UNIVERSITAT POLITÈCNICA DE CATALUNYA (UPC) - BARCELONATECH

UNIVERSITAT DE BARCELONA (UB)

UNIVERSITAT ROVIRA I VIRGILI (URV)

MASTER'S THESIS

Quantum Machine Learning

Author: Jordi RIU

Advisor: Dr. Artur GARCIA
Institution: Barcelona Supercomputing Center (BSC)

Tutor: Dr. Ulises CORTÉS
Department: Computer Science (UPC)

MASTER IN ARTIFICIAL INTELLIGENCE

Facultat d'Informàtica de Barcelona (FIB)
Facultat de Matemàtiques (UB)
Escola Tècnica Superior d'Enginyeria(URV)

April 14, 2019

UNIVERSITAT POLITÈCNICA DE CATALUNYA (UPC) - BARCELONATECH

Abstract

Facultat d'Informàtica de Barcelona (FIB)

MASTER IN ARTIFICIAL INTELLIGENCE

Quantum Machine Learning

by Jordi RIU

This project applies reinforcement learning techniques to optimize a quantum algorithm. The upcoming quantum computers are capable of optimization tasks in an exponentially large parameter space. Having access to such a large optimization space potentially allows an advantage of Quantum methods over classical algorithms, but at the price of problems in the optimization process -just to name a few, the presence of local minima and the low convergence speeds-. This project explores the application of reinforcement learning techniques to control and optimize the operation of a quantum computer solving a hard combinatorial problem, namely the MaxCut problem. Our methods use a collection of quantum observables to feed the state of the agent, inspired on a similar classical approach to the problem applied to the antiferromagnetic Ising model lattice. Our results show how an agent using these observables can optimally control the operation of a quantum device to obtain quasi-optimal solutions when in combination with the Quantum Approximate Optimization Algorithm (QAOA).

Acknowledgements

I would like to thank Prof. Rossend Rey for supporting me, everyone at the Quantic group for welcoming me, and specially Prof. Ulises Cortés for his supervision of the project.

Finally, my most sincere gratitude to my advisor, Dr. Artur Garcia. He has contributed greatly to the project with fantastic ideas both for the Quantum Mechanics and Machine Learning related areas. It has been fantastic working with him.

Contents

List of Figures

List of Tables

Dedicated to my family.

Chapter 1

Introduction

Artificial intelligence techniques, and machine learning specifically, have been proven to be very useful and effective when applied to a wide variety of technological and scientific fields. Examples of such applications range from the automotive industry to health care.

Quantum machine learning, which combines both quantum physics and machine learning disciplines, arises in the late 90's and early 2000's. The field encompasses multiple areas of research such as the use of machine learning algorithms for the analysis of classical data on quantum computers, the application of classical machine learning algorithms on quantum problems or the intersection between learning theory and quantum information.

Analyzing classical data using quantum computers is expected to allow the recognition of more complex patterns in such datasets, through the use of quantum mechanical effects such as coherence and entanglement. In this regard, using quantum algorithms can also result in both improving the accuracy of the existing classical algorithms and also its computational performance. The latter is known as quantum speedup.

There are well-known examples of quantum speedup phenomena both for supervised and unsupervised tasks. Lloyd et al. [1] performed quantum principal component analysis of classical data which achieved exponential speedups with respect to classical PCA. Quantum exponential speedups have also been observed in supervised learning algorithms like support vector machines [2]. The aforementioned algorithms require the loading of classical data into quantum computers, which can take up to exponential time [3]. This issue can be handled using qRAM [4], although limitations appear on the amount of data that can be handled.

Dunjko et al. [5] presented a general agent-environment framework for quantum reinforcement learning algorithms with quadratic speedups in learning efficiency, i.e., number of interaction steps with the environment required by the agent.

Quantum algorithms for deep learning have also been explored with moderate success. The use of quantum annealers, which are special-purpose quantum information processors, to perform quantum sampling during the pre-training phase of a DBN yielded close to state of the art accuracy for the MNIST dataset with fewer training iterations [6]. Amin et al. [7] proposed a more general algorithm, called Quantum Boltzmann Machines, with promising performances on small datasets. Moreover, the authors argued that QBM training could in principle be conducted on quantum annealers, and D-wave in particular, with small hardware modifications.

Of particular interest to our work is the Quantum Approximate Optimization Algorithm (QAOA) [8]. The algorithm allows reaching approximate solutions for combinatorial optimization problems, such as the MaxCut problem, by applying qubit rotations that depend on the clauses of the problem and some optimizable parameters. We seek to study the performance of the QAOA when using deep reinforcement learning as its optimization strategy.

This thesis will be structured in 2 main blocks. The first one will be focused on the design of a gym environment that mimics the Ising Model (using ferromagnetic coupling) and the study of its performance when using reinforcement learning to solve it. This environment should allow us to get an insight on the possible policies, architectures and algorithms that are most likely to succeed combined with the QAOA when applied to the MaxCut problem, which will be our focus in the second part of this work.

There are various reasons behind the selection of the Ising Model as our testing model. First and foremost, the Ising Model with nearest neighbours interactions is a particularly well-behaved case of the MaxCut problem. Moreover, the 2-Dimensional Ising Model with ferromagnetic coupling is known to have a phase transition between the ordered and disordered phases which can be computed analytically. Therefore, it will be clear whether our reinforcement learning approach yields satisfactory results or not.

Finally, and although it is not the goal of this work, using reinforcement learning to simulate the dynamics of the Ising Model could allow us to perform future research regarding the possibility of using the Ising Model to generate time series with statistical properties similar to the ones of financial time series as in [9].

Chapter 2

Reinforcement Learning in a Classical Environment

2.1 Ising Model

The Ising model is a simple classical physics model which was originally used in statistical mechanics to describe ferromagnetism, but nowadays it is used in a wide range of social and physical problems.

It consists of a large number of spins interacting within a lattice. Each spin can only take the value +1 or -1 (depending on the direction in which it points) and is only allowed to interact with its nearest neighbours, tending to align with them (see Figure 2.1). Therefore, it combines the effect of temperature (disordering force) and the tendency of a physical system to reach its most stable (lowest energy) state, which for the ferromagnetic Ising model coincides with the state in which all spins are aligned. The Hamiltonian of the system is the following:

$$\mathcal{H}_{Ising} = -J\sum_{n.n}\sigma_i\sigma_j, \tag{2.1}$$

where n.n. stands for nearest neighbours, J>0 (ferromagnetic coupling) and σ_i is the value of the ith spin of the lattice. The Ising model does not have any specific dynamics, only an established behaviour at equilibrium. For this reason, there exist multiple possibilities to model its dynamical behaviour. At equilibrium though, the ferromagnetic 2-D Ising Model with isotropic coupling is known to exhibit a phase transition between the ordered and disordered phases (Onsager's Exact Solution).

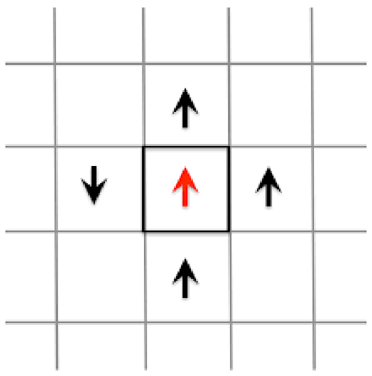

FIGURE 2.1: Graphical representation of the 2D-Ising lattice. A spin from the lattice (in red) can only interact with the ones closer to it (in black). In 2D the number of nearest neighbours is 4.

Specifically, such transition is observed at

$$k_B T_{critical} = \frac{2J}{ln(1+\sqrt{2})}. \tag{2.2}$$

For J=1, $T^*_{critical} = k_B T_{critical} \approx 2.26$. Our first goal is to perform a simple experiment that shows the capacities for reinforcement learning approaches to simulate the dynamics of the model while respecting its behaviour at equilibrium.

Next, we will try to solve a trivial case of the MaxCut problem, the 2-D Ising Lattice, also through the use of RL. Ideally the agent will be able to transform the lattice from a random initial state, to the highest energy configuration (antiferromagnetic state) by flipping the minimum number of spins. This second experiment should provide us with some insight on which RL strategies can be successful when using the QAOA with RL optimization to solve the MaxCut problem.

2.2 Q-Learning

Temporal-Difference(TD) learning algorithms combine the advantages of Monte Carlo algorithms, such as being able to learn from an environment whose dynamics model is unknown, as well as Dynamic Programming with for example bootstrapping, i.e., performing estimations using other previously learned estimates. These conditions allow TD algorithms to learn on-line, at each step of training, without the need to wait until the end of an episode (which is not very well defined for our problem).

A particularly relevant TD control algorithm to our work is the Q-Learning algorithm [10]. Q-learning is an off-policy TD control algorithm which can be presented in several forms (even for continuous action spaces as we will see in the second part of the thesis). The simplest one, one-step Q-learning, uses the following update rule for the state-action value function $\mathcal{Q}$ of the problem:

$$\mathcal{Q}(s_t, a_t) \leftarrow \mathcal{Q}(s_t, a_t) + \alpha \left\{ r_{t+1} + \gamma \max_a \mathcal{Q}(s_{t+1}, a) - \mathcal{Q}(s_t, a) \right\} \tag{2.3}$$

With r_{t+1} being the reward of applying action a_t at time t with the environment being at state s_t, γ the discount factor for future gains and α the learning rate. This update rule is based on the fact that the optimal Q function satisfies the Bellman Equation.

Since Q-learning is an off-policy method, its performance is independent on the chosen policy. Not only that but also Q-learning is known to converge with probability 1 to the optimal action-value function. The only requisite for this convergence to be achieved is that all state-action pairs are visited and updated, policy selection thus remains somewhat relevant.

DQN

A more sophisticated Q-learning algorithm which uses neural networks as approximation to the action-value function is the Deep Q-Network (DQN) algorithm [11]. DQN solves the instabilities of standard online Q-learning using neural networks as interpolates of the action-value function by reducing the correlation of the data sequence through experience replay.

Moreover, the weights of the target network used as ground truth for the training of the Q-network are only updated every n steps (using the weights of the Q-network) while kept frozen during the rest of individual updates, thus reducing the correlation with the target as well.

The loss function that incorporates both ideas is shown in equation (2.4). It is the squared difference between the Q value computed using the target network, with weights θ_i^- and the one provided by the Q-Network.

For each iteration i, the algorithm uses the observation of the current state of the environment, s, as input to the Q-network which yields the state-action values Q(s,a;θ_i) for all the possible actions, a, that can be performed in that state with θ_i being the parameters of the network.
Then, following the defined policy, an action is chosen from those Q values and performed on the environment, obtaining reward, r, and the observation of the new state of the environment s′. The experience tuple formed by (s,a,r,s′) is stored in a Data set (memory) with the previous ones. Finally, a mini-batch of experiences is sampled randomly using a uniform distribution and it is used to compute the loss of the model, and update the Q-network weights. Every n iterations, the weights of the target network are updated with the ones from the Q-network.

$$L_i(\theta_i) = \mathop{\mathbb{E}}_{(s,a,r,s')} \left\{ \left(r + \gamma \max_{a'} Q(s', a'; \theta_i^-) - Q(s, a; \theta_i) \right)^2 \right\} \tag{2.4}$$

Double Q-Learning

Q-learning and DQN in particular can suffer from large overestimations due to using the same parameters (or network) both to evaluate the action-value function and select the action. For this reason, the Double Q-Learning algorithm is proposed [12].

For the Double Q-Learning algorithm, the target value to which we want our state-action function to converge is computed as:

$$Y_{target} = r + \gamma Q(S_{t+1}, \arg\max_a Q(S_{t+1}, a; \theta_i); \theta_i') \tag{2.5}$$

Thus, two different networks are used to select which action is the one with highest state-action value. The first network decides which action is the optimal one, and the second is used to asses its value fairly. From this idea the Double DQN algorithm arises [13]. In it, the authors suggest using the target network as responsible for the assessment of the state-action value once the Q-network chooses the action with maximum expected value. Again, the weights of the target network can be updated periodically with the ones from the Q-network (hard update). Another possibility is to perform a soft update on the weights of the target network such that

$$\theta_i^- = \tau\theta_i + (1-\tau)\theta_i^- \tag{2.6}$$

Dueling Network Architectures

In [14] a new strategy consisting on two streams within a network, one focused on the computation of the average state-action value of the state s, $V^\pi(s) = \mathbb{E}[Q^\pi(s,a)]$, and the other, on the relative advantage for each action with respect to the average value (2.7).

$$A^{\pi}(s,a) = Q^{\pi}(s,a) - V^{\pi}(s) \tag{2.7}$$

Thus, the first stream will produce a unique scalar value corresponding to the value of the input state of the network while the other will produce several outputs, as many as available action for the state. It might seem immediate to recover the Q value arising from such strategy as the sum of the values of both streams(2.8), once the action selection policy is applied on the A stream, with α the parameters of the V(s) stream network and β for the A(s,a) network equivalently. However, the authors mention such equation is unidentifiable as adding a constant to either V or A and subtracting it to the other would yield the same Q-value and this multiplicity would damage the performance of the dueling strategy.

$$Q(s,a;\theta,\alpha,\beta) = V(s;\theta,\beta) + A(s,a;\theta,\alpha) \tag{2.8}$$

For this reason, two alternative definitions are proposed (2.9). The first one ensures that the estimator for Q(s,a) equals V(s) for the best action, eliminating the multiplicity since the advantage A(s,a) for the optimal action is forced to be 0 as point of reference. The second one does not tackle the multiplicity problem but allows for a more stable optimization as the advantage estimator is only required to evolve as fast as the mean instead of compensating for changes in the optimal advantage value.

$$\begin{aligned} Q(s,a;\theta,\alpha,\beta) &= V(s;\theta,\beta) + \left(A(s,a;\theta,\alpha) - \max_{a'} A(s,a';\theta,\alpha)\right) \\ Q(s,a;\theta,\alpha,\beta) &= V(s;\theta,\beta) + \left(A(s,a;\theta,\alpha) - \frac{1}{m}\sum_{a'=1}^{m} A(s,a';\theta,\alpha)\right) \end{aligned} \tag{2.9}$$

Continuous Deep Q-learning: NAF & DDPG

For the second part of the project we will apply deep Q-learning in a continuous action space. Tackling this problem requires additional developments for the agent as the well-known action selection policies such as, for example, ϵ-greedy and Boltzmann can no longer be applied.

In this regard, Gu et al [15] suggest the Normalized Advantage Function (NAF) approach. The method is based on the idea of finding a representation for the state-action value function Q for which its maximum can be determined analytically.

The NAF algorithm uses a naive dueling strategy with $Q(s,a) = A(s,a) + V(s)$ but A(s,a) is constructed as

$$A(s,a;\theta^{A}) = -\frac{1}{2}(a-\mu(a;\theta^{\mu}))^{T}P(s;\theta^{P})(a-\mu(a;\theta^{\mu})) \tag{2.10}$$

with

$$P(s;\theta^{P}) = L(s;\theta^{P})L(s;\theta^{P})^{T} \tag{2.11}$$

a positive-definite square matrix and $L(s;\theta^{P})$ a lower-triangular matrix constructed with the outputs of the network. Therefore, the network will now have three streams, V, μ and L.

By forcing A(s,a) to have this form, it is ensured that the maximum value for Q will be obtained when a = μ as otherwise, A is always negative.

At each step, the agent obtains μ from the observation of the state and selects the action a by adding noise to it (it is suggested to use an Ornstein-Uhlenbeck process for noise generation). Afterwards the action is applied to the environment and the tuple (s_t, a, r, $s_t + 1$) is stored in the memory to perform experience replay during the optimization phase. The target value for Q used as optimization goal is set to r + $V(s_t + 1)$ and the Q value is constructed using the V(s), L(s) and $\mu(s)$ streams.

In [16] the authors suggest a two network actor-critic strategy, Deep DPG. The critic network is used to compute Q(s,a) while the actor network establishes the action selection policy through the computation of $\mu(s)$. To increase exploration, a random term sampled from a random process is added to the actor value in the same way than for NAF. The target value used to compute the loss function is then: $y_t = r + \gamma Q'(s_{t+1}, \mu'(s_{t+1}; \theta^{\mu'}); \theta^{Q'})$.

2.3 Agent & Environment Implementation

Gym package (python) is used to set up the Ising environment for both problems.

2.3.1 Simulating the Dynamics of the Ising Model with RL

For the first problem, we want to reproduce the behaviour at equilibrium of the Ising Model when simulated using an RL approach.
The environment stores the physical properties of the model, i.e., its current temperature and the size of the lattice as well as its configuration (which spins are up and which ones are down).

The environment is initialized with all the spins pointing at the same direction or with random directions according to a random variable that follows a uniform distribution between [-1 and 1], with negative values resulting in -1 orientation for the spins and the other way around for positive values.There are only two possible actions to be performed on the environment for each step: Flipping a spin or maintaining its orientation.

The *Metropolis* algorithm [17] defines the acceptance probability of a transition between lattice configurations, which for single-flip dynamics corresponds to the probability of flipping a certain spin ($P(\uparrow\downarrow)$), to be equal to Equation (2.12) in order to satisfy detailed balance at equilibrium.

$$P(\uparrow\downarrow) = min(1, e^{-\beta \Delta E}), \tag{2.12}$$

Note that $\beta = \frac{1}{k_B T}$ and ΔE is difference of energy between the new state (after the flip is performed) and the previous state.

Inspired by this probability definition, the rewards of the environment are set to:

- *Flip Action*: $-\Delta E$.
- *No-Flip Action*: *0*.

With this reward definition, unfavorable energy states are in principle not encouraged by the environment. Note that for $\gamma = 0$ the Q-values for each action will be equal to $-\Delta E$ for any flip action and 0 for the no-flip action. With such reward definition, using a Softmax policy for action selection and taking into account that the configuration probability at equilibrium follows the Boltzmann distribution, the detailed balance condition is fulfilled ((2.13)).

$$P(\{\sigma\}_1)P(\{\sigma\}_1 \rightarrow \{\sigma\}_2) = e^{-\beta E_1}\frac{e^{-\beta(E_2-E_1)}}{e^{-\beta(E_2-E_1)}+1} = e^{-\beta E_2}\frac{1}{e^{-\beta(E_2-E_1)}+1}$$
$$= e^{-\beta E_2}\frac{1}{e^{-\beta(E_2-E_1)} * (1+e^{\beta(E_2-E_1)})} = e^{-\beta E_2}\frac{e^{-\beta(E_1-E_2)}}{1+e^{-\beta(E_1-E_2)}} = P(\{\sigma\}_2)P(\{\sigma\}_2 \rightarrow \{\sigma\}_1). \tag{2.13}$$

The agent will follow a single spin-flip strategy per step. Each spin will be selected randomly following a uniform distribution across all the spins in the lattice. Therefore, the energy variation of the system per step depends uniquely on the orientation of the target spin and its nearest neighbours. Each of these 5 spins can only take 2 orientation values, yielding a total amount of 32 different states for the environment to be in (if each spin is identified with a specific location). Moreover, the state space of the environment can be reduced to only 5 states by using the existing symmetries on the energy definition. For example, the state $\{\sigma_i\}_{i\in n.n.} = [-1,-1,1,-1,-1]$ should behave in the same way that the state $\{\sigma_j\}_{j\in n.n.} = [1,1,-1,1,1]$ as for both states flipping the central spin would result in a decrease of -4J of the energy of the system. For this reason, the observation obtained from the environment will be a one-hot encoding of these 5 energy states.

In the same way, each state can only transition to two other states, either itself (if the action is no-flip) or the state which energy has the same absolute value but different sign than the original. Therefore, we will be working with a deterministic classical and discrete environment.

Finally, the agent will use one-step Q-learning as RL algorithm, with Softmax selection policy. The action-value function will be computed with a simple single-layer neural network, which takes as inputs the observation of the environment, i.e., a one-hot encoded vector of the energy value of the configuration. Also, the reduced temperature of the system $T^* = k_B T$ will be used as τ for the Softmax policy.

2.3.2 Obtaining the Antiferromagnetic State of the 2-D Ising Lattice

For this second problem, the goal of the agent is to transform the 2-D Ising lattice, starting from a random configuration, to the antiferromagnetic state. That is, all the nearest neighbour spins have opposite orientation to the central spin. To do so, the agent will perform single spin-flips at each step. Note that the antiferromagnetic state is the state with the highest energy of the system, with value equal to twice the number of spins in the lattice, n_{spins}, if J equals one. In order to consider the solution to be relevant, it has to be reached in less steps than the amount of spins in the lattice.

The environment stores the current configuration of the lattice as well as the number of flips that did not increase the energy of the system, considered incorrect.

Each step, the agent chooses a spin and flips it. Therefore, the action space has dimension n_{spins}. The resulting reward is defined as $\frac{\Delta E}{\Delta E^{max}}$. For the 2-D Ising lattice the maximum variation of energy between two states due to a spin-flip is 8J. An episode

ends whenever the antiferromagnetic state is reached, providing an additional reward equal to K divided by the number of incorrect flips, with K being a positive arbitrary parameter. Thus, the agent obtains the maximum possible reward for an episode, when it reaches the final state without making any mistakes.
The observation returned to the agent is the array of spin orientations after the spin is flipped and has dimension n_{spins} x n_{spins}.

Regarding the agent, the DQN Agent implementation from the keras-rl package will be used. The network architecture depends on the size of the lattice (as seen in the next section), and combines both convolutional and fully-connected layers.

2.4 Experiments & Results

2.4.1 Simulating the Dynamics of the Ising Model with RL

We train the agent at different temperatures, starting from T* = 1.5 and up to T* = 3.0 with an increase of 0.1 each iteration. At each temperature the training is performed for 10.000 steps and tested over 300.000 steps. For J=0, $T^*_{critical}$ should be close to 2.26 in a lattice with infinite dimension. However, since the lattice used will have a limited size of 20x20 and 40x40 spins, $^T*_{critical}$ value will be modified due to finite-size effects. To reduce such effects, periodic boundary conditions will be applied. Also, the lattice is initialized at the ferromagnetic state, which is the equilibrium state of the lattice for temperature below the critical temperature.

One-step Q-learning with learning rate α = 1 is used as the RL algorithm for the agent. The action-value function is approximated by a simple neural network that has a 5 dimensional vector as input, which corresponds to the one-hot encoded energy state, and 2 outputs, the Q values for the possible actions. The network is optimized minimizing the loss function in (2.14) using the gradient descent algorithm with 0.1 learning rate.

$$L_i(\theta_i, s, a) = \left(r + \gamma \max_{a'} Q(s', a'; \theta_i) - Q(s, a; \theta_i) \right)^2 \tag{2.14}$$

For each lattice size, multiple discount factors are tested for the agent. The magnetization of the system is sampled every 300 testing steps and its mean absolute value is computed at the end of the testing phase per each temperature.

The results for the 40x40 lattice are shown in Figure 2.2. The agent is able to reproduce the behaviour at equilibrium of the system as the transition between the ordered and disordered phases is clear and found around $T^* \sim 2.4$. The fact that the agent is able to control the temperature of transition is particularly interesting, reducing its value as γ increases.

The same behaviour is observed for the 20x20 lattice (Figure 2.2) but the finite-size effects are far more relevant. For instance, it is clear that the mean absolute magnetization for the disordered phase is higher than for the previous case and the transition is less sharp.

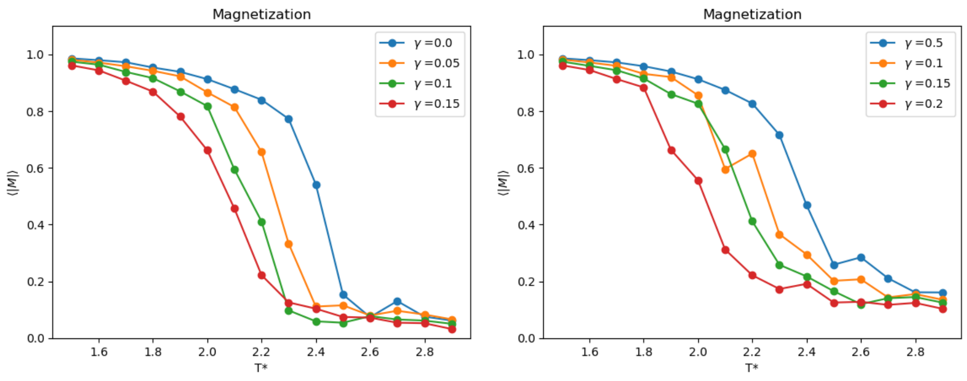

FIGURE 2.2: Mean absolute magnetization for a spin of a 40x40 Ising lattice (left) and a 20x20 Ising lattice (right) for multiple discount factors γ. $T^*_{critical}$ decreases as γ increases.

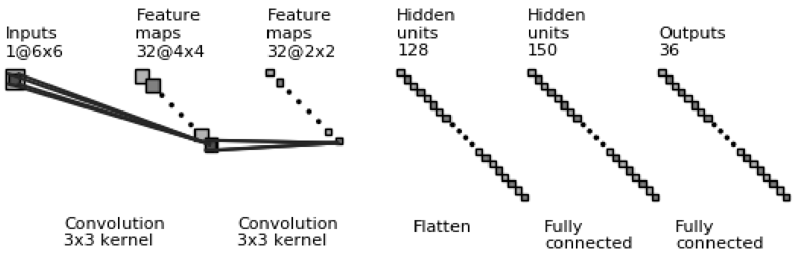

FIGURE 2.3: Q-Network model for a 6x6 Ising Lattice.

2.4.2 Obtaining the Antiferromagnetic State of the 2-D Ising Lattice

For this problem, we perform a proof of concept with 6 x 6 2-D Ising Lattice first and an 8x8 afterwards. The optimal network architecture is shown in Figure 2.2. It consists of two two-dimensional convolutional layers with *ReLU* activation function [18] and a kernel of 3 by 3 as the interactions between spins in the layer have this range of effect. In addition, the previous outputs are flattened and introduced into a fully-connected layer with *ReLU* activation and an output layer, that has linear activation function.

The optimization algorithm used is the *Adam* [19] algorithm with 0.001 learning rate. During training, the agent has 100 steps per episode to transform the initial random lattice configuration, sampled from a uniform distribution, into the antiferromagnetic state. This tends to generate configurations with average Magnetization close to 0, but with some variation between the energies of multiple initial states.

For a lower number of steps per episode, the convergence of the algorithm is slightly slower as the exploration of the state space is reduced. The exploration range of the state space becomes specially salient when taking into account that the initial configuration of the lattice has the previously mentioned restrictions. We would expect that a wider variety in initial configurations would make this difference disappear.

On the other hand, during the testing phase of the model, the agent is only allowed to perform a maximum of 36 flips per episode, which is the number of spins in the lattice. This restriction is imposed as a non-intelligent agent, that flipped a spin up and the next one down in order, would reach the antiferromagnetic state in that

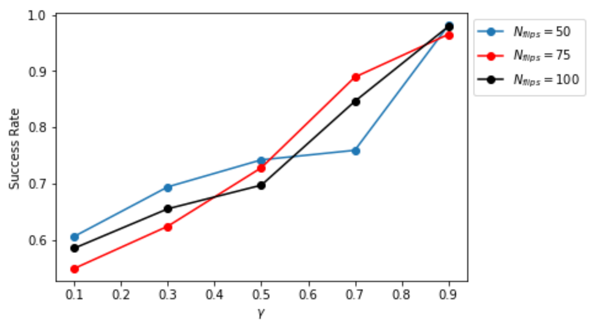

FIGURE 2.4: Agent success rate dependent on the discount factor and number of steps per episode. 1000 test episodes with random initial configuration. 200.000 training steps per parameter set.

amount of steps.

The other crucial parameter for the result of the network is the discount factor γ (Figure 2.4). The higher the discount factor, the better results the agent is able to produce. Specifically, the success rate of the agent, i.e., the amount of episodes that reach the antiferromagnetic configuration, for $\gamma = 0.6$ is around 60% while for $\gamma = 0.9$ the success rate is higher than 96%. However, for $\gamma > 0.9$ the model displays convergence issues during training.

This behaviour is due to the additional reward obtained by the agent if it succeeds on reaching the final state. A small value of the discount factor reduces the influence of this final reward on the state-action value. Therefore, the incentive for the agent to end the episode is lower. On the other hand, for very high discount factors, such final reward becomes dominant in the Q value computation, which might compensate the negative immediate reward of incorrect flips.
Also, we use an ϵ-greedy policy with $\epsilon = 0.1$, train the agent over 200.000 steps and test it over 1000 episodes. The target network weights are updated using a soft update with $\tau = 0.01$.

In Figure 2.5 the obtained results are displayed. The first one is an histogram on the amount of errors per episode made by the agent during testing. Note that a flip producing a null variation in energy for the system is also considered an incorrect flip. The agent makes ≤ 5 mistakes per episode with probability close to 0.7. The second figure is a histogram on the amount of steps performed per episode. Again the agent is able to solve the problem in ≤ 20 steps with a probability close to 0.7. However, for 35 of the 1000 episodes the agent is not able to solve the problem, yielding a success rate of 0.965.

In Figure 2.7 a random sample of 50 testing episodes is displayed. We note that there is no clear correlation between the initial energy of the lattice and the performance of the agent. On the other hand, and as expected, the episodes with higher incorrect steps also have a higher total amount of steps.

For the 8x8 lattice we use the same parameters with the following slight modifications: The number of outputs is set to 64, and the input size is an 8x8 array. In addition, the number of steps per episode during training is set to 500 as the state space dimension is also increased. For the test phase, the number of steps per episode is

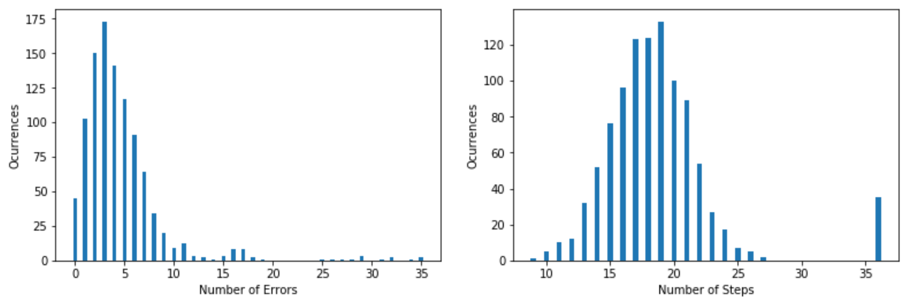

FIGURE 2.5: Left: Histogram of the number of steps per episode with reward lower or equal to zero (errors). Right: Histogram of the total number of steps per episode. Both for an 6x6 Ising lattice.

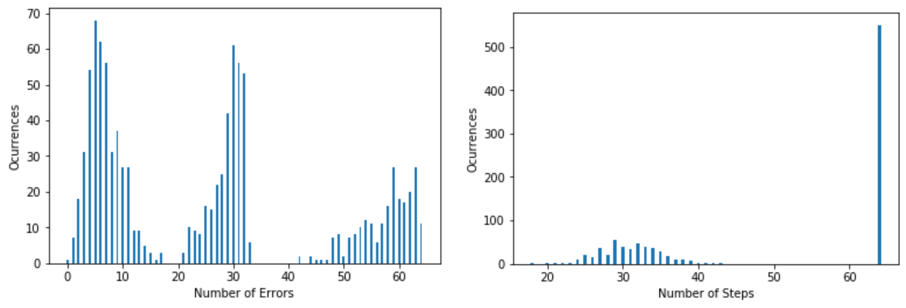

FIGURE 2.6: Left: Histogram of the number of steps per episode with reward lower or equal to zero (errors). Right: Histogram of the total number of steps per episode. Both for an 8x8 Ising lattice.

set to 64. Moreover, we expand the exploration of the action space by increasing the ϵ parameter of the policy to 0.25. Finally, we train the agent for 500.000 steps instead of the previous 200.000.

The best obtained results are displayed in Figure 2.6. The success rate of the agent is around 40%, much lower than for the previous lattice. Also the number of errors per episode is much higher. A sample of the test episode results is displayed in Figure 2.7. It is clear that scaling this approach for solving the MaxCut problem is complex and its capacity is limited. The optimization of the 8x8 lattice would already require an in-depth parameter exploration which is not the goal of this thesis.

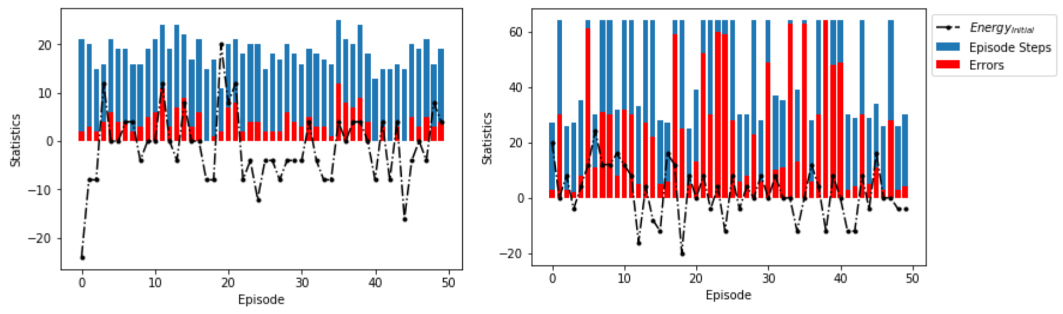

FIGURE 2.7: Results obtained for the 6x6 (left) and 8x8 (right) Ising Lattice. Steps taken (blue), errors (red) and energy of the initial configuration (black).

Chapter 3

Deep Reinforcement Learning in a Quantum Environment

3.1 Quantum Approximate Optimization Algorithm

The Quantum Approximate Optimization Algorithm (QAOA) [8] is a very recent algorithm (2014) aimed towards obtaining quasi-optimal solutions for combinatorial problems. The algorithm uses classical-quantum approach based on the classical optimization of quantum states. Combinatorial problems are defined by a set of m binary clauses, C_α, each of which is either fulfilled or not by a set of n bits, z. If a clause is fulfilled it yields a value of 1, otherwise its value is 0. The goal of the QAOA is to find a z that satisfies as many clauses as possible. Thus, the objective function to maximize, C(z), is defined as Equation (3.1).

$$C(z) = \sum_\alpha C_\alpha(z). \tag{3.1}$$

Note that the algorithm will likely not produce the optimal solution but rather an approximate to it within a reasonable amount of time.

The QAOA is based on a series of pairs of unitary transformations applied to a quantum system of n qubits. Each qubit is a two-level quantum system within a 2-dimensional Hilbert space ($\mathscr{H}^2$). At the beginning, every qubit is initialized as $|+\rangle = \frac{1}{\sqrt{2}}|\uparrow\rangle + \frac{1}{\sqrt{2}}|\downarrow\rangle$, where $|\uparrow\rangle$ and $|\downarrow\rangle$ are the eigenstates of the σ_z matrix and form a basis of the space. Therefore, the whole 2^n-dimensional initial system state becomes $|s\rangle = |+\rangle \otimes |+\rangle \otimes ... \otimes |+\rangle$. Note that storing the wave function of the state in a classical computer would require an exponential increase in memory when n increases, while that is not the case if a quantum computer is used.

The algorithm applies p pairs of unitary transformations, $U(B,\beta)$ and $U(C,\gamma)$ each of which depends on a different operator and a parameter that acts as a scaling factor for the rotation, β and γ respectively. While C is the quantum form of the objective function, B is independent on the combinatorial problem we want to solve and its form is given by (3.2).

$$B = \sum_{j=1}^{n} \sigma_j^x, \tag{3.2}$$

With σ_j^x the x-axis Pauli matrix for the j-th qubit. After such transformations, and for a certain p, the final angle dependent quantum state of the n-qubit system becomes:

$$|\gamma,\beta\rangle = U(B,\beta_p)U(C,\gamma_p)U(B,\beta_{p-1})U(C,\gamma_{p-1})...U(B,\beta_1)U(C,\gamma_1)|s\rangle\,. \tag{3.3}$$

The form of these unitary transformations are shown in Equation (3.4)

$$\begin{aligned} U(B,\beta) &= e^{-i\beta B} = e^{-i\beta\sum_{j=1}^{n}\sigma_j^x} = \prod_j e^{-i\beta\sigma_j^x} \\ U(C,\gamma) &= e^{-i\gamma C} = e^{-i\gamma\sum_\alpha C_\alpha} = \prod_\alpha e^{-i\gamma C_\alpha} \end{aligned} \tag{3.4}$$

Therefore, the quantum state obtained after these rotations uniquely depends on 2p parameters and can be generated with a quantum circuit of depth mp+p at most. Let F_p be the average value of the objective operator in this state

$$F_p = \langle \gamma, \beta | \, C \, | \gamma, \beta \rangle \, , \tag{3.5}$$

The algorithm requires an optimization strategy that finds the optimal angle configuration that yields the maximum expectation value M_p. Notice that $M_p \geq M_{p-1}$ as β_p and γ_p can be set to 0 such that $U(B,\beta_p) = U(C,\gamma_p) = \mathbb{I}$.
Once M_p is found, and due to the form of F_p, a quantum computer will be able to generate a string of bits z for which C(z) is close to M_p or larger in an efficient way.

The goal of this second part of the project is to use reinforcement learning approaches as the optimization strategy for the QAOA when applied to the MaxCut problem.

MaxCut

The goal of the QAOA algorithm will be to find the MaxCut for a graph with bounded degree. The graph will have n vertices each of which will have a bounded number of edges connecting itself to other vertices in the graph. For r-regular graphs, all vertex will have exactly r connections.

Finding the MaxCut of the graph means dividing the vertices into two types, +1 and -1 for example, such that the amount of connections between vertices of different type is maximal. To do so using the QAOA algorithm, each vertex will be encoded by a qubit, and each state of the computational basis will identify the type of the vertex once it is measured.

With this setup, the objective function to be maximized can be expressed in terms of quantum operators as

$$C = \sum_{\langle jk \rangle} \frac{1}{2}(-\sigma_j^z \sigma_k^z + I), \tag{3.6}$$

with $\sum_{\langle jk \rangle}$ being the sum over all the edges of the graph and j and k the identifier of the vertices connected by the edge. Since the eigenvalues of the σ^z are +1 and -1, each edge will contribute to either 1 or 0 to the objective function depending on whether it connects qubits measured in different eigenstates or not.

Using (3.6), operator $U(C,\gamma)$ can be constructed in the following way

$$U(C,\gamma) = \prod_{\langle jk \rangle} e^{-i\gamma \frac{-\sigma_j^z \sigma_k^z + I}{2}} \tag{3.7}$$

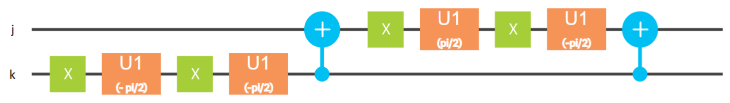

FIGURE 3.1: Example of the quantum circuit required to generate the operator in (3.8) with $\gamma = \pi = -2\phi$ using the IBM Quantum Experience. Note that R_ϕ = U1.

Each of the terms of the product, that corresponds to the operator that arises due to each edge between qubits j and k, can be expressed as follows

$$e^{-i\gamma\frac{-\sigma_j^z\sigma_k^z+I}{2}} = e^{-i\gamma\frac{-\begin{pmatrix}1&0\\0&-1\end{pmatrix}\otimes\begin{pmatrix}1&0\\0&-1\end{pmatrix}+\begin{pmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&0&1\end{pmatrix}}{2}} =$$
$$= e^{-i\gamma\frac{-\begin{pmatrix}1&0&0&0\\0&-1&0&0\\0&0&-1&0\\0&0&0&1\end{pmatrix}+\begin{pmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&0&1\end{pmatrix}}{2}} = e^{-i\gamma\begin{pmatrix}0&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&0&0\end{pmatrix}} = \tag{3.8}$$
$$= \begin{pmatrix}1&0&0&0\\0&e^{-i\gamma}&0&0\\0&0&e^{-i\gamma}&0\\0&0&0&1\end{pmatrix}$$

This operator can be constructed combining the following one-qubit and two-qubits quantum gates in Equation (3.9) as shown in Figure 3.1 [20]. Note that the matrix for the CNOT gate corresponds to a CNOT gate in which the second qubit (qubit k) is the control qubit.

$$\begin{aligned} X &= \begin{pmatrix}0&1\\1&0\end{pmatrix}\\ R_\phi &= \begin{pmatrix}1&0\\0&e^{i\phi}\end{pmatrix}\\ CNOT &= \begin{pmatrix}1&0&0&0\\0&0&0&1\\0&0&1&0\\0&1&0&0\end{pmatrix}\\ R_x(\theta) &= \begin{pmatrix}\cos\frac{\theta}{2}&-i\sin\frac{\theta}{2}\\-i\sin\frac{\theta}{2}&\cos\frac{\theta}{2}\end{pmatrix}\end{aligned} \tag{3.9}$$

By sequentially introducing this gates between each of the connected qubits of the graph, the unitary transformation $U(C,\gamma)$ is applied. On the other hand, $U(B,\beta)$, can be implemented as the $R_x(\theta)$ gate. Once all these gates are applied for all angles, the state $|\gamma,\beta\rangle$ for the MaxCut problem is obtained.

Evaluating the goodness of the obtained solution is another issue that needs to be addressed. The MaxCut for the Ising model, which corresponds to a 4-regular graph, could be computed in a very simple way (2*n) due to the geometry of the graph. However, this might not always be the case due to frustrations of edges that arise

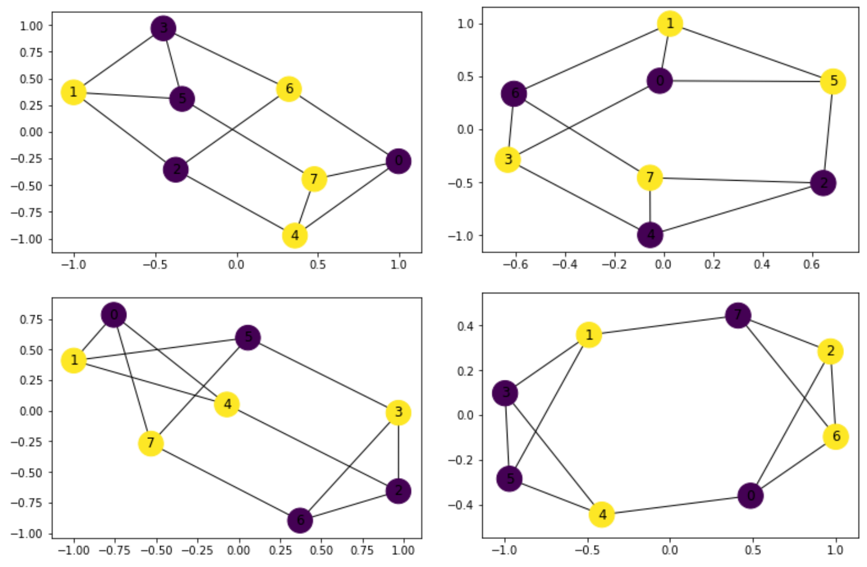
FIGURE 3.2: Examples of a 3-regular graph with 8 vertices.

from the geometry of the problem. Examples of that are shown in Figure 3.2 for a 3-regular graph with 8 vertices. Note that the MaxCut for this type of graph would be 12 for the ideal case (a cube) but due to the geometry of the graph, such cut is not achievable for the cases in 3.2 which have a MaxCut of 10 edges.

One way to evaluate the correctness of the cut obtained with the QAOA would be to compare it with the results of other well-established classical algorithms, but since we will be working with small graphs it will not be required.

Finally, it is also important to define the range of values that will be explored for γ and β by the optimization agent. In general, γ should be explored within the interval $[0, 2\pi)$ since C has integer eigenvalues, while β should lie between 0 and π. For the specific case of the MaxCut problem there are additional symmetries that allows the restriction of both parameters into a smaller range. Specifically, since $U(B, \frac{\pi}{2}) = (\sigma^x)^{\otimes N}$ it commutes through the circuit and thus it is possible to define the range of β as $[0, \frac{\pi}{2})$.

3.1.1 State of the Art

In [8] the authors suggest the possibility of evaluating $F_p = \langle \gamma, \beta | C | \gamma, \beta \rangle$ in a fine grid as long as the partitions are polynomial in n and m. Additionally, for the MaxCut problem, they present an optimization strategy consisting on dividing the whole graph in subgraph types which size depends on p, optimizing these subgraphs independently, and weighting its cut using the number of occurrences of each subgraph on the whole graph. These subgraphs arise when analyzing each clause C_{jk} in F_p. Specifically, due to the commutation of the Pauli-X and Pauli-Z operators applied to different qubits as in (3.10) (exemplified for p = 1).

$$\begin{aligned}\langle\gamma,\beta|C_{jk}|\gamma,\beta\rangle &= U^{\dagger}(C,\gamma)U^{\dagger}(B,\beta)(-\sigma_j^z\sigma_k^z+I)U(B,\beta)U(C,\gamma) = \\ &= U^{\dagger}(C,\gamma)\prod_l e^{i\beta\sigma_l^x}(-\sigma_j^z\sigma_k^z+I)\prod_l e^{-i\beta\sigma_l^x}U(C,\gamma) = \\ &= U^{\dagger}(C,\gamma)e^{i\beta(\sigma_j^x+\sigma_k^x)}(-\sigma_j^z\sigma_k^z+I)e^{-i\beta(\sigma_j^x+\sigma_k^x)}\prod_{l\neq j,k}e^{i\beta\sigma_l^x}e^{-i\beta\sigma_l^x}U(C,\gamma) \\ &= U^{\dagger}(C,\gamma)e^{i\beta(\sigma_j^x+\sigma_k^x)}(-\sigma_j^z\sigma_k^z+I)e^{-i\beta(\sigma_j^x+\sigma_k^x)}U(C,\gamma)\end{aligned} \tag{3.10}$$

In the same way, all the terms in $U(C,\gamma)$ that do not involve operators σ_j^z or σ_k^z will also cancel out. Thus, for p=1 each clause is only affected by its edge and its immediately adjacent edges. In general, for any p, each clause would be affected by those that are at distance smaller or equal than p.

In the same publication, it is also shown that the mean C(z) obtained sampling m^2 times from the state $|\gamma,\beta\rangle$ will be within 1 of $F_p(\gamma,\beta)$ with probability $1-\frac{1}{m}$. Moreover, the generated classical bit strings will then likely produce C(z) close to $F_p(\gamma,\beta)$. Further proof on the concentration of the objective function value is provided in [21].

Finally, 2-regular graphs (ring of disagrees) and 3-regular graphs are studied. It is concluded that, for the former, the algorithm obtains a cut of size $n\frac{2p+1}{2p+2}-1$ or bigger for the optimal configuration of parameters, which is close to the optimal one of n. Whereas for the latter, the algorithm produces a cut with size of at least 0.6924 of the optimal cut.

In [22] the authors first use BFGS to perform brute force optimization of the parameters for 3-regular graphs, obtaining for all graphs a non-degenerate global maximum and finding some tendencies on the evolution of γ_p and β_p as p increases. Specifically, γ is found to increase smoothly while β decreases smoothly with p. Moreover, for regular graphs of the same degree, the optimal parameters seem to occupy approximately the same region of the space independent of the graph geometry. From this tendencies, the authors propose an heuristic strategy to perform an optimization in polynomial time instead of the exponential time of the brute force approach. Essentially, they use the value of the angles for p to determine a good starting point for the search of the optimal parameters for p+1. To do so, the angular parameters are represented with the following Fourier-like transformation:

$$\begin{aligned}\gamma_i &= \sum_{k=1}^{q} u_k \sin\left[\left(k-\frac{1}{2}\right)\left(i-\frac{1}{2}\right)\frac{\pi}{p}\right] \\ \beta_i &= \sum_{k=1}^{q} v_k \cos\left[\left(k-\frac{1}{2}\right)\left(i-\frac{1}{2}\right)\frac{\pi}{p}\right]\end{aligned} \tag{3.11}$$

The optimized angular parameters for p will then be described by the tuple $(\vec{u^*},\vec{v^*})$. Using these tuple as heuristics to initialize γ_{p+1} and β_{p+1} (for O(poly(p)) iterations) proves to beat a brute force approach with random initialization, unless $2^{O(p)}$ number of executions are performed.

A different optimization approach is followed by Crooks [23]. Instead of optimizing the angle parameters for each specific graph, what is suggested is finding a global optimization protocol for an entire class of problem instances (for example graphs with the same edge probability). Thus, the optimal parameters are such that

$$(\beta^*, \gamma^*) = -\arg\min_{\beta,\gamma} \frac{1}{|T|} \sum_{C \in T} \langle \gamma, \beta | C | \gamma, \beta \rangle \tag{3.12}$$

With T the training set of problems and ∣T∣ the total number of problems in the training set. The quantum circuit defining the objective function is implemented into quantum virtual machine on top of Tensorflow and optimized using stochastic gradient descent and back-propagation. Note that since gradient descent is used the objective function multiplied by -1 so that the MaxCut for the graphs becomes the minimum of the objective function. Results show that the MaxCut obtained by the QAOA for p=8 is already better than the ones with Goemans-Williamson [**Goeamans**], shown for graphs with number of vertices $n \in [8, 17]$.

Guerreschi et Matsuura [24] study the computational time requirements of the QAOA algorithm when implemented on real quantum hardware and compare them with the AKMAXSAT classical solver [25]. The time required for the calculation of a single instance $|\gamma, \beta\rangle$, T is equal to

$$T = T_I + c_{depth} \cdot T_G + T_M \tag{3.13}$$

with c_{depth} the depth of the quantum circuit, T_I being the time required to prepare the initial state, T_G the time duration of the quantum gates of the system averaged over all types of gates used, and T_M the time required to measure the qubits. The total amount of time will then be multiplied by the total number of instances used during optimization. The projection of the results show that the computation time required for the QAOA using Nelder-Mead optimization would only surpass classical approaches when several hundreds of qubits would be used.

Wang et al. [26] present an analytical formula to compute classically $\langle \gamma, \beta | C_{jk} | \gamma\beta \rangle$ for $p = 1$ (3.14). Using this expression, the optimization of F_p can be performed in a fully classical manner.

$$\begin{aligned} \langle \gamma, \beta | C_{jk} | \gamma\beta \rangle = \frac{1}{2} + \frac{1}{4} \sin(4\beta) \sin(\gamma) \left[\cos^{d_j}(\gamma) + \cos^{d_k}(\gamma) \right] \\ - \frac{1}{4} \sin^2(\beta) \cos^{d_j + d_k + \lambda_{jk}}(\gamma) \left[1 - \cos^{\lambda_{jk}}(2\gamma) \right] \end{aligned} \tag{3.14}$$

$d_j + 1$ is the number of edges connected to vertex j and λ_{jk} the number of triangles in the graph that contain edge jk. Through this formula, the authors reach an expression (3.15) for F_p for triangle-free n-regular graphs, which will be useful when testing our approach on the ring of disagrees problem.

$$F_p = \frac{m}{2} (1 + \sin(4\beta) \sin(\gamma) \cos^(n - 1)(\gamma) \tag{3.15}$$

With m the number of edges in the graph. The previous expression is maximal for

$$F_p^* = \frac{m}{2} \left[1 + \frac{1}{\sqrt{n+1}} \left(\frac{1}{n+1} \right)^{\frac{n}{2}} \right]. \tag{3.16}$$

Finally, we mention one of the immediate applications of the QAOA algorithm presented in [27]. The authors use the QAOA algorithm to perform image segmentation in medical images. To do so, each image is cropped into smaller-scale images of 3x3 and 4x4 pixels. Each pixel in the image is associated with a node of the graph and every edge is weighted using the intensity of the pixel. Note that the problem would have the same geometry than the 2-D Ising Lattice. The problem is then assimilated to the max-flow min-cut problem, and successfully solved for a Coronary angiogram image.

3.2 QAOA Environment

3.2.1 Discrete

The first environment to be tested will try to optimize a discrete version of the QAOA algorithm. There are several reasons that lead us to choose this as starting point. Firstly, and although we will be using a quantum computer simulator as back-end, if we were to utilize this environment in combination with a real quantum computer, we might not have the tools to generate all the continuous angular spectrum for γ and β. Secondly, training the discrete environment should be easier than a continuous one while obtaining relevant results.

The environment stores the number of qubits of the graph and its edges, and the number of partitions into which each angle dominion is discretized. The environment has the following properties:

- *Initial State*: Initialize each qubit into $|+\rangle$ state. And return the first observation.
- *Agent's action*: Select a site of the discretized (γ, β) lattice defining a value for γ_p and β_p. Apply operators U(C,γ_p) and U(B,β_p) to the current quantum state.
- *Action effect on the environment*: Quantum state modified from $|\gamma, \beta\rangle_{p-1}$ to $|\gamma, \beta\rangle_p$.
- *Observation*: There are different observations that could be relevant in principle. Firstly, and as suggested by [22], γ increases steadily with p while β decreases. Therefore, it would make sense to include the angles obtained in p-1 to predict the ones in p (not the whole set of angles since the dimension of the input would then be different at each step). Moreover, although we want to maximize $\langle\gamma, \beta| C |\gamma, \beta\rangle$, there are many other properties from the quantum state at our disposal, that can be obtained measuring with different operators. Specifically, we plan on using $\{\sigma_i^x\}$, $\{\sigma_i^y\}$ and $\{\sigma_i^z\}$. Using these operators, the initial observation used by our agent will be:

$$\begin{aligned} \langle +|_i \sigma_i^z |+\rangle_i &= 1, \\ \langle +|_i \sigma_i^y |+\rangle_i &= 0, \\ \langle +|_i \sigma_i^x |+\rangle_i &= 0, \forall i \in [1, n] \end{aligned} \tag{3.17}$$

  With n the number of qubits. We can also use other observables generated with combinations of products of Pauli matrices for multiple qubits. On the other hand, although using the whole wave function of the state is theoretically possible, the dimension of the observation would then increase exponentially with the number of qubits on the graph, making the classical optimization infeasible.

- *Reward*: For the reward system we also have several logical options. First and foremost, since the length of the episodes is fixed at p steps, we can define a reward of 0 for all steps before step p and a reward of F_p for that last episode. With this approach we expect the algorithm to be less likely to get stuck in optimal states achieved for smaller p, making the rest of rotations irrelevant (with angles equal to 0).
  On the other hand, we could also define an incremental reward for each step of the episode (F_1, $F_2 - F_1$, $F_3 - F_2$, and so on), which would result in the same total reward than the first definition but increasing the likelihood of finding optimal configurations for smaller p. Additionally, the reward could also be computed as the difference between the obtained F_p and the best F_p observed during training.

As for the implementation of the quantum circuit, a universal quantum source language called *Qibo* will be used in combination with a quantum virtual machine *vqm-lite* as they are developed by our group and thus their intricacies are better known. Other options would include *Project-Q* [28] or *QCGPU* [29] both of which are open-source and also implemented in our environment, but not tested.

It is important to note that since we are working with a quantum computer simulator, we can obtain the exact wave function of the $|\gamma, \beta\rangle$ state and thus compute also F_p exactly for this state (with vector-matrix products). On the other hand, when using a quantum computer, evaluating F_p would be done by averaging the cut of the bitstrings, obtained when measuring the $|\gamma, \beta\rangle$ on the computational basis, over several iterations.

Agent Properties

Regarding the agent structure, a dueling network architecture will be used, with average dueling type as in equation (2.9), and trained using the DDQN algorithm. Such agent will be implemented using the DQN agent from the Keras-rl package. Note that for such implementation the network for both streams shares the same weights except for the output layer. This might limit the network performance, but should suffice for the goals of this work. As it will be seen in the results, the size of the network (both its number of units and layers) will increase as p increases. Moreover, the agent will use the Boltzmann policy as action selection policy, a soft-update of the network weights, and Adam as optimization algorithm. Finally, it is relevant to point out that the Keras-rl DQN agent uses the Huber-loss function as minimization objective.

3.2.2 Continuous

The continuous version of the discrete environment will have the exact same properties except for the fact that the angle parameters applied to the environment will not be obtained from a discretized lattice.

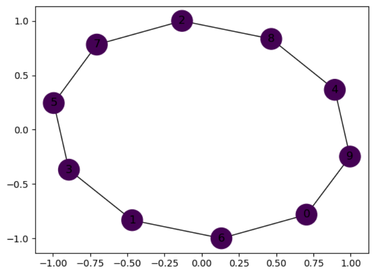

FIGURE 3.3: Unsolved ring of disagrees graph.

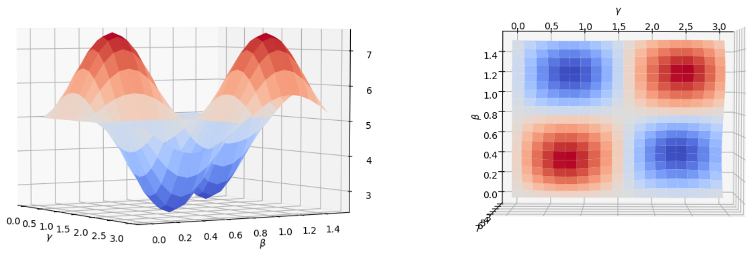

FIGURE 3.4: F_1 for the ring of disagrees graph.

Agent Properties

For the continuous version of the environment we will work with a DDPG agent again implemented using the Keras-rl package. The network architecture will depend on the value of p, but all hidden layers will use ReLU as its activation function (as for the discrete case), soft updates for the target network will be used and Adam as optimization algorithm. The random term added to $\mu(s)$ will be sampled from a Ornstein-Uhlenbeck process.

3.3 Results

3.3.1 Ring of Disagrees

We will start testing our approach on the ring of disagrees graph, Figure 3.3, as its maximum value for F_p is known exactly to be $n\frac{2p+1}{2p+2}$. To begin with, the dependence of F_p on γ, β is studied for $p = 1$ evaluating F_1 in a discretized γ, β lattice with 20 partitions (Figure 3.4). Two maximums and the same amount of minimums are observed, and the maximum value for F_1 is 7.5 as expected for a graph of n = 10 nodes.

Now that we know which are the optimal angles for $p = 1$, we start using our discrete environment for the same value of p. The network has a unique hidden layer with 64 units. The learning rate used for the Adam algorithm is 1e-3 which is also the τ_{update} used for the soft update oF the target network weights.

The agent is trained over 10.000 steps with a memory limit of 500 steps. Each angular parameter is divided in 20 partitions. The observations used by the agent are the angles for the previous step (with initial value of 0 for both) and the average value of σ_i^x, σ_i^y and σ_i^z.

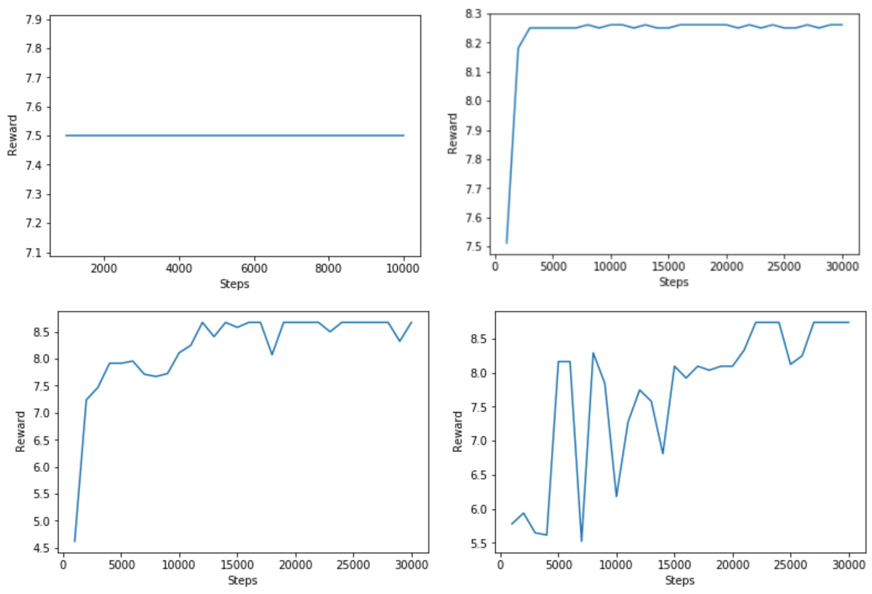

FIGURE 3.5: Evolution of the reward during training, tested every 1.000 steps. $P = 1$ (Top-left), $p = 2$ (Top-right), $p = 3$ (Bottom-left), $p = 4$ (Bottom-right).

We observe that the network is particularly sensible to the τ_B parameter of the action selection policy used, which controls the balance between exploration and exploitation of the algorithm. For higher τ_B values the agent visits higher reward states in less number of steps, but the algorithm is highly unstable and the weights of the network do not converge to the desired ones. This instability increases as p increases as we will see next.

For $\tau_B = 0.55$ and sampling 32 instances from the memory at each step the algorithm converges rapidly towards the state in Table 3.1.The evolution of the reward during training is displayed in Figure 3.5. We observe that the algorithm is stable.

For $p = 2$ we try to initiate the training using the same network structure and using the weights of the optimal configuration for $p = 1$ as initialization. This approach is repeated for the next p as well (using the optimal configuration for p-1). However, it performs poorly in all cases. There are two reasons that justify this poor performance. First and foremost, by looking at the optimal results shown in Table 3.1, it is clear that γ_i of the optimal state for different p values is modified. This is also the main issue to be dealt with when trying to use the network trained for a certain p to obtained results for higher p values. Second, we will see that the architecture of the network changes with p and, therefore, its weights can not be initialized with the previous p weights.

If we initialize the weights randomly and train for 30.000 steps with $\tau_B = 0.35$ the results obtained are shown in Figure 3.5. The network consists of two hidden layers of 64 units each. Note that the maximum observed value is obtained before 10.000 steps of training (5.000 episodes), while if we tried to evaluate every point of the discretized 4-dimensional angular lattice we would require 160.000 episodes.

Ring of Disagrees Results				
p	γ	β	F_p	$Max(F_p)$
1	$\gamma_1 = 2.3561$	$\beta_1 = 1.1781$	7.5	7.5
2	$\gamma_1 = 1.2566$ $\gamma_2 = 2.5133$	$\beta_1 = 1.2566$ $\beta_2 = 0.7069$	8.262	$8.\bar{3}$
3	$\gamma_1 = 2.3562$ $\gamma_2 = 0.6283$ $\gamma_3 = 1.2566$	$\beta_1 = 0.6283$ $\beta_2 = 0.3142$ $\beta_3 = 1.1781$	8.670	8.75
4	$\gamma_1 = 2.5132$ $\gamma_2 = 2.1991$ $\gamma_3 = 2.1991$ $\gamma_4 = 1.8849$	$\beta_1 = 0.9424$ $\beta_2 = 1.0996$ $\beta_3 = 1.0996$ $\beta_4 = 1.2566$	8.74	9

TABLE 3.1: Ring of disagrees results with 20 partitions per angle.

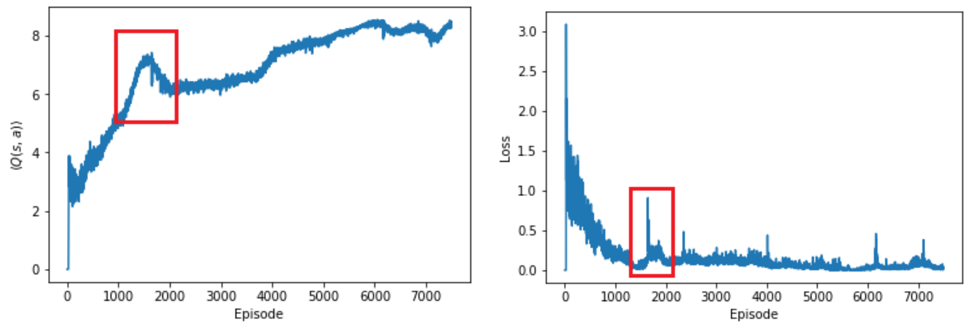

FIGURE 3.6: Mean Q-value (left) and Huber loss (right) for each step of an episode with τ_B=0.20 and a network with 4 hidden layers and 128 units.

For $p = 3$ similar results are found, with simply increasing the number of hidden layers to 3 and the number of units per layer to 96. The algorithm is a bit more unstable but a great result is obtained with less than 20.000 steps (around 7.000 episodes).

For $p = 4$ the algorithm is not able to converge for tau_B = 0.35 with networks of <5 layers of depth. Reducing the τ_B parameter to 0.20 allows for the algorithm to converge with only 2 layers of 64 units per layer depth, but in expense of reducing exploration and an F_p of 8.65 is obtained after 40.000 steps. If we increase the number of layers to 4 and the number of units to 128 we reach the results in Figure 3.5, with the the algorithm being less stable. The evolution of the mean Q(s,a) value of each episode for this configuration is shown in Figure 3.6, as well as the evolution of the loss function. We observe that the source of the oscillation in the reward is due to the overestimation of a local optima (red box). Also, for $p = 4$ the memory limit is increased to 1.000 steps to maintain a reasonable amount of episodes.

In Figure 3.7 the optimal states for each p are sampled 100.000 times and the probability of obtaining a cut depending on its size is shown. As predicted by the theory, the cut size of the samples is very close to the F_p value of the state. The obtained bitstring with maximum cut for the ring of disagrees problem is shown in Figure 3.8.

As a final remark, it is relevant to point out that the results obtained by the network might vary within each execution as the algorithm might get stuck in local

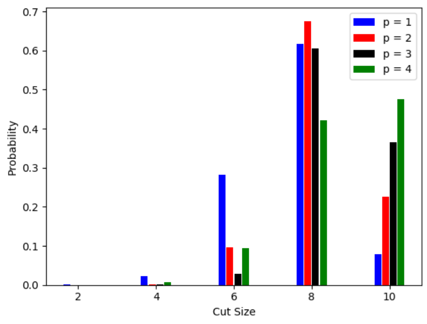

FIGURE 3.7: Probability of obtaining a certain cut size of the ring of disagrees problem for $p = 1$ (blue), $p = 2$ (red), $p = 3$ (black) and $p = 4$ (green).

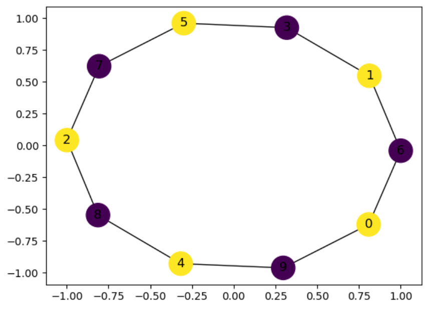

FIGURE 3.8: Maximum cut for the ring of disagrees.

optima. Nonetheless, although we might not reach the exact optimal value at each execution, very similar results are obtained.

3.3.2 Irregular Graph

After testing the algorithm for a 2-regular graph we will do the same for a graph in which all nodes have 2 edges except for two nodes with a degree of 3. The structure of the graph is shown in Figure 3.9. Again we explore F_1 within a lattice of 20 partitions (Figure 3.13) and observe that for this problem there is only a global maximum and a global minimum, and additionally, there is a local maximum and a local minimum.

We use the same parameter configuration than for the previous graph (as they are pretty similar with the irregular having less nodes). The results are shown in Table 3.2 and Figure 3.11. Again we are able to achieve a pretty stable training with better results for higher p value.

In Figure 3.12 the probability of obtaining a bitstring yielding a certain cut is shown. As for the ring of disagrees , the produced bitstrings are concentrated around F_p.

With these results, both for a regular and irregular graph, we have performed a proof of concept of the possibility of using RL as optimization algorithm for the QAOA for

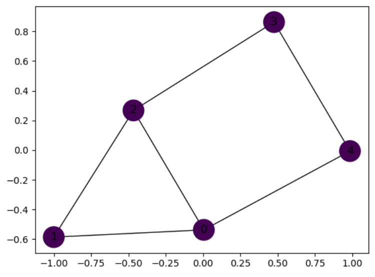

FIGURE 3.9: Structure of the irregular graph.

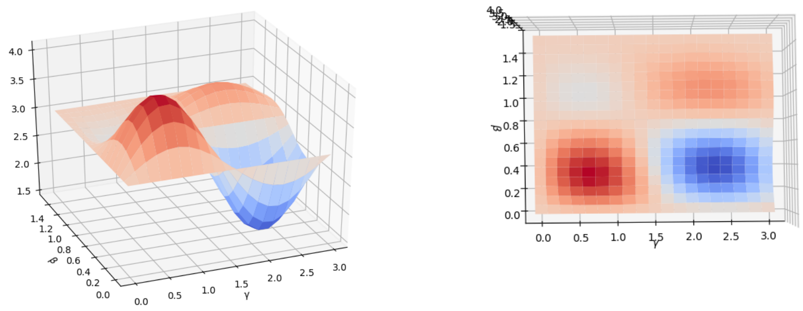

FIGURE 3.10: F_1 for the irregular graph.

a certain p. However, there is a further development that could be done, which is to train the network to be able to optimize all p values at the same time. Ideally, we would like our network to be able to extrapolate the optimization strategy to any p even if we train it for smaller p values. In this regard we propose a training strategy in which per each episode the p value (number of steps per episode) is chosen randomly within a certain range.

To get an insight on the difficulties that might arise, we plot in Figure 3.13 the interpolated Q(s,a) function of the first set of parameters, γ_1 and β_1, for the optimal networks obtained at each p. It is clear that the surface changes with p as the path to reach better states for higher p is different. However, the localization of the maximum is more or less stable. Further tests need to be done for the other episode steps in order to validate the proposed training strategy.

Finally, for this irregular graph we perform some tests using the continuous environment. We test the continuous approach for $p = 2$ and $p = 3$ without using the wave function of $|\gamma, \beta\rangle$ as inputs. For $p = 2$ using 2 hidden layers with 32 neurons per layer and a learning rate of 0.001 for the Adam optimization algorithm we obtain a maximum F_p of 4.190. For $p = 3$ only increasing the amount of units per layer to 100 we are able to reach a maximum F_p of 4.665. Further exploration of the continuous remains to be done. The graph configuration producing maximum cut for the irregular graph is shown in Figure 3.14.

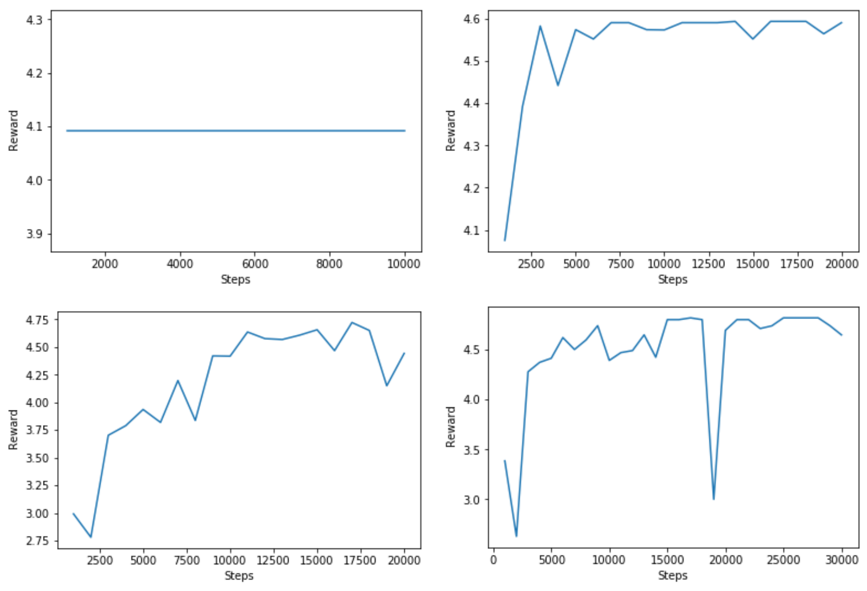

FIGURE 3.11: Evolution of the reward during training for the irregular graph, tested every 1.000 steps. $P = 1$ (Top-left), $p = 2$ (Top-right), $p = 3$ (Bottom-left), $p = 4$ (Bottom-right).

Irregular Graph Results				
p	γ	β	F_p	$Max(F_p)$
1	$\gamma_1 = 0.6283$	$\beta_1 = 0.3142$	4.091	-
2	$\gamma_1 = 0.6283$ $\gamma_2 = 1.0995$	$\beta_1 = 0.5498$ $\beta_2 = 0.3142$	4.593	-
3	$\gamma_1 = 0.6283$ $\gamma_2 = 0.9425$ $\gamma_3 = 0.9425$	$\beta_1 = 0.4712$ $\beta_2 = 0.3142$ $\beta_3 = 0.1571$	4.721	-
4	$\gamma_1 = 0.4712$ $\gamma_2 = 0.7854$ $\gamma_3 = 0.6283$ $\gamma_4 = 0.6283$	$\beta_1 = 0.4712$ $\beta_2 = 0.3142$ $\beta_3 = 0.1571$ $\beta_4 = 0.1571$	4.818	-

TABLE 3.2: Irregular graph results with 20 partitions per angle.

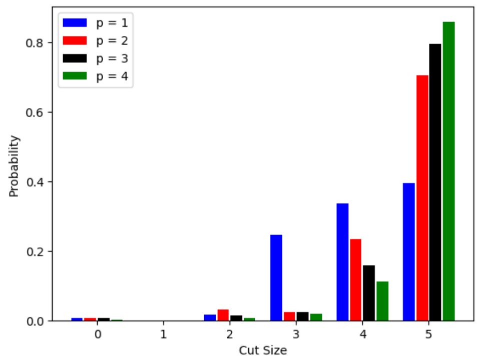

FIGURE 3.12: Probability of obtaining a certain cut size of the irregular graph for $p = 1$ (blue), $p = 2$ (red), $p = 3$ (black) and $p = 4$ (green).

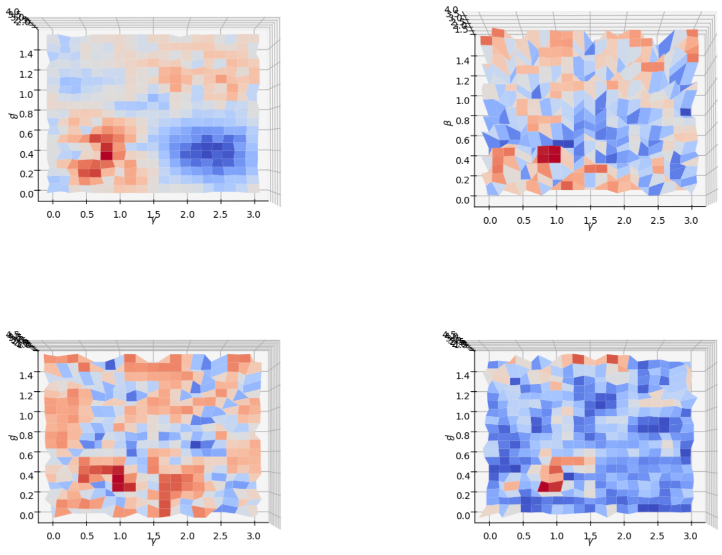

FIGURE 3.13: Interpolated Q value function for the first set of parameters (γ_1, β_1) for the irregular graph. P=1 (top-left), p=2 (top-right), p=3 (bottom-left), p=4 (bottom-right).

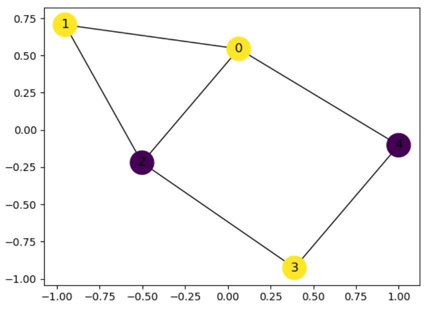

FIGURE 3.14: Maximum cut for the irregular graph.

Chapter 4

Conclusions

In the first part of this work, we have designed a RL strategy to solve the MaxCut problem for the 2-D Antiferromagnetic Ising Model, yielding good performance for small lattices. The approach consists on using the lattice configuration as input to the optimization agent.

Similarly, and inspired on the previous approach, we suggest characterizing the $|\gamma, \beta\rangle$ state with the use of either the expected value for the Pauli operators for each qubit, or its complete wave function, and then employ it as observation for the optimization of the QAOA algorithm. This represents a novel formulation, in contrast to the variational eigensolver approaches typically used for the optimization of quantum problems (*Reinforcement Learning Methods for Quantum Approximate Optimization Algorithm*, Artur Garcia and Jordi Riu, in preparation).

As proof of concept, we tested this modus operandi both for small regular and irregular graphs, using discretized and continuous RL algorithms, and for several p-levels, obtaining close to optimal results with few steps of training. We are currently working on performing further exploration with higher complexity graph and bigger p values.

Additionally, more complex state representations using products of these operators for several qubits could be used. Specifically, it seems promising to use the expected value of each individual clause of the objective function as input, or an analogous operator using σ^x and σ^y.

The obtained results suggest the possibility of using a global training strategy for all p. Ultimately, we would want our network to receive p as an input value and return the optimal angle configuration for it. To do so, we plan on using a training strategy in which the length of the episode is chosen at random at the start of the episode. As seen in the results for the irregular graph, the optimal angle configuration at a certain step varies with the length of the episode, which suggests that the optimization problem will require more sophisticated network models to be reproduced.

Another relevant characteristic of our work is the fact that all the computation of the quantum estimates have been performed using the exact wave function of our state. For higher number of qubits or when using a quantum computer, these quantities have to be estimated taking several measures of the properties of the state. Thus, the amount of samples becomes a new parameter to take into consideration for the performance of the agent.

In conclusion, this work represents the first step towards finding a general RL approach for the optimization of the QAOA algorithm for the MaxCut problem.

Bibliography

[1] Seth Lloyd, Masoud Mohseni, and Patrick Rebentrost. “Quantum principal component analysis”. In: *Nature Physics* 10 (2014), pp. 631–633. DOI: 10.1038/nphys3029. arXiv: 1307.0401 [quant-ph].

[2] Patrick Rebentrost, Masoud Mohseni, and Seth Lloyd. “Quantum Support Vector Machine for Big Data Classification”. In: 113, 130503 (2014), p. 130503. DOI: 10.1103/PhysRevLett.113.130503. arXiv: 1307.0471 [quant-ph].

[3] S. Aaronson. “Read the fine print”. In: *Nature Physics* 11 (Apr. 2015), pp. 291–293. DOI: 10.1038/nphys3272.

[4] Vittorio Giovannetti, Seth Lloyd, and Lorenzo Maccone. “Quantum Random Access Memory”. In: 100, 160501 (2008), p. 160501. DOI: 10.1103/PhysRevLett.100.160501. arXiv: 0708.1879 [quant-ph].

[5] Vedran Dunjko, Jacob M. Taylor, and Hans J. Briegel. “Quantum-Enhanced Machine Learning”. In: *Phys. Rev. Lett.* 117 (13 2016), p. 130501. DOI: 10.1103/PhysRevLett.117.130501. URL: https://link.aps.org/doi/10.1103/PhysRevLett.117.130501.

[6] Steven H. Adachi and Maxwell P. Henderson. “Application of Quantum Annealing to Training of Deep Neural Networks”. In: *arXiv e-prints*, arXiv:1510.06356 (2015), arXiv:1510.06356. arXiv: 1510.06356 [quant-ph].

[7] Mohammad H. Amin et al. “Quantum Boltzmann Machine”. In: *arXiv e-prints*, arXiv:1601.02036 (2016), arXiv:1601.02036. arXiv: 1601.02036 [quant-ph].

[8] Edward Farhi, Jeffrey Goldstone, and Sam Gutmann. “A Quantum Approximate Optimization Algorithm”. In: *arXiv e-prints*, arXiv:1411.4028 (2014), arXiv:1411.4028. arXiv: 1411.4028 [quant-ph].

[9] Jordi Riu and Rossend Rey. “Microscopic Models Applied to Financial Markets”. In: (2015).

[10] Christopher Watkins. “Learning From Delayed Rewards”. In: (Jan. 1989).

[11] Volodymyr Mnih et al. “Human-level control through deep reinforcement learning”. In: *Nature* 518.7540 (Feb. 2015), pp. 529–533. ISSN: 00280836. URL: http://dx.doi.org/10.1038/nature14236.

[12] Hado V. Hasselt. “Double Q-learning”. In: (2010). Ed. by J. D. Lafferty et al., pp. 2613–2621. URL: http://papers.nips.cc/paper/3964-double-q-learning.pdf.

[13] Hado van Hasselt, Arthur Guez, and David Silver. “Deep Reinforcement Learning with Double Q-learning”. In: *CoRR* abs/1509.06461 (2015). arXiv: 1509.06461. URL: http://arxiv.org/abs/1509.06461.

[14] Ziyu Wang, Nando de Freitas, and Marc Lanctot. “Dueling Network Architectures for Deep Reinforcement Learning”. In: *CoRR* abs/1511.06581 (2015). arXiv: 1511.06581. URL: http://arxiv.org/abs/1511.06581.

[15] Shixiang Gu et al. "Continuous Deep Q-Learning with Model-based Acceleration". In: *arXiv e-prints*, arXiv:1603.00748 (2016), arXiv:1603.00748. arXiv: 1603.00748 [cs.LG].

[16] Timothy P. Lillicrap et al. "Continuous control with deep reinforcement learning". In: *arXiv e-prints*, arXiv:1509.02971 (2015), arXiv:1509.02971. arXiv: 1509.02971 [cs.LG].

[17] Metropolis NS et al. "Equation of State Calculations by Fast Computing Machines". In: *Journal of Chemical Physics* 21 (Jan. 1953), pp. 1087–1092.

[18] Richard H. R. Hahnloser et al. "Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit". In: *Nature* 405 (2000), pp. 947–951.

[19] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *arXiv e-prints*, arXiv:1412.6980 (2014), arXiv:1412.6980. arXiv: 1412.6980 [cs.LG].

[20] Patrick J. Coles et al. "Quantum Algorithm Implementations for Beginners". In: *arXiv e-prints*, arXiv:1804.03719 (2018), arXiv:1804.03719. arXiv: 1804.03719 [cs.ET].

[21] Fernando G. S. L. Brandao et al. "For Fixed Control Parameters the Quantum Approximate Optimization Algorithm's Objective Function Value Concentrates for Typical Instances". In: *arXiv e-prints*, arXiv:1812.04170 (2018), arXiv:1812.04170. arXiv: 1812.04170 [quant-ph].

[22] Leo Zhou et al. "Quantum Approximate Optimization Algorithm: Performance, Mechanism, and Implementation on Near-Term Devices". In: *arXiv e-prints*, arXiv:1812.01041 (2018), arXiv:1812.01041. arXiv: 1812.01041 [quant-ph].

[23] Gavin E. Crooks. "Performance of the Quantum Approximate Optimization Algorithm on the Maximum Cut Problem". In: *arXiv e-prints*, arXiv:1811.08419 (2018), arXiv:1811.08419. arXiv: 1811.08419 [quant-ph].

[24] G. G. Guerreschi and A. Y. Matsuura. "QAOA for Max-Cut requires hundreds of qubits for quantum speed-up". In: *arXiv e-prints*, arXiv:1812.07589 (2018), arXiv:1812.07589. arXiv: 1812.07589 [quant-ph].

[25] Adrian Kuegel. "Improved Exact Solver for the Weighted MAX-SAT Problem". In: EPiC Series in Computing 8 (2012). Ed. by Daniel Le Berre, pp. 15–27. ISSN: 2398-7340. DOI: 10.29007/38lm. URL: https://easychair.org/publications/paper/p3wf.

[26] Zhihui Wang et al. "Quantum approximate optimization algorithm for Max-Cut: A fermionic view". In: *Physical Review A* 97, 022304 (2018), p. 022304. DOI: 10.1103/PhysRevA.97.022304. arXiv: 1706.02998 [quant-ph].

[27] Lisa Tse et al. "Graph Cut Segmentation Methods Revisited with a Quantum Algorithm". In: *CoRR* abs/1812.03050 (2018). arXiv: 1812.03050. URL: http://arxiv.org/abs/1812.03050.

[28] Damian S. Steiger, Thomas Häner, and Matthias Troyer. "ProjectQ: An Open Source Software Framework for Quantum Computing". In: *arXiv e-prints*, arXiv:1612.08091 (2016), arXiv:1612.08091. arXiv: 1612.08091 [quant-ph].

[29] Adam Kelly. "Simulating Quantum Computers Using OpenCL". In: *arXiv e-prints*, arXiv:1805.00988 (2018), arXiv:1805.00988. arXiv: 1805.00988 [quant-ph].