Télécom ParisTech

Semester Project Final Report:

Quantum Reinforcement Learning and Projective Simulation

Sofiène Jerbi

February 5, 2018

Abstract

At the premises of the *quantum supremacy* era, quantum computing is spiking increasingly more interest. With already a few algorithmic applications surpassing their actual classical counterparts in theory and in a few practical proof-of-principle realizations, one is tempted to anticipate the creation of a large-scale universal quantum computer by applying the powerfulness of quantum computation to another emerging field, *Reinforcement Learning*. This branch of Machine Learning takes an approach to learning based on an autonomous agent situated in an environment and learning to perform a task through trial and error. In this work we try to explore the possibilities of such a combination, and we especially deal with the *Projective Simulation* model, for which a quantum algorithm has already been devised, allowing quadratic enhancement over the classical model in "thinking" time of the agent. We also show through computer simulations that these enhancements are robust to noise errors in the specific *trapped-ions* implementation of a quantum computer.

Outline

1 Introduction

In the last few decades, the use of quantum mechanics has undoubtedly provided significant enhancement to communication and information processing, leading to the now well-established fields of *quantum information* and *quantum computation.* Interest in these fields especially spiked with the discovery of *Quantum Teleportation* (transmitting a quantum qubit state between two parties through an entangled pair of qubits and two classical bits) [1] and *Superdense coding* (transmitting two classical bits using one qubit and an entangled pair) [2] in the former ; *Grover's search algorithm* (introducing a quadratic speed-up in the number of queries to search marked elements in an unordered list) [3] and *Shor's factoring algorithm* (giving an exponential time-complexity speed-up to the best classical algorithm) [4] in the latter. Moreover, these discoveries didn't come alone as they were enriched by a deeper characterization of the underlying quantum complexities, both algorithmic: it is known that super-polynomial speed-ups for query complexity problems are only possible for *promise problems* [5] and that in general they require a *substantial combinatorial or algebraic structure* [6]; and communication-wise: for instance, *Holevo's theorem* [7] ensures that n qubits cannot transmit more that n bits of information (even though some bipartite problems, such as *Distributed Deutsch-Jozsa* [8], can be solved with far fewer qubits exchanged than classical bits). These promising results make it very tempting to consider using the advantageous properties of quantum mechanics in another related and ever-growing field: *Artificial Intelligence (AI).* This idea seems to gather more and more talented minds on the subject, reportedly mostly dealing with enhancements to what we'll call next *applied AI* tasks.

We come a long way from the general AI problem of producing a genuine artificial consciousness, set in the early times of computer science. Research evolved by shifting interest to the development of specific AI tasks such as binary classification, data clustering... an area we refer to as *applied AI.* However this shift of emphasis to more refined problems doesn't imply any regression or any lack of general competence, but instead just as in order to produce a flying machine we had to stop trying to replicate how birds fly and start focusing on aerodynamics, in order to design genuine AI we need to understand the building blocks of intelligence first. For that matter, we started developing *Machine Learning (ML)* theories to empower machines with the ability to learn how to perform applied AI tasks, and which are usually divided into 3 categories:

- **Supervised learning**: the program receives a set of training samples with their expected output and has to train until it reaches a success rate threshold. Its performance is then tested on a test set, measured by its mean error.
- **Unsupervised learning**: the program doesn't get any labeling on the samples like in the supervised case. It needs to recognize patterns in the samples and cluster them by similarities. Testing the performance of a program becomes trickier as the definition of error needs to take into account the number of clusters formed by the program.
- **Reinforcement learning**: we define an agent and an environment, interacting through the agent's sensors and actuators. The agent acts on the environment through a set of actions, and the environment reacts through percepts and rewards sent back to the agent.

This latter category, often regarded as a completely separate branch of ML, focuses on the study of intelligent agents that operate autonomously in complex and changing environments, such as traffic, trading, or the internet. Here, intelligence is to be understood as the ability to perceive and act on the environment in a way that maximizes the chances of performing an intended task correctly. The field was significantly impacted by the study of *embodied cognitive science* [9] that provides a new framework to model and analyze both biological and artificial intelligent entities. *Creativity,* i.e. the capability to show ingenious behavior in unprecedented situations by relating them with other conceivable situations, can then be seen as a major aspect of intelligence.

During the characterization and development of Reinforcement Learning (RL) theory, many eminent challenges arose and are still of great consideration even today. For instance, in many real-world tasks, the agent only has partial scope of the environment with which it interacts and, to be able to learn, the agent then needs to take into account not only its current observation but also past ones to better define its current state. This process can be very memory-consuming and can slow down decision-making, therefore introducing the need to *synthesize* remembered experiences and learned state-action associations. Another important aspect is the balance between *exploitation* and *exploration*: RL is a continuous trial-and-error based learning, in which the agent has the choice either to focus on what it has already learned to accumulate rewards from known highly rewarding paths, or either discover new paths by taking unprecedented actions, thus risking to diverge from his highly-rewarding state-action space. A good trade-off is then to be found between these two strategies, which can be set for example through tuning of an exploration-parameter. Also, similarly to biological intelligent entities that relate what they already learned to understand faster how to perform new resembling tasks, *multi-task learning* is an important aspect in RL and learning new tasks shouldn't be linear in the number of training samples or computational-time. A good agent should be able to generalize its knowledge and skills to use it across a variety of tasks.

The notion of simulation, i.e. the power of physical systems to simulate other systems, has become a very important topic in modern physics, especially in the fields of quantum information and computation, pursuing the *universal quantum simulator* [10], which later inspired the *universal quantum computer* [11] and is capable of mimicking the time evolution of any other quantum system (i.e. simulating its Hamiltonian). *Projective Simulation* is a physically-oriented approach to embodied agent design and a natural candidate for quantization (i.e. generalization to a quantum algorithm) thanks to its underlying random walk process, a mathematical model allowing the agent to replay sequences of episodes from memory before taking a decision on its next action.

In our current global effort to create a large-scale universal quantum computer with interesting applications, showing robustness to the noise errors generated by specific implementations of a QC is crucial in the development of quantum algorithms. It helps establish a common ground of research between algorithmic theorist and practical experimentalists who then share their needs and constraints to seek more efficiently a path towards their shared goal of progress in the field.

2 Reinforcement Learning

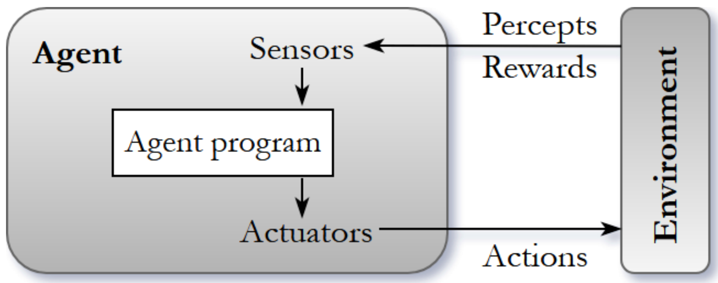


Figure 2.1: An agent is situated in an environment. It has sensors, through which it receives percepts and rewards from the environment, and actuators, through which it acts on the environment. Based on its perceptual input, the agent engages an internal processing and outputs an action.

The specificity of the Reinforcement Learning model, i.e. what distinguishes it from the simple agent/environment scenario, is the reward the environment sends back to the agent when it performs the "correct" action for a specified state and that it is supposed to use in order to learn.

2.1 Formal setting

Let us give a formal definition of the Reinforcement Learning model:

Definition 2.1. A reinforcement learning agent is defined by a sextuplet $(\mathscr{S},\mathscr{A},\mathscr{C},\Lambda,\mathscr{D},\mathscr{U})$ where:

- $\mathscr{S} = \{s_1,...,s_n\}$ is the set of percepts (which can also be seen as the environment's state) the agent receives from the environment through its sensors
- $\mathscr{A} = \{a_1,...,a_m\}$ is the set of actions the agent performs on the environment through its actuators
- $\mathscr{C} = \{c_1,...,c_p\}$ is the set of internal states of the agent
- $\Lambda = \{0,1\}$ is the set of rewards issued by the environment (we consider binary rewards for simplicity, but this can easily be generalized)
- $\mathscr{D} : \mathscr{S} \times \mathscr{C} \rightarrow \mathscr{A}$ is the decision function, i.e. the agent's internal program outputting an action for a given percept and state
- $\mathscr{U} : \mathscr{S} \times \mathscr{A} \times \Lambda \times \mathscr{C} \rightarrow \mathscr{C}$ is the update function which updates the agent's state based on its last state-action-reward sequence

A commonly used model for the environment is the *Markov Decision Process (MDP)*, which is a formal model for decision making problems. Its is basically an extension of a Markov chain (a stochastic model assessing that the probability of a future state is independent of the past given the present), the difference being the addition of actions that the agent can perform given a state, and rewards. This implies that the evolution of the environment from a certain state remains partly random, but is also impacted by the decision (i.e. action) taken by the agent at that state. The rewards are only here to enable learning.

2.2 Quantum Reinforcement Learning

Inspecting the RL model depicted in Fig. 2.1, one clearly sees a bipartite setting. When we talk of *Quantum Reinforcement Learning* we should then specify which part(s) of the whole RL setting is (are) indeed *quantum* and which remain classical.

Where should be the quantum? 3 scenarios:

1. **Quantum agent / Classical environment**:
   This is case of *quantum-enhanced AI* (or *quantized AI*), where the internal program of the agent is the only thing that processes quantum information. Here we are interested in quantum speed-ups of classical programs, currently mostly inspired by *Grover's algorithm* because these programs often imply search in an unsorted space in some sort or another. This is the case we'll be interested in when dealing with *Quantized Reflecting Projective Simulation* and for which we achieve a quadratic speed-up in the "thinking" time or *deliberation time* of the agent.

2. **Classical agent / Quantum environment**:
   This configuration is found when the environment is for example a quantum state or a set-up for a quantum experiment and the agent acts on it through measurements, laser interactions, or modifications of the set-up. This branch aims at developing protocols for Quantum Information Processing (QIP) (for example by means of *Measurement-based Quantum Computation* [12, 13]) and optimization of quantum state preparation and control.

3. **Quantum agent / Quantum (oracularized) environment**:
   This last branch studies the framework of a completely quantum agent, environment and agent/environment interaction. The environments in question would be *oracularized*, meaning that the agent would be allowed to query its environment in a quantum superposition. The framework is not easy to set, with multiple obstacles along the way, as for example the impossibility to store the complete history of states/actions of the agent/environment without destroying an entanglement or a superposition and not infringing the no-cloning theorem [14], thus very limiting when we want to record the performance of an agent.
   But V. Dunjko and H. Briegel resolved these issues in [15], achieving a quadratic speed-up in learning time (or number of interactions) for so-called *luck-favoring* settings (basically an agent that had luck interacting with an environment for t time steps will on average perform better in the future than an agent with the same model that had been unlucky during its interaction with the same environment for the same t time steps). This speed-up is again based on Grover-like search for a rewarded sequence of actions through interaction with the oracularized quantum environment, then *fast* training (this is when the luck-favoring aspect is important) of an *"artificially-lucky"* simulated classical agent to perform with the corresponding classical environment.

   V. Dunjko then recently developed some ideas for super-polynomial and exponential separations in learning times for *genuine* environments [16] in the same framework defined in the previous publication. This *genuineness* comes from three characteristics:

   a) delayed rewards and a large MDP rewarding diameter (minimal number of interactions between two rewards)
   b) the agent's actions genuinely influence subsequent states and the reward value
   c) optimal behavior explicitly depends on the state label and not just an interaction step counter

   The constructed task environments this time specifically benefit from oracles based on the *Recursive Fourier Sampling (RFS)* problem in the super-polynomial case, and from oracles based on Simon's problem in the exponential case, which are oracular problems with respectively super-polynomial and exponential separations between classical and quantum algorithms. These two cases in fact only serve as *proof of principle* for a general method (based on an underlying structure) to create oracles for quantum-enhanced RL with these types of speed-ups, yet to come.

2.3 Projective Simulation

Projective Simulation (PS) is a general reinforcement learning model first introduced in [17] that allows many additional features developed in [18, 19, 20] but which conceptual idea is to represent the agent's internal memory in a network of episodes (so called *clips*) built upon experience through simple creation of new percept-clips (once a new percept is perceived) or variation and composition of previous clips. The agent's deliberation will then consist in hopping randomly between clips until an action-clip is hit. This Episodic Compositional Memory (ECM) allows the agent to simulate itself into future situations before taking a real action by replaying clips representing previous situations during its deliberation.

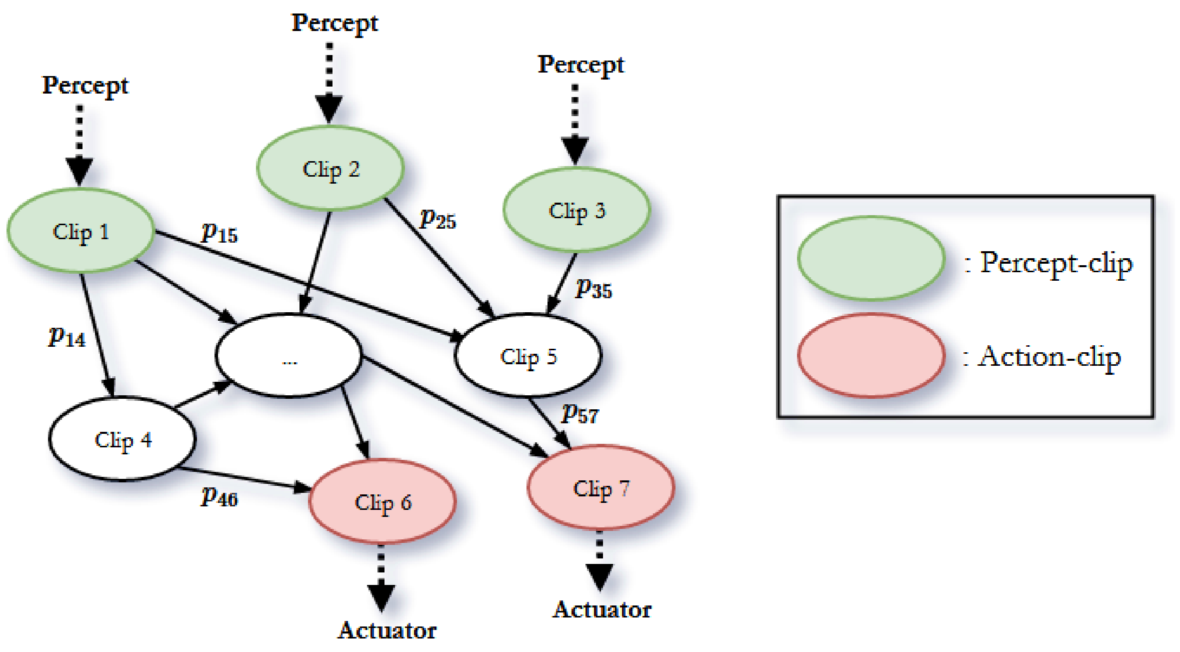


Figure 2.2: **The ECM network model.**

The ECM forms a connected weighted graph. How the the deliberation process works is that for a given perceptual input of the agent, the associated percept-clip is triggered, which launches a *random walk* through the ECM until an action-clip is hit and the associated action is then coupled to the actuator of the agent. Based on the reward of that action, the weights of the graph are adjusted (through *Bayesian updating*) in order to increase the probability of the traversed edges. Additional features come in three kinds:

- The first has to do with the updates in the weights of the ECM after a percept-action-reward sequence. The simplest weight update procedure would be to add the reward λ from that last sequence to the edges activated during the random walk that lead from the percept-clip to the action-clip while leaving the other edges unchanged. A first enhancement would be to add a *damping* parameter $\gamma \in [0,1]$, determining how much the agent should forget the previous weights; a way to balance the value given to past versus recent experience, independently from the last reward received. On a similar scheme, an *afterglow* damping mechanism (set by a parameter $\eta \in [0,1]$) applied on the edges (or possibly the clips) can help determine the number of elapsed percept-action-reward sequences since the edge/clip was last activated during a random walk, which then can be used to adjust its weight update. This last feature is especially useful in the case of a delayed-reward task when it takes several sequences for the agent to receive a positive reward.
- The second type deals with the creation of new clips in the ECM. We already talked about how new percept-clips are created every time a new percept is perceived; add to that the initially preset action-clips given to the agent and you already get a so-called "two-layered" ECM capable of learning how to perform simple tasks. But in a concern for efficiency of

learning more complex tasks, for which many similar percepts may lead to the same action or a same percept may lead to many similar actions, one should consider a way to synthesize these categorizations in order to speed-up deliberation. This can be done for example in the first case through composition of percept-clips into a fictitious intermediary clip and forcing the agent perceiving one of the percepts leading to the same action to hop by the new clip; or in the second case through creation of new action-clips by composition of similar and sufficiently rewarded (by a same percept) former action-clips. More elaborated methods, such as generalization [19], exist but we won't go through them here.

- One additional feature, providing an enhancement similar to the damping/glowing kind, and that will be of great interest for the quantum agent is called *reflection* with *flags* (or *emoticons*). This feature, based on recent experience, adds an emoticon (basically "good" or "bad") to transitions between percept-clips and action-clips as a way to add some sort of short-term memory to the agent. This latter is enhanced with the ability to reflect (meaning to perform another random walk on the ECM) multiple times until a promising (i.e. "good") action is found. Like the weights of the network, these flags are updated based on the reward of the last percept-action sequence. The agent initially starts with all flags as "good" and modifies them at each interaction step. Once all flags turn "bad" (indicating an environment change), they are all reset to "good" again.

The PS theory takes an approach to intelligence that is strongly behavior-based, with particular focus on *creativity* (i.e. the capability of dealing with unprecedented situations by relating them to previous experience), *planning* and *predicting*. It also puts strong emphasis on a few basic principles that come from the study of biological agents, such as *cognitive embodiment* (an agent is situated in a physical environment, with which it actively interacts), *adaptivity to unknown dynamic environments* and finally *homogeneity* (similarly to artificial neural networks, where cognitive units are modular and all arise as possible configurations of a few underlying systems).

3 Classical & Quantum Walks

An essential element of the PS model is certainly the deliberation process during which the agent browses through its memory, performing a simulation of its upcoming experience in search for the best action to output. This process consists in a random walk through the agent's internal stochastic network, the ECM. Fortunately there is a broad classical theory describing random walks and a quantum generalization of that theory providing a quadratic speedup over the former.

3.1 Classical Random Walks Complexity

Consider an *undirected non-stochastic* connected graph with N vertices, an ϵ-fraction of which are *marked*. The goal is to output a marked vertex. We start by defining a general *cost* of our algorithms, which can be seen as the number of queries (i.e. calls) to an oracular form of our graph or the number of elementary operations performed before halting; our objective being to get the smallest cost possible.
A basic classical random walk would be:

```
Start at some vertex y of the graph.
Repeat until halting:
    Check if y is marked.
    If so, output y. Otherwise update y to one of its neighbors at random.
```

Such a naive algorithm would only need to keep track of the current vertex y (supposing that the neighbors of any vertex are efficiently computable, insuring that there is no need to store the entire graph in memory) using space $O(\log(N))$ and if left to run for poly(N) steps will end up finding a marked element.

We now take a more global view on our graph and define a transition matrix P whose elements $P_{i,j}$ are the probabilities of our previous algorithm at current vertex i to take its next vertex to be j, forming a $N \times N$ symmetric matrix. For example, in the case of a d-regular graph without self-loops, each column (line) would have exactly d non-zero elements all equal to $1/d$ whose line (column) indices would correspond to those of its neighbors. By taking a probability distribution vector $v \in [0,1]^N$ (for example, at the initialization of our previous algorithm, v would have a 1 at position y and 0s elsewhere) one would be able to apply one step of random walk on that vector when multiplying it by P except that Pv would be a distribution over all possible outcomes of the random draw of the next vertex y. If we consider the vector v given as an example earlier and the case of a d-regular graph, Pv would be the uniform distribution over the neighbors of y (reached in one step), i.e. the line P_y of P.

We can then consider a very important property of a graph in the context of random walks: its *speed of convergence* (in number of applications of P) to the uniform distribution u. It is determined by the difference between the first and second largest (in absolute value) eigenvalues $1 = \lambda_1 > \lambda_2 \geq ... \geq \lambda_N$ of P, i.e. the *spectral gap* $\delta = 1 - \max_{i\geq 2}|\lambda_i|$ (λ_1 is clearly equal to 1 because it corresponds to the eigenvalue of the uniform distribution, and $\lambda_{i\neq 1} \in]-1,1[$).
Further calculation shows us that: $||P^k v - u||^2 \leq ||v||^2(1-\delta)^{2k}\ \forall v$, i.e. the closer δ is to 1 the quicker is the convergence to the uniform distribution u, regardless of the starting distribution v. Then, at uniform distribution, we have probability ϵ of hitting a marked vertex.
One may wonder that the same result could have been achieved without all this random walk process just by directly picking uniformly at random a vertex of the graph, but that may not always be possible, for instance if the graph is encoded only implicitly in its oracular form.

Let's define the cost **S** of setting up an initial state v, the cost **U** of one application of P and the cost **C** of checking if a given vertex is marked.
A classical random walk-based search algorithm would be:

```
Start with some arbitrary distribution v.
Repeat until hitting a marked vertex (roughly $1/\epsilon$ times):
    Run a random walk for roughly $1/\delta$ steps (updating v).
    Pick a random vertex y according to the distribution v.
    Check if y is marked.
```

The expected cost of such an algorithm before halting would then be in the order of:

$$\mathbf{S} + \frac{1}{\epsilon}(\mathbf{C} + \frac{1}{\delta}\mathbf{U}) \tag{3.1}$$

3.2 Quantum Walks Complexity

In this section we will present the so-called *Szegedy*-type quantum random walk, thoroughly developed in [21, 22], and which can be seen as a generalization of the classical algorithm preceding (3.1). The building block of the algorithm shifts from the distribution-preserving transition matrix P to the norm-preserving (i.e. a unitary) walk-operator $W(P)$ which doesn't act solely on one register that would represent the distribution over *current* vertices but on a two register system representing the distributions over *current* and *previous* vertices of our walk (i.e. a distribution over edges of the graph G). In fact, quantum operations act on Hilbert spaces and hence must be unitary, but a unitary on an N-dimensional Hilbert space $\mathscr{H}$ is not capable of representing all the entries of the transition matrix P (i.e. all transitions). For this reason, we must use two copies $\mathscr{H}_\text{I}$ and $\mathscr{H}_\text{II}$ of $\mathscr{H}$, the Hilbert space of vertices, to accommodate for all the required degrees of freedom.

To get an intuition on what this walk operator $W(P)$ does, we'll break it down as in Fig. 3.1:
Define $|p_x\rangle = \sum_y \sqrt{P_{xy}}\,|y\rangle$ to be the uniform superposition over the neighbors of a vertex x, and $\mathscr{A} = \text{span}\{|x\rangle_\text{I}\,|p_x\rangle_\text{II},\ x \in G\}$, $\mathscr{B} = \text{span}\{|p_y\rangle_\text{I}\,|y\rangle_\text{II},\ y \in G\}$ subspaces of $\mathscr{H}_\text{I} \otimes \mathscr{H}_\text{II}$. Then ref($\mathscr{A}$) (ref($\mathscr{B}$)) denotes the reflection over subpsace $\mathscr{A}$ ($\mathscr{B}$), i.e. $\forall v \in \mathscr{A}$ ref$(\mathscr{A})v = v$ and $\forall w \in \mathscr{A}^\perp$ ref$(\mathscr{A})w = -w$.
ref($\mathscr{A}$) is built with the *diffusion unitary* U_P, acting on the two register system I/II (but modifying only register II), and that can be seen as one step of the classical random walk. Homologously, ref($\mathscr{B}$) is built with the *diffusion unitary* V_P, acting on the two register system I/II (but modifying only register I), and can be seen as one step of a classical random walk but when time is *reversed*, which is exactly the same as a classical random walk step when we have an undirected graph (symmetric transitions) but in the case of a non-symmetric graph, one should picture transitions made from the end to the origin of edges (a more complete characterization is given at the end of this section).
The actions of U_P and V_P are given by:

$$U_P\,|x\rangle_\text{I}\,|0\rangle_\text{II} = |x\rangle_\text{I}\,|p_x\rangle_\text{II} \tag{3.2a}$$

$$V_P\,|0\rangle_\text{I}\,|y\rangle_\text{II} = |p_y^*\rangle_\text{I}\,|y\rangle_\text{II} \tag{3.2b}$$

where $|p_y^*\rangle$ are *reversed-time* neighbors of y, and turn out to be $|p_y\rangle$ in the symmetric case (then V_P can be obtained from U_P by swapping registers I and II before and after an application of U_P).
To get the reflection ref($\mathscr{A}$) (ref($\mathscr{B}$)), one needs to apply $U_P^\dagger$ ($V_P^\dagger$) over the two register system I/II, then a reflection D_0 over the state $|0\rangle_\text{II}$ ($|0\rangle_\text{I}$) and finally apply the inverse operation U_P (V_P). This can be seen as first "undoing" one step of the classical walk (in normal time for ref($\mathscr{A}$) and reversed time for ref($\mathscr{B}$)) so all edges between a vertex and its neighbors $|x\rangle\,|p_x\rangle$ are transformed into the "empty" state $|x\rangle\,|0\rangle$ (while other edges are displaced elsewhere), we then reflect over that state, and finally we perform back the step of the classical walk giving us a state where all $|x\rangle\,|p_x\rangle$ have a negative phase added to them (they are marked).
Overall, the sequential application of ref($\mathscr{A}$) and ref($\mathscr{B}$) (giving $W(P)$), because it highlights both normal-time and reversed-time edges, performs one step of *quantum* walk.

Our resulting quantum walk-based search algorithm will actually be very similar to the classical one, with some more convenient aspects: we can encode a distribution over vertices/edges in a quantum superposition and measuring that superposition naturally performs a random sampling; but also some constraints: measurements collapse quantum superstitions. But most importantly, the central problem for both cases is still a *search* over an ensemble of vertices and fortunately QIP provides us with *Grover's search algorithm*, from which we are going to gain a first speed-up on the ϵ-dependency.
We start by defining the superpositions over "good" states (for an edge $|x\rangle_\text{I}\,|y\rangle_\text{II}$ with a vertex $x \in M$, the set of marked vertices) and "bad" states (other edges):

$$|G\rangle = \frac{1}{\sqrt{|M|}} \sum_{x\in M} |x\rangle_\text{I}\,|p_x\rangle_\text{II} \text{ and } |B\rangle = \frac{1}{\sqrt{N-|M|}} \sum_{x\notin M} |x\rangle_\text{I}\,|p_x\rangle_\text{II} \tag{3.3}$$

And the uniform distribution over all edges:

$$|U\rangle = \frac{1}{\sqrt{N}} \sum_x |x\rangle_\text{I}\,|p_x\rangle_\text{II} \tag{3.4}$$

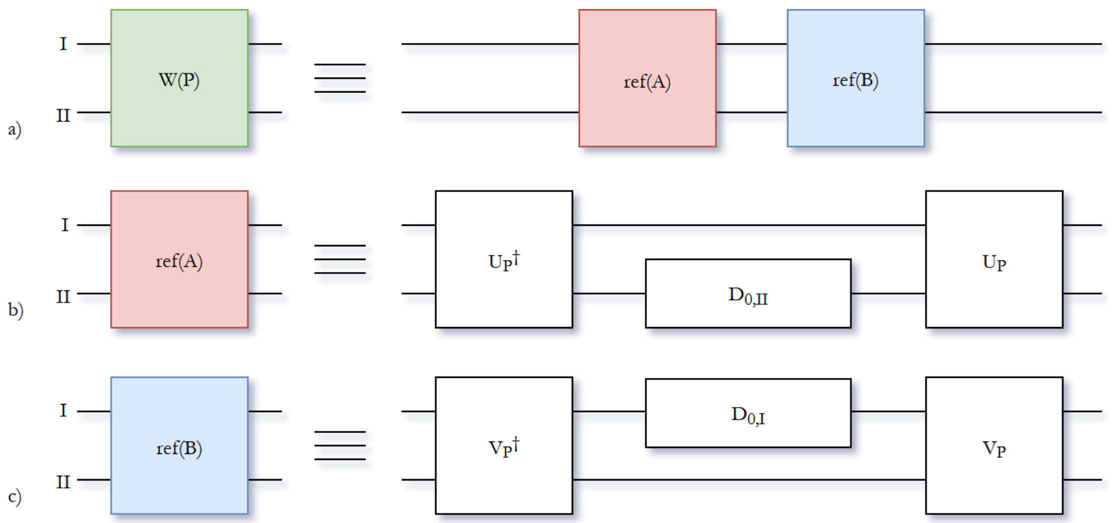


Figure 3.1: **Szegedy Walk operator.**

Then the algorithm can be written:

```
    Start by setting up the state $|U\rangle$.
    Repeat roughly $1/\sqrt{\epsilon}$ times:
        Reflect over $|B\rangle$.
        Reflect over $|U\rangle$.
    Measure the register I and check if the resulting vertex x is marked.
```

Reflection over $|B\rangle$ is quite straightforward. What needs more explanation is the construction of the reflection over $|U\rangle$. For this we use a nice property of the walk operator $W(P)$: it has a unique eigenvalue-1 eigenstate, which is $|U\rangle$, and its other eigenvalues are of the form $e^{\pm 2i\theta_i}$ (where $\theta_i = \arccos(\lambda_i)$ and λ_i are the eigenvalues of the associated transition matrix P), verifying $|\theta_i| \geq \sqrt{2\delta}$. Then all we need to do is use Kitaev's *phase detection algorithm* [23] to perform an *approximate reflection* over the uniform distribution as depicted in Fig. 3.2: it will, through controlled applications of powers of $W(P)$, encode the phases θ_i of the eigenvalues $e^{\pm 2i\theta_i}$ in auxiliary qubits, thereby entangling them to the encoded distribution in registers I/II, then reflect over phase-0 (i.e. eigenvalue-1) in the auxiliary qubits, and finally undo the phase detection. Because the non-zero phases $|\theta_i| \geq \sqrt{2\delta}$, approximating the phases of the eigenvalues within $\sqrt{\delta}/2$ (by using $\log_2(1/\sqrt{\delta})+1$ auxiliary qubits and as many controlled-W^{2^k} operations) is good enough.

Again let's define the cost **S** of setting up an initial state $|U\rangle$, the cost **U** of one application of U_P, V_P or their inverses and the cost **C** of reflecting over $|B\rangle$. The expected cost of our quantum walk algorithm would then be in the order of:

$$\mathbf{S} + \frac{1}{\sqrt{\epsilon}}(\mathbf{C} + \frac{1}{\sqrt{\delta}}\mathbf{U}) \tag{3.5}$$

Now let's point out that in the case of our ECM in the RPS model, our graph is neither undirected nor non-stochastic (not even connected but we'll work on connected percept-specific sub-graphs, constructed according to the procedure shown in Fig. 3.3). It is actually an *ergodic* (*irreducible* and *aperiodic*) and usually *time-reversible Markov Chain*. Fortunately, our results (3.1) through (3.5) still hold, and we just need to replace in our reasoning *uniform distribution* by *stationary distribution* defined as the unique (because of the ergodicity) eigenvalue-1 eigenvector of the transition matrix P of the MC, to which we necessarily converge after a certain number of applications of P, called the *mixing time* (homologous to the speed of convergence we had before), fixed by the

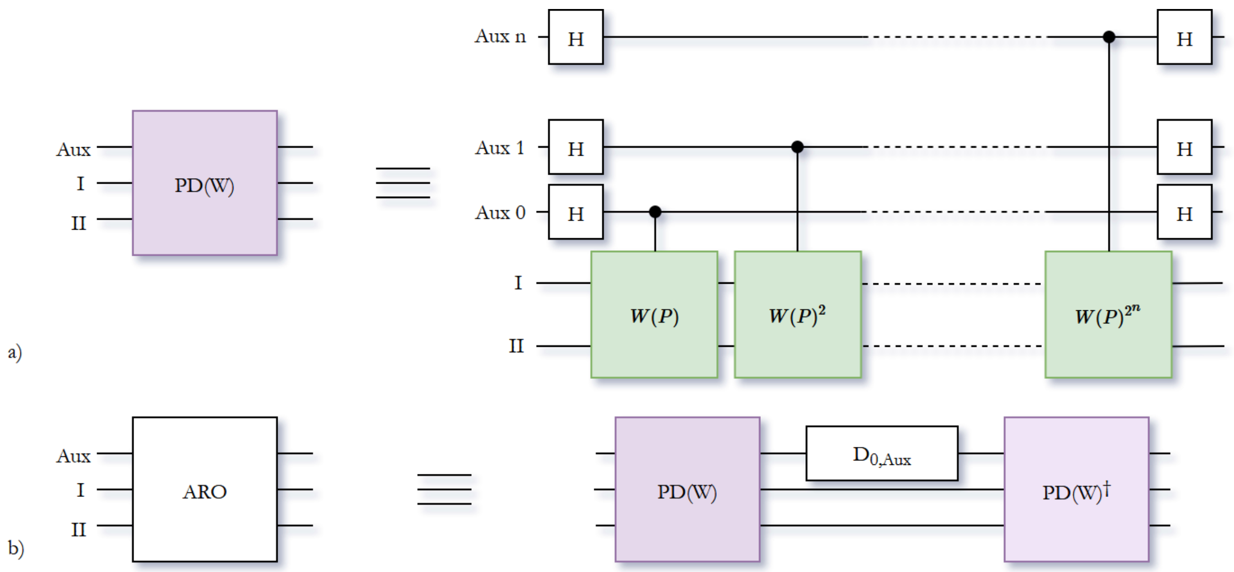


Figure 3.2: **Kitaev's Phase Detection algorithm and the Approximate Relfection Operator.**

spectral gap (in the order of $O(1/\delta)$ for the classical case and $O(1/\sqrt{\delta})$ for the quantum case). For certain reasons, explained in Sec. 4.3, we will loose the time-reversible property of our MC at some point. In that case we need to define the time-reversed transition matrix P^*, whose elements are $p^*_{ij} = p_{ji}(\pi_P)_i/(\pi_P)_j$ (where π_P denotes the stationary distribution and p_{ji} the elements of P), needed to construct the time-reversed diffusion operator V_P. This construction reproduces exactly the picture introduced earlier of taking the edges from target-endpoint to source-endpoint, but because we don't have a uniform distribution over the vertices anymore, we weight the vertices according to the stationary distribution π_P, thus re-calibrating the edges accord to the ratio of stationary probabilities of the endpoints.

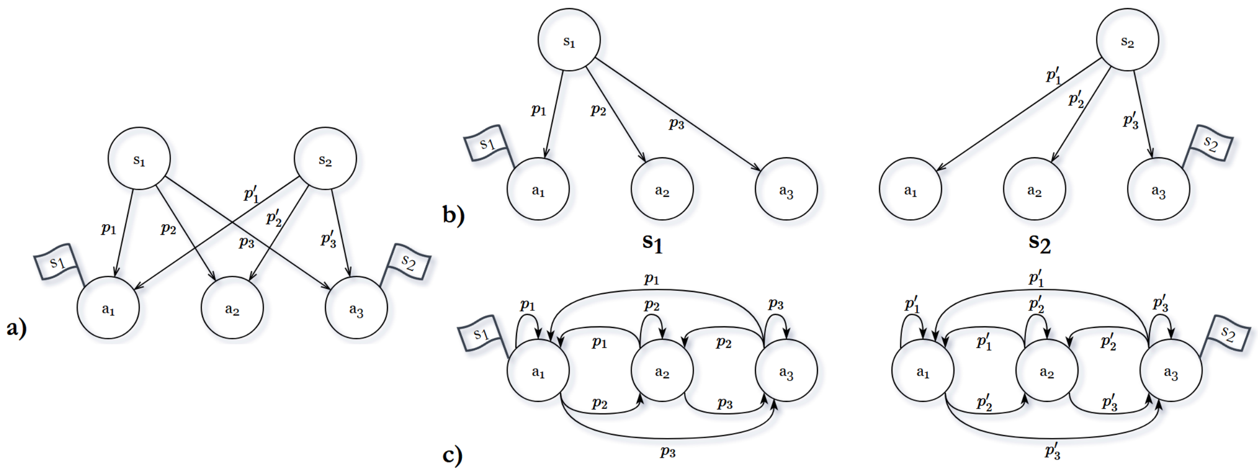


Figure 3.3: **Construction of the RPS percept-specific sub-graphs.**
a) is the original PS model with two percept-clips and three action-clips, supplemented with percept-specific flags. **b)** is an intermediary construction step where we break up the graph a) into two graphs by duplication of action-clips. **c)** is the final construction of our RPS model. To each percept-clip we assign an MC over the action-clips only, such that its stationary distribution recovers the initial output probabilities.

4 Quantized Reflecting Projective Simulation

4.1 Testbed application: Invasion Game

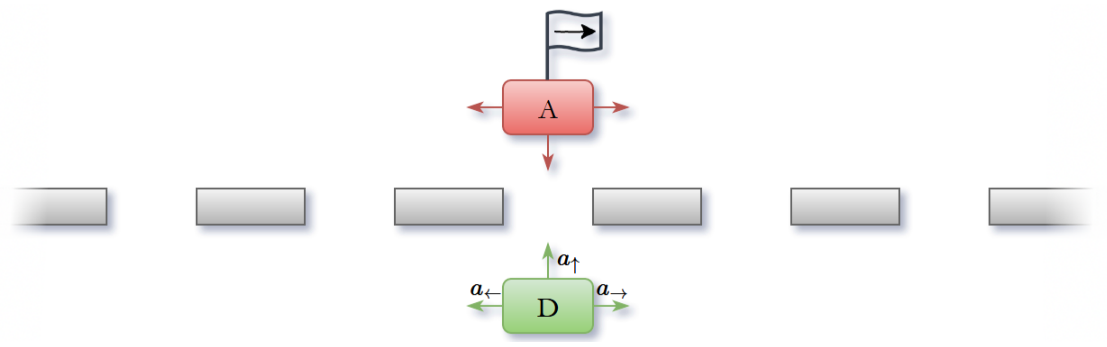


Figure 4.1: **Invasion game.**

In order to show the speed-up of our quantum algorithm, we choose a very simple RL game which presents enough degrees of freedom to be able to work on solely a few qubits: the *Invasion game*, depicted in Fig. 4.1. In this game, the task of the Defender (i.e. the agent) is to guard a region of space against an Attacker (i.e. the environment) that tries to enter the region through a series of openings. The game takes place in rounds, during which the Attacker has three possible moves: move left, move right or enter at its current position; and the Defender attempts to block it by matching its opponent's move. For there to be a learning aspect in this game, we assume that the Attacker *hints* the agent by sending him a signal prior to its move, indicating its next location. Of course the Attacker is allowed to have a strategy and change the meanings of his hints during the game (forcing the agent to adapt).
The set of percepts of the agent are then $\{\uparrow, \leftarrow, \rightarrow\}$ and its set of action-clips are $\{c_1(a_\uparrow), c_2(a_\leftarrow), c_3(a_\rightarrow)\}$, which leaves us with a *two-layered* ECM network, where every percept-clip is directly linked to all the action-clips with no intermediary clips. Two classical bits (or two qubits) are then sufficient to represent all the action-clips. And for there to be a meaning to all the possible two-bit values, we assign the natural binary integer encodings to their respective action-clips and additionally assign the value 11_2 to action-clip $c_3(a_\rightarrow)$.

4.2 Rank-one Reflecting PS

The aim of the following part is to show how our quantum-enhancement allows to keep one of the good features of PS, its *reactivity to environment changes*, even when adding an additional structure (Reflecting PS with flags, explained in Sec. 2.3) that increases learning efficiency but induces longer deliberation times, especially in the case of changes in the environment.

Let us set the context and explain this further:
We consider the case of a "two-layered" RPS agent, where each transition matrix P_s associated to the MC of a percept-specific sub-network G_s of the ECM G has rank one. We focus on the sub-network associated to one percept (an then for simplicity call its transition matrix P) but we would get analog results for the other percepts.
This rank-one property leads to many simplifications:

- The columns of P are all identical, equal to the stationary distribution, so the unitaries U_P and V_P are equal (disregarding the swapping of registers I/II) and act the same way on all clip states
- P has only one eigenvalue (+1) and one eigenvector (the stationary distribution) which leads to a spectral gap $\delta = 1$
- The Markov chain mixes in one step
- The Szegedy Walk operator becomes $W(P) = UD_0U^\dagger$ and is hermitian, then exact reflection over the stationary distribution can be achieved by applying $W(P)$ once on only one register

These simplifications will shorten considerably our quantum circuit for the associated simulations, removing the need for ancilla qubits and for controlled-operations, which is of great benefit in the number and complexity of the performed operations.
In these simulations, we deal with the simple "Invasion game" explained in Sec. 4.1 with a set of three percepts and three actions. We define the stationary action probabilities $\pi = (\pi_1, \pi_2, \pi_3)$ and add the *flags* (or *emoticons*) structure to the network, increasing the learning efficiency of the agent but inducing longer deliberation times to output an action because of the multiple reflections the RPS agent has to go through to output a *flagged* action. This number of reflections is determined by $\epsilon = \sum_{i \in F} \pi_i$ the relative probability of flagged actions within the stationary distribution, scaling as $\tilde{O}(1/\sqrt{\epsilon})$ for the quantum RPS agent and as $\tilde{O}(1/\epsilon)$ for the classical RPS agent.

To exhibit the better reactivity of quantum RPS to environment changes, which happens to be the case when epsilon is the smallest (so when the difference in the dependencies on ϵ is the largest), we deal with a change in the strategy of the adversary (i.e. the environment) consisting in a permutation of the meanings of its hints (i.e. the agent's percepts). Suppose the adversary pursues a consistent strategy (i.e. gives the same hints, or percepts, to the agent before each of its moves) for a long period of time. The agent would have learned well, which means that for a given percept he would have associated with high probability an action (i.e. $\pi_3 >> \pi_1 + \pi_2$ for example) and only the action-clip associated to π_3 would be flagged in the sub-network associated to that percept. Then the adversary changes his strategy, which means that now the flagged actions are reset (after one unsuccessful trial on the action associated to π_3), and become the action-clips associated to π_1 and π_2, with cumulated probability $\epsilon = \pi_1 + \pi_2 << 1$.

From here we are all set for the first part of our simulations, presented in Sec. 5.3.

4.3 Rank-two Reflecting PS

We showed in the previous section how our quantum-enhancement of the RPS model allowed to keep the good reactivity of the agent to environment changes. Let us now show another good feature of the RPS model that the quantum-enhancement preserves: the *good interplay between planning and real-time action selection.*
Indeed, in a more general case than the rank-one RPS, the ergodic MCs of the percept-specific subnetworks of the ECM don't completely mix in one step. This is because the *mixing time* of the MC (the number of applications of the transition matrix P on an initial distribution to get a "good" approximation of the stationary distribution) is inversely dependent on the spectral gap $\delta = 1 - |\lambda|$ (where λ is the largest eigenvalue of P in absolute value after 1). But this mixing time is precisely the planning in the future that the agent performs as, the longer the MC is mixed, the better the stationary distribution is approximated and the more accurate is the planning of the agent. The mixing time or the number of application of P (or $W(P)$ for the quantum agent) is determined by the spectral gap δ scaling as $\tilde{O}(1/\sqrt{\delta})$ for the quantum RPS agent and as $\tilde{O}(1/\delta)$ for the classical RPS agent.

To show these dependencies, we consider a transition matrix P with eigenvalues 1 and λ only, with λ set arbitrarily through a transformation of the rank-one transition matrix we had above:

$$\begin{pmatrix} p_1+\lambda & p_1 & p_1 & p_1 \\ p_2-\lambda & p_2 & p_2 & p_2 \\ p_3 & p_3 & p_3 & p_3 \\ p_4 & p_4 & p_4 & p_4 \end{pmatrix} \tag{4.1}$$

where (p_1, p_2, p_3, p_4) refers to the stationary distribution in the previous case.

Then the simplifications made in the first part disappear, we need to define different diffusion unitaries (corresponding to U_P) U_1 and $U_{\{2,3,4\}}$ for the first and second, third and fourth percepts respectively and apply the general Szegedy quantum walk operator and Phase Estimation algorithm as they were presented. The stationary distribution becomes $(\frac{p_1}{1-\lambda}, p_2 - p_1 \times \frac{\lambda}{1-\lambda}, p_3, p_4)$. The constraints we have on λ (supposing it is positive) are:

$$\begin{cases} \frac{p_1}{1-\lambda} \leq 1 \\ p_2 - p_1 \times \frac{\lambda}{1-\lambda} \geq 0 \\ p_1 + \lambda \leq 1 \\ p_2 - \lambda \geq 0 \end{cases} \tag{4.2}$$

Which sum up to $\lambda \in [0, min(1-p_1, p_2, \frac{p_2}{p_1+p_2})]$. That can be further simplified if we take $p_1 < p_2$ and $p_1 + p_2 < 1$ to $\lambda \in [0, p_2]$. We now have a way to set λ arbitrarily close to 1 (i.e. δ close to 0).

In order to get rid of the ϵ-dependency in our simulation, we make it equal to 0.25 (by marking only the action-clip c_1 and setting $\frac{p_1}{1-\lambda} = 0.25$ in the stationary distribution), which induces the need for only one application of the two reflections of Grover's algorithm to output the marked action-clip with probability 1.

It is important to notice that this rank-two case will never happen on a two-layered case (at least not in the symmetric way in which we define the percept-specific subnetworks in the RPS model), this is just a simple construction used as a proof-of-principle that allows to set the spectral gap δ arbitrarily close to 0. However, certainly these values of the spectral gap will appear in arbitrary "higher"-layered cases.

5 Implementation & Computer Simulations

5.1 Coherent Controlization

As mentioned in Sec. 3.2, the implementation of the unitary quantum walk-operator $W(P)$ requires the use of two copies $\mathcal{H}_{\mathrm{I}}$ and $\mathcal{H}_{\mathrm{II}}$ of the N-dimensional Hilbert space $\mathcal{H}$ over vertices of the MC associated to the ECM. For its construction, we use the underlying diffusion unitaries U_P and V_P given by Equations (3.2). For simplicity, let us just consider the unitary U_P, but a similar construction can be done for V_P.

How U_P works is that, conditioned on the value of the register I (the clip $|c_i\rangle_{\mathrm{I}}$), the unitary U_i (which first column is the square root of the i^{th} column of the transition matrix P associated to the MC) is applied to the register II, first initialized at $|0\rangle_{\mathrm{II}}$. Which gives:

$$U_P |c_i\rangle_{\mathrm{I}} |0\rangle_{\mathrm{II}} = |c_i\rangle_{\mathrm{I}} U_i |0\rangle_{\mathrm{II}} = |c_i\rangle_{\mathrm{I}} \sum_{j=1}^{n} \sqrt{P_{ij}} |c_j\rangle_{\mathrm{II}} \tag{5.1}$$

But because the register I can be in a quantum superposition of different clips c_i (and we want to keep that superposition), the respective unitaries U_i also need to be applied conditionally on each clip in the superposition. To implement such a mapping, we need a physical procedure that enables quantum control of individual elementary operations. Fortunately we can use *coherent controlization*, depicted in Fig. 5.1, and which basically recursively decomposes the application of U_P in elementary controlled rotations.

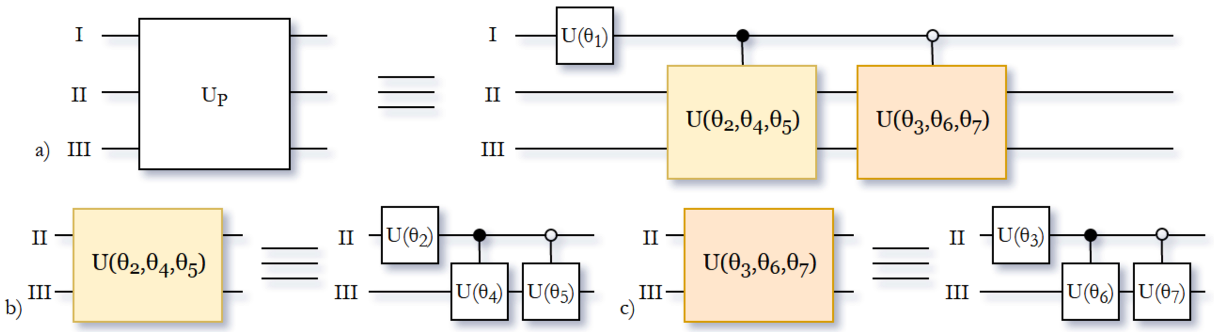


Figure 5.1: **Coherent controlization.**

The implementation shown in Fig. 5.1 shows the construction for a three-qubit (i.e. 8-clip) probability unitary, which is here only to show what we mean by *recursion*. Let us instead start with the simplest single-qubit (i.e. 2-clip) probability unitary: for this one it is sufficient to define a single-qubit Y-rotation $U(\theta_1)$ with $p_1 = \cos^2(\theta_1/2)$ and $p_2 = \sin^2(\theta_1/2)$ (thus setting the parameter θ_1) the probabilities of transition from clips 1 to 2 and 2 to 1 respectively. Setting the state $|0\rangle$ ($|1\rangle$) as the state of percept 1 (2), then applying $U(\theta_1)$ on the qubit performs one step of the classical random walk on our 2-clip network.
We can then extend this construction to the two-qubit (i.e. 4-clip) probability unitary $U(\theta_1, \theta_2, \theta_3)$ with probability distribution $\{p'_1, p'_2, p'_3, p'_4\}$ by starting again with the unitary $U(\theta_1)$ applied to the first qubit where $p_1 = p'_1 + p'_2$ and $p_2 = p'_3 + p'_4$, and followed by two controlled Y-rotations on the second qubit ($U(\theta_2)$ and $U(\theta_3)$), conditioned on the value of the first qubit being respectively $|0\rangle$ and $|1\rangle$. We set the parameters θ_2 and θ_3 with $\cos^2(\theta_2/2) = p'_1/(p'_1 + p'_2)$ and $\cos^2(\theta_3/2) = p'_3/(p'_3 + p'_4)$. All together this forms the correct probability unitary. This last construction is the one depicted in Fig. 5.1.b), given we reset the indices of the θ_is correctly.
Further extension to a higher number of qubits is possible through recursion of this construction, but in our simulation we work only with 4-clip networks, so the two-qubit construction is enough.

5.2 Trapped ions implementation *first order* error model

At this stage of advancement in the field of Quantum Computation, showing the feasibility of the implementation with specific architectures is a very important aspect of algorithmic development. How this is done is through embedment of error models in the basic constitutive quantum operations of the computational realization in question, designed to mimic the sources of errors in real physical systems. The parametrization of these error models and the characterization of optimal quantum circuit constructions comes from the interplay between computer science theorists and physical experimentalists, both applying their expertise and sharing their needs and constraints for their common goal of robust practical QC applications.

The particular implementation we are going to focus on is the *Trapped ions universal quantum computer*, where cooled ions are confined and suspended in free space using electromagnetic

fields. Qubits are encoded in the nuclear spin states of individual ions and they interact via the lowest quantized collective vibrational modes (i.e. phonons) of the conjointly trapped ions. Arbitrary transformations can be achieved through application of lasers on the ions to induce coupling between the qubit states (for single-qubit operations) or coupling between the internal qubit states and the external motional states (for entanglement between qubits).
In order to perform arbitrary quantum operations, one only needs to be able perform a finite set of quantum gates, called a *universal set of quantum gates*. One universal set of particular interest in our implementation with trapped ions is composed of the two-qubit C-Z gate (a controlled phase-flip operation where the first qubit controls if the phase of the second qubit is flipped) and single-qubit $R_X(\theta_X)$ and $R_Y(\theta_Y)$ rotations by arbitrary angles θ_X and θ_Y about the X-axis and Y-axis respectively (forming both together a universal set of single-qubit operations). All of these are performed with laser pulses, of particular intensity and duration, aimed at ions individually or collectively. The C-Z gate is actually further decomposed into the construction given in Fig. 5.2.b), taken from [24], and based on the use of the Mølmer-Sørensen (MS) gate, more natural to implement and whose matrix representation is:

$$MS(\chi) = \begin{pmatrix} \cos(\chi) & 0 & 0 & -i\sin(\chi) \\ 0 & \cos(\chi) & -i\sin(\chi) & 0 \\ 0 & -i\sin(\chi)) & \cos(\chi) & 0 \\ -i\sin(\chi) & 0 & 0 & \cos(\chi) \end{pmatrix} \tag{5.2}$$

This way, all our gates are characterized by an angle parameter.

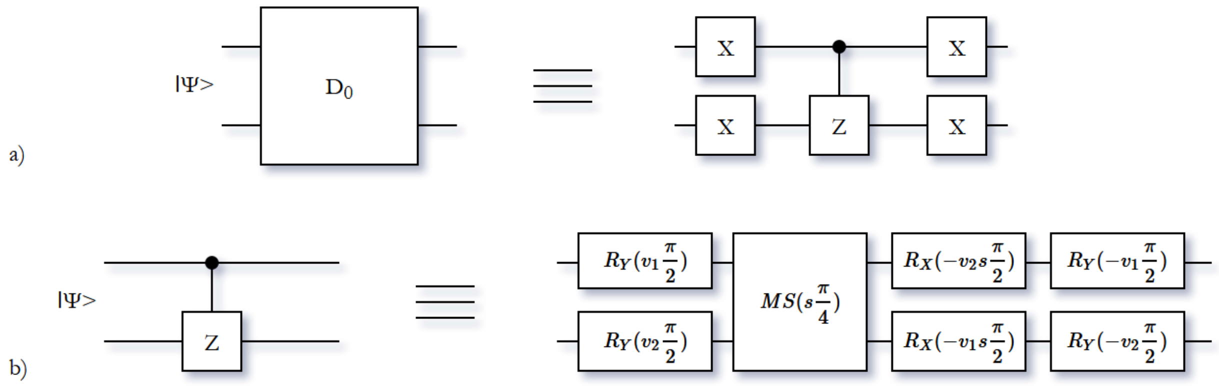


Figure 5.2: **Construction of the D_0 reflection over state $|00\rangle$ (a) and of the C-Z operation (b).** Up to an irrelevant global phase, the X-gate is equal to $R_X(\pi)$

Defining an error model close enough to what happens in real physical systems can be complicated, and usually there is a need to consider certain simplifications so that the computer simulations remain computationally feasible. For that matter we only consider a *first order approximation* error model, for which the characterization of gates by angle parameters only is of great help. We assume that, for each operation of our universal set that we perform, the time (i.e. the phase) of the laser pulses jitters, that is the angle parameter jitters, but remains correctly centered on its axis (we neglect errors on other axes than the one concerned with our operation). We then assume that this jitter is distributed according to a Gaussian with some variance σ. So the actual realized angle is sampled from the correctly centered Gaussian distribution, with its variance quantifying the amount of noise of the system.
By decomposing the gates in the circuits of Figs. 5.1.b), 3.1 and 3.2 (H = $R_Y(-\pi/2)R_X(\pi)$) according to the construction of Fig. 5.2 and the use of $MS(\chi)$, $R_X(\theta_X)$ and $R_Y(\theta_Y)$ gates, we can then

perform our simulations with the error model we described above. Results are given in the next section.

5.3 Results

In order to show the dependencies on parameters ϵ and δ for the quantized case we run two sets of simulations (inspired by [25]) but which are inherently very similar.
The first set of simulations (based on the rank-one RPS, Sec. 4.2) focuses on the parameter ϵ, chosen to be $\epsilon = p_1 + p_2$ (only the action-clips 1 and 2 are flagged) and that we can set arbitrarily close to 0. We first initialize our register to the stationary distribution $|\pi_P\rangle = U(\theta_1, \theta_2)|0\rangle$, then chose a random integer $m \in [\![0, m_\epsilon]\!]$ (where $m_\epsilon = \lceil 1/\sqrt{\epsilon} \rceil$) and apply the operations ref($\mathscr{A}$) and $UD_0U^\dagger$ together (from the quantum walk algorithm in Sec. 3.2) a total of m times. We then measure our register in the clip basis (the computational basis here). If the resulting clip is not marked then we pick another random m and perform the whole procedure again until a marked action-clip is hit. Along the process we count the number N_U of applications of the unitary U. To build up statistics, for each value of ϵ we perform this whole procedure with 10^3 agents. The linked dots we can see on Fig. 5.3 are the average numbers of application N_U of all these agents for each value of ϵ and the vertical bars represent the standard deviation of each batch. The step-like pattern we can distinguish in the evolution of the data comes from the integer steps the parameter m_ϵ takes as ϵ decreases.
We also perform a regression of our curve linearly in $1/\sqrt{\epsilon}$ (represented in orange) to show the resulting dependency, which turns out to have a very good coefficient of determination R^2 (very close to 1).

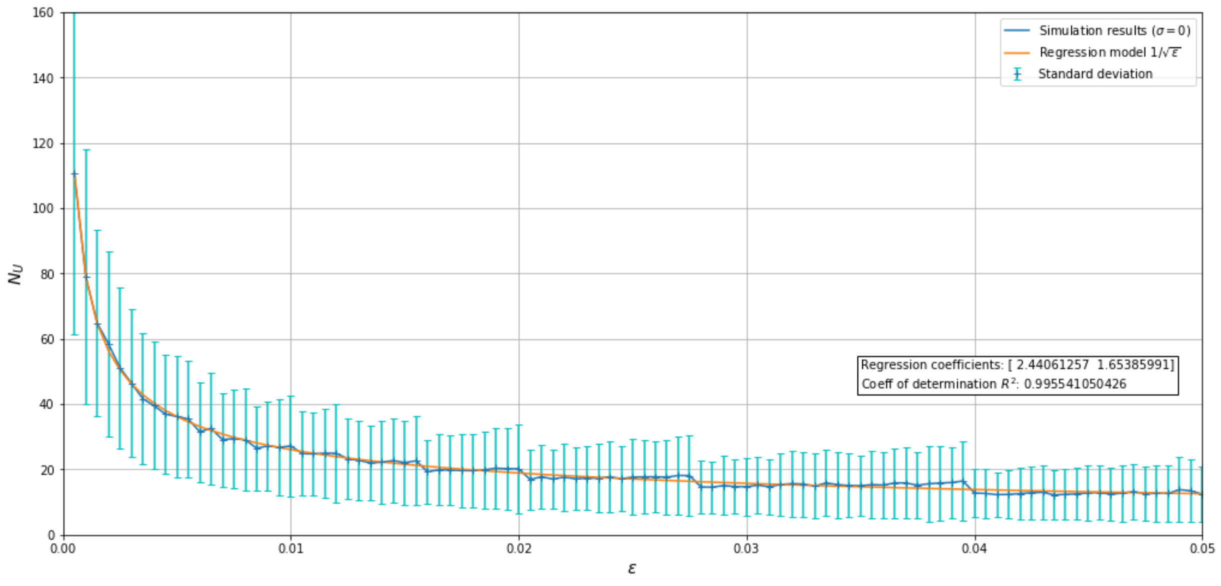


Figure 5.3: **Numerical simulation with no error model to show the $O(1/\sqrt{\epsilon})$ dependency.**

In order to show the quadratic speed-up of our quantized RPS over the classical one even better, we emulate the latter by running the same process previously described but with $m_\epsilon = 0$ constantly, that is preparing the state $U|0\rangle$ then measuring it, repeatedly until a marked action is hit, while counting the number N_U of applications of the unitary U along the process. This allows us to draw the comparison graph shown in Fig. 5.4, again with regressions of our curves linearly in $1/\sqrt{\epsilon}$ for quantized RPS and in $1/\epsilon$ for the classical one to show the resulting dependencies.

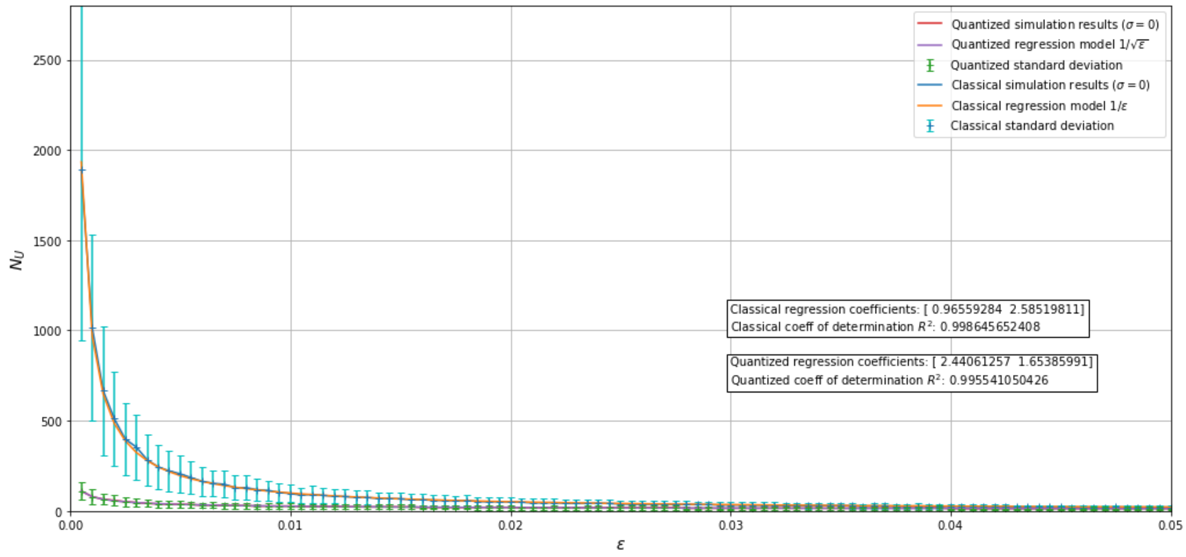


Figure 5.4: **Comparison of numerical simulations without error model of classical and quantized RPS.**

Now the second set of simulations (based on the rank-two RPS, Sec. 4.3) focuses on the parameter δ, only this time all the simplifications of the circuit we had for the previous case disappear and we need to apply it exactly as it is depicted in Figs. 3.1 and 3.2. This means that the simulations become a lot more computationally expensive (especially because of the number of ancilla qubits used, equal to $\log_2(1/\sqrt{\delta})$+1, and the difficulty to simulate operations on a large number of qubits) implying that we cannot run them for very small values of δ.
But it is possible to get more points in our graph without simulating our algorithm for smaller values of δ. All we need to do is produce a better granularity of our δs. In order to do so, we don't take only integer values of n (formally the number of ancilla qubits and corresponding to 2^{n+1} calls to the Szegedy walk operator $W(P)$). What we do is we take integer values of the number of calls c to $W(P)$, and the *logical* number of ancilla qubits associated to that value will be $k = \log_2(c)$-1 (so actually $\lceil \log_2(c) - 1 \rceil$ ancilla qubits are used because we need an integer number of qubits). Then all we have to do is not to allow a number of calls to $W(P)$ that is greater than c in the Phase Detection algorithm (Fig. 3.2.a)), as if we would have taken k ancilla qubits.

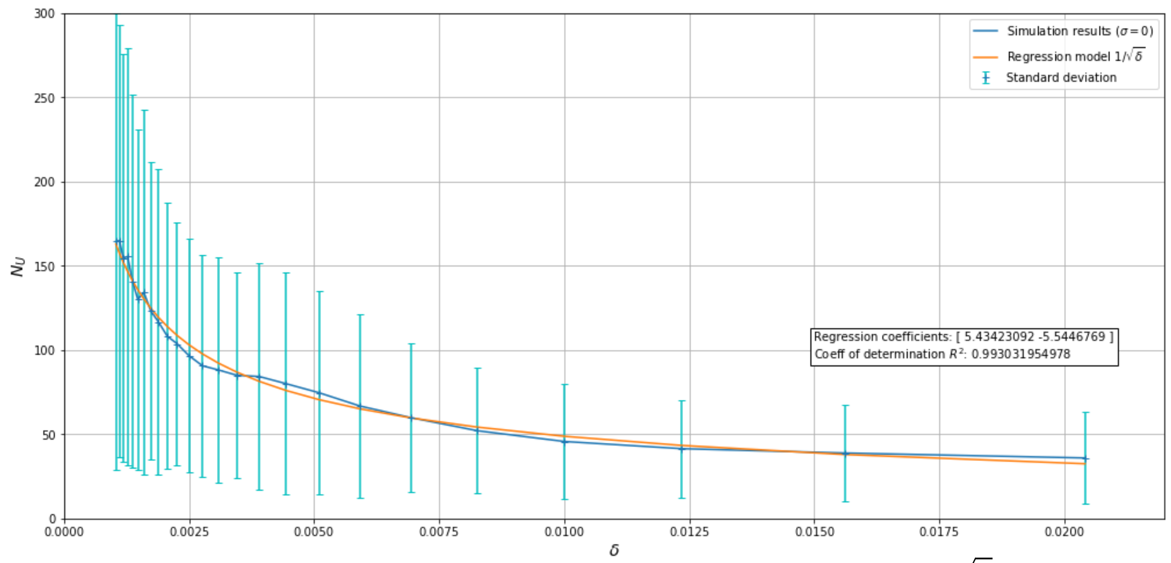
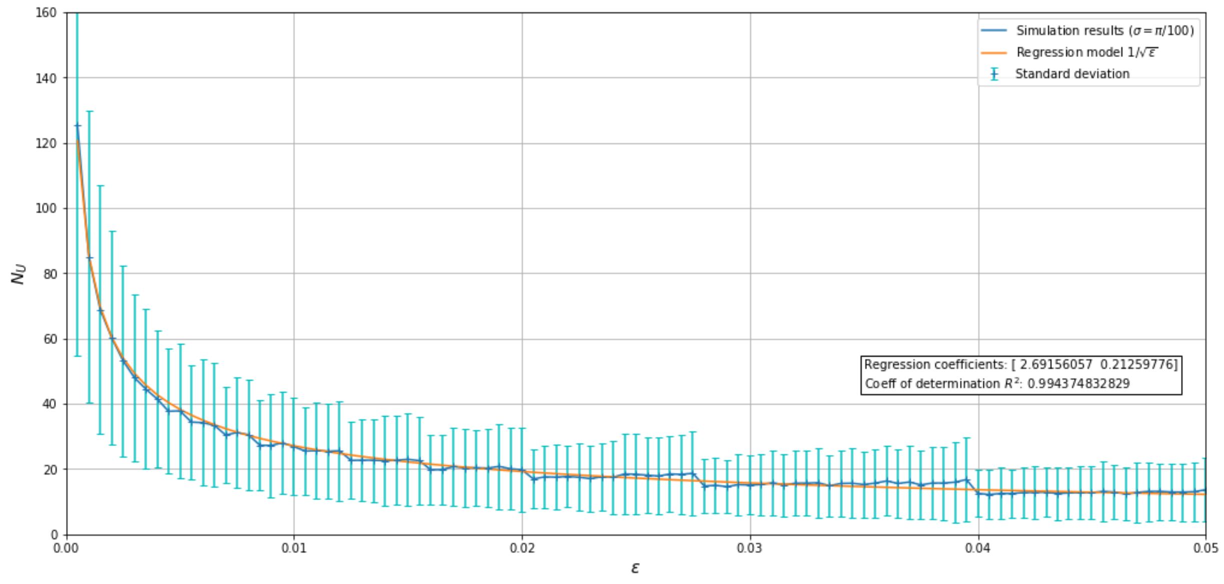


Figure 5.5: **Numerical simulation with no error model to show the $O(1/\sqrt{\delta})$ dependency.**

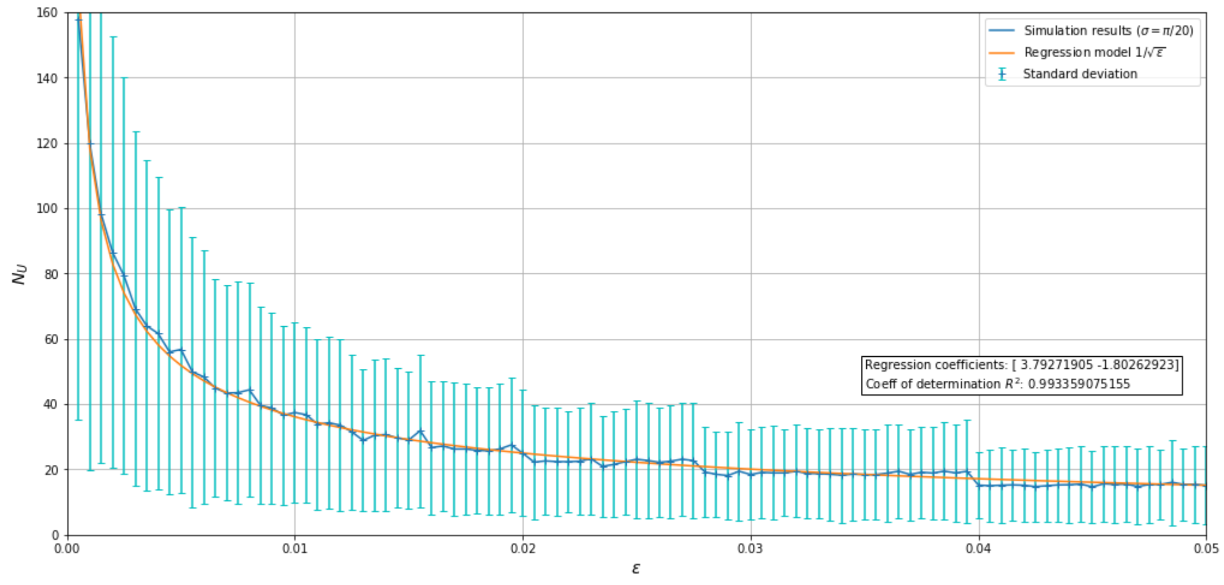
Finally we apply the Grover "one-shot" amplitude amplification algorithm that, as the name indicates, needs only one application of the two reflections (over the flagged action-clips of cumulated probability $\epsilon = 0.25$ and over the stationary distribution, as in the quantum walk algorithm) to output a distribution with probability 1 among flagged action-clips (the only flagged action-clip being c_1 here).
We again perform a regression of our curve linearly in $1/\sqrt{\delta}$ (represented in orange) to show the resulting dependency.

All of these simulations indicate that our computer simulations work correctly without sources of errors in our program. In the remaining simulations (Fig. 5.6 and 5.7), we will introduce the error model described in Sec. 5.2 and show the robustness of our algorithm to sources to noise. What these last curves show, for values of standard deviation σ of the Gaussian-distributed error in the gates' angle parameters ranging from $\pi/100$ to $\pi/10$, is that the inverse square root dependency on both ϵ and δ still remains on average, with still very good coefficients of determination R^2. We can notice though that the standard deviation of each batch (of 10^3 agents for each point) is the only thing that tends to increase with the amount of noise introduced in the model.

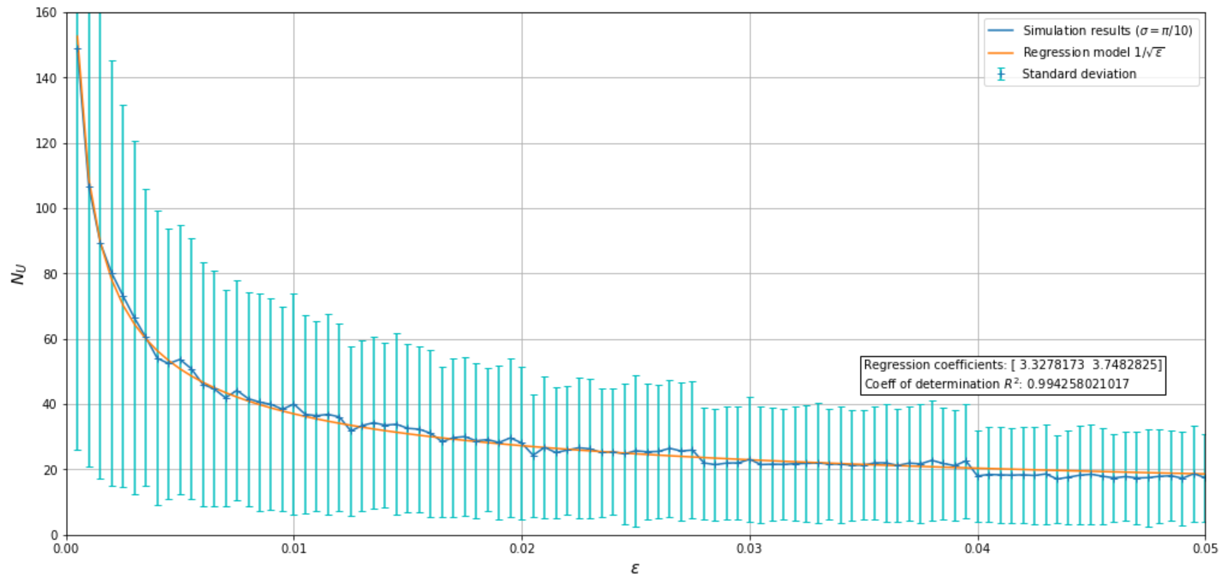
The Python code behind all these simulations and based on the QuTip library, can be found in the form of a Jupyter iPython Notebook at [26].



(a)

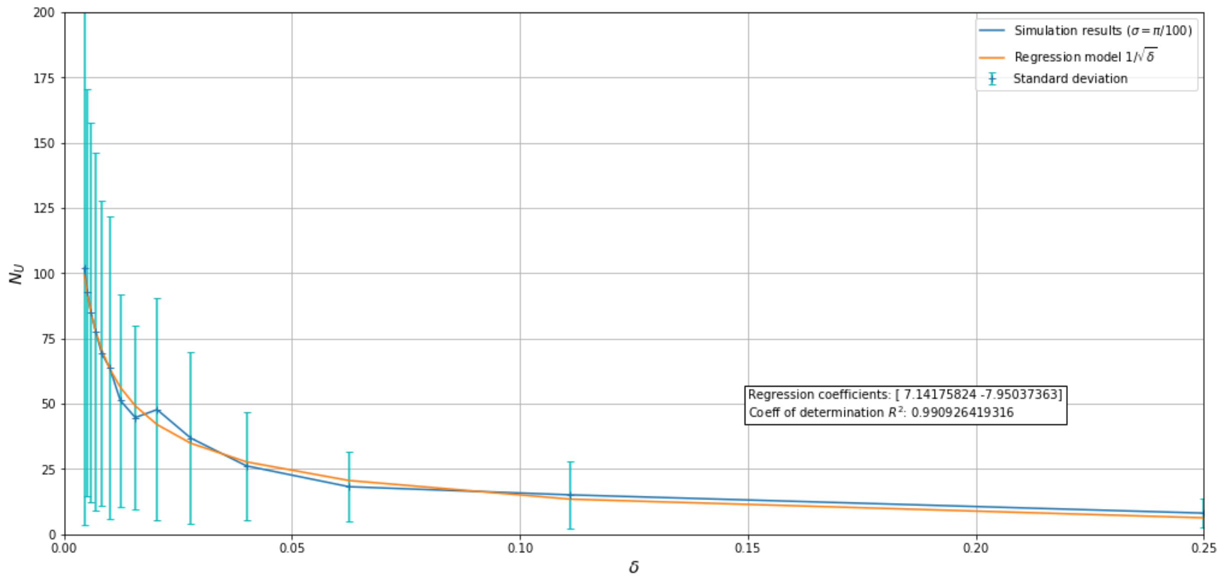


(b)

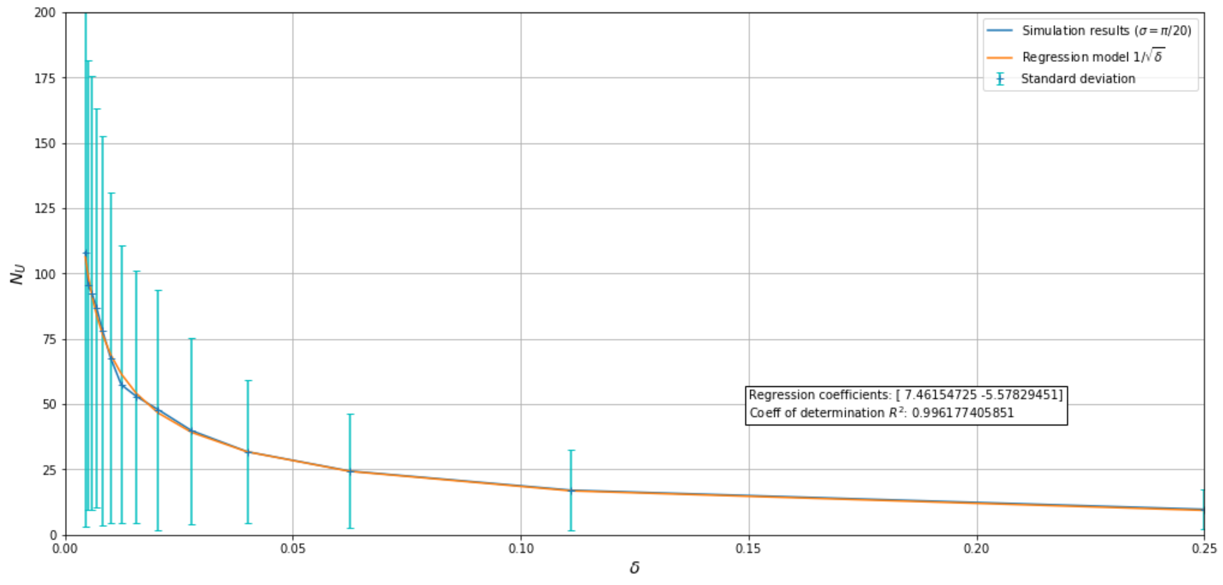


(c)

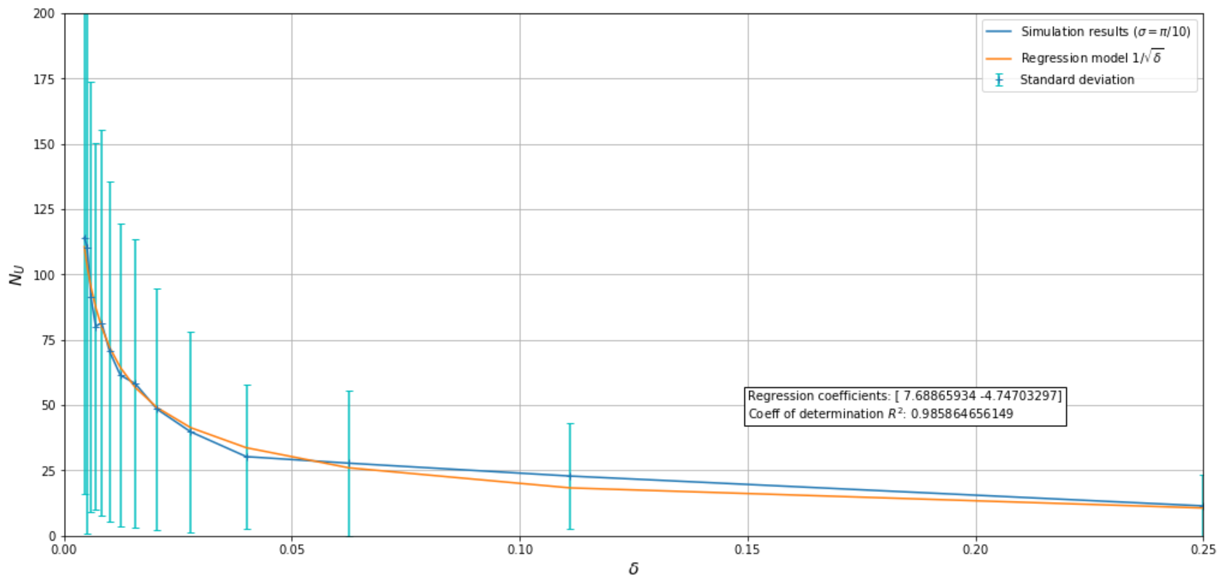
Figure 5.6: **Numerical simulation with error to show the $O(1/\sqrt{\epsilon})$ dependency.**



(a)



(b)



(c)

Figure 5.7: **Numerical simulation with error to show the $O(1/\sqrt{\delta})$ dependency.**

6 CONCLUSIONS & PROSPECTS

Throughout this work, we set the context for quantum-enhancements to the field of Reinforcement Learning and explained how different approaches (based on what we consider as *quantum* in our model) aiming at different applications can provide improvements to our classical means. We then introduced *Projective Simulation*, an RL model prone to quantization by virtue of the random walk process on which it is based. From there we developed the quantum generalization of classical walk theory, and defined the prerequisites for our computer simulations, aiming to show the feasibility of our PS algorithm in erroneous implementations of a QC and especially the robustness of our quadratic speed-ups to a first-order approximation of sources of noise in the trapped-ions realization.

But as explained before, the error model used in our simulations can always be perfected. For instance, a next step would be to use again the Mølmer-Sørensen gate to introduce error in the controlled operations of Kitaev's Phase Detection algorithm (Fig. 3.2.a)) and Coherent Controlization (Fig. 5.1) or also find a convenient construction to perform the $D_{0,Aux}$ reflection over $|0...0\rangle_{\text{Aux}}$ used for the Approximation Reflection Operation (Fig. 3.2.b)) were possible solutions would be either to use CNOT (controlled-X) operations (realized by the MS gate) to build Toffoli gates (two-qubit AND operation) and then use Toffoli gates and ancilla qubits to realize an n-qubit controlled-Z gate as a generalization of Fig. 5.2.a), or, another option would be to use *global MS* gates [27], leading to more efficient realizations of n-qubit controlled gates. But most importantly, one should consider *global dephasing* through varying detuning of the laser (its frequency shifts off slightly from the ions' resonant frequency, hence having the same effect on all the ions together) in its error model, as it is also a main source of noise for trapped ions.

Furthermore, some questions about quantized RPS where not treated in this work. First, in our quantum walk algorithm in Sec. 3.2 we start by setting the state $|U\rangle$ uniformly distributed over all vertices, but in the case of an ECM we have to set the coherent encoding of the stationary distribution $|\pi_P\rangle$ instead, which is not a trivial task and is in the order of $O(1/\delta)$ in complexity for the classical case. Fortunately, this subject was dealt with in [28] and in [29], achieving again a $O(1/\sqrt{\delta})$ complexity to generate the stationary distribution in the quantum case.
Another open question is the weight-update of the ECM after a percept-action-reward sequence in the quantum case. Indeed, the problem of the quantum walk in an n-layered RPS case is that we cannot keep track of the path of the agent in the graph (it doesn't even exist), so when the question of updating the weights of the traversed edges (in order to learn) comes up, finding an update procedure is still an open problem.

Finally one major challenge in RL that we did not talk about and is particularly impacting PS is the *curse of dimensionality*. Indeed, representing every state-action association in the memory of an agent is cumbersome and does not allow it to deal efficiently with unprecedented situations, especially in the case of a game where the state space is of high dimensionality (such as Chess or Go) where the probability of seeing again in a different game a same state (ex. the same board in Chess) that we obtain after at least a few state-action-reward sequences in a first game is practically zero, rendering learning very hard with an agent representing exhaustively its state-action associations in its memory. The use of composition (presented in Sec. 2.3) is of good help, but one needs to consider even more powerful techniques. In classical RL today, we deal with the curse of dimensionality by using *Convolutional Neural Networks*, a type of Neural Networks capable of dealing with high-dimensional inputs such as images to output a coinjoined (continuous) probability distribution over states and actions, a way to interpolate what the agent learned to all input states in the state-space. It is used for example to teach an Deep Reinforcement Learning (DRL) agent superhuman performance in Atari Games [30] or Go [31]. One idea to make PS benefit from such an advantageous property would be for example to embed *Deep Boltzmann Machines*, a similar sort of Recurrent Neural Networks modeling conjoined distributions, in the PS model; a subject yet to be studied.

References

[1] Charles H. Bennett, Gilles Brassard, Claude Crépeau, Richard Jozsa, Asher Peres, and William K. Wootters. Teleporting an unknown quantum state via dual classical and einstein-podolsky-rosen channels. *Physical Review Letters*, 70(13):1895–1899, 1993.

[2] Charles H. Bennett and Stephen J. Wiesner. Communication via one- and two-particle operators on einstein-podolsky-rosen states. *Physical Review Letters*, 69(20):2881–2884, 1992.

[3] Lov K. Grover. A fast quantum mechanical algorithm for database search. *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing - STOC 96*, 1996.

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

[5] R. Beals, H. Buhrman, R. Cleve, M. Mosca, and R. De Wolf. Quantum lower bounds by polynomials. *Proceedings 39th Annual Symposium on Foundations of Computer Science (Cat. No.98CB36280)*.

[6] S. Aaronson and A. Ambainis. The need for structure in quantum speedups. In *Theory of Computing*, volume 10, pages 133–166. 2014.

[7] A.S. Holevo. Bounds for the quantity of information transmitted by a quantum communication channel. *Problems Inform. Transmission*, 9(3):177–183, 1973.

[8] Harry Buhrman, Richard Cleve, and Avi Wigderson. Quantum vs. classical communication and computation. *Proceedings of the thirtieth annual ACM symposium on Theory of computing - STOC 98*, 1998.

[9] Rolf Pfeifer, Christian Scheier, Alexander Riegler, and Isabelle Follath. *Understanding intelligence.* The MIT Press, 2005.

[10] Richard P. Feynman. Simulating physics with computers. *International Journal of Theoretical Physics*, 21(6-7):467–488, 1982.

[11] D. Deutsch. Quantum theory, the church-turing principle and the universal quantum computer. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 400(1818):97–117, Aug 1985.

[12] Robert Raussendorf, Daniel E. Browne, and Hans J. Briegel. Measurement-based quantum computation on cluster states. *Physical Review A*, 68(2), 2003.

[13] H. J. Briegel, D. E. Browne, W. Dür, R. Raussendorf, and M. Van Den Nest. Measurement-based quantum computation. *Nature Physics*, 5(1):19–26, 2009.

[14] W. K. Wootters and W. H. Zurek. A single quantum cannot be cloned. *Nature*, 299(5886):802–803, 1982.

[15] Vedran Dunjko, Jacob M. Taylor, and Hans J. Briegel. Framework for learning agents in quantum environments, 2015.

[16] Vedran Dunjko, Yi-Kai Liu, Xingyao Wu, and Jacob M. Taylor. Super-polynomial and exponential improvements for quantum-enhanced reinforcement learning, 2017.

[17] Hans J. Briegel and Gemma De Las Cuevas. Projective simulation for artificial intelligence. *Scientific Reports*, 2(1), 2012.

[18] Julian Mautner, Adi Makmal, Daniel Manzano, Markus Tiersch, and Hans J. Briegel. Projective simulation for classical learning agents: A comprehensive investigation. *New Generation Computing*, 33(1):69–114, 2015.

[19] Alexey A. Melnikov, Adi Makmal, Vedran Dunjko, and Hans J. Briegel. Projective simulation with generalization. *Scientific Reports*, 7(1), 2017.

[20] Alexey A. Melnikov, Adi Makmal, and Hans J. Briegel. Projective simulation applied to the grid-world and the mountain-car problem. *Artificial Intelligence Research*, 3(3), 2014.

[21] M. Szegedy. Quantum speed-up of markov chain based algorithms. *45th Annual IEEE Symposium on Foundations of Computer Science.*

[22] Frederic Magniez, Ashwin Nayak, Jeremie Roland, and Miklos Santha. Search via quantum walk. *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing - STOC 07*, 2007.

[23] A. Yu. Kitaev. Quantum measurements and the abelian stabilizer problem, 1995.

[24] Dmitri Maslov. Basic circuit compilation techniques for an ion-trap quantum machine. *New Journal of Physics*, 19(2):023035, 2017.

[25] V Dunjko, N Friis, and H J Briegel. Quantum-enhanced deliberation of learning agents using trapped ions. *New Journal of Physics*, 17(2):023006, 2015.

[26] Sofiène Jerbi. Quantum-enhanced reinforcement learning and projective simulation github repository. github.com/sjerbi/Quantum-Reinforcement-Learning. *GitHub*.

[27] Dmitri Maslov and Yunseong Nam. Use of global interactions in efficient quantum circuit constructions. *New Journal of Physics*, 2017.

[28] V Dunjko and H J Briegel. Quantum mixing of markov chains for special distributions. *New Journal of Physics*, 17(7):073004, Mar 2015.

[29] Vedran Dunjko and Hans J. Briegel. Sequential quantum mixing for slowly evolving sequences of markov chains, 2015.

[30] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning, 2013.

[31] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, and et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359, 2017.