit. Multi-level systems can be used as well if they possess two states that can be decoupled from the rest. Some common examples are the polarization of a photon where the dual states are encoded as the horizontal and vertical polarization of the light [10], the spin of an electron [11], and the supercurrent phenomenon that flows through a Josephson junction [12].

The ground state of the two-level system can be expressed as the zero state $|0\rangle$ and the first excited state as the $|1\rangle$ state which form a orthonormal basis of an Hilbert space and is called the *Computational basis*, being isomorphic to $\mathbb{C}^2$. The state of a qubit is represented by a linear combination of the basis states $|0\rangle$ and $|1\rangle$. According to the Born rule, with probability $|\alpha_0|^2$ we observe the state $|0\rangle$ and with $|\alpha_1|^2$ we observe $|1\rangle$, following the rule $|\alpha_0^2| + |\alpha_1^2| = 1$. The state can also be represented in the standard basis as the vectors:

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \qquad |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \tag{15}$$

Another representation is the *Bloch Sphere* and can be parametrized as

$$|\psi\rangle = e^{i\phi}(cos\frac{\theta}{2}|0\rangle + e^{i\varphi}sin\frac{\theta}{2}|1\rangle) \tag{16}$$

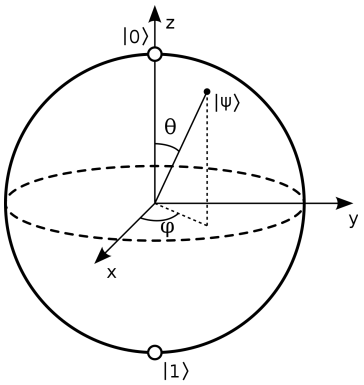


Figure 1: Bloch Sphere. Image from Wikipedia.

Note that, because α and β are complex numbers, we should expect four degrees of freedom. However, one degree of freedom is eliminated by the normalization constraint leaving us with equation 16. Additionally, for a single qubit, the overall phase of the state $e^{i\phi}$ has no physical observables resulting in $e^{i\varphi}$ being the only significant phase, presented in figure 2.
We can represent a classical bit as being in either the north or south pole of the sphere, pure qubit states are spanned over the surface of the sphere and a mixed state can only be represented inside the sphere.
In the *circuit model of quantum computation*, gates (unitary matrices that emulate quantum operators) operate over qubits mapping them to an arbitrary state.

2.2.2 *Quantum Gates*

The computation is performed by quantum logic gates. These are operators described as unitary matrices relative to some basis (usually the computational basis). This property makes these gates reversible meaning that, if we consider a state $|\psi_0\rangle$ and an unitary operator $\mathcal{O}$, we can write $|\psi_1\rangle = \mathcal{O}\,|\psi_0\rangle$ and $|\psi_0\rangle = \mathcal{O}^{-1}\,|\psi_1\rangle$.

The most known single qubit gates are the *Pauli matrices*:

$$\begin{aligned} \sigma_x &= \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} && = X \\ \sigma_y &= \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} && = Y \\ \sigma_z &= \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} && = Z \end{aligned} \tag{17}$$

In general, a single qubit phase shift can be written as $P(\varphi) = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\varphi} \end{pmatrix}$ and

$$
\begin{aligned}
Z &= \begin{pmatrix} 1 & 0 \\ 0 & e^{i\pi} \end{pmatrix} = P(\pi) \\
S &= \begin{pmatrix} 1 & 0 \\ 0 & e^{i\frac{\pi}{2}} \end{pmatrix} = P\left(\frac{\pi}{2}\right) = \sqrt{Z} \\
T &= \begin{pmatrix} 1 & 0 \\ 0 & e^{i\frac{\pi}{4}} \end{pmatrix} = P\left(\frac{\pi}{4}\right) = \sqrt{S}
\end{aligned}
\tag{18}
$$

The σ_x acts as a classical *NOT* gate inverting the state (often called the *bit-flip gate*)

$$
\sigma_x |0\rangle = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix} = |1\rangle \tag{19}
$$

The σ_z acts as a *phase-flip gate* inverting the phase of a quantum state $|1\rangle$ while leaving the state $|0\rangle$ unchanged

$$
\sigma_z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix} = |0\rangle \tag{20}
$$

$$
\sigma_z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} = - \begin{pmatrix} 0 \\ 1 \end{pmatrix} = - |1\rangle \tag{21}
$$

In this case the global phase is irrelevant, in fact, if we measure a state $|1\rangle$ transformed by a gate σ_z resulting in $-|1\rangle$, we will measure it with the square amplitude, which in this case results in measuring the state $|1\rangle$ all the time. Mind you that this is true if we do not perform any computation after flipping the phase; Grover's Algorithm is a clear example of an algorithm where the global phase is determinant as we will see in Section 2.4.1.

Rotation gates correspond to single qubit rotations on the corresponding axis of the Bloch Sphere. Equations 22, 23 and 24 refer to rotations around the $\hat{x}$, $\hat{y}$ and $\hat{z}$ axis respectively

$$
R_x(\theta) = \begin{pmatrix} cos(\theta/2) & -isin(\theta/2) \\ -isin(\theta/2) & cos(\theta/2) \end{pmatrix} \tag{22}
$$

$$R_y(\theta) = \begin{pmatrix} cos(\theta/2) & -sin(\theta/2) \\ sin(\theta/2) & cos(\theta/2) \end{pmatrix} \tag{23}$$

$$R_z(\theta) = \begin{pmatrix} e^{-i\theta/2} & 0 \\ 0 & e^{i\theta/2} \end{pmatrix} \tag{24}$$

All single-qubit rotation gates can be translated into Pauli-matrices, using an angle of $\theta = \pi$ as represented in equation 25

$$\begin{aligned} R_x(\pi) &= -iX \\ R_y(\pi) &= -iY \\ R_z(\pi) &= -iZ \end{aligned} \tag{25}$$

An arbitrary single-qubit gate can be parametrized by three Euler angles as

$$U(\phi, \theta, \varphi) = \begin{pmatrix} cos(\theta/2) & -e^{i\phi}sin(\theta/2) \\ e^{i\varphi}sin(\theta/2) & e^{i(\theta+\phi)}cos(\theta/2) \end{pmatrix} \tag{26}$$

The Hadamard gate is probably the most important logic gate, being responsible for creating quantum uniform superpositions over the basis state:

$$H|0\rangle = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \frac{1}{\sqrt{2}} \left[\begin{pmatrix} 1 \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ 1 \end{pmatrix} \right] = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \tag{27}$$

$$H|1\rangle = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \frac{1}{\sqrt{2}} \left[\begin{pmatrix} 1 \\ 0 \end{pmatrix} - \begin{pmatrix} 0 \\ 1 \end{pmatrix} \right] = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \tag{28}$$

An arbitrary superposition state can be created using the R_y gate as in equation 29

$$R_y(\theta)|0\rangle = \begin{pmatrix} cos(\theta/2) & -sin(\theta/2) \\ sin(\theta/2) & cos(\theta/2) \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = cos(\theta/2)|0\rangle + sin(\theta/2)|1\rangle \tag{29}$$

From here is a matter of solving the equations $\alpha_0 = cos(\theta/2); \alpha_1 = sin(\theta/2)$ for said superposition $|\psi\rangle = \alpha_0|0\rangle + \alpha_1|1\rangle$.
To operate over multiple qubits we have the fundamental two-qubit gate *CNOT*, *controlled-NOT gate* or *CX*. This gate does a bit-flip on a qubit (target) dependent on the state of another one (control)

$$CNOT = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \tag{30}$$

In other words, if the control is zero, it applies the identity matrix on the target or applies a not gate if the control is one. In general, this gate can be modified to apply any other single-qubit gate $\mathcal{U}$. Writing in Dirac notation

$$CNOT = |0\rangle \langle 0| \otimes \mathbb{1} + |1\rangle \langle 1| \otimes U \tag{31}$$

The CNOT gate in conjunction with the Hadamard is able to entangle qubits

$$CNOT(H \otimes \mathbb{1})(|0\rangle \otimes |0\rangle) = CNOT\left(\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \otimes |0\rangle)\right) = \frac{1}{\sqrt{2}}(|0\rangle \otimes |0\rangle + |1\rangle \otimes |1\rangle) \tag{32}$$

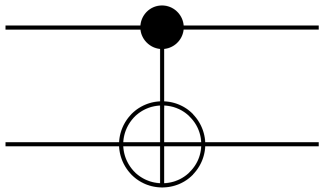


Figure 2: CNOT circuit representation. Image from Wikipedia.

Lastly, is the *Toffoli* gate also known as CCNOT being the quantum analog to the classic AND gate. This gate has 2 control qubits and flips the third one if the controls are both in the state $|1\rangle$. The Toffoli gate needs to be decomposed into simpler gates to be implementable on actual hardware. The canonical Toffoli gate is decomposed as in figure 3, and further decomposed as in Figure 4 for an arbitrary number of control qubits.

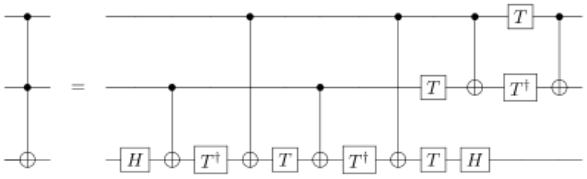


Figure 3: Toffoli gate representation and its decomposition with elementary gates. Image from [1].

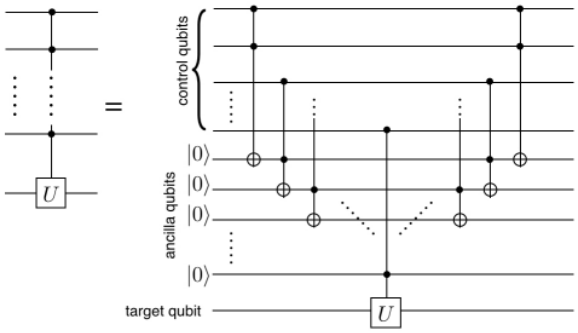


Figure 4: V-Shape decomposition for multi control gates. Image from [2].

2.3 INITIALIZATION

The most known state preparation methods are the *Basis encoding* and *Amplitude encoding* but there are others such as *Qsample encoding*, *dynamic encoding*, *squeezing embedding*, *Hamiltonian encoding* [13]. In this work, we will use the basis encoding method because of the constraints imposed by the algorithm that will be proposed later. First, let´s consider a classical binary input data set of M elements, $D = \{x_0, \cdots, x_{M-1}\}$ where $x_m = \{0,1\}^n$, n being the bitstring length.

2.3.1 *Basis Encoding*

This method is the most straightforward, it just associates each element of the data set with a computational basis state of a qubit system. In this case, the amplitude of each quantum state carries no information and is only used as a way to ”mark” the state for posterior processes.
An entire data set will be represented as:

$$|D\rangle = \frac{1}{\sqrt{M}} \sum_{m=0}^{M-1} |x_m\rangle \tag{33}$$

For example the data set of the string $x_0 = 01$ and $x_1 = 10$ is represented as

$$|D\rangle = \frac{1}{\sqrt{2}} |01\rangle + \frac{1}{\sqrt{2}} |10\rangle \tag{34}$$

and it is achieved by applying the following operations

$$\begin{aligned} CNOT(H \otimes X)(|0\rangle \otimes |0\rangle) &= CNOT(\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle) \otimes |1\rangle \\ &= \frac{1}{\sqrt{2}} CNOT(|01\rangle + \frac{1}{\sqrt{2}}|11\rangle) \\ &= \frac{1}{\sqrt{2}}(|01\rangle + \frac{1}{\sqrt{2}}|10\rangle) \end{aligned} \tag{35}$$

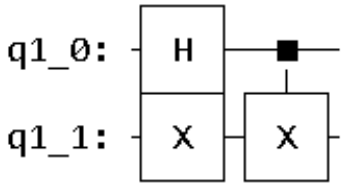


Figure 5: Initialization circuit for the specific state presented in equation 35.

For this example, the initialization circuit is very simple to realize. A more generalized way of creating such state preparation circuits is described in [14]. A better approach to basis encoding is to encode the indexed positions of the array instead of the elements. If we take the array, $D = \{010101, 101111\}$, it can be encoded by $|0\rangle \mapsto 010101$ and $|1\rangle \mapsto 101111$. This way it is possible to encode repeated binary strings [15] and use fewer qubits.

2.3.2 *Amplitude Encoding*

This technique, encodes the information on the amplitude of the sub-states of a quantum state. A normalized dataset of M elements, $D = x_0, \cdots, x_M$ will be represented by the quantum state as

$$|D\rangle = \sum_{i=1}^{N} \frac{a_i}{\sqrt{norm}} |\psi_i\rangle \tag{36}$$

where $|\frac{a_i}{\sqrt{norm}}|^2$ corresponds to the amplitude of element x_i of the dataset D and $|\psi_i\rangle$ is a quantum state of $log(M)$ qubits. To exemplify, let us say we want to amplitude encode the dataset $x = [2, 2.2, 4, 0]$. First we calculate the norm, $|D| = 24.84$, the normalized array becomes $x_{norm} = \frac{1}{\sqrt{24.84}}[2, 2.2, 4, 0]$ and now we can represent this array in the state

$$|D\rangle = \frac{1}{\sqrt{24.84}}[2|00\rangle + 2.2|01\rangle + 4|10\rangle] \tag{37}$$

Note that the $\sqrt{24.84}$ appears due to the measurement postulate. Again for this example in particular, is relatively easy to come up with a set of operators that initializes $|D\rangle$

$$\begin{aligned} CR_y(\theta_2)(R_y(\theta_1) \otimes H)(|0\rangle \otimes |0\rangle) &= CR_y(\theta_2)(cos(\theta_1/2)\,|0\rangle + sin(\theta_1)\,|1\rangle)(\frac{1}{\sqrt{2}}\,|0\rangle + \frac{1}{\sqrt{2}}\,|1\rangle) \\ &= \frac{1}{\sqrt{2}}cos(\theta_1/2)\,|00\rangle + \frac{1}{\sqrt{2}}cos(\theta_1/2)\,|01\rangle + \frac{1}{\sqrt{2}}D \end{aligned} \tag{38}$$

where $D = sin(\theta_1/2)\,|1\rangle\,(cos(\theta_2/2)\,|0\rangle + sin(\theta_2/2)\,|1\rangle - sin(\theta_2/2)\,|0\rangle + cos(\theta_2)\,|1\rangle)$ resulting in the final state

$$\begin{aligned} |\psi\rangle = &\frac{1}{\sqrt{2}}cos(\theta_1/2)\,|00\rangle + \frac{1}{\sqrt{2}}cos(\theta_1/2)\,|01\rangle + \\ &[sin(\theta_1/2)cos(\theta_2/2) - sin(\theta_1/2)sin(\theta_2/2)]\,|10\rangle + \\ &[sin(\theta_1/2)sin(\theta_2/2) + sin(\theta_1/2)cos(\theta_2/2)]\,|11\rangle \end{aligned} \tag{39}$$

Lastly we can get θ_1 and θ_2 by solving the equations

$$\begin{aligned} \frac{1}{\sqrt{2}}cos(\theta_1/2) &= \frac{2}{\sqrt{24.84}} \\ \frac{1}{\sqrt{2}}cos(\theta_1/2) &= \frac{2.2}{\sqrt{24.84}} \\ sin(\theta_1/2)cos(\theta_2/2) - sin(\theta_1/2)sin(\theta_2/2) &= \frac{4}{\sqrt{24.84}} \\ sin(\theta_1/2)sin(\theta_2/2) + sin(\theta_1/2)cos(\theta_2/2) &= 0 \end{aligned} \tag{40}$$

Again, this is a very naive way of obtaining an initialization circuit but is an excellent exercise to practice the application of the gates presented in the chapter subsection 2.2.2.
A more generic technique consists of mapping any arbitrary state to the ground state [16]. It means that to be used as a state preparation routine, we need to invert every gate and apply them in inverse order.

This strategy has the advantage of only needing $\log(N)$ qubits - N being the total number of elements to encode - and can encode different data types, i. e., integer or floating-point. On the other hand, it creates deep circuits due to the use of multi-controlled gates (see figure 3 for the decomposition of a multi-controlled gate which can be decomposed as in figure 4).

2.4.1 *Grover's Search and Amplitude Amplification*

One solution

Consider the problem: given a discrete function $y = f(x)$ and a search value y_i, find x such that $f(x) = y_i$. Classically, a search performed on an array of N ordered elements will return the solution, i.e., the index i of the array that corresponds to x in $\mathcal{O}(\log N)$ steps. However, in an unordered set of elements, no classical algorithm can find i with probability higher than $\frac{1}{2}$ without looking at half the elements on average.
Consider the elements are encoded on a uniform superposition. Doing a measurement will collapse the state to one of the sub-states that represent one of the data points encoded in the quantum state, possibly one of the states that we are not interested in. What if we want to measure a particular sub-state? We can measure the quantum state multiple times, which is costly because the quantum state needs to be created every time after we measure it. Alternatively, we can amplify the amplitude of that particular sub-state with the algorithm proposed in [17], reducing the number of circuits we would have to run. This is a procedure used in a lot of other algorithms, such as the well-known *Grover's Algorithm*. Suppose we have a state $|\psi\rangle$ and an operator $\mathcal{A}$ that transform $|\psi\rangle$ into a superposition of its sub-states, encoding an arbitrary set of data and performs no measurements. In [18] the authors mention a Boolean function $\chi : \mathbb{Z} \to \{0, 1\}$ that induces the Hilbert space, to which $|\psi\rangle$ belongs, into a sum of two subspaces, the *good* and the *bad*. The good subspace is the subspace spanned by the set of basis states $|x\rangle \in H$ for which $\chi(x) = 1$. The amplification process is done by applying repeatedly the operator

$$\mathcal{Q} = -\mathcal{A}S_0\mathcal{A}^{-1}S_\chi \tag{41}$$

also known as a grover iteration. The operator S_χ is the oracle that changes the sign of the *good* states,

$$|x\rangle \to \begin{cases} -|x\rangle, & \text{if} \quad \chi(x) = 1 \\ |x\rangle, & \text{if} \quad \chi(x) = 0 \end{cases} \tag{42}$$

the operator S_0 changes the sign of the amplitude if the state $|x\rangle$ is in the ground state and the operator $\mathcal{A}^{-1}$ is the inverse of $\mathcal{A}$.
A graphical explanation of a Grover iteration can be seen on the Figure 6

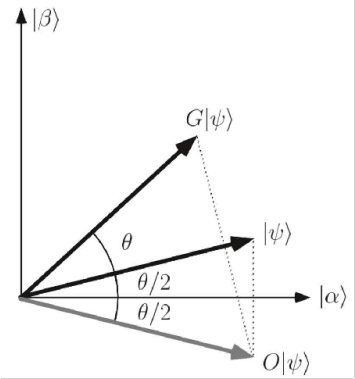


Figure 6: Geometric representation of the first iteration of Grover's Algortihm. Image from [3].

The state ψ is a superpositon of the solution state $|\beta\rangle$, and $|\alpha\rangle$, the superposition of all non-solution states. S_χ (O in the image 6) acts as a reflection operator, rotating $|\psi\rangle$ around the bad states α (fliping their phases in the process), resulting in $|\psi\rangle'$ ($O|\psi\rangle$ in the same image). Then, the operators $-\mathcal{A}S_0\mathcal{A}^{-1}$, where S_0 is the reflection around the all-zero state, transforming $|\psi\rangle'$ in $\mathcal{G}|\psi\rangle$. In sum, the reflection around the $|\psi\rangle$ state is described as,

$$-\mathcal{A}S_0\mathcal{A}^{-1} = 2|\psi\rangle\langle\psi| - \mathbb{I}^{\otimes n} \tag{43}$$

An interesting property of the operator $\mathcal{Q}$ appears when S_χ marks exactly 25% of the search space, on a uniform superposition. If that happens, the reflection around $|\psi\rangle$ eliminates all the "bad" states producing a uniform superposition over the "good" states, guaranteeing that we will measure only the ones marked by the oracle S_χ. On the other hand, if we mark $\frac{1}{2}$ of the states in a uniform superposition the algorithm will not work. These properties emerge due to the fact that, after the operator S_0 acts as an inversion about the mean. As an example lets suppose that we have the state $|\psi\rangle = \mathcal{A}|\psi_0\rangle$. Here $\mathcal{A}$ is a set of operations that encode our data and $|\psi_0\rangle = |0\rangle^{\otimes 2}$.

$$|\psi\rangle = \frac{1}{\sqrt{4}}\sum_{i=0}^{3}|x_i\rangle = \frac{1}{2}(|00\rangle + |01\rangle + |10\rangle + |11\rangle) \tag{44}$$

Supose that S_χ marks the state $|00\rangle$ leaving $|\psi\rangle' = \frac{1}{2}([-|00\rangle + |01\rangle + |10\rangle + |11\rangle]$.
Next, the inversion about the mean is performed. Inverting around the mean is to find a different sequence of numbers such that:

- the new numbers have the same distance as before from the mean, but inverted and,
- the new sequence has the same mean as before.

Suppose for a given sequence of amplitudes the mean is $\mu = \frac{\sum_{i=0}^{N} \alpha_i}{N}$. For every amplitude α_i the distance to the mean will be $\alpha_i - \mu$. After the inversion about the mean the resulting amplitude of each state will be given as

$$2\mu - \alpha_i \tag{45}$$

If we calculate the distance to the mean of every new amplitude we get $2\mu - \alpha_i - \mu = -(\alpha_i - \mu)$ proving the first point. For the second point we have

$$\mu_{new} = \frac{1}{N}\sum_{i=0}^{N} 2\mu - \alpha_i = 2\mu - \frac{\sum_{i=0}^{N} \alpha_i}{N} = 2\mu - \mu = \mu \tag{46}$$

Applying this to our example we get

$$\begin{aligned} |\psi\rangle'' &= \left(\frac{2}{4} + \frac{1}{\sqrt{4}}\right)|00\rangle + \left(\frac{2}{4} - \frac{1}{\sqrt{4}}\right)|01\rangle + \left(\frac{2}{\sqrt{4}} - \frac{1}{\sqrt{4}}\right)|10\rangle + \left(\frac{2}{4} - \frac{1}{\sqrt{4}}\right)|11\rangle \\ &= |00\rangle \end{aligned} \tag{47}$$

We can conclude from these examples that for the case we mark $\frac{1}{4}$ of the total state space we can make sure that the measurement will result in one of the marked ones.

For the next example, we will mark half of the states, $|01\rangle$ and $|11\rangle$, resulting in the following: $\mu = 0$, $|\psi\rangle' = \frac{1}{\sqrt{4}}(|00\rangle - |01\rangle + |10\rangle - |11\rangle)$ and applying equation 45 we end up with

$$\left|\psi''\right\rangle = \frac{1}{\sqrt{2}}(-|00\rangle + |01\rangle - |10\rangle + |11\rangle) \tag{48}$$

With this, we showed that the algorithm will just swap the amplitude of the marked states with the "bad" ones when half of the states were marked keeping their probability of being measured the same as before the amplification. More precisely, if we have the state $|\psi\rangle = sin(\frac{\theta}{2})|\beta\rangle + cos(\frac{\theta}{2})|\alpha\rangle$, we see in Figure 6, that a single application of the operator $\mathcal{G}$ rotates the state by 3 times the initial angle resulting in

$$|\psi\rangle = sin(\frac{3\theta}{2})|\beta\rangle + cos(\frac{3\theta}{2})|\alpha\rangle \tag{49}$$

or, more generally

$$\mathcal{G}^j = sin((2j+1)\theta)|\beta\rangle + cos((2j+1)\theta)|\alpha\rangle \tag{50}$$

So, we have to apply $\mathcal{G}$ j times such that the probability of measuring a good state is close to 1

$$sin^2((2j+1)\theta) \approx 1 j \quad = \lfloor \frac{\pi}{4\theta} \rfloor \tag{51}$$

In general, the reader might be tempted to iterate this algorithm j number of times to get the amplitude of the desired state close to one but, be careful with the amplitude of said state, if it is too high the mean after its inversion might be negative, decreasing the amplitude after the inversion about the mean.

Multiple number of Solutions

Let us suppose our problem has t solutions in a search space of N elements. *Grover's Algorithm* can be easily adapted for this problem, we just need to create the oracle, i.e., the operator S_χ, to mark all solutions and follow it with the reflection over the mean. This process may be iterated $j = \frac{\pi}{4}\sqrt{\frac{N}{t}}$ times to ensure that we measure a solution with maximum probability [17].

Unknown number of Solutions

The case of the unknown number of solutions was first presented for a classical search in [19]. A quantum adaptation of the *exponencial search* algorithm created by Jon Bentley and Andrew Chi-Chih Yao was proposed in [17], where the authors proved that it still converges in $\mathcal{O}(\sqrt{\frac{N}{t}})$. Because we do not know the number of solutions we can not apply a priori the exact number of Grover's operators, j, that maximizes the amplitude of the solutions. The algorithm solves this by selecting the number of operators to be applied randomly from a set that grows exponentially with each iteration until it finds a solution.

Algorithm 1 Qsearch

$m \leftarrow 1$; maxiterations $\leftarrow \sqrt{N}$; $\lambda \leftarrow \frac{6}{5}$; A[N]
while $m \leq maxiterations$ **do**
  $it \leftarrow random(1, m)$;
  $Create\ |\psi\rangle = \frac{1}{\sqrt{N}} \sum_i^N |i\rangle$;
  Apply it iterations of Grover's operator, Equation 41;
  $Measure\ |\psi\rangle\ , i \leftarrow |i\rangle$;
  if $A[i] == x$ **then**
    $return$ i;
  else
    $m \leftarrow min(\lambda * m, maxiterations)$;
  end if
end while
$return$ Null

Notice that the parameter λ must be a number between 1 and $\frac{4}{3}$.

2.4.2 *Quantum Maximum Finding*

Algorithm 1 can be used as a subroutine to find the minimum element in an unordered list as proposed in [20]- Although, in the article, the authors designed the algorithm to find the minimum, a magnitude comparator oracle that marks the states $|x\rangle$ such that $A[x] \geq A[y]$, where y is a threshold, can be trivially devised and was suggested in [21].
The algorithm uses the Qsearch as a subroutine. We start by randomly choosing one of the elements from the set to be the threshold of the Qsearch (y). The Qsearch is then applied to find an element that is larger than the threshold which becomes the new threshold.
The authors claim that the probability is $1 - \frac{1}{2^c}$ where c is the number of iterations of the quantum maximum finding algorithm. The algorithm finds a solution in $\mathcal{O}(\sqrt{N})$ iterations.

Algorithm 2 QMF(Quantum Maximum Finding)

$y \leftarrow random(0, ..., N-1); it \leftarrow 0; A[N];$
while $it \leq \sqrt{N}$ **do**
  $Create\ |\psi\rangle = \frac{1}{\sqrt{N}} \sum_{i=0}^{N-1} |i\rangle$;
  Apply the magnitude comparator oracle;
  Apply the Qsearch Algorithm 1;
  Measure $|x\rangle, x^{'} \leftarrow |x\rangle$;
  if $A[x^{'}] > A[y]$ **then**;
    $y \leftarrow x^{'}$;
  end if
end while
Return y

3

REINFORCEMENT LEARNING

Reinforcement Learning (RL) is an area of machine learning together with *supervised learning* and *unsupervised learning*.
In supervised learning, the agent receives a labeled dataset {x,y} where y is a "flag" that classifies x. The goal of the agent is to design a function $f : x \mapsto y$, such that it learns all the correct labels associated with x. The goal is to generalize such functions to unseen data.
In unsupervised learning, the parity present in the training set does not exist, and so, the agent has to find the patterns in the data.
RL differs from the previous two. The RL agent receives information from the environment, namely the state is fully observable and more importantly the reward. The agent needs to interact with the surrounding environment and learn by trial and error. RL algorithms aim to maximize a reward which is a signal that indicates the degree of goodness of the performed action.
RL techniques have a trade-off, the exploration v.s. exploitation problem, since the agent has no prior knowledge of the environment. This has been thoroughly studied through the multi-armed bandit problem and for finite state space, MDPs in [22]. Should we take an action that we already know the outcome or should we risk-taking an unknown action expecting a bigger reward?
For RL problems, the environment can be *fully observable*, the agent can directly observe the environment, for example, the game of chess; or, it can be *partially observable*, where the agent has access to a particular piece of the entire environment like the game of poker. In this thesis, we will consider fully observable environments, which can be stated as Markov Decision Processes (MDP).

3.1 MARKOV DECISION PROCESSES

MDPs consist of states, actions, transitions between states, and a reward function definition. They form the basis of RL since it gives a clear definition of an environment and its dynamics. An MDP is described as a tuple $\langle S, A, T, R \rangle$. An additional parameter might be added to

this tuple, the discounting factor γ, as we will see later.
A set of **states**, S, is defined as the finite set $S = \{s^1, \cdots, s^N\}$. For example, in chess, any possible configuration of pieces disposed on the board is a state.
A set of **actions**, A, is defined as $A = \{a^1, \cdots, a^K\}$. The set of actions that can performed in a state is described as A(s).
Applying the action $a \in A$, in the state $s \in S$, the system performs a **transition** form s to a new state $s^{'} \in S$, based on a probability distribution over the set of possible transitions.The transition function T is defined as $T : S \times A \times S \to [0, 1]$, i.e., the probability of ending up in state $s^{'}$ after taking action a in state s is denoted $T(s, a, s^{'})$. For each state-action pair, T needs to be normalized $0 \leq T(s, a, s^{'}) \leq 1$ and $\sum_{s^{'} \in S} T(s, a, s^{'}) = 1$.
The system being controlled is an MDP if the result of an action does not depend on the previous actions and visited states (history), but only depends on the current state, also known as the *Markov Property*

$$P(s_{t+1}|s_t, a_t, s_{t-1}, a_{t-1}, \cdots) = P(s_{t+1}|s_t, a_t) \tag{52}$$

Here the time is discrete, s_t denotes the state at time t. The idea of Markovian dynamics is that the current state s gives enough information to make an optimal decision. It is not important which states and actions preceded s.
More general models can be characterized by being *k-Markov*, i.e., the last k states are enough.
The **Reward Function** is responsible to assign rewards to the agent and can be designed mainly with two distinct strategies, namely, state-dependent rewards or state-action dependent rewards. The state reward function is defined as $R_{sa} : S \to \mathcal{R}$ specifying the rewards for being in a state. Other definitions are: $R : S \times A \to \mathcal{R}$ or $R_{sa} : S \times A \times S \to \mathcal{R}$. The first gives rewards for acting in a state, and the second gives rewards for particular transitions between states.
The transition function T and the reward function R together define the model of the MDP. Often MDPs are depicted as a state transition graph where the nodes correspond to states/actions and (directed) edges denote transitions[23]. The reward is added to this representation that tells how good it is in a given state. The acumulated reward defines a trajectory in the graph, G_t, the return of some sequences of rewards, $G_t = R_{t+1} + \cdots + R_T$. This formulation may be problematic, in the case $T = \infty$, the reward will be infinite. The discount factor ensures that – even with an infinite horizon – the sum of the rewards obtained is finite. In episodic tasks, the discount factor is not needed, however, it is useful when $h \to \infty$ to prevent large reward summation that may difficult the optimization reward.To solve this, is added the discounting factor γ resulting in $G_t = R_{t+1} + \gamma R_{t+2} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$. There are three models of optimality in the MDP: the finite horizon, where the *expected return* is

given by $E\left[\sum_{t=0}^{h} r_t\right]$; the discounted infinite horizon, $E\left[\sum_{t=0}^{\infty} \gamma^t r_t\right]$, and the average reward, $\lim_{h\to\infty} E\left[\frac{1}{h}\sum_{t=0}^{h} r_t\right]$.

3.1.1 *Policies*

Given a MDP M, a policy π is a computable function that outputs for each state $s \in S$ an action $a \in A(s)$. A *deterministic* policy is described as $\pi : S \to A$ or $\pi : S \times A \to \delta_{SA}$ (for every $s \in S$, $\pi(s,a) \geq 0$ and $\sum_{a\in A} \pi(s,a) = 1$. A *stochastic* policy is defined as $\pi : S \times A \to [0,1]$.
Sampling trajectories under a policy π is done by:

- a start state s_0 from the initial state distribution I is generated
- the action $a_0 = \pi(s_0)$, calculated by the policy π, is performed
- Based on the transition function T and reward function R, a transition is made to state s_1, with probability $T(s_0, a_0, s_1)$ and a reward is received

If the task is *episodic*, the process ends after some steps or if the agent reaches a terminal state.

3.1.2 *Value functions and Bellman Equations*

A value function represents an estimate of how good it is for the agent to be in a certain state. The value of a state s under a policy π, $V^\pi(s)$ is the expected return when starting in s and following π. Considering the infinite horizon model, this can be expressed as

$$V^\pi(s) = E_\pi\{\sum_{k=0}^{\infty} \gamma^k r_t | s_t = s\} \tag{53}$$

A similar state-action value function $Q^\pi(s,a)$, or *Q-function*, can be defined as the expected return starting from state s, taking action a and thereafter following policy π

$$Q^\pi(s,a) = E_\pi\{\sum_{t=0}^{\infty} \gamma^t r_t | s_t = s, a_t = a\} \tag{54}$$

One fundamental property of value functions is that they satisfy certain recursive properties. For any π and any state s, $V^{\pi}(s)$ can be recursively defined as a *Bellman equation*:

$$\begin{aligned} V^{\pi}(s) &= E_{\pi}\{r_t + \gamma_{t+1}^{1} + \gamma^2 r_{t+2} + \cdots | s_t = s\} \\ &= E_{\pi}\{r_t + \gamma V^{\pi}(s_{t+1}) | s_t = s\} \\ &= \sum_{s'} T(s, \pi(s), s') \left(R(s, a, s') + \gamma V^{\pi}(s') \right) \end{aligned} \tag{55}$$

This property denotes that the expected value of state is defined in terms of the immediate reward and values of possible next states weighted by their transition probabilities, and additionally a discount factor. Multiple policies can have the same value function, but for a given policy π, V^{π} is unique.
The goal for any MDP is to find the optimal policy, the policy that maximizes the value function $V^{\pi}(s)$ for all $s \in S$, i.e., maximizes the reward.
An optimal policy, π^*, is such that $V^{\pi^*}(s) \geq V^{\pi}(s)$, $\forall s \in S$ and $\forall \pi \in \Pi$. The *Bellman optimality* equation

$$V^*(s) = max_{a \in A} \sum_{s'} T(s, a, s') \left(R(s, a, s') + \gamma V^*(s') \right) \tag{56}$$

states that the value of a state under the optimal policy must be equal to the expected return for the best action in that state. To select an optimal action given the optimal state value function V^* the following rule can be applied:

$$\pi^*(s) = argmax_a \sum_{s'} T(s, a, s') \left(R(s, a, s') + \gamma V^*(s') \right) \tag{57}$$

π^* is called the *greedy* policy because it allways selects the best action using the value function V. The analogous optimal state-action value is:

$$Q^*(s, a) = \sum_{s'} T(s, a, s') \left(R(s, a, s') + \gamma max_{a'} Q^*(s', a') \right) \tag{58}$$

Q-functions are useful because they make the weighted summation over different alternatives. This is the reason for their preference in the model-free setting, i.e., in case T and R are unknown. The relation between Q^* and V^* is given by

$$Q^*(s, a) = T(s, a, s') \left(R(s, a, s') + \gamma V^*(s') \right) \tag{59}$$

$$V^*(s) = max_a Q^*(s, a) \tag{60}$$

And the optimal action (best policy) can be written as

$$\pi^*(s) = argmax_a Q^*(s,a) \tag{61}$$

3.2 DYNAMIC PROGRAMMING

Dynamic Programming (*DP*) is a method of solving a complicated problem by breaking it down into simpler sub-problems in a recursive manner. In the context of RL, the Bellman Equation acts as a recursive structure and its solution can be solved with DP techniques. There are two main models used, *Policy Iteration* and *Value Iteration*.
We will assume a standard MDP $< S, A, T, R, \gamma >$ where the state and action sets are finite and discrete such that they can be stored in tables. The reward and value functions are assumed to store values for all states and actions separately.

3.2.1 *Policy Iteration by Howard*

This technique is composed of two stages: the *policy evaluation* stage computes the value function of the current policy and the *policy improvement* stage computes an improved policy by a maximization over the value function. This is repeated until converging to an optimal policy. The first step is to find the value function V^π of a fixed policy π. This is called the prediction problem.
The Bellman equation is transformed into an update rule, turning the actual value function V_t^π into V_{t+1}^π, the one-step lookahead.

$$\begin{aligned} V_{t+1}^\pi(s) &= E_t\{r_t + \gamma V_k^\pi(s_{t+1}) | s_t = s\} \\ &= \sum_s^{'} T(s, \pi(s), s^{'})(R(s, \pi(s), s^{'}) + \gamma V_k^\pi(s^{'})) \end{aligned} \tag{62}$$

The update rule is applied to each state $s \in S$ in each iteration. The old value-function is replaced with a new one based on the expected value of possible successor states, intermediate rewards, and weighted by their transition probabilities. This operation is called a full backup because it is based on all possible transitions from that state.
At the end of policy evaluation, we ought to improve $V^\pi(s)$.First, we identify the value of all actions:

$$\begin{aligned} Q^\pi(s,a) &= E_t\{r_t + \gamma V_t^\pi(s_{t+1}) | s_t = s, a_t = a\} \\ &= \sum_{s'} T(s, a, s^{'})(R(s, a, s^{'}) + \gamma V_k^\pi(s^{'})) \end{aligned} \tag{63}$$

If $Q^\pi(s,a)$ is larger than $V^\pi(s)$ then a is chosen to update the policy for state s. This is done for every state of the MDP, generating our improved policy. That is, we can compute

the *greedy* policy, selecting the best action in each state, based on the current value function V^{π}:

$$\begin{aligned} \pi'(s) &= argmax_a Q^{\pi}(s,a) \\ &= argmax_a E\{r_t + \gamma V^{\pi}(s_{t+1}) | s_t = s, a_t = a\} \\ &= argmax_a \sum_{s}^{'} T(s,a,s^{'})(R(s,a,s^{'}) + \gamma V_k^{\pi}(s^{'})),\ \forall s \in S \end{aligned} \tag{64}$$

If $\pi'(s) = \pi(s), \forall s \in S$ then the agent converged to the optimal policy. For a finite MDP each π^{t+1} is strictly better than π^t unless $\pi^t = \pi^*$ in which case the algorithm stops. The algorithm is presented in Algorithm 3.

Algorithm 3 Policy Iteration (PI)

Require: $V(s) \in \mathbb{R} and \pi(s) \in A(s) for all s \in S$
  Policy Evaluation
  repeat
    $\Delta \leftarrow 0$
    for all $s \in S$ **do**
      $v \leftarrow V^{\pi}(s)$
      $V(s) \leftarrow \sum_s^{'} T(s,\pi(s),s^{'})(R(s,\pi(s),s^{'}) + \gamma V(s^{'}))$
      $\Delta \leftarrow max(\Delta, |v - V(s)|)$
    end for
  until $\Delta < \sigma$
  Policy Improvement
  $p \leftarrow true$
  for all $s \in S$ **do**
    $b \leftarrow \pi(s)$
    $\pi(s) \leftarrow argmax_a \sum_s^{'} T(s,a,s^{'})(R(s,a,s^{'}) + \gamma V(s^{'}))$
    if $b \neq \pi(s)$ **then**
      $p \leftarrow false$
    end if
  end for
  if p **then**
    halt
  else
    go to Policy Evaluation
  end if

3.2.2 *Value Iteration*

This algorithm blends the policy improvement step into it.

$$\begin{aligned} V_{t+1}(s) &= max_a \sum_{s}^{'} T(s,a,s^{'})(R(s,a,s^{'}) + \gamma V_t^{\pi}(s^{'})) \\ &= max_a Q_{t+1}(s,a) \end{aligned} \tag{65}$$

The *value iteration* algorithm spends less time in the evaluation part by removing the *improvement* step and, instead of initializing a policy randomly and alternating between *evaluation* and *improvement*, it initializes the value function arbitrarily.
Both algorithms theoretically converge to the optimal policy, however, the difference in the update, makes the Value Iteration converge to the optimal policy faster. These algorithms operate over the entire state space of the MDP which, for some applications, may be too expensive.

3.3 MODEL-FREE TECHNIQUES

In contrast with the algorithms discussed in the previous section, model-free methods do not rely on the availability of a priori model of the MDP. This generates a need to sample the MDP to gather statistical knowledge about this unknown model.
The first one is to learn the transition and reward model from interacting with the environment. Once our model is sufficiently correct, the methods from the previous section can be applied. This is called model-based RL or *indirect* RL.
The second option, *direct* RL, is to estimate values for actions, without even estimating the model of the MDP. Additionally, mixed forms between these two can exist. For example, one can still do a model-free estimation of action values, but use an approximated model to speed up value learning by using this model to perform more, and in addition, full backups of values.
Aditionally, it is difficult to assess the utility of some action, if the real effects of this particular action can only be perceived much later. One possibility is to wait until the end of an episode and punish/reward specific actions along the path taken. However, this will take a lot of memory and, often, it is not known beforehand whether, or when, there will be an "end". Instead, similar mechanisms as in value iteration can be used to adjust the estimated value of a state based on the immediate reward and the estimated (discounted) value of the next state. This is generally called *temporal difference learning*.
The general class of algorithms that interact with the environment and update their estimates after each horizon are called *online* RL. Here, the agent selects an action based on its current state, gets feedback in the form of the resulting state and associated reward, and updates its

estimated values stored in V^{π} and Q^{π}. The selection of the action is based on the current state s and the value function.

Algorithm 4 Online RL

for each episode **do**
  $s \in S$ is the starting state
  $t \leftarrow 0$
  repeat
    choose an action $a \in A(s)$ from $\pi(s)$
    perform action a
    observe the new state $s^{'}$ and reward r
    update T, R, Q and/or V
    using the experience $\langle s, a, r, s^{'} \rangle$
    $s \leftarrow s^{'}$
  until $s^{'}$ is the goal state/terminal state or we reached the horizon
end for

Since model-free algorithms need to balance exploration and exploitation, the agent needs an action-selection mechanism that balances effectively, between informed actions and random actions.
The most basic exploration strategy is the $\epsilon-$greedy, where, the learner takes its current best action with probability $1-\varepsilon$ and other randomly selected action with probability ε. One natural way of balancing exploration/exploitation is through the Boltzmann exploration strategy where the agent stochastically selects an action from the probability distribution π in equation 66.

$$\pi(a|s) = \frac{e^{\frac{Q(s,a)}{T}}}{\sum_i e^{\frac{Q(s,a_i)}{T}}} \tag{66}$$

where T is known as the inverse temperature $\beta = \frac{1}{T}$. Higher values of T will move the selection strategy more towards a purely random strategy and lower values will move to a fully greedy strategy. We can combine the $\epsilon - greedy$ and the strategy in equation 66, selecting the best action with probability $1-\epsilon$ and an action computed by the Boltzmann exploration strategy, with probability ϵ.

3.3.1 *Monte-Carlo Methods*

Monte-Carlo(MC) algorithms treat the long-term reward as a random variable and take as its estimate the sampled mean. It estimates values based on *averaging sample returns* observed during an interaction. Monte Carlo methods are defined only for episodic tasks, they are

incremental in an episode-by-episode sense, but not in a step-by-step (online) sense.
As more returns are observed, the average should converge to the expected value.
Suppose we are trying to estimate $v_\pi(s)$, the value of a state s under policy π, given a set of episodes obtained by following π. Each occurrence of state s in an episode is called a *visit* to s. The state s may be visited multiple times during an episode so let us call the first time it is visited *first visit to s*. The *first-visit MC* method estimates $v_\pi(s)$ as the average of the returns following first visits to s, whereas the *every-visit MC* method averages the returns following all visits to s. Both these methods converge as the number of visits (or first visits) to s tends to infinity.
The computational expense of estimating the value of a single state is independent of the number of total states.
An interesting application of Monte-Carlo methods is the Monte-Carlo Tree Search where each episode is divided into four steps: selection, expansion, simulation and backpropagation. This technique was used to design software that was able to beat the world champions in games like chess and Go (see [24]).

3.3.2 *Temporal Difference Learning (TD)*

An advantage of TD methods is that they are naturally implemented in an online, incremental fashion such that they can be easily used in various circumstances. Only along experienced paths do values get updated, and updates are affected after each step. This is an advantage over Monte Carlo methods where we have to wait until the end of an episode. Also, Monte Carlo methods must ignore or discount episodes on which experimental actions are taken, which can greatly slow learning.
TD(0) is used to evaluate a policy through the use of the following update rule:

$$V_{k+1}(s) = V_k(s) + \alpha(r + \gamma V_k(s^{'}) - V_k(s)) \tag{67}$$

where $\alpha \in [0, 1]$ is the *learning rate* that determines how much the values get updated. The difference with DP backups is that the update is still done by bootstrapping but is based on an observed transition, it uses a sample backup instead of a full backup. Only the value of one successor state is used, instead of a weighted average of all possible successor states.
The learning rate α has to be decreased appropriately for learning to converge.
Q-learning incrementally estimate Q-values for actions, based on feedback (rewards) and the agent's Q-value function [25].

$$Q_{k+1}(s_t, a_t) = Q_k(s_t, a_t) + \alpha(r_t + \gamma max_a Q_k(s_{t+1}, a) - Q_k(s_t, a_t)) \tag{68}$$

The agent makes a step in the environment from state s_t to s_{t+1} using action a_t while receiving reward r_t. The update occurs on the Q-value of action a_t in the state s_t from which this action was executed.
SARSA is a *on-policy* algorithm that learns the Q-value function for the policy the agent is executing.

$$Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \alpha(r_t + \gamma Q_t(s_{t+1}, a_{t+1}) - Q_t(s_t, a_t)) \tag{69}$$

a_{t+1} is the action that is executed by the current policy for state s_{t+1}. Note that the max-operator in Q-learning is replaced by the estimate of the value of the next action according to the policy. This learning algorithm will still converge to the optimal value function (and policy) under the condition that all states and actions are tried infinitely often and the policy converges in the limit to the greedy policy.

4

QUANTUM REINFORCEMENT LEARNING AND TREE SEARCH

The article that served as inspiration for this thesis, [4], deals with the branching factor of a search tree and its consequences and tries to answer questions like what are the ramifications of a variable such as the branching factor? Would the system require the use of a constant branching factor? In this thesis we approach only problems with constant branching factors, however, the strategy used has its genesis in an attempt to answer the previous questions.

4.1 BRANCHING FACTOR RAMIFICATIONS

In theoretical computer science, one possible way to measure a problem's complexity consists in assessing how long a given algorithm takes to find a solution. This time complexity is dependent on hardware factors so, it is more suitable to find the total amount of items that are needed to be evaluated. In the case of a classical tree search, this equates to the number of nodes to take into account. From a classical tree search perspective, complexity is expressed in terms of b, the branching factor; d, the depth of the shallowest goal node and m, the maximum length of any path in the state space.
A tree with constant branching factor b and depth d will have a total of b^d leaf nodes.
Suppose the search will be performed up to a depth level d. Each sequence of actions will contain d elements/actions with each one requiring $n = \lceil log_2(b) \rceil$ bits. In total the binary string will employ $n \times d$ bits. We are able to construct a quantum superposition $|\psi\rangle$ that encompasses all the actions to be applied up to a depth level d (basis encoding)

$$|\psi\rangle = \frac{1}{\sqrt{2^{n*d}}} \sum_{x=0}^{2^{n*d}-1} |x\rangle \tag{70}$$

Consider the example given by the authors in [4], section 2.2. In figure 7, the authors present a tree with an action set $a = a_0, a_1, a_2, a_3, a_4$. In this case, we have two distinct types of branching factor, the maximum branching factor $b_{max} = 5$ and the average branching factor of each node $b_{avg} = 2$. In order to encode each of the possible actions we require $n = \lceil log_2(b_{max}) \rceil = 3$ bits. If we build a quantum superposition state $|\psi'\rangle$, if we take

into account a depth $d = 3$, it would have $2^{n*d} = 2^9 = 512$ possible states when in reality there is only 7 different states. Here, $n = \lceil log_2(b_{avg}) \rceil$ bits might have sufficed to encode all different set of actions; by employing $n = \lceil log_2(b_{max}) \rceil$ bits, the search space is extended and in the process some of the speedup provided by Grover's algorithm is lost. Considering, $b_{avg} < b_{max}$, when does Grover's algorithm stop providing a speedup over classical approaches?

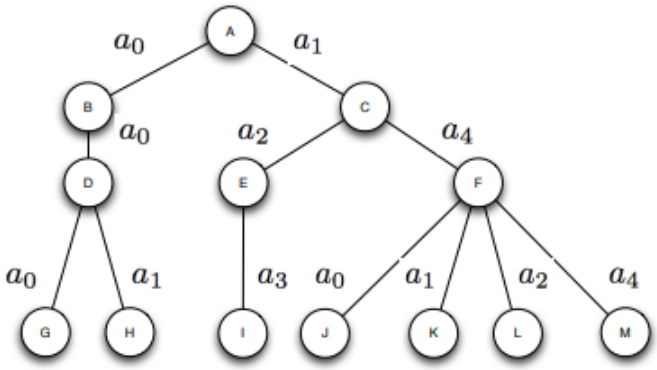


Figure 7: Search tree with a non constant branching factor. Image from [4]

The number of times to apply Grover's operator is

$$\begin{aligned} |\mathcal{G}| &= \sqrt{N} \\ &= \sqrt{2^{n*d}} \\ &= \sqrt{2^{\lceil log_2(b_{max}\rceil)*d}} \\ &= 2^{\frac{\lceil log_2(b_{max})\rceil * d}{2}} \end{aligned} \tag{71}$$

To determine where the threshold lies between the number of elements of a classical search, b_{avg}^d, and the total number of times to apply Grover's iterate we need to solve

$$\begin{aligned} b_{avg}^d = |\mathcal{G}| &\Leftrightarrow b_{avg}^d = 2^{\frac{\lceil log_2(b_{max}\rceil)*d}{2}} \\ &\Leftrightarrow b_{avg} = 2^{\frac{\lceil log_2(b_{max})\rceil}{2}} \end{aligned} \tag{72}$$

Accordingly, when $b_{avg} < 2^{\frac{\lceil log_2(b_{max})\rceil}{2}}$ then the total number of nodes evaluated classically will be less than the number of times to apply Grover's Iterate and, when $b_{avg} > 2^{\frac{\lceil log_2(b_{max})\rceil}{2}}$, the Grover's search will yield a speedup over classical search algorithms.
The authors follow this by studying each b_{max} and its associated range of b_{avg} values.
To basis encode any b_{max} we require $n = \lceil log_2(b_{max}) \rceil$ bits. The use of the ceiling function forces certain ranges of b_{max} to require the same number of n bits.

Following this, the authors study what happens when a transition is made, for example, from n to $n+1$ bits. In this case the b_{avg} value grows from $2^{\frac{n}{2}}$ to $2^{\frac{n+1}{2}}$ which differ by a factor of $\sqrt{2}$, $b_{avg}^{n+1} = \sqrt{2} b_{avg}^{n}$.

4.2 QUANTUM HEURISTIC

During the search process, it would be useful to know which action might produce a state which is closer to a goal state. This process may be described as trying to determine the quality of a path of actions with the optimal solution having the lowest cost among all solutions. This is conceptualized as an *heuristic*, a function $h(n)$ responsible for presenting an estimate of the distance that a given state n is relative to a goal state. The function $h(n)$ is defined depending on the problem at hand.
In the paper, the authors reflect the heuristic function as a unitary operator U to be employed by Grover's algorithm, flipping the amplitude of the solution states. We can opt to consider only the states whose heuristic outputs a value below a certain threshold T.
Consider that $|n\rangle$ is the current node being processed and a_i, action at depth i. The operator U is simply described as

$$U|a_1 a_2 \cdots a_d\rangle = \begin{cases} -|a_1 a_2 \cdots a_d\rangle \text{ if } h(a_1, a_2, \cdots, a_d) \leq T \\ |a_1 a_2 \cdots a_d\rangle \text{ otherwise} \end{cases} \tag{73}$$

The authors extend this idea to a heuristic distribution, the heuristic function employed has a probabilistic distribution, either discrete or continuous. Because in this work we will deal with discrete probabilistic distributions, the other will be ignored.
Let us consider a discrete random variable X. The sum of the set containing all possible probabilities is 1, $\sum_{i=1}^{n} P(X = x_i) = 1$.
The probability that a random variable X takes on a a value that is less than or equal to x is referred to as the *cumulative distributive function* F, $F(x) = P(X \leq x)$. For a discrete random variable X, $F(a)$ is a simple sum of the values up to element a $F(a) = \sum_{x \leq a} P(X = x)$.
To calculate $F(x) = p$ or, in other words, which values the cumulative distribution function equals a certan probability, the *quantile function* is used $F^{-1}(p) = x$. In discrete random variables, the cumulative distribution function may contain gaps between values of the domain so that the quantile function is given by $F^{-1}(p) = inf\{x \in \mathcal{R} : p \leq F(s)\}$ where inf is the infinity operator (see [26]).
Knowing that when a fourth of the states are marked, Grover's algorithm needs only one iteration to obtain one of the marked states, the authors fine-tune the operator U with the assistance of the quantile function.
Suppose the heuristic function $f : X \to Y$. We are interested in marking as a solution the state that produces the smallest values of codomain Y. From a probabilistic point-of-view, it

is possible to check if the heuristic value is less than or equal to the quantile function output for a probability of 25%, transforming the equation 73 in

$$U\left|a_1a_2\cdots a_d\right\rangle = \begin{cases} -\left|a_1a_2\cdots a_d\right\rangle \text{ if } h(a_1,a_2,\cdots,a_d) \leq F^{-1}(0.25) \\ \left|a_1a_2\cdots a_d\right\rangle \text{ otherwise} \end{cases} \tag{74}$$

This notion can be expanded to contemplate different sections of a probability distribution, obtaining a superposition of the states that lie between the heuristic values $F^{-1}(a)$ and $F^{-1}(b)$ where a and b denote two different probabilities.

$$U\left|a_1a_2\cdots a_d\right\rangle = \begin{cases} -\left|a_1a_2\cdots a_d\right\rangle \text{ if } h(a_1,a_2,\cdots,a_d) \in [F^{-1}(a),F^{-1}(b)] \\ \left|a_1a_2\cdots a_d\right\rangle \text{ otherwise} \end{cases} \tag{75}$$

In conclusion, the paper verifies, in the case of a non-constant branching factor, whether is worth using Grover and, presents a way to reduce the search space by evaluating the states that are within a given threshold.
This thesis differs from the above in the fact that the selection is done classically as opposed to the paper where the authors present a quantum oracle that marks a fourth of the states. The quantile function provides no value to this thesis: the marked states are simply sampled from the current heuristic which is not encoded in the quantum state, effectively selecting a fourth of the total states to be marked instead of the ones in which the quantile function is less than or equal to 25%. With this, we just make sure that the sequences which were observed as having the highest reward have a higher probability of getting selected.

5

PROPOSED ALGORITHM: G25

5.1 QUANTUM MAXIMUM FINDING WITH G25

A sequence of actions is encoded as a binary string of length $h * log_2 A$ where h is the horizon or, the number of actions that the agent will perform and A is the possible alternative actions available at each state s (also referred to as the branching factor), resulting in a total of $N = A^h$ alternative sequences.
There are cases, however, where $N = A^h$ might be a loose upper bound on the size of the search domain. These cases include:

- different states s_j having a different number of available actions, A_j, with $A_j < A$, which results in a total number of actions sequences significantly less than N. A small search space might render this thesis strategy useless;
- having some a priori knowledge that certain actions on certain states will lead, with high probability, to a larger reward; this a priori knowledge would allow pruning the decision tree (by not taking into account the potentially less rewarding actions), thus reducing the search domain.

This dissertation does not directly handle any of the previous cases. A constant branching factor is used and no a priori knowledge on the rewards of the actions is assumed. It takes, however, a different approach as a first step towards reducing the search domain. A uniform probability distribution P is initially prepared over all N sequences of actions. 25% of these sequences are stochastically selected and marked for evaluation with the QMF algorithm. QMF returns a given sequence and P is updated proportionally to the associated reward. A new subset of 25% of all possible sequences of actions is selected again from P and processed using QMF.
The goal is, by updating the distribution, at least one of the sequences of actions that have the highest reward possible will have the highest weight and, then will be selected as the solution to the problem.
The hypothesis is that the query complexity of the circuit acting on 25% of the actions

sequences is less than the query complexity of the circuit acting on the totality of actions sequences. More importantly, it is hypothesized that this query complexity is maintained even though the proposed algorithm has to iterate multiple times. This hypothesis is experimentally tested in the next chapter. If it holds it will enable a reduction in the total number of queries required to find an optimal action for a given horizon. Additionally, it will allow for better decision-making by increasing the horizon while maintaining a similar number of queries as the original algorithm. This procedure is shown in Algorithm 5.

Algorithm 5 Proposed Algorithm: G25

Initialize the distribution: $P = \sum_{i=1}^{N} \frac{1}{N}$;
repeat
  Initialize the quantum state $|\psi\rangle = \mathcal{R}(\sum_{i=0}^{N} \frac{1}{\sqrt{N}} |a_0 \cdots a_h\rangle \otimes |0 \cdots 0\rangle)$;
  Selection: Sample a fourth of the sequences from P;
  Apply the G25 operator;
  *Search*Apply c iterations of the Quantum Maximum Finding;
  Measure the state;
  Update the distribution P;
until A set number of iterations were performed;

5.1.1 *Selection*

The *Selection* stage is just a classical sampling from the current distribution P, choosing a fourth of the total number of sequences. The distribution is initialized as a uniform discrete distribution over all the possible sequences of actions and updated over the course of the iterations performed.

5.1.2 *Search*

This step is performed by the quantum machine and to do so, the quantum state is initialized, followed by the application of the G25 operator and it is ended with the QMF algorithm. The initialization of the quantum state is done by creating a superposition over the action register. Then is applied an operator that entangles the action register with the reward register, $\mathcal{R}$. By doing this we map each sequence of actions to its reward.
Next, we apply the *G25 operator*. This operator is just a Grover's operator (41), with exactly a fourth of the search space marked. A single application of this operator results in a superposition over the states that were marked by the oracle reducing the amplitude of the unmarked ones to 0.
Next the QMF algorithm is applied, to be more precise, 3 iterations of this algorithm will

be executed, giving a probability of the result being the maximum reward present in the superposition of 87,5% as was mentioned in section 2.4.2. Notice that, even though we may measure a sequence with the highest reward after just 1 or 2 iterations, the algorithm always performs the 3 iterations because the agent does not know a priori the maximum reward that it may obtain and will keep searching for a better one.

5.1.3 *Update*

The updating is done by adding to the measured sequence the respective reward obtained and subtracting the adequate quantity to normalize the distribution.

Normalization

The measured sequence "weigth" is updated by (76):

$$p^{sel} = \frac{1 + r_{t-1}^{sel} + r^{sel}}{N + R_{t-1} + r^{sel}} \tag{76}$$

where *sel* refers to the measured sequence, r_{t-1}^{sel} is the reward the sequence *sel* has acumulated, r^{sel} is the measured reward and R_{t-1} is the total acumulated reward by the measured sequences after t iterations of the G25 algorithm.
The rest of the sequences are updated by equation (77):

$$p^{seq} = \frac{1 + r_{t-1}^{seq}}{N + R_{t-1} + r^{sel}} \tag{77}$$

It can be proven that for each update of the distribution, the sum of the "weights" is 1:

$$p^{sel} + \sum_{seq \in S'} p^{seq} = \frac{1 + r_{t-1}^{sel} + r^{sel}}{N + R_{t-1} + r^{sel}} + \times \frac{\sum_{seq \in S'} (1 + r_{t-1}^{seq})}{N + R_{t-1} + r^{sel}} \tag{78a}$$

$$= \frac{1 + r_{t-1}^{sel} + r^{sel} + (N-1) + \sum_{seq \in S'} r_{t-1}^{seq}}{N + R_{t-1} + r^{sel}} \tag{78b}$$

$$= \frac{N + r^{sel} + (\sum_{seq \in S'} r_{t-1}^{seq} + r_{t-1}^{sel})}{N + R_{t-1} + r^{sel}} \tag{78c}$$

$$= \frac{N + R_{t-1} + r^{sel}}{N + R_{t-1} + r^{sel}} \tag{78d}$$

where S' represents all the sequences except the one that was measured by the algorithm at the iteration t. Notice that in 78c, $\sum_{seq \in S'} r_{t-1}^{seq} + r_{t-1}^{sel} = R_{t-1}$.
After t iterations of the algorithm, the distribution is given by equation (79):

$$\mathcal{P} = \sum_{seq} \frac{1 + x^{seq} \times r_t^{seq}}{N + R_t} \tag{79}$$

where x^{seq} is the number of times the sequence seq was measured, N is the total number of possible sequences of actions and R_t is the reward acumulated over all measured sequences after t iterations of the algorithm, $R_t = \sum_{seq} x^{seq} \times r^{seq}$. As any discrete probabilistic distribution, $\sum p = 1$:

$$\begin{aligned} \mathcal{P} &= \sum_{seq} \frac{1 + x^{seq} \times r^{seq}}{N + R_t} \\ &= \frac{N + \sum_{seq} x^{seq} \times r_t^{seq}}{N + R_t} \\ &= \frac{N + R_t}{N + R_t} = 1 \end{aligned} \tag{80}$$

Since the exact number of iterations of the G25 algorithm to achieve close to 100% success rate is hard to be determined mathematically, the experiments consist of applying 1 to *it* iterations of the G25 algorithm (see Chapter 6). After each experiment, the sequence with the maximum "weight" is picked as the output of the c iterations of the algorithm ($seq_{out} = argmax\mathcal{P}$).
Then the success rate, specifically, the number of experiences that result in a sequence with the maximum reward possible for that environment, is compared with the QMF algorithm. Notice that, in the latter, there is no distribution, so the 3 iterations of the QMF are applied and the result is the output.

5.2 COMPLEXITY

Being a hybrid algorithm we will take into account the classical and quantum complexity of the algorithm.
Since the algorithm proposed aims to reduce the number of queries, this will be the metric used to study its complexity. Specifically, in the quantum setting, this query complexity refers to the number of Grover's operators that any given experiment is expected to utilize. The classical complexity is simply the number of elements to be searched in both the selection and update steps. It is easy to see that both these steps have a complexity of $\mathcal{O}(N)$:

- the algorithm has to select exactly a fourth of the sequences resulting in a complexity of $\mathcal{O}(\frac{N}{4}) = \mathcal{O}(N)$ when $N \to \infty$;
- in the updating step, the machine will have to search the measured sequence over N and, for the same reasons as in the select step, this grows linearly with the increasing of the search space

The sequences generation was ignored. For the quantum computation, because we make use of the Quantum Maximum Finding algorithm, it inherits its complexity of $\mathcal{O}(\sqrt{N})$. Notice that, after the G25 operator we end up with a fourth of the original number of sequences so the complexity will be $\mathcal{O}(\sqrt{N'})$ where $N' = \frac{N}{4}$.
Despite the gain in the quantum complexity, the size of the circuits might be higher with the application of the G25 operator. For this thesis, only the number of queries (applications of the operator Q in the QMF algorithm) will be taken into account in the calculation of the complexity.
If we look at the algorithm globally, its complexity will be $\mathcal{O}(N)$ (when $N \to \infty$) due to the logistics surrounding the distribution.
How to solve this?
For the selection, we can circumvent this by, instead of selecting a fourth of the search space every iteration, applying this selection only for the first iteration and in the next ones, substituting a set number of sequences of the previous selected with different ones resulting in a constant complexity, $\mathcal{O}(1)$. The previous approach was not tested so its impact on the behavior of the algorithm and its results are not known.
The second item was ignored by hashing the distribution.

6

CASE STUDY

6.1 SIMULATIONS: THE GRIDWORLD

The modeled environment chosen was the Gridworld. As the name suggests, the environment is a grid/matrix of chosen dimensions where the states are each cell of the matrix, and the agents have four available actions at each time-step: move left, right, up, or down. There can be "blocked" states, which can simulate a wall/object; goal states and we can add rewards to each state. The goal of the agent is to find the path that accumulates the most reward (if that is the case) or reaches a goal state avoiding blocked states along the path.
This type of environment is easily scalable making it a perfect fit for this thesis.
Our set of actions $A = \{left, right, up, down\}$ is basis encoded as the states $A = \{left \rightarrow |10\rangle, right \rightarrow |00\rangle, up \rightarrow |01\rangle, down \rightarrow |11\rangle\}$. For this, we need two qubits for each step of the MDP, i.e, the horizon of each episode. In this work, we take into account the actions performed at states in the edge of the environment that lead the agent in that direction, for example, if the agent is in the bottom left entry of the grid and moves to the left, he will remain in the same state but the time step will continue.
The goal state is different from the others by being the only one that offers a reward (this is not true for the 4x4 Gridworld for reasons mentioned later).
The minimum number of alternative sequences of actions is given by $N = A^h$, where A is the number of alternative actions available at each state (also referred to as the branching factor) and h is the horizon (i.e., the length of the sequence of actions).
It was considered that the maximum amount of reward collected at each timestep is 1 leading to the possible maximum reward to be collected being given by h so that $log_2(h) + 1$ qubits are needed to encode the reward of each sequence of actions.
Another important consideration is the characteristic of the simulation. Because we are dealing with deterministic problems, the need to encode the states is removed, and only the actions are taken into account. Thereafter, the implementation uses only one register for the actions with $log_2(actions) * horizon$ qubits and one register for the rewards with $log_2(h) + 1$ qubits.

6.1.1 *Studied Gridworlds*

2x2 Grid

The first case study was the 2x2 Gridworld. The starting state is in the bottom left corner and the goal state is in the top right. In this case, the agent only needs 2 steps to reach the end goal. However, it was simulated a horizon of three because if we only considered 2 actions, we would have a superposition of 16 quantum states. Performing the operator G25 in a superposition of 4 states. If the selected states contain more than 1 encoded sequence with a reward greater than the given threshold, the quantum maximum finding algorithm will not work (the amplification over the mean does not work for $n = \frac{N}{2}$ selected sequences, see 2.4.1).
In this case, there are 4 sequences with the maximum reward of 2, 12 with reward 1, and the rest offer no reward. The total number of sequences is $4^3 = 64$.

Figure 8: The simulated 2x2 Gridworld. In black is the starting state and in blue the goal state.

4x4 Grid

The last environment simulated was a GridWorld 4x4. In this case, we have two states that offer reward 1 and the rest give no reward.
The simulation was divided into two cases: one where the maximum reward is two and the other has a maximum reward of three. The reason is that in the first case it was intended that the reward availability was the same as in the 2x2 Gridworld so that the difference in results would not interfere in the discussion; for the second case, the idea is to study the behavior with a different set of rewards.
As we can conclude from the images 9 and 10, for the agent to collect the desired rewards, two and three respectively, the horizon of each episode should be five resulting in a total of $4^5 = 1024$ different sequences.
In the case of Figure 9 there are 6 sequences of actions with reward two and 42 with reward 1.

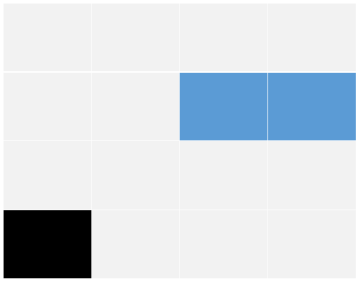


Figure 9: The simulated 4x4 Gridworld with a maximum reward of 2.

In the case of Figure 10 there are 3 desired rewards (reward 3), 31 with reward 2 and 121 with reward 1.

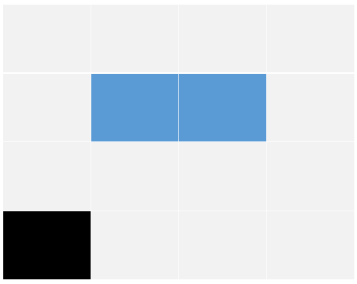


Figure 10: The simulated 4x4 Gridworld with a maximum reward of 3.

We can see that the number of solution sequences (sequences with the maximum reward) is closely the same between every simulation, however the pool of sequences from which we have to search increases from 64 in the 2x2 case to 1024 in the 4x4 case.

6.2 RESULTS

For each of the previously described experimental environments, results are presented for different numbers of iterations of the proposed algorithm, specifically, 1 to 5 iterations. This algorithm is labeled as G25. Each experiment was performed 50 times. Tables 1,2 and 3 present how many times (out of the 50 experiments) each possible reward was measured. The "Success rate" column presents the ratio the maximum possible reward was measured over the total number of experiments. The columns $\mu_{queries}$ and $\sigma_{queries}$ present the average number of queries (executions of Grover's operator) per experiment and the respective standard deviation. The row labeled with 0 refers to the original algorithm, I.e., performing the search over the total number of possible sequences of actions. Comparing the original with the G25 algorithm for the 2x2 grid world (see table 1) a similar average number of queries is achieved for 3 iterations of the latter. However, under these conditions, the former

#Iterations of G25	Reward 0	Reward 1	Reward 2	Success	$\mu_{queries}$	$\sigma_{queries}$
1	1	20	29	58%	10	2
2	0	7	43	86%	19	3
3	0	3	47	94%	29	4
4	0	1	49	98%	37	5
5	0	0	50	100%	46	5
0	0	0	50	100%	26	7

Table 1: Results for the 2x2 Gridworld

achieves a 100% success rate, whereas the latter succeeds at a lower rate of 94%.
It is fair to conclude that in this case, the G25 algorithm provides no benefits compared to just performing a normal quantum search with the QMF. It actually is detrimental; to achieve close to 100% success rate, the results show the need for at least 4 iterations which already have a higher query usage as may be seen in Figure 11.
What if the search space has more sequences of actions? In the 4x4 Gridworld, $N = 1024$ and by the Qsearch algorithm the number of Grover operators (queries) to be applied can reach up to $\sqrt{N} = 32$; adding to this, if the Qsearch does not return a solution (a sequence with reward higher than the threshold passed to the algorithm), the QMF algorithm will apply the Qsearch $\sqrt{N} = 32$ times. Using the G25 operator these numbers are reduced to only 16 hopefully reducing the queries used.

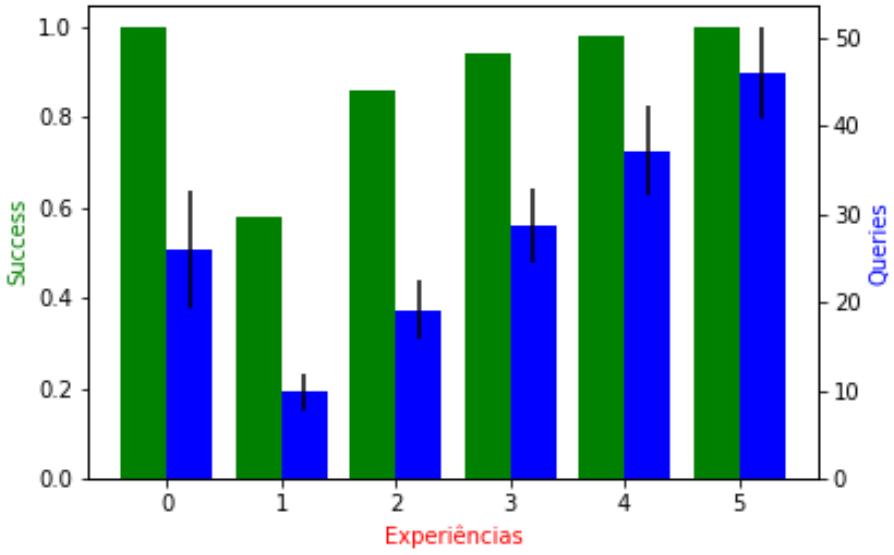


Figure 11: Success Rate v.s. Number of queries used for the 2x2 Gridworld

In the table 2 and figure 12 we can observe that the G25 algorithm achieves a success rate of 100% with only 3 iterations and with a less average number of queries, showing some potential.

G25 Iterations	Reward 0	Reward 1	Reward 2	Success	$\mu_{queries}$	$\sigma_{queries}$
1	0	15	35	70%	139	43
2	0	1	49	98%	258	55
3	0	0	50	100%	384	70
4	0	0	50	100%	507	82
5	0	0	50	100%	624	101
0	0	0	50	100%	463	123

Table 2: Results for the Gridworld 4x4 with a maximum reward of 2.

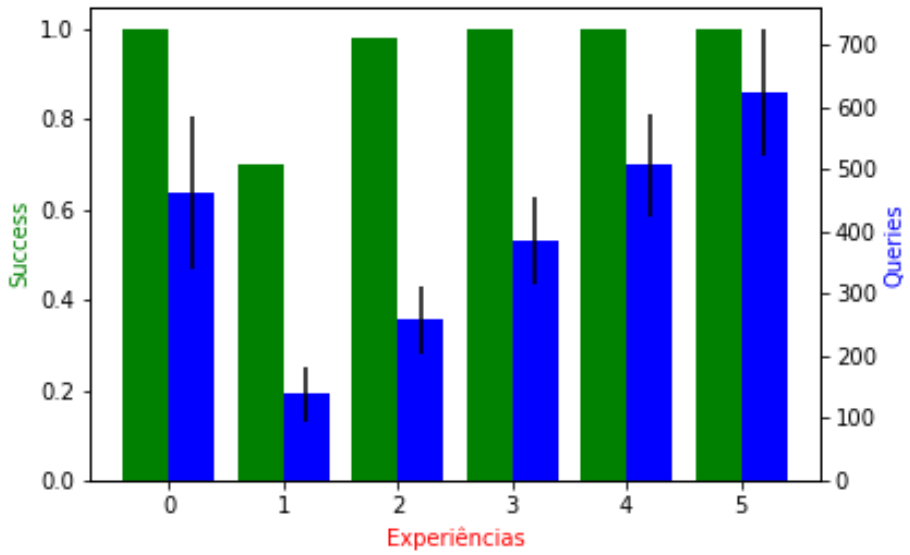


Figure 12: Success Rate v.s. Number of queries used for the Gridworld 4x4 with a maximum reward of 2.

Lastly, it was simulated the same environment 4x4 but where the agent can get a maximum reward of 3 with 5 actions. Table 3 and Figure 13 show some interesting behaviour.
Firstly, we see that the algorithm offers, again, no advantages.
Secondly, the case where the quantum search is applied in the entire search space results in a higher standard deviation than the average meaning that in some experiments were needed a lot fewer queries, while in others this number is extremely high.

Why does this happen? Recall that the QMF is iterated 3 times and each iteration only terminates if a measurement retrieves a reward larger than the current threshold, otherwise,

G25 Iterations	Reward 0	Reward 1	Reward 2	Reward 3	Success	$\mu_{queries}$	$\sigma_{queries}$
1	0	0	26	24	48%	91	52
2	0	0	10	40	80%	160	69
3	0	0	6	44	88%	221	82
4	0	0	4	46	92%	268	87
5	0	0	1	49	98%	328	98
0	0	0	0	50	100%	139	185

Table 3: Results for the 4x4 Gridworld with a maximum reward of 3.

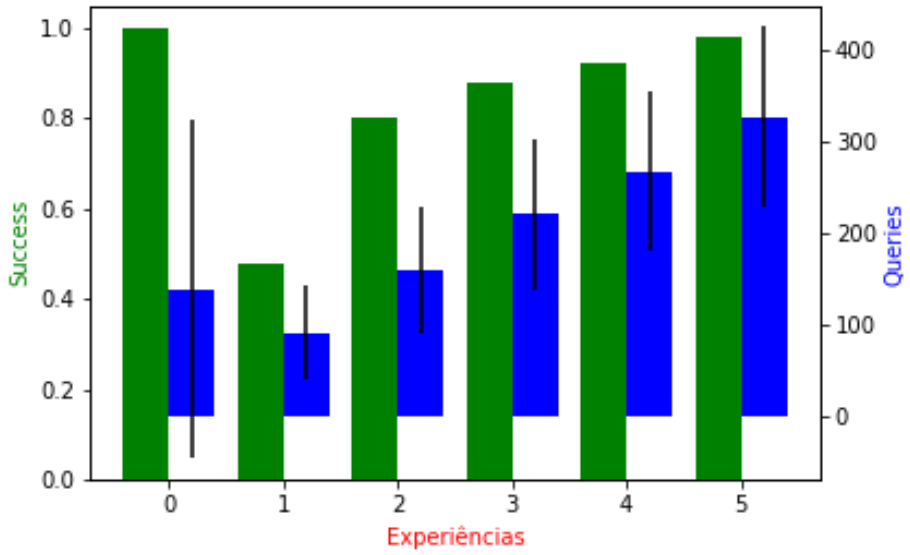


Figure 13: Success Rate v.s. Number of queries used for the Gridworld 4x4 with a maximum reward of 3.

it is performed $\sqrt{N}$ searchs which can apply from 1 to $\sqrt{N}$ Grover operators (queries) (Remember: this number is chosen randomly and the interval grows with each iteration). The cases where a significantly low number of queries used are a result of in each iteration of the QMF was found a reward larger than the threshold, while on the other side of the spectrum, the high number of queries used are a product of the maximum reward being measured in the first iteration of the QMF causing the other 2 to follow the second halt rule. Tables 5 and 4 show a "dry running" of 2 different experiments done without the G25 operator where, in table 5, was used a number of queries were in the low side of the spectrum because 3 solutions were found (lines in bold), while, in the table 4, resulted in a high number of queries due to only 2 solutions being found. Notice that the first solution is a reward of 2 which means that, in the next attempts only a sequence with reward 3 will halt the iteration and the last iteration of the QMF will execute the $\sqrt{N}$ Qsearch evaluations.

Additionally, in the first table, it did not take too long to find a solution which can be seen by the low number of lines between each solution, while in the second table this number is higher between the first solution found and the second which, in conjunction with the first argument, contributes for the high number of queries used in some experiments.

#Queries used	Sequence measured	Reward obtained	Threshold
1	**0100011000**	**2**	**0**
1	1000000110	0	2
1	0100100111	0	2
1	1001000001	1	2
1	0101010011	1	2
3	0111010011	0	2
5	1011111111	0	2
4	1011010011	0	2
12	**0100010010**	**3**	**2**
1	0100100101	0	3
1	0110111010	0	3
1	0010101100	0	3
2	1000111101	0	3
3	0110001111	0	3
6	0110111000	0	3
3	0111100001	0	3
1	1100010101	1	3
17	0000010111	1	3
22	1000001001	0	3
20	0100101101	0	3
20	1100100101	0	3
8	0101110101	0	3
28	1011001000	0	3
3	1000100010	0	3
8	0100100000	0	3
7	1111011001	0	3
11	0011111101	0	3
26	0100101111	0	3
2	1001100111	0	3
29	1010000011	0	3
21	1001100010	0	3
24	0010010100	1	3
27	0011110000	0	3
25	1000000011	0	3
4	0000000100	0	3
4	1110011010	0	3
2	1110110111	0	3
6	1010110011	0	3
10	0111000001	0	3
25	1010010110	0	3
24	1101000010	0	3

Table 4: Dry running of an iteration of the search without applying the G25 operator. Here is shown a high number of circuit executions which result in a total number of queries much higher than the mean.

# Queries used	Sequence Measured	Reward Obtained	Threshold
1	**1001000110**	**1**	**0**
1	**0110000100**	**2**	**1**
1	1100101000	0	2
1	1111111010	0	2
1	**0001010010**	**3**	**2**

Table 5: Dry running of an iteration of the search without applying the G25 operator. Here is shown a low number of circuit executions and in each of them only one query was needed resulting in a total of 5 queries to obtain the maximum reward.

Circuit average query complexity

Circuit depth is tied with the decoherence times of current quantum computers. The deeper the circuit the higher the chance that the information is lost to the environment.
Despite, this being a real problem, in this work the size of the circuit was not taken, directly, into account. This would be a very difficult metric to quantify due to the dynamics of the algorithm. To be more precise, this dynamic is a result of three factors:

- The G25 operator contains an oracle that marks a fourth of the sequences, which in turn, are sampled for every iteration of the algorithm resulting in an oracle that can differ for each iteration.
- The number of Grover operators applied in each Qsearch is also dynamic, adding another layer to the variance of the circuit size.
- The oracle used in the QMF also changes depending on the current threshold.

Instead, was adopted the average number of queries per circuit which are presented in the next table.

Environment	With G25	Without G25
2x2	1	2
4x4 (Rmax = 2)	5	11
4x4 (Rmax = 3)	4	7

Table 6: Average number of queries evaluated in each circuit.

The results presented in the table 6 show that, by reducing the search space using the proposed G25 algorithm, on average, the number of queries performed per circuit is less than searching the entire space. This reduced average number of queries may translate into the proposed algorithm being more suitable to near/medium term noisy quantum hardware. It must be kept in mind that the G25 algorithm requires an additional initial step to prepare a uniform superposition over the 25% selected basis states. The above-mentioned potential

advantage for NISQ systems depends on actual circuit depths, which haven't been evaluated. This will be proposed as future work.

7

CONCLUSION AND FUTURE WORK

7.1 CONCLUSION

To summarize, the proposed approach aimed to reduce the number of queries needed to evaluate the best possible sequence of actions an agent will need to retrieve the most reward possible. To achieve this, the algorithm takes advantage of reducing the total number of sequences to search to exactly a fourth of all available to the agent in a given environment. The strategy uses the characteristic of only needing one Grover's operator to create a uniform superposition over the marked states while reducing the amplitude of the unwanted states to zero. The QMF subroutine is then performed and the information retrieved serves to update a distribution stored in a classical memory. This process is iterated and is expected that at least one of the sequences with the highest reward, will have the highest "weight".
The data obtained, specifically, the case of the Gridworld 4x4 with a maximum reward of 2, shows some advantages by utilizing the proposed algorithm. For low search spaces, as in the case of the 2x2 Gridworld, the algorithm showed no advantage.
Another observed aspect is the interaction between the number of iterations of the QMF subroutine and the total number of different rewards which is another aspect to take into account. The fact that the agent is dealing with a model-free environment makes the calculation of an optimal number of iterations of the QMF quite difficult or even impossible.
In the case studies presented, the results allow us to also conclude that the supposition that the algorithm might be able to see "a bigger picture" of the environment while using the same queries of its competitor can not be achieved; the case that shows some advantages uses a similar number of queries as applying the QMF by itself. If the horizon was increased, the algorithm would eventually use more queries, considering that the number of iterations remains the same as in the results presented than its competitor.

The inconclusive results might indicate a need for extensive empirical work to be done. Realizing more experiments to increase the analyzed data might help get a better idea of the potential of the algorithm.
Increasing the search space is also critical to get a better understanding of the scalability of the algorithm. As mentioned before, the interaction between the measured rewards and the number of QMF iterations can also be taken into account.
As mentioned before, in this work the branching factor remains constant, each state has the same number of possible actions available. Experiments with a non-constant branching factor would be extremely valuable to further the knowledge on the behavior of the algorithm. This might show up to not be a problem, the only constraints on the algorithm are that the total search space has to be multiple of 4 and the environment must be deterministic.
Additionally, regarding the size of the circuits, it could be presented the average depth in terms of the number of gates instead of the average number of queries per circuit.
Some questions arise if some pieces of the puzzle are ignored, what if instead of applying the G25 operator to create a uniform superposition over the sampled sequences, it was utilized an algorithm that designs a circuit that prepares said superposition. With this approach, we could even create a superposition over any fraction of the search space because we are no longer constrained by Grover's operator. Would this preparation circuit be less deep, in terms of the number of gates, than this thesis approach? How would the algorithm behave with various fractions of the search space selected?
Lastly, the way the sequences are sampled could be improved. For example, sequences that are selected too often or, provide the same reward as previous measurements, could have diminishing returns, either, less chance of being selected in future iterations or have their reward being reduced for the update of that particular iteration. This could be implemented in the *Updating* phase, instead of just adding/subtracting a "weight" dependent only on the reward of the sequence, the updating would be dependent, additionally, on the number of times the sequence was selected.

BIBLIOGRAPHY

[1] Vivek V. Shende and Igor L. Markov. On the CNOT-cost of TOFFOLI gates. *arXiv e-prints*, March 2008.

[2] Keisuke Fujii. Quantum Computation with Topological Codes: from qubit to topological fault-tolerance. *arXiv e-prints*, April 2015.

[3] Rituparna Maji, Bikash Behera, and Prasanta Panigrahi. Solving linear systems of equations by using the concept of grover's search algorithm: An ibm quantum experience. 12 2017.

[4] Luís Tarrataca and Andreas Wichert. Tree search and quantum computation. *Quantum Information Processing*, 10(4):475–500, Nov 2010.

[5] Lov K. Grover and Jaikumar Radhakrishnan. Is partial quantum search of a database any easier? *arXiv e-prints*, July 2004.

[6] David J. Griffiths and Darrell F. Schroeter. *Introduction to Quantum Mechanics*. Cambridge University Press, 3 edition, 2018.

[7] D. Bacon. Decoherence, Control, and Symmetry in Quantum Computers. *arXiv e-prints*, May 2003.

[8] Vedran Dunjko and Hans J Briegel. Machine learning & artificial intelligence in the quantum domain: a review of recent progress. jun 2018.

[9] Manuel Vogel. Quantum computation and quantum information, by m.a. nielsen and i.l. chuang. *Contemporary Physics - CONTEMP PHYS*, 52, 11 2011.

[10] C. Adami and N. J. Cerf. Quantum computation with linear optics. June 1998.

[11] L. M. K. Vandersypen and M. A. Eriksson. Quantum computing with semiconductor spins. 60, 2019.

[12] T. P. Orlando, J. E. Mooij, Lin Tian, Caspar H. van der Wal, L. S. Levitov, Seth Lloyd, and J. J. Mazo. Superconducting persistent-current qubit. *Physical Review B*, 60(22):15398–15413, Dec 1999.

[13] Seth Lloyd, Maria Schuld, Aroosa Ijaz, Josh Izaac, and Nathan Killoran. Quantum embeddings for machine learning. *arXiv e-prints*, January 2020.

[14] V.V. Shende, S.S. Bullock, and I.L. Markov. Synthesis of quantum-logic circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 25(6):1000–1010, Jun 2006.

[15] Maria Schuld and Francesco Petruccione. Supervised learning with quantum computers. chapter 3. Springer Publishing Company, Incorporated, 1st edition, 2018.

[16] Mikko Mottonen, Juha J. Vartiainen, Ville Bergholm, and Martti M. Salomaa. Transformation of quantum states using uniformly controlled rotations. *arXiv e-prints*, July 2004.

[17] Michel Boyer, Gilles Brassard, Peter Høyer, and Alain Tapp. Tight bounds on quantum searching. *Fortschritte der Physik*, Jun 1998.

[18] Gilles Brassard, Peter Høyer, Michele Mosca, and Alain Tapp. Quantum amplitude amplification and estimation. *Quantum Computation and Information*, page 53–74, 2002.

[19] Jon Bentley, Bruce Weide, and Andrew Yao. Optimal expected-time algorithms for closest point problems. *ACM Trans. Math. Softw.*, 6:563–580, 12 1980.

[20] Christoph Durr and Peter Hoyer. A Quantum Algorithm for Finding the Minimum. *arXiv e-prints*, July 1996.

[21] Ashish Ahuja and Sanjiv Kapoor. A Quantum Algorithm for finding the Maximum. *arXiv e-prints*, November 1999.

[22] Michael N. Katehakis Apostolos N. Burnetas. Optimal adaptive policies for markov decision processes. 1997.

[23] M. van Otterlo, M.; Wiering. *Reinforcement learning and markov decision processes. Reinforcement Learning. Adaptation, Learning, and Optimization*. 2012.

[24] G.M.J.B. Chaslot; M.H.M. Winands; J.W.H.M. Uiterwijk; H.J. van den Herik; B. Bouzy. Progressive strategies for monte-carlo tree search. 2008.

[25] R. S. Sutton and A. G. Barto. Reinforcement learning: An introduction. 1998.

[26] Warren Gilchrist. *Statistical modelling with quantile functions*. Chapman and Hall/CRC, 2000.