Thesis Dissertation

Quantum Approximation Optimization Algorithms Implementation for Travelling Salesman Problem

Ioannis Pastellas

UNIVERSITY OF CYPRUS

DEPARTMENT OF COMPUTER SCIENCE

May 2021

UNIVERSITY OF CYPRUS

DEPARTMENT OF COMPUTER SCIENCE

Quantum Approximation Optimizations Algorithms Implementation for The Travelling Salesman Problem

Ioannis Pastellas

Advisor
Associate Professor Anna Philippou

Thesis submitted in partial fulfilment of the requirements for the award of

degree of Bachelor in Computer Science at University of Cyprus.

May 2021

Acknowledgements

First and foremost, I would like to thank my professor Dr. Anna Phillipou, for accepting to be my advisor for this thesis dissertation. She helped me a lot through this work, by being patient (especially on some difficult moments) and providing a lot of practical resources. In this point I ought to thank all the professors that equipped me with an unmeasurable knowledge through this 4-year journey and made me reinvent the view about what Computer Science really is. I would also like to thank my family and friends, who provide support and stayed close to me through the most difficult parts of the 4 year and especially on this work. Finally, I would like to thank Sotiris Loizidis, who gave me useful insights that were necessary to complete this study both in the subject of the study and on the design of the thesis.

We acknowledge the use of IBM Quantum services for this work. The views expressed are those of the authors and do not reflect the official policy or position of IBM or the IBM Quantum team.

Abstract

Quantum computing is an uprising technology, which combines two of the most marvellous and powerful sciences, Physics and Computer Science and it is making huge steps towards commercialization and becoming the new black. Although it is, as of this date, more on the scientific and research side, it started to become commercialised as tech giants(Google, IBM, Amazon) invested to it and provided their resources through the Cloud. However, the quantum computing is still at its infancy stages. Throughout the thesis, this statement was made clear to our eyes. But Quantum Computing will definitely be the reason why many fields will rise through it in the near future.

This thesis aims to provide first, an introduction to the basics of Quantum Computing, the Quantum Approximation Optimization Algorithm, and the Travelling Salesman Problem, and second the implementation of QAOA approaches of solving the Travelling Salesman Problem on the IBM Q machines.

Overall, we conclude that, while Quantum computing is still far from be commercialized, there are some benefits and resources available, mainly through quantum services that are available through the cloud which can be used to construct viable solutions to real-world problems. Also, we give our insight about the results of the algorithms both in simulator and real quantum machines. But there are a lot of limitations and problems with the existing hardware available in the cloud that restrict future applications, that’s why the current milestones of Quantum Computing providers are to overcome these limitations.

Contents

Introduction

1.1 **Motivation**

The greatest motivation for working with quantum computer can be easily determined. It is nothing else than its computational power and its great potential of solving hard complex problems that nowadays computers struggle to solve. Computing was introduced many years ago to solve problems much more easily and much more efficiently than humans could do. And has successfully done that in the years that it followed. Yet classical computing while it can solve a lot of problems efficiently, there are still problems that even a computer cannot solve, at least in any reasonable time. These problems are also known as NP hard problems.

Another obstacle in classical computing is that the hardware available has seem to find its limit. Moore's Law which states that the power of computing hardware doubly increasing seems to be frozen and slowed down.

These limitations of classical computing led scientist to think and find an alternative way of doing computation and that uses different hardware. The beginning of the Quantum computing can be marked by two critical moments in the timeline. The first can be found in the 1980 when Paul Benioff(physicist) presented a universal quantum mechanical model for computation where he showcased the theoretical feasibility of Quantum Computers to be reversible. A few years later, in 1982, the whole concept of computer that could simulate physical phenomena and used as a tool of computation was introduced by the famous physicist Richard Feynman, who stated also that the existing computers couldn't efficiently simulate the evolution of quantum systems and hence, quantum physical phenomena which are the core of Physics. But Quantum Computing started attracting attention, when a practical quantum algorithm was developed, Shor's Algorithm. Shor developed an efficient quantum algorithm for prime factorization, which is the essential problem of breaking various cryptosystems and especially the famous RSA encryption. This was enough for quantum computing to gain popularity and to raise money for further research.

The simulation of a Quantum System as Richard Feynman stated is grand example of the limitations of the Classical Computers. Classical Computing is based on Newtonian Mechanics where its state is deterministically determined and processed thus having a sequential nature. But when we want to describe the dynamics of a Quantum System and we have to take into account the concepts of "Superposition" and "Entanglement" and this fact proposed a state space much larger than the classical one. But these concepts are the gust of the computational potential of Quantum Computers.

In addition to that, new algorithms and quantum computing schemes were developed over the years, such as Grover's search, Adiabatic Quantum Computation, Variational Quantum eigen solver etc., and set the pillars of reimagining the fields of Cybersecurity, Optimization, Chemistry, Drug Development and Artificial Intelligence.

Surely, the scenario of personal Quantum computers is very unlikely to come to reality, at least not for many years. But we live in the age of emerging technologies such as Cloud Computing Services, which will be the main channel to real quantum systems, where practical applications of Quantum Computing will be developed. Right now, technological giants like Google, Microsoft, Amazon, IBM and many others are investing in Quantum Computing research and development and when this happens you have to pay attention. Not only that but many other quantum-based start-ups are slowly forming the Quantum Computing market.

For this undergraduate thesis, the goal is to learn about Quantum Computing what are its promises on the computation and the NP-hard problems, get familiar with the basic and essential quantum computing concepts as well as with the quantum computing programming. Through implementation of quantum algorithms on both simulators and on real quantum machines, the scope is to see the capabilities of the existing algorithms and hardware on solving hard problems.

1.2 **Thesis Overview**

Quantum Computing is a revolutionary concept that combines two of the most important Sciences, Physics and Computer Science, in a completely novel way (I don't mention Mathematics because its already the heart of both of them). This thesis focusing on the implementation of some algorithms using the quantum programming tool Qiskit on Python. Based on that implementation we run some various experiments to test the implementation and to gain insights about the algorithms, quantum computation and the hardware available. We construct the experiments on Jupyter Notebook which is a tool for a more interactive engagement with Python. Notebooks are a way of interaction with the IBM Q Cloud Systems. Before proceeding with the implementation and the experiments we present the theoretical background needed for the experiments.

Before proceeding with Qiskit experimentation we must state a few disclaimers as well. Below we provide what is **not** a goal of this thesis.

- This thesis is not claiming to **rigorously** prove physical proofs of why these Quantum Systems work or how they work. Whatever we provide, is based on our level of perception and understanding of how the mechanisms and algorithms work.

1.3 **Thesis Contribution**

Our contribution is to provide the implementation of some algorithms that may not be available using the IBM's Qiskit. The code for the implementation can be found in our repository in github, https://github.com/ipaste01/ThesisUCY-QAOA. Beside the implementation of the algorithms, we provide our insight on the algorithms through experimental evaluation of them on different instances and different configuration. This insight is based in our perspective and on our understanding of the algorithms.

1.4 **Thesis Outline**

In the first chapter, the incentive of this thesis and the current picture of Quantum Computing was presented. We also raised some research questions that will be answered by the end of this thesis. In the second chapter, we will talk about some previous work that is associated with Quantum Computing and Travelling Salesman Problem. In addition, we will present the recommended background knowledge required for this thesis. In the third chapter, we will present the primers of Quantum Computing, the basic elements that are used to perform this fascinating computation and it will complete the background. In the fourth chapter, we will talk about some basic Quantum Complexity Theory and how Quantum Computing impacts the Complexity Domain. In the fifth chapter, we introduce the famous Travelling Salesman Problem, its description and previous attempts (both classical and quantum) of solving the problem. In the sixth chapter, we introduce and explain the Quantum Approximate Optimization Algorithm, an algorithm for solving combinatorial problem like TSP, and also the new variation of the algorithm. In the seventh chapter, we showcase and implement a QAOA approach for solving the problem using the standard version of QAOA. In the eighth chapter, we another QAOA approach for the TSP but using the the Alternating version of QAOA and a constraint sub-space of feasible solutions as well as some experiments to understand both the algorithm and the Quantum Computing in general. That chapter will be the core of the code that is on GitHub. In the ninth chapter, we present our technical and genral conclusions concerning the hardware of Quantum Computation and discuss limitations of the current hardware, especially when it comes to algorithms such as the QAOA and of course future work.

Chapter 2

Related Work and Background

2.1 **Related Work**

Since this thesis is multi-purposed, there is a number of different works that are close to our efforts.

2.1.1 **Quantum Computing Programming tools**

One of the goals of this thesis was the implementation of some quantum algorithms. To do that, I needed to pick a Quantum Programming tool of the ones that were available.

The quantum programming tools that I consider was IBM's Qiskit, Righetti's Pyquil and Xanadu's Pennylane. At that time of decision, only the two first have real quantum machines available on the Cloud. So, the decision was between the two. I selected IBM's Qiskit over Pyquil for various reasons. First is that Qiskit provide a textbook that introduce various quantum computing concepts and algorithms among with their implementation in Qiskit and you can learn and practice at the same time. In general, for a beginner with Quantum Computing, Qiskit might be more suitable tool.

But the right tool depends on what is the target of someone. For example, when implementing and evolving circuits consist of Pauli matrices which is much easier for the QAOA to be build. PyQuil may be better library for implementing these kinds of circuits on a lower level. If the target is Quantum Machine Learning, then Xanadu's Pennylane is more specialized tool. Xanadu lately started to offer some services in the cloud.

2.1.2 IBM Quantum Experience and Qiskit

Qiskit is an open-source framework developed by IBM Research. Qiskit is mainly developed in Python and is cross-platform compatible and it is pipelined in other libraries like Pennylane. The Qiskit textbook is a very useful supplement to quantum computing courses. It provides some very useful quantum computing material that can prepared you for the field much more sooner. All of Qiskit can be encapsulated in the IBM Q platform, . IBM offers free experimentation on real quantum systems, as well as simulators that work under the Qiskit framework. All of the above can be found here [25], [29], [30]

IBM Q offers various machines of 5 qubits with tolerable error rate. It also offers a system with 15 qubits which usually is the one someone will want to use since it can provide larger problems. This system, however, has more error rate than the other machines and the larger the circuit the more visible this error is. In IBM Q, one places circuits (jobs) to a queue of the machine he/she selects to run them. The more jobs in the queue the longer the time for the job to completed.

2.2 Background

2.2.1 Methodology

For this thesis, we have worked with IBM'S Qiskit Library for implementation of the algorithms on simulators and using real machines via IBM Q Experience. Before implementing something, we try to understand it. We have deployed a repository in GitHub where we keep all our code there. It also includes a Readme.txt where it describes the way of running the code.

2.2.2 Important Python Packages for Quantum Computing

Besides the libraries and the frameworks that enable you perform Quantum Computation, there are some important libraries that someone definitely will use when experimenting with Quantum Computing.

1) NumPy

It provides a lot of functionality in working with high-dimensional arrays and in the domain of Linear Algebra and Matrices. For quantum computing, this is very important as the process involves a lot of Linear Algebra operations. In addition to that, NumPy is great on dealing with parameter and function optimization arrays and to create parameters grid which in this thesis was very useful. The full documentation can be viewed [31] .

2) Matplotlib

Matplotlib is library for the visualization part of the experimentation of Quantum Computing. While it is not as important and frequent as NumPy, it is used to visualize the results of the experiments with plots, or in some cases to gain sone insights about some other fractions of the experiment.The full documentation can can be viewed [30].

2.3 Mathematical and Theoretical Physics Background

2.3.1 Quantum Complexity

Just like the classical computation complexity theory it is important to the world of classical computing, the same importance lie also in the world of Quantum Computing.

In this chapter we mention some important things about the Quantum Complexity theory that will be useful for this thesis and in general for public knowledge.

In this point is crucial to remember some classical and general complexity classes. PSPACE is the most important in this paragraph. It is the class of decision problems which can be solved by a Turing Machine using polynomial space and an amount of time. P is the class of problems that can be solved on a classical computer/Turing Machine in an efficiently amount of time(polynomially). NP is the set of problems whose solution is verified efficiently on classical computer/TM. But its solution isn't efficiently found.

Also, the most famous unsolved problem in computer science is to determine if the the last two classes are different (or the other way):

$$P \neq NP$$

But most widely believed is that these classes are different, and NP contains problems that are not in P and that are "hard" to compute.

The huge interest on Quantum Computing is to solve some of these problems.

The main class of Quantum Complexity is the BQP class. It is the class of decision problems that can be solved by quantum computers with a bounded probability of error using efficient-polynomial size of quantum circuit. For instance, Shor's Algorithm mentioned before lies to this category.

2.3.2 **Complex Numbers**

In the quantum computing and general in the quantum mechanics the numbers that are being used are the Complex Numbers. Complex numbers are constit of two parts the real part and the imaginative one. We can think complex as an extension of the real numbers that we are all familiar, thus this extension provides us with the ability to perform some actions that with real numbers we cannot. Complex numbers are superset of the Real numbers since for any real number is complex number without the imaginary part. What complex numbers in quantum mechanics is to provide a mathematical tool where the calculations done are easily trackable because of the algebraic closure of Complete Field.

2.3.3 **Linear Algebra**

For one to be familiarised with Quantum Computing one should first learn some other topics, at least some basic concepts of them. And if someone looks deep down and on the low-level of Quantum Computing it will see that Quantum Computing is nothing less than pure Linear Algebra. All the gates, that are used to manipulate the states of the are operators or matrices that transform the states like in linear algebra. The states though are characterized by vectors and a quantum program is a sequence of operators that transform the states to other states. The vector space that these vectors belong to, is the complex vector subspace.

Some important terms of linear algebra that will be critical in this thesis are firstly the **basis.** The basis is set of vectors that no vector in the set can be written as a scalar product with respect with some another, and also every vector in the space can be written as linear combination of these vectors where the coefficients are complex numbers. Although there can be a lot of basis in a space, we will mostly care about the basis that consists of vectors that represents the possible solutions of a problem. We will refer to this as the computational basis.

Dirac Notation. In quantum computing and in quantum mechanics in general, the vector that describes a state of a system is represented using the Dirac Notation. Suppose the vector $\vec{v}$ the using the Dirac notation,

$$\vec{v} = |v\rangle = \begin{bmatrix} x \\ y \\ \cdot \\ \cdot \\ \cdot \\ z \end{bmatrix}, \qquad \vec{v}^{\dagger} = \langle v| = [x* \quad y* \;.\;.\;.\; z*]$$, (x^* is the conjugate of *x)*

$|v\rangle$ it is called the "ket", $\langle v|$ the "bra".

Inner product. Inner product is a binary operation on two vectors in complex(or real) space that returns a complex(or real) scalar. Our interest is on the complex definition of inner product.

$$\langle \varphi \mid \psi \rangle = = [\ \varphi 1 \quad \varphi 2 \ldots \quad \varphi n] \begin{bmatrix} \psi 1 \\ \psi 2 \\ \cdots \\ \psi n \end{bmatrix} = \varphi 1\psi 1 + \varphi 2\psi 2 + \cdots + \varphi n\psi n = c$$

Where *c* is a complex number

Essentially, the value is way of comparing two vectors in the space. Inner product can be regarded as the probability amplitude of going to the $|\varphi\rangle$ state if we are first in $|\psi\rangle$ state.

Note that this amplitude can take negative value, that's why in order to find the probability as we know it, we take this amplitude in the square.

Eigenvector and Eigenvalue. When a matrix/operator A acting on a vector and the only effect on the vector it is the change of its length (by a scalar) but not its direction the vector is called eigenvector of that matrix. The scalar that alters the length (making larger or smaller) it is called eigenvalue. $AV = \lambda \cdot V$

Hermitian. An n-by-n matrix A is called Hermitian if A† = A. In other words, A[j, k] = A[k, j]* , where '*' means conjugate. Hermitian has only real values since if the conjugate of a complex number equals to original value if and only if this number is real.

Unitary. An n-by-n matrix U is called Unitary if $UU^{\dagger} = U^{\dagger}U = I_n$. In other words, the conjugate transpose of a unitary matrix "undo" the original matrix and vice versa.

Tensor (Kronecker's) Product. Suppose two vectors (can be more than 2) $\vec{v}, \vec{w}$ $\vec{v} = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$ and $\vec{w} = \begin{bmatrix} w_1 \\ w_2 \end{bmatrix}$. Then the tensor product of the two vectors creates the following:

$$\vec{v} \otimes \vec{w} = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} \otimes \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} = \begin{bmatrix} v_1 \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} \\ v_2 \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} \end{bmatrix} = \begin{bmatrix} v_1 w_1 \\ v_1 w_2 \\ v_2 w_1 \\ v_2 w_2 \end{bmatrix}$$

Tensor product expands the dimension of the two vectors by combining their corresponding dimensions and values. The new dimensions are the product of the row dimensions of each vector involved in the product × product of the column dimensions of each vector involved in the product. More specifically for two vectors with a × b and c × d dimensions respectively, the tensor product will yield a new vector with ac × bd dimensions.

2.3.4 Basic Quantum Mechanics

The hardware of quantum computing is nothing like a classical computer. It is in fact a quantum mechanical system and obeys its postulates that they described it:

1. *At each time, the state of a system is described by a ket($|\psi\rangle$) which belongs to the Hilbert Space.*

Hilbert Space is nothing more than a complex vector space that is equipped with an additional information, the inner product between two states ($\langle \psi \mid \phi \rangle$) which is a single complex number and what it says is the probability amplitude of going to the state $|\psi\rangle$ if we are in the state $|\phi\rangle$

2. *. The time evolution of the state of a quantum system is described by $|\psi(t+1)\rangle = U|\psi(t)\rangle$, where U is an unitary operator.*

That is the state at next time is determined by an unitary Operator. As also due to the definition of unitary operators ($UU^\dagger = U^\dagger U = I$), one can determined the state on a previous time step by acting the conjugate transpose of the U operator on a state, $|\psi(t-1)\rangle = U^\dagger|\psi(t)\rangle$ and hence the reversibility of quantum computing

3. *Every observable of a physical system is described by an operator that acts on the states that describe the system.*

For example, an observable of the system could be the energy which is an operator/matrix which has diagonal elements the values of energy of each state. Observables have eigenstates that form a basis.

4. *The only possible result of the measurement of an observable O is one of the eigenvalues of the corresponding operator O.*

And since we are measuring only real values (the eigenvalues are real) the operator O is actually Hermitian, and the eigenstates of a Hermitian that is an Observable have some good properties, they are orthogonal, $\langle \psi \mid \phi \rangle = 0$, and thus they form a basis (every state is written as linear combination of them). Hermitian operators and their basis states will play an important role throughout this thesis.

5. *When a measurement of an observable O is made on a generic state $|\psi\rangle$, the probability of obtaining an eigenvalue corresponding to an eigenstate is given by the square of the inner product of $|\psi\rangle$ with the eigenstate $|i\rangle$, $\langle i \mid \psi \rangle^2$*

We will see how we can capture that probability later when we performed the experiments. In general, this means that the higher the probability of a vector state to an eigenstate, the "closer" is that vector to the eigenstate vector geometrically.

2.3.5 **Hamiltonian**

Hamiltonians are probably the most important thing in this thesis as they are the heart of the algorithms that will be used and generally of the physics laws that governed those algorithms. A perfect view on the Hamiltonians can be seen in the lectures of one of the greatest physicists we have witnessed, Richard Feynman on [18]

$$i\hbar \frac{dC_i(t)}{dt} = \sum_j H_{ij}(t) C_j(t)$$

Here $C_i(t)$ is the amplitude of an arbitrary state to to be in the state $|i\rangle$ at time t. More specifically is $C_i(t) = \langle i \mid \psi \rangle$ at time t, where $|\psi\rangle$ is an arbitrary state.

The idea of Hamiltonian matrix is if we have a basis of states (which in our case these states will be possible solutions(bitstrings) then the H is a matrix that describe the physical laws of a system. If one knows the Hamiltonian(s) that describe a system, then one knows what its about to happen to that system as the system varies over time or even if it's independent of time which simplify the process. And the hardest part is to find these Hamiltonians cause more of the times you don't know how a particle's properties are, especially if that it's a new one.

The $H_{ij}(t)$ hence, telling us how these laws change over the time.

If these matrix does not depend on time, then the things are even better. The equation describing the changes in the amplitudes is

$$i\hbar \frac{dC_i}{dt} = \sum_j H_{ij} C_j$$

Which is more clear and more easily solvable.

Another important question one may ask (and this thesis we will ask) is what is the expectation value of a the Hamiltonian if that is expressed by a Hermitian and diagonal matrix. More specifically, for example what is the energy of the system (what we expect it to be if is in an arbitrary state $|\psi\rangle$).

$$\langle \psi | H | \psi \rangle$$

But for our case (for the computer scientists' perspective) the things will a lot easier than those of theoretical and experimental physicists. Since we love abstraction, our particle will be only one (the qubit which will see later) and will be the analogue to the bit we already know. So, a system that consists of multiple qubits will be a system whose states are bitstrings which encode a solution to a problem as we are familiar with. So, the Hamiltonians in our case will describe the physical laws of the solutions to a problem. For example, the Hamiltonian could describe a function that evaluate a bitstring.

2.4 Qubit

The classical computation has as an atom of computation the bit. In Quantum Computing, the atom of computation can be an actual atom, and that's why there are various qubit implementations using different particles. For example, Xanadu has quantum hardware consisting of photons. But for this thesis and Computer Science in general this can be abstracted as it does not really affect us. Whatever particle is used we will be named it qubit.

What differentiates the qubit from the bit is that it is equipped with the aforementioned properties of quantum mechanics. More specifically qubit have two states that form a basis

$$|0\rangle = \begin{bmatrix}1\\0\end{bmatrix} \text{ and } |1\rangle = \begin{bmatrix}0\\1\end{bmatrix}$$

Qubit can be in any of those states and also to be in the two states simultaneously with amplitudes telling the chance on be a specific state. Any state is witten as a linear combination of these two states.

These states are described using Dirac notation as shown above. So, an arbitrary state can be represented as follows:

$|\psi\rangle = c_0|0\rangle + c_1|1\rangle$ where $c_0,\ c_1$ are complex numbers. Also, it can be represented as a standard vector. $|\psi\rangle = \begin{bmatrix}c_0\\c_1\end{bmatrix}$

When the system consists of multi-qubits (which this is the case) then the state of the system is determined by the tensor product of the between the state of each qubit

So, if the system consists of two qubits, for instance, with each qubit be in a state $|\psi_1\rangle$ and $|\psi_2\rangle$ respectively then the total state of the system is $|\psi_1\rangle \otimes |\psi_1\rangle$. If $|\psi_1\rangle = c_0|0\rangle + c_1|1\rangle$ and $|\psi_2\rangle = g_0|0\rangle + g_1|1\rangle$ then the the state of the system will be:

$$|\psi\rangle = c_0g_0|00\rangle + c_0g_1|01\rangle + c_1g_0|10\rangle + c_1g_1|11\rangle = \begin{bmatrix} c_0g_0 \\ c_0g_1 \\ c_1g_0 \\ c_1g_1 \end{bmatrix}$$

The dimension of a system consisted of two qubits has now $4 = 2^2$ possible states than can be. And in general, for a system with n qubits, the possible states are 2^n. When number of qubits grows linearly, the possible states grow exponentially. This one of the reasons why Quantum Computing show this enormous computational potential.

2.4.1 **Bloch Sphere**

the Bloch sphere gives a geometrical representation and useful visualization of the qubit. From the figure below we can observe three axes. X, Y, and Z (hence the X, Y, Z gate are rotation gates around the corresponding axe). On the surface of the sphere are the superposition states (with the equal superposition to lie on the equator). On the "north pole" is the $|0\rangle$ state while on the "south" pole the $|1\rangle$. As we can see there is also the φ parameter that determines the phase. Phase plays extremely important role in the quantum algorithms and s especially when it is the tool for the interference.

But using the Bloch sphere we can write any state as following:

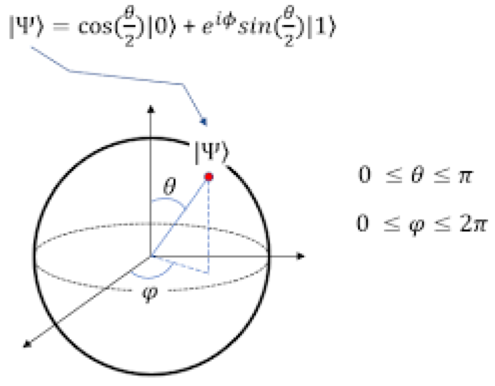

$|\Psi\rangle = \cos(\frac{\theta}{2})|0\rangle + e^{i\phi}sin(\frac{\theta}{2})|1\rangle$
$|\Psi\rangle$
θ
φ
$0 \leq \theta \leq \pi$
$0 \leq \varphi \leq 2\pi$

Figure 3.1 – Bloch Sphere[27]

The Figure 3.1 shows a visualization of a qubit. The dot on the surface of the sphere represents an arbitrary state. The θ angle tells in which of computational states $|0\rangle$, $|1\rangle$ the state of the qubit is closer to be projected into, while the φ angle showcases the phase. Two states that have the different phase can have the same probabilities in yielding to $|0\rangle$, $|1\rangle$ but there are different states.

Chapter 3

Quantum Computing Primers

3.1 **Key Concepts**

Quantum Computing is considered more powerful than classical computation because it has concepts that the classical counterpart has not. In addition to the qubit concept (as spoken previously our atom of computation) there are few key concepts:

- Superposition
- Entanglement
- Quantum Interference

3.1.1 **Superposition**

Superposition is the first property of Quantum Computing which sparks its theoretical supremacy. Superposition is the ability of a quantum system to exist in multiple possible states simultaneously. In Quantum Computing with n qubits, this means the system can be a superposition of all possible 2^n states at once. In classical Computing we only process a system of bits sequentially and one at a time, while superposition gives us the ability to process all states together. This property is useful in many Quantum Algorithms and it is used almost in all algorithms. Its mathematical expression can be shown below (for equal superposition):

$$|\psi_1\rangle = \frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} |x\rangle$$

Where $|x\rangle$ defines a possible state of the system

3.1.2 **Entanglement**

Entanglement is perhaps the most fascinating (and mysterious) concept of Quantum Mechanics. It says that two or more particles are somehow correlated between them and they cannot exist independently of each other. If one acts in a way, that means that the others will act in correlated way, even are far away. For example, if we entangle two qubits and measure the one qubit in the $|0\rangle$ state, that means its entangled qubit will also be in the $|0\rangle$ state without having to measure it or do anything (as well as with the $|1\rangle$ state). And even if you move the entangled particles far away from each other this extraordinary property will still exist. “Spooky action at a distance” as Albert Einstein describe it.

3.1.3 **Quantum Gates**

The reader should be familiar with the classical gates of computation such as the AND, OR, XOR, NAND etc. The purpose of quantum gates is to manipulate the state of a system (e.g. single or multiple qubits) and try to guide the system into measuring something that we want. What quantum gates essentially do, is rotate the qubit to a state (that is in Qiskit the U3(θ,φ,λ) gate exists which can represent any gate if the three parameters are set accordingly). All quantum gates basically rotate the qubit around the three axes shown in the Bloch sphere to reach a distinct state. Quantum gates are unitary matrices Why Unitary gates? Because they preserved norms and so probability amplitudes. That the probability amplitudes will sum up to 1. Each time a gate operation is performed on a qubit, we can calculate the new state of the qubit by matrix and vector multiplication to find the new state. Some very important gates that will be very useful on the rest of the thesis are the Rx(θ), Ry(θ), Rz(θ), which depend on one parameter angle theta, and basically are rotations around the corresponding axis around θ radians.

3.1.4 **X gate**

The NOT gate of Quantum Computing. The X gate acts on a single qubit and switches from state $|0\rangle$ to $|1\rangle$ and vice versa.

The X matrix: $\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$

Applying an X gate to a qubit that is in state $|0\rangle$ will then have the following outcome:

$$X|0\rangle = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} = |1\rangle$$

In general, the gate changes the values of the amplitudes for each of the computational states. For example, if for an arbitrary state $|\psi\rangle = c_0|0\rangle + c_1|1\rangle$ the effect will be

$$X|\psi\rangle = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} c_0 \\ c_1 \end{bmatrix} = \begin{bmatrix} c_1 \\ c_0 \end{bmatrix} = c_1|0\rangle + c_0|1\rangle$$

3.1.5 **Rx/z gate**

This gate is very similar to X gate, but it is something more. First, it is parametrized that means depends on a parameter which is an angle. This angle determines the rotation of a qubit around X-axis.

$$Rx(\theta) = \exp\left(-i\frac{\theta}{2}X\right) = \begin{bmatrix} \cos\frac{\theta}{2} & -i\sin\frac{\theta}{2} \\ -i\sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{bmatrix}$$

The same for the Rz gate which rotates the qubit over the z-axis over an angle parameter.

$$R_z(\theta) = \begin{bmatrix} e^{-i\frac{\theta}{2}} & 0 \\ 0 & e^{i\frac{\theta}{2}} \end{bmatrix}$$

These two gates will be the most frequent gates used in the circuits in this thesis, especially the Rz gate.

3.1.6 **U3 Gate**

We mentioned a 3 parameter gate U3((θ,λ,φ) that can be represent any gate if the parameters set accordingly.

$$U(\theta,\phi,\lambda) = \begin{bmatrix} \cos\left(\frac{\theta}{2}\right) & -e^{i\lambda}\sin\left(\frac{\theta}{2}\right) \\ e^{i\phi}\sin\left(\frac{\theta}{2}\right) & e^{i(\lambda+\phi)}\cos\left(\frac{\theta}{2}\right) \end{bmatrix}$$

For example, if $\theta = \pi$, φ = 0, λ = π the gate will be

$$\begin{bmatrix} \cos\left(\frac{\pi}{2}\right) & -e^{i\pi}\sin\left(\frac{\pi}{2}\right) \\ e^{i0}\sin\left(\frac{\pi}{2}\right) & e^{i\pi}\cos\left(\frac{\pi}{2}\right) \end{bmatrix} = \begin{bmatrix} 0 & -e^{i\pi} \\ e^{i0} & 0 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} (e^{i\pi} = -1, \text{Euler's Formula})$$

As we can see the above gate with these parameters acts as the X gate.

A good question is how someone find the correct configuration for the gate of its interest. My opinion is that one approach is, since the gate is parametrized, to use optimizers to adjust the right values depending on what the circuit should do.

3.1.7 **CNOT Gate**

The controlled-NOT gate is a 2 qubit gate since it needs two qubits to operate on. One qubit is the control while the other is the target qubit. Whenever the control qubit is in the $|1\rangle$ state and X operation is applied to the target qubit. If control is in $|0\rangle$ nothing happens (or we apply the identity I gate). Later we will see how this gate helps us achieve entanglement. The gate is defined as follows:

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

3.1.8 **Hadamard Gate**

The Hadamard gate might be the most important operator. It is the gate responsible of performing the most important mechanism of Quantum Computing, the superposition.

The H Matrix: $\frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$

Applying an H gate to a qubit that is in state $|0\rangle$ will then have the following outcome:

$$H|0\rangle = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 \\ 1 \end{bmatrix} = |+\rangle$$

$$H|1\rangle = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \cdot \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 \\ -1 \end{bmatrix} = |-\rangle$$

Both $|+\rangle$ and $|-\rangle$ are states which are essentially in a superposition of exactly equal probability of yielding 0 or 1, but their different states. The difference lies to the fact that the latter creates a negative phase which might be use for destructive interference when try for example to exclude some solutions.

Using this gate on a qubit q. And then the CNOT gate on two qubits with the control qubit the q, we can entangle these bits.

3.2 **Quantum Circuits**

Quantum circuits are a model in which a computation is performed with a sequence of quantum gates, which are reversible and unitary transformations on a quantum mechanical. An n-qubit register is quantum system of n qubits. When measured (in Z basis), it will yield a state of the computational states (which represent a sequence of 1s and 0s as we know a n-bit register looks like). However, measurement can be performed in any basis but usually it is the computational basis that make us interested. Before the measurement the system can be in any superposition of all possible sequences of n 1s and 0s. The basic components of a quantum circuits are the gates or matrices, barriers which everything before that must be executed and finished and then the circuit proceeds. and measurements which will collapse the system into a sequence of 1s and 0s as a classic register.

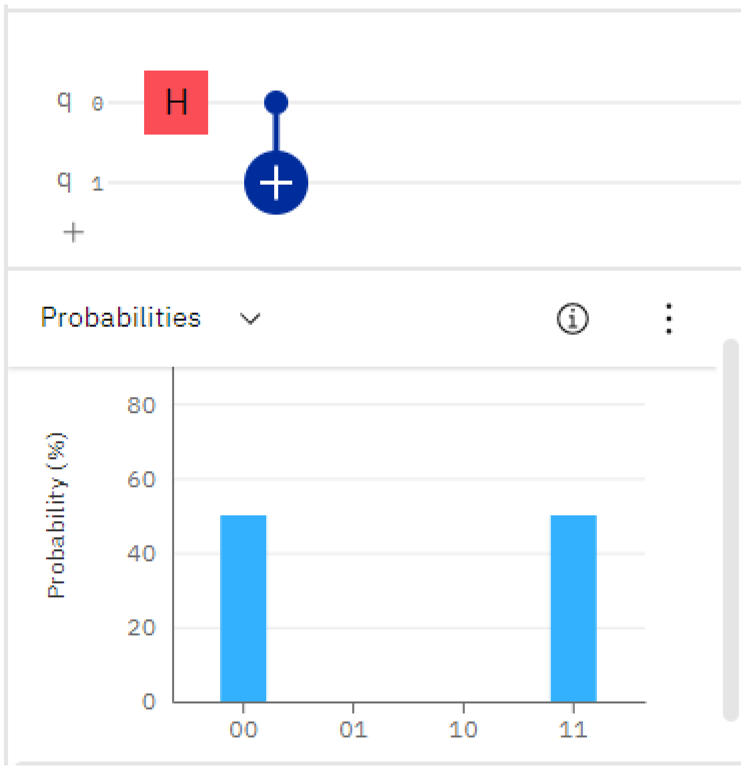


Figure 3.1- Bell state, Entangled state

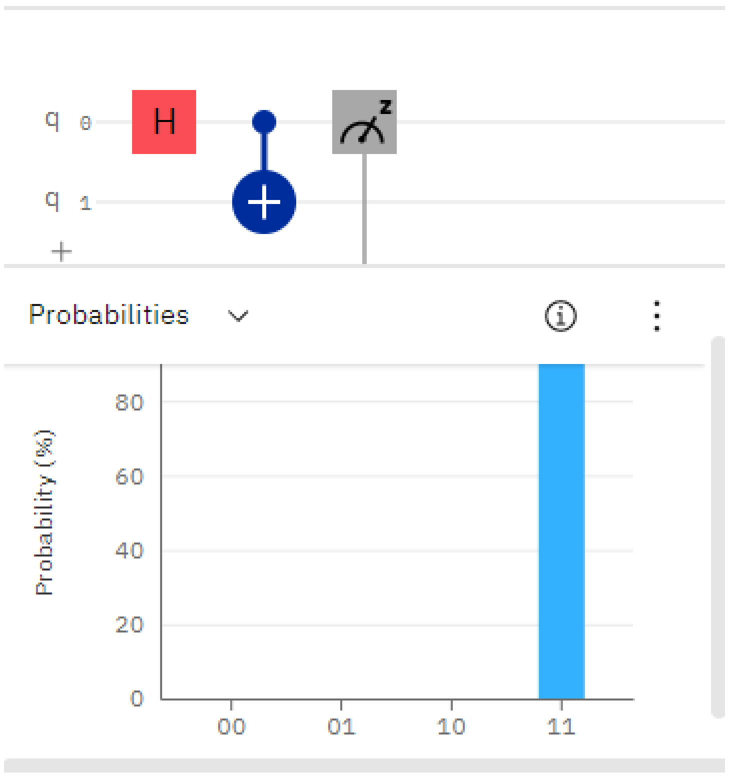


Figure 3.2 – bell state with measurement on first qubit

In figure 3.1 we can see a simple example of a circuit. This is also known as the Bell State. These two qubits are entangled, that means are somehow related as we can see with probabilities. That either must be both 0 or both 1 if we performed measurement.

In figure 3.2 we place a measurement on first qubit. With measurement the state of the qubit should collapse into one of the basis states $|0\rangle$ or $|1\rangle$. We see that the qubit collapses to $|1\rangle$ state and automatically the other qubit also collapses into the same state without measuring it. That's the power of Entanglement.

3.2.1 **Quantum Interference**

Remember that each state in qubit has a phase factor, beside the amplitudes for each of the computational states, $|0\rangle$ and $|1\rangle$. In a multi-qubit system usually the start is to create a superposition of all possible states. Then you perform such operations such that the amplitudes and the phases of some state changes in order to cancel out with each other and leave you with a state or a set of states which is either the solution or is the way of getting the solution. Quantum interference is the point where it gets you out of the superposition and leads you to the answer. It is as important as Superposition and Entanglement.

.

Chapter 4 **Travelling Salesman Problem**

4.1 **Description**

Well if you study Computer Science, sometime in the long run you might heard(or you will) about a problem named The Travelling Salesman Problem. And the reason is because it is considered one of the most famous NP-complete problems of the Computer Science and Mathematics in general. NP-Completeness states that there is no known algorithm that can solve it in polynomial time and that every other problem in NP is efficiently (requires polynomial time) reducible to the TSP problem.

The problem is defined as follows. A salesman person must travel through a known set of cities starting from one specific city and return to the city he/she started. But he/she must visit each city once (excluding the starting city). The goal of this problem is to find the route with the minimum total cost (this cost may be time, length, etc) possible. This route (passing though all cities once) is called Hamiltonian Cycle.

It may seem simple but if the graph that is formed by the cities and their connections between them are forming a fully-connected graph then they are (n-1)! possible solutions!!! Which if n tends to be large it grows more than exponential. ((n-1)! /2 if it's undirected graph).

Such it is where the quantum computing hopefully will step up a provide a more efficient solution.

4.2 **Previous Attemps(Classical Computing)**

As this is considered one of the most famous and NP complete problems in Computer Science there have been several attempts of solving it and with various ways. Here we will state some of the important ones as well as with some quantum approaches.

A) Greedy Algorithm/Nearest Neighbour

The first thought that may come to someone mind(well except from brute force is to visit at every stop in the tour the city with the minimum cost/distance from the current city in the stop. This approach is very efficient as it requires O(n**2) steps to solve but it does not produce optimal solutions as the total path taken from another neighbour (if that one was chosen instead)possibly could lead to a better solution. There is also a variation of this algorithm where in the start of the algorithm the set of edges are sorted in ascending order. And chooses the first/lowest elements to build the Hamiltonian cycle. The complexity of this is O($\log_n$ * n**2), but still is not guaranteed to find optimal solution

B) Held-Kerp Algorithm

This is a dynamic programming algorithm to the problem. Its intuition is based on the fact that every subpath of a path that is the solution and is of a minimum distance, that subpath is also of a minimum subpath. This intuition is also the optimization property of this dynamic algorithm. Consider V = {X_1, X_2, X_3, . . . , Xn}

Let S ⊂ V \{X_1} ≡ {X_2, X_3, . . . , Xn} be a subset of the graph that defines the problem with starting node to be X_1. The recursion cost of finding the solution of this problem its defined as:

$$\text{cost}(X_i , S) = \min X_j \{\text{cost}(X_j , S\setminus\{X_i\}) + D_{ij}\}$$

which states the shortest hamiltonian path from X_1 to X_i

When S becomes {X_2, X_3, . . . , Xn} what is left to find is :

$$\text{Minimum Hamiltonian cycle} = \min X_i \{\text{cost}(X_i , S) + D_{i1}\}$$

The complexity of this algorithm is of O (n^2 * 2^n) which is much better than the brute-force approach but worse than the greedy. Of course, this is guaranteed to find an optimal solution while the greedy the more probably will yield a suboptimal solution.

C) Genetic and Evolutionary Algorithms

Since TSP is a complex optimization problem evolutionary algorithm are a reasonable idea to consider.

Evolutionary Algorithms are derived from the function and evolution of natural and biological populations. Their common function is that they maintain a population of candidate solutions (initialised randomly) and each iteration produces a new more suitable population consisting of more desirable solutions. This new population is evolved (thus evolutionary) by performing some basic operations that derive from natural evolution organisms (e.g mutation, crossover etc.) and then selection to decide what to keep and what to discard. The selection is based on an objective value or fitness which in the case of TSP will be the overall path cost. Those with low fitness will be eliminated and those with high fitness will generate the new population with the above operations.

So, an approach to solve TSP is to use a Genetic Algorithm which is an evolutionary algorithm. It is based on how organisms are created and evolve through the DNA. What is needed is to find an encoding of solution to the problem and cost function to act as the fitness of these solutions. The candidate solutions are called chromosomes which are basically bitstrings encoding the solution. In each iteration, the algorithm evaluates the fitness of each chromosome and then the "best" chromosomes are selected, and crossover and mutation are performed to the selected chromosomes. Crossover is the interchange of some corresponding substring between chromosomes, while mutation is the flip of some bits of a chromosome.

The complexity of evolutionary algorithms is not quite straight-forward. This depends on the iterations will let the algorithm run in order to find a quality solution.

We will see that an encoding of an algorithm we will more analytically see later can be fed easily to a Genetic Algorithm.

4.2.1 **Previous Attempt(Quantum Computing)**

A) Grover's Search for minimum

One way to solve the problem quantum mechanical, is to search the set of the possible solution and find the state/ eigenvector with the lowest cost as presented in the paper [6].

To do that with quantum computing is to encode the distances between cities as phases. The reason for encoding the cost as phases is to acquire the ability of constructing unitary operations from the adjacency matrix to manipulate the qubits of the system. Using these different Unitary operations, the algorithm tries to build a space where states represent different solutions. Each solution corresponds to an eigenvalue which is the total cost of the traversal encoded in phases. So, for each for these eigenstates to find the phase value of the cost the algorithm uses the famous Quantum Phase Estimation to estimate the value of the phase cost-eigenvalue. Then using a variation of the Grover's Search Algorithm , the algorithm searches for the optimal value and its corresponding eigenvector.
The steps can be generalised as follows:

1. Prepare the matrix $B_{ij} = e^{i(adj_{ij})}$ where adj is the adjacency matrix with the normal weights. (phase encoding)
2. Prepare the unitary operations U_j as follows:

$$[U_j]_{kk} = \frac{1}{\sqrt{N}}[B]_{jk} \,,$$

where j, k >=0 and 1<= j, k <=N and N = number of cities

3. Prepare a new unitary U as the tensor products of the above unitary operations,

$$U = U_1 \otimes U_2 \otimes \ldots U_N$$

4. Out of the N^N diagonal elements in U , extract the (N-1!) possible eigenvectors that correspond to the possible Hamiltonian cycles .
5. Use Quantum Phase Estimation to estimate the phase of these eigenvectors.
6. Use the algorithm [15] of finding the index of the minimum element (variation of the Grover's Search) to find the eigenvector with the lowest cost.

The complexity of this algorithm is the complexity of the algorithm of finding the state with the minimum value which is Quadratic speedup because is based on the famous Grover's Search quantum algorithm that is proven to have quadratic speedup. Grover's search can be also used to search for the solution in the dynamic programming algorithms so they can be benefited by that speedup.

4.3 Applications of TSP

The problem has not so many real -world practical applications mainly because no efficient solution exists. But still there are nowdays applications and who knows with Quantum Computing new applications could be created. First of all, it is used for optimal traffic and vehicle routing. There is already, a practical quantum computing application for vehicle routing demonstrated and used by Volkswagen [34], showcasing the strong promise of Quantum Computing. Another application of TSP is on drilling in circuits and TSP can be used to minimise the time needed for drilling the desired points which are represented by nodes. Another potential application of TSP is in DNA sequencing. Each node corresponds to a DNA fragment and the cost of edges a similarity measure of these fragments.

Chapter 5 Quantum Approximation Optimization Algorithm and the Alternating Ansatz

5.1 Introduction

In this section the Quantum Approximate Optimization Algorithm is introduced and discussed. QAOA is the technique in which the algorithms implemented in this thesis are based so a complete chapter for it, is necessary. This algorithm was introduced by Edward Farhi and Jeffrey Goldstone at 2014([9]) and is one of the most known and important quantum algorithms right now, and maybe the most important among the variational and h.

Before diving in a more comprehensive overview of the algorithm it is important to see the idea behind this algorithm. We start with the Adiabatic Quantum Computation. That if we have a system that can be described with a simple Hamiltonian (let’s say H_S) with a known ground state (the eigenstate with the lowest eigenvalue-energy) and another Hamiltonian (H_C) whose ground state is wanted, then we can form another Hamiltonian in the form of

$$H(t) = (1-t)H_S + tH_C$$

Where t can be any physical quantity, but in our case will be the time. If we start from t = 0 (the system is completely described by H_S and slowly-sufficiently increase it to 1 then the system will stay to the ground state of *H(t)* throughout the process and at the end the system will be in the ground state of H_C.

So, what is needed is to find a way to encode a problem to a Hamiltonian (again to find the Hamiltonian(s) that describe the system is really the most important task)

In the case of the combinatorial optimization problems this can be done easily by diagonalizing the cost function that needed to be optimized:

$$H = \sum_{x \in \{0,1\}^n} C(x)|x\rangle\langle x|$$

Additionally, typically the cost function in combinatorial optimization can be written as a sum of subsequent Cost terms that involve only a subset of the bitstring that encodes a solution

Now that the cost function consists of a sum of subsequent binary variables (x_i) we can map the cost function as a sum of Pauli Z operators by doing the following step:

$$x_i = (1 - z_i)/2$$

where z_i is replaced by the Pauli Z operator.

Now we have the Hamiltonian, it remains to evolve the state over the time or some parameters. Remember from before that solving the Schrodinger's equation gives us the evolved state at specific time over the Hamiltonian if we start from initial state $|\psi(0)\rangle$

$$|\psi(t)\rangle = \exp(-\mathrm{i}Ht)|\psi(0)\rangle$$

The final thing we mention is the Trotterization or Suzuki-Trotter Expansion when it is used to simulate a complex Hamiltonian (e.g a time-dependent Hamiltonian) by approximating it. This Approximation can be done by interleaving the evolved operation for very small-time difference and many times starting from t=0 to the desired t. And if the total Hamiltonian describing the system is sum of other Hamiltonians/Hermitians like below:

$$H = \sum H_m$$

then the approximation of the time evolution of the system is H is determined by:

$$e^{\Sigma H_m} = \lim_{n\to\infty}\left(e^{\frac{H1}{n}}e^{\frac{H2}{n}}\dots e^{\frac{H_m}{n}}\right)^n$$

.

5.2 **Brief Overview**

Now considering all of the above, the full QAOA algorithm will be introduced here.

QAOA is belongs to the class of variational, hybrid, and approximate quantum algorithms and it is suitable for solving combinatorial optimization algorithms like the Travelling Salesman Problem. Variational because it has parameters which the optimal selection of this is wanted to minimise/maximise the cost function of the combinatorial problem. Hybrid because the structure of the algorithm includes both quantum computing parts and classical computing parts. Finally approximate, because the solution it produced is approximation of the optimal solution, thus the solution approximates the best solution, but may not be the best.

What QAOA generally do is to approximate the Quantum Adiabatic Path mentioned above using the Trotter-Suzuki evolution. It has the total time-dependent Hamiltonian of the system described by the sum of two independent Hamiltonians H_C and H_B. H_C is the Hamiltonian that describes the cost function of the problem and whose ground state is wanted. H_B is the Hamiltonian whose ground-state is known but also can be used as mixer Hamiltonian where it “mixes” and creates superposition over the states of the system. By repeating p times consecutive executions of the evolutions of these two Hamiltonian over some parameters acting on an initial state (the ground/high energy state of H_B) the algorithm approximates the quantum adiabatic evolution of the overall system and in the end it finds a state that is close to the ground state of the H_C with an approximation ratio.

Below is the equation that describes the quantum part of the algorithm.

$$\left|\psi_p(\vec{\gamma},\vec{\beta})\right\rangle = e^{-i\beta_p H_B}e^{-i\gamma_p H_C}\dots e^{-i\beta_1 H_B}e^{-i\gamma_1 H_C}|s\rangle$$

where $|s\rangle$ is the initial state and it is the ground state of the mixer Hamiltonian.

$(\overrightarrow{\gamma}\overrightarrow{\beta})$ is the parameter vector $(\gamma_1,\dots,\gamma_p,\beta_1,\dots,\beta_p)$ and which are angles in which the gates used to generate the above state are rotated over the axes of the involving qubits.

From the above equation the only thing is missing is H_B.

$$H_B = \sum_{i=1}^{n} X_i$$

The above Hamiltonian has many advantages on why it is used. First its evolution on some parameter is simple to implement.

$$\exp(-\mathrm{i}H_B\theta) = \sum_{i=1}^{n} R_x(2\theta)$$

Additionally, we know the ground state of that Hamiltonian. Since the eigenvectors of the Pauli X operator are $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle), \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$. with eigenvalues 1, -1 respectively so for n-qubits the H_B has a ground/high energy state of the equal superposition of the n-qubits.

So the initial state of the QAOA will be the below

$$|s\rangle = \frac{1}{\sqrt{2^n}} \sum_{x} |x\rangle$$

H_B has another advantage that mixes over all the states and it allows the escape of local minimum in the optimization function that will see below.

The algorithm thus creates a function which with the help of a classical optimization tries to approximate the optimal value:

$$F_p(\vec{\gamma}, \vec{\beta}) = \langle \psi_p(\vec{\gamma}, \vec{\beta}) |\mathrm{Hc}| \psi_p(\vec{\gamma}, \vec{\beta}) \rangle$$

The above function is the expectation value of H_C and is created evaluating the cost function multiple times by using bitstrings measured by the state $|\psi_p(\vec{\gamma}, \vec{\beta})\rangle$ in the computational/Z basis.

Here it is important to mention. That for the evolution of the cost Hamiltonian which is written as a sum of weighted Pauli Z matrices is enough to append the Rz gate mentioned previously for each term in the equation and its corresponding weight times the angle as a parameter. If the term has product of Pauli Z matrices acting on different qubits then we entangle these qubits and put an Rz gate to one of the corresponding qubits

Then using classical optimizer, the parameters change so that $F_p(\vec{\gamma}, \vec{\beta})$ maximizes/minimises. Then after the optimization we expect that when we do another measurement in the state it will produce a bitstring, that has cost evaluation close or equal to the max/min cost of the system, with high probability.

Furthermore, if O_p is the most optimal expected value of cost function in respect of the parameter vector $(\vec{\gamma}, \vec{\beta})$

$$O_p = \underset{\gamma,\boldsymbol{\beta}}{opt}\ F_p(\gamma, \beta)$$

Then if we have p to go to infinity, we have the expected value equal the actual optimal value of the cost function.

$$\lim_{p\to\infty} O_p = \underset{x\prime}{opt}\ C(x')$$

Where *x'* is the bitstring/solution that produces the actual optimal cost

Now the general steps of the algorithms will be introduced.

5.2.1 **Algorithm steps**

The input of this algorithm is generally the p value. The number of layers the circuit will have. Any other inputs may depend on the problem. For example, in our case another input will be a text file that have details on a graph instance. The output of the algorithm also may vary. Most of the cases will be the most probable bitstring. It can also be the probability distribution of all the bitstrings.

1. Encode the cost function of the combinatorial problem into sum of Pauli Z operators. This is done by changing the binary variables in the function (assuming that the cost function is written in binary variables) to
(1-z_i /2).
2. Produce the unitary operators $U_1(H_{C,}\,\gamma)$ = exp(-iγHc), for some parameter γ, and $U_2(H_{B,}\,\beta)$ = exp(-iβH) for some parameter β. Also prepare the initial state $|s\rangle$ to the equal superposition between all possible states by simply applying Hadamard gate to each qubit.
3. Select the value of p(the number of layers and the number of the parameter vector and prepare the state
$|\psi_p(\vec{\gamma}, \vec{\beta})\rangle = e^{-i\beta_p H_B} e^{-i\gamma_p H_C} \dots e^{-i\beta_1 H_B} e^{-i\gamma_1 H_C} |s\rangle$ by using repeated actions of U_1 and U_2
4. Evaluate the expected value of the $F_p(\vec{\gamma}, \vec{\beta}) = \langle \psi_p(\vec{\gamma}, \vec{\beta}) | \text{Hc} | \psi_p(\vec{\gamma}, \vec{\beta}) \rangle$, and find optimal $(\vec{\gamma}, \vec{\beta})$ that maximises/minimises that expected value using classical

optimizers. Repeat this step until some fixed number of trials or until a satisfied level of convergence.

5. To find the optimal solution, measure the $|\psi_p(\vec{\gamma},\vec{\beta})\rangle$ (with optimal $\vec{\gamma},\vec{\beta}$) to computational basis.

The $F_p(\vec{\gamma},\vec{\beta})$ can be computed as follows. After the state $|\psi_p(\vec{\gamma},\vec{\beta})\rangle$ is prepared, we measure in the computational basis multiple times (e.g 128 shots) to retrieve some bitstrings with their counts (how many times a specific bitstring is yielded). The we compute the expected cost as the weighted (with probability to yield count(x) / shots) sum over the evaluation of cost function over the bitstrings.

QAOA is different from Adiabatic Quantum Computation, although it takes some ideas of it. The AGC starts from the known Hamiltonian and slowly go to the wanted Hamiltonian, while QAOA its traverses in interleaved execution of the two Hamiltonians. Also, QAOA may work well even for a relatively small number of the value p while AGC needs its similar parameter which is the time T usually has to be large. In the original paper of QAOA, the authors mention that sometimes even if T is large the accuracy of the algorithm drops, while they stated QAOA doesn't drop its accuracy with large p.

5.3 Quantum Alternating Operator Ansatz

A new variation of the initial QAOA was introduced by ([10]) which concentrates mainly on the mixer Hamiltonian H_B and the only thing that truly differs from the algorithm steps mention above is the implementation of U_2 and the initial state $|s\rangle$.
The reason behind this alternating ansatz is that for many combinatorial problems the possible feasible solutions of the problem are forming a subspace in the Hilbert space thus the superposition over all 2^n states is not enough.
Although, as we will see later, one can give a better chance for feasible solutions to be sampled by using the original QAOA by penalising the infeasible solutions in the cost function so that the final measurement after the classical optimization won't yield them. We will see later what a better approach is.

While the H_B in the original QAOA is standard and simpler, the construction of it in this Alternating Ansatz varies among the problems and are more complicated in both finding and implementing it. Generally, there different classes of this mixers and they will not be presented here except of those that will be used in the algorithms in this thesis. But the paper mentioned in this section is a very good source of getting more details.

There are some criteria and constraints when this approach is selected. First, both the mixer Hamiltonian and the initial state must be sufficiently prepared and implemented. Thus, some bounds must be set when it comes to complexity or even the circuit depth when it comes to how evolution of the mixer Hamiltonian is simulated. For example, the initial state and the mixer Hamiltonian should be found within polynomial bounds otherwise this approach will not do any good. In addition to that, the depth of the circuit used to simulate the evolution of the mixer Hamiltonian should also have some polynomial bounds, since depth is a way of measuring the complexity of a quantum algorithm and the hardware right now is very limited.

There are two approaches about what the initial state should be according to this approach. First is to create an equal superposition of all the feasible solutions, but this is quite difficult to achieve especially for a complex problem. Second approach is to find a feasible solution which is easy to implement by just using X gates.

From the initial state the mixer Hamiltonian can be determined as by every feasible solution, other feasible solution can be determined by flipping a number of bits of it.

For both ways about the initial state there must be a way of finding a feasible solution in efficient way. And according to the paper, for every NP problem there is a an efficient validate solution that checks if a bitstring corresponds to a valid solution to a problem.

<u>1st Approach</u>

Suppose starting in the state of equal superposition in all states in 2^n and an extra ancilla qubit. Suppose V is an operator that acts as an oracle and evaluates as one in that ancilla bit if a state is a solution to a problem and leave it 0 if else.

$$\sum_{x=0}^{2^n} |x\rangle \otimes |0\rangle \rightarrow \mathrm{V} \rightarrow \sum_{i \neq Feasible} |x\rangle \otimes |0\rangle + \sum_{i \in \mathrm{Feasible}} |x\rangle \otimes |1\rangle$$

Then using $I \otimes |1\rangle\langle 1|$ we get the $\sum_{i \in \mathrm{Feasible}} |x\rangle$ initial state.

But the validate oracle implemented in the quantum computer is non-trivial.

2nd Approach

We found a validate solution classically (supposing this can be done in efficient way), then from the $|0\rangle^{\otimes n}$ we flip the appropriate qubits using the X gate to form the initial state.

Chapter 6

Algorithm 1 for Travelling Salesman Problem:

6.1 **Intro-Description**

.

This chapter and the next will focus on two different ways of encoding and solving the Travelling Salesman Problem using QAOA that introduced by different. In this chapter the original QAOA will be used and in the next chapter the Alternating Ansatz will be used. The chapters will present them as long as their implementations using IBM's Qiskit using both simulators and running them on real quantum machine in IBM Q experience.

When it comes to the original QAOA, the beautiful thing is that what you need to find is the mapping of cost function to a sum of pauli Z operators with the way mentiond before. Furthermore, there ways which make this part easier and save time. For example, there is IBM library Docplex that if you write the cost function using binary variables then it can convert it to Pauli Z representation.

6.2 **Cost function and the map to the Pauli Z sum**

We need a way of describing the problem in terms of binary variables and bitstrings. The intuition in the approach found in the [8], is that we encode the the solution in n^2 bits. It's better to visualise it as an array bitstring,

$$x = \begin{bmatrix} 0\,1\,0\,0 \\ 0\,0\,1\,0 \\ 1\,0\,0\,0 \\ 0\,0\,0\,1 \end{bmatrix}$$

where each row corresponds at the timestep/order of the traversal and each column corresponds to the cities in the traversal and in general all the cities of the problem. So

x[t, i] = 1 , means that the ith city is visited at the t time step or order of the traversal else ***x[t, i] = 0***. The above example corresponds to following tour:

city 2 -> city 3 -> city 1 -> city 4

Suppose we have a distance matrix n × n adj, where it gives us the distance/cost of the adjacent nodes with the entry, e.g. adj[i, j] = the distance of the edge in our graph problem that connects city i and city j (if such edge don't exist then we need to set matrix entry a value bigger than the ,maximum distance the existing edges)

The cost function can be then evaluated for each bitstring as

$$C(x) = \sum_{i,j} adj_{i,j} \sum_t x_{t,i} x_{t+1,j}$$

Is it clear that this approach has 2^{n^2} possible states/bitstrings which is much bigger than the (n-1)! Hamiltonian cycles and feasible solutions and thus we know that since the standard QAOA will be in superposition between all states then a lot of states will not form a Hamiltonian cycle and a true solution to the problem. So, this approach adds penalty to the cost function for a feasible solution violation as follows:

1) Each city must be visited once in the whole route, so the columns include only one 1 entry. This constraint can be encoded and added to the previous cost function

$$\sum_i \left(1 - \sum_t x_{t,i}\right)^2$$

2) Each timestep one and only one city must be visited. The adding penalty to the cost function is given by

$$\sum_t \left(1 - \sum_i x_{t,i}\right)^2$$

So now if we add the following terms to the C(x) we will get the full cost function with the penalizing constraints so the bitsring that will be yield, is good and validate Hamiltonian cycle.

The final cost function of the problem is:

$$C(x) = \sum_{i,j} adj_{i,j} \sum_t x_{t,i}x_{t+1,j} + P\ (\sum_i \left(1 - \sum_t x_{t,i}\right)^2 + \sum_t \left(1 - \sum_i x_{t,i}\right)^2\ )$$

Where P is the penalty value, normally a very huge number.

Now the cost function must me mapped to a sum of pauli Z matrices so it can be implemented in quantum computing. Using the $x = (1- z)/2$ transformation and getting the values of *adj* and setting the penalty value P then we get a new equation with a sum of weighted Pauli Z matrices. Modelling this to the Docplex library can yield the Pauli Z representation with one line.

6.3 **Circuit**

Now since we have the Cost function expanded in a sum of weighted Pauli Matrices it is time to implement the circuit that will describe the QAOA algorithm.

Like mentioned before, the beauty on this algorithm (standard QAOA) is the fact that then only thing that changes in the steps and the only different thing we must calculate is the circuit that simulates the evolution of the cost Hamiltonian. To do that the cost Hamiltonian must be transformed into sum of weighed Pauli Z operators and then using the Rz(weight*θ) (where weight is the weight in front of Pauli Z op and θ the parameter angle) just like it presented before to construct the circuit. Then at the end we append a Rx(2θ) gate to each qubit. This is a layer of the total circuit. Depending on the value of p we have that number of layers where the only difference is the different angle parameters.

. Then this total circuit(with p layers) will be added to the Hadamard gate sequence which creates a superposition, each H gate to each qubit

Below is a picture of the circuit of the particular problem (one layer):

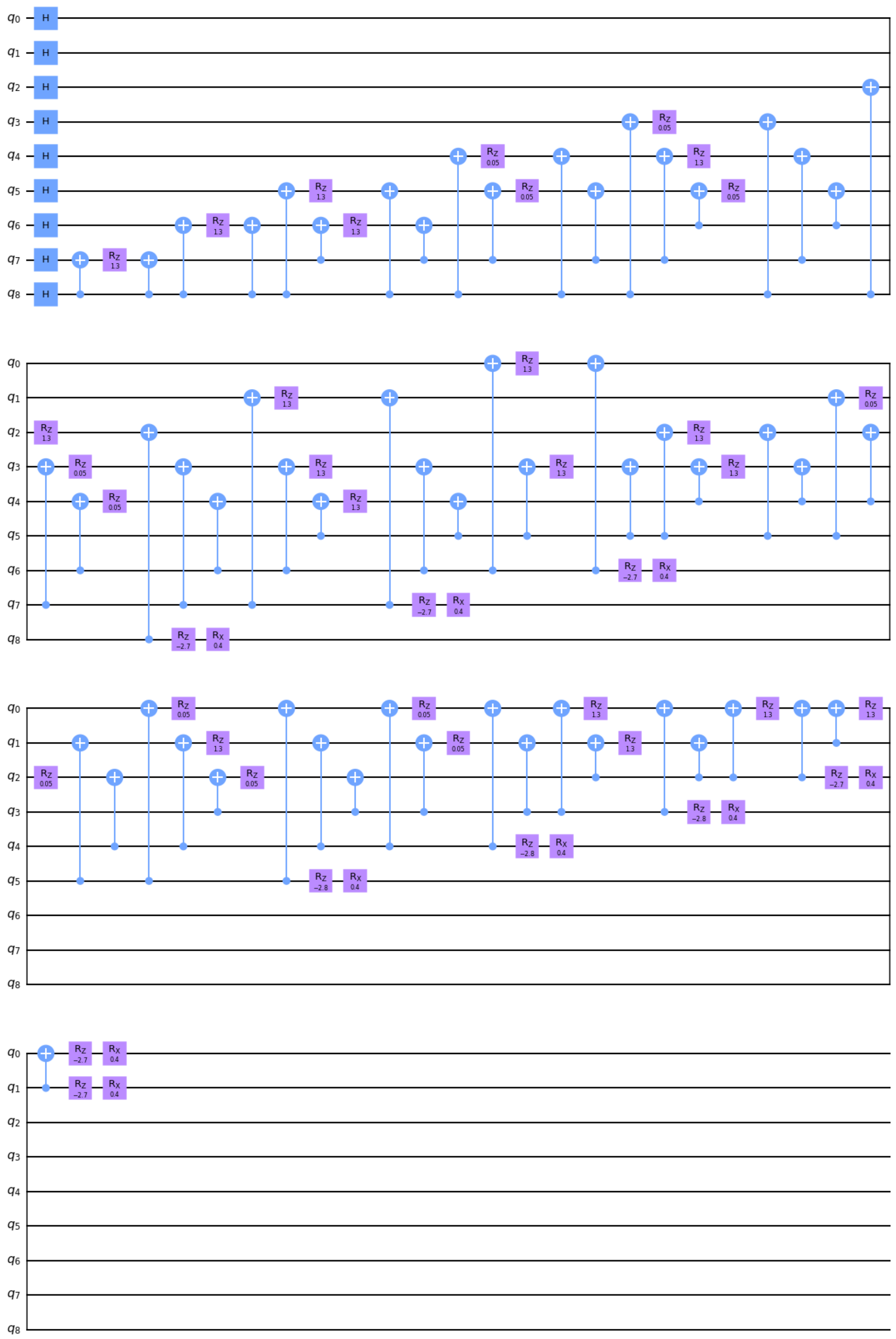


Figure 6.1- one layer of the algorithm circuit

The circuit, as shown above, consist mostly of entangled Rz gate connection with different weight in the parameter part of the gate. These different connections are connections between the calculation of the distance and the penalty for the constraints. That's why the in the some Rz gates we see bigger values in the parameters as the constraints give bigger weights.

Of course, in the end of all layers and the circuit must be measurements on all qubits in the Z basis (computational basis) to yield a bitstring.

In general, the depth of the total circuit is depended on p times the number of constraints/clauses in the cost function, which most of the times is polynomial. If the cost function cannot express in polynomial group of constraints and terms, then we have to build the entire diagonal cost Hamiltonian matrix for each bitstring which has an exponential cost in complexity.

6.4 **Experiments**

Now you begin with the experiments of the algorithm where we will run different graph instances on both simulator and real quantum machines.

Due to limited number of qubits available in the systems of IBM, we run this algorithm for instance with a small number of cities since the encoding of the solutions are quite sparse. As stated before, the encoding of the solution requires n*n qubits to deploy QAOA.

1st Experiment (n = 3)

This particular algorithm was very problematic in many experiments that was run. problematic in the sense that the results wasn't the expected as other problems. For instance, the results of the next algorithm and the Max-cut problem (the hello world of QAOA) that were also tested were. The only conclusion that can be derived is that the optimizers used (COBYLA, SPSA, ADAM, P_BFGS) cannot find the optimal gamma and beta so that the cost expectation function is low to the optimal. This probably due the different and more complex landscape created where optimizers struggle to find the optimal parameters as they stuck to locals' minimums.

The only thing left to do is to try to find the optimal parameters through a grid search as it proposed in the original paper of the QAOA.

We performed a grid search approach of finding the optimal parameters using a grid of [0,2π] [0,π] with step size 0.1

Consider the graph G = {0,1,2,3} with edges E= {(0,1,1.0), (0,2,1.0), (1,2,1.0)} as can be seen below :

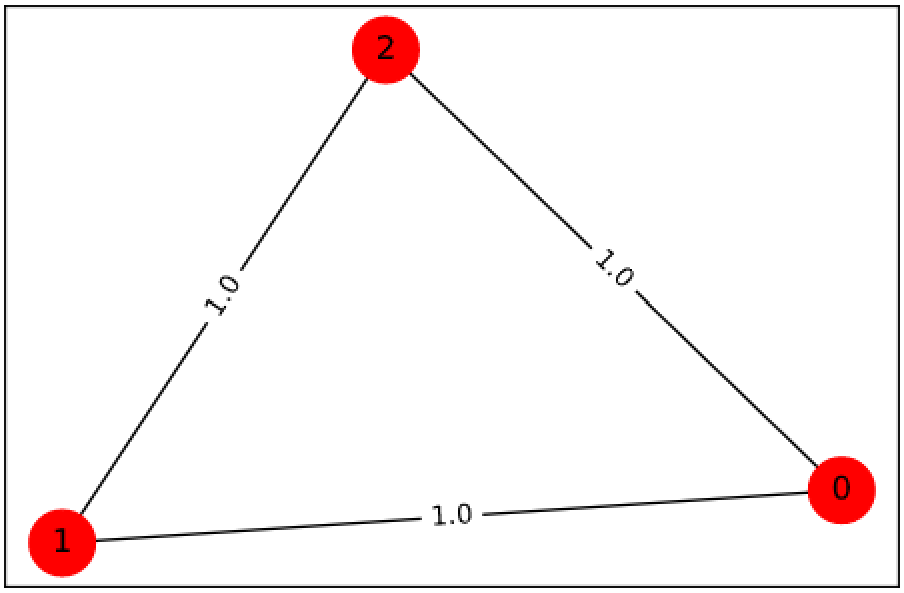


Figure 6.2 – Simple Graph instance

From this example and the way, the algorithm encodes the solution the correct possible answers are 6. More specifically ret = {100010001, 100001010, 010100001, 010001100, 001100010, 001010100}. In general, there is only one Hamiltonian Cycle. This encoding taking into consideration the order of the cities and that's why in this example the 6 solution is the Hamiltonian Cycle just written in different order.

We expect that the grid search will give these solutions high probability and since they have the same cost, we expect it to yield the with similar probability.

The grid search to find the optimal parameters took time around the 10.60 minutes or 636.56 secs. The results are below:

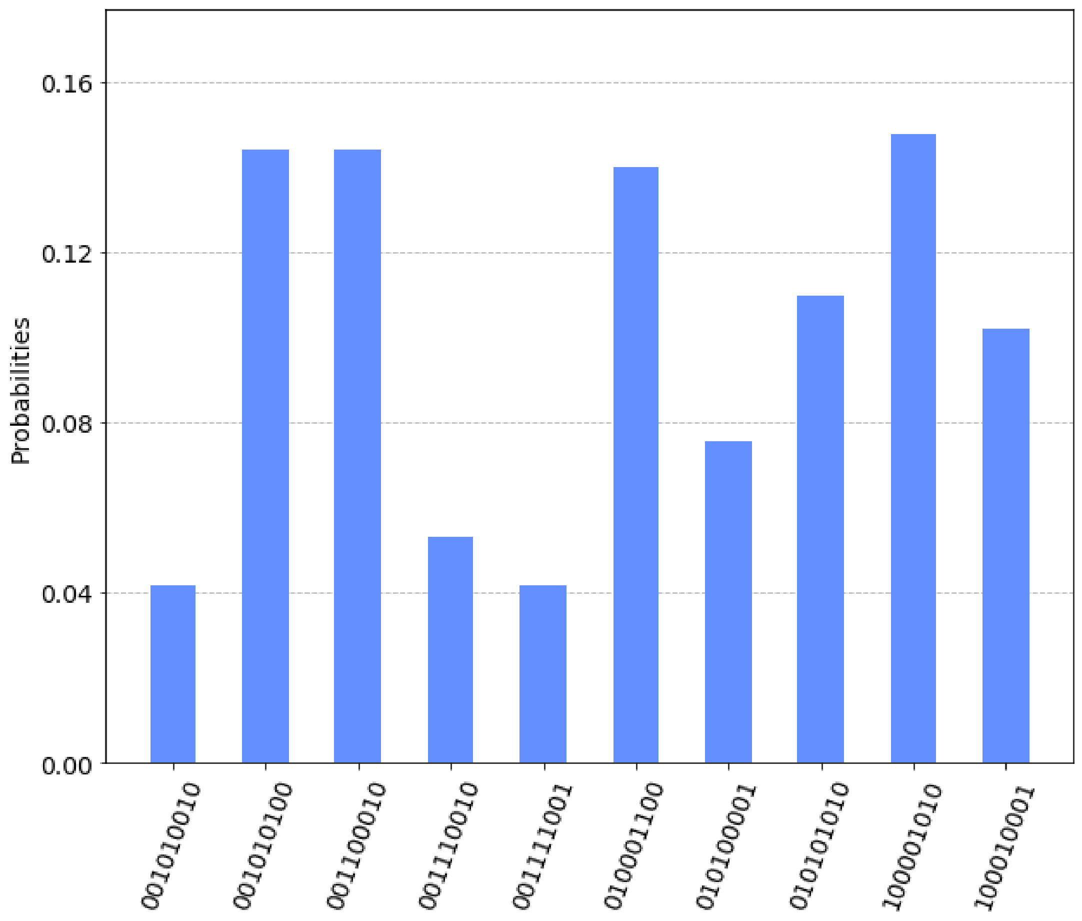


Figure 6.3 - 10 highest probabilities for bitstrings

The above figure shows the top 10 high probable bitstrings. The distribution was far larger but for visualization reasons, we selected the top 10.

As we can see the 4 top highest in probability to be yielded are all correct solutions. Furthermore, the other two solutions are also yielded just with a lower probability (see 7th and last bitstring starting from left).

The other bitstring are solutions that are almost correct, but they are violating one or two constraints.

So, the simulator with grid search approach with even p = 1 works well, while using built-in classical optimizers fails. Grid search seems to be more accurate but at the same time more costly.

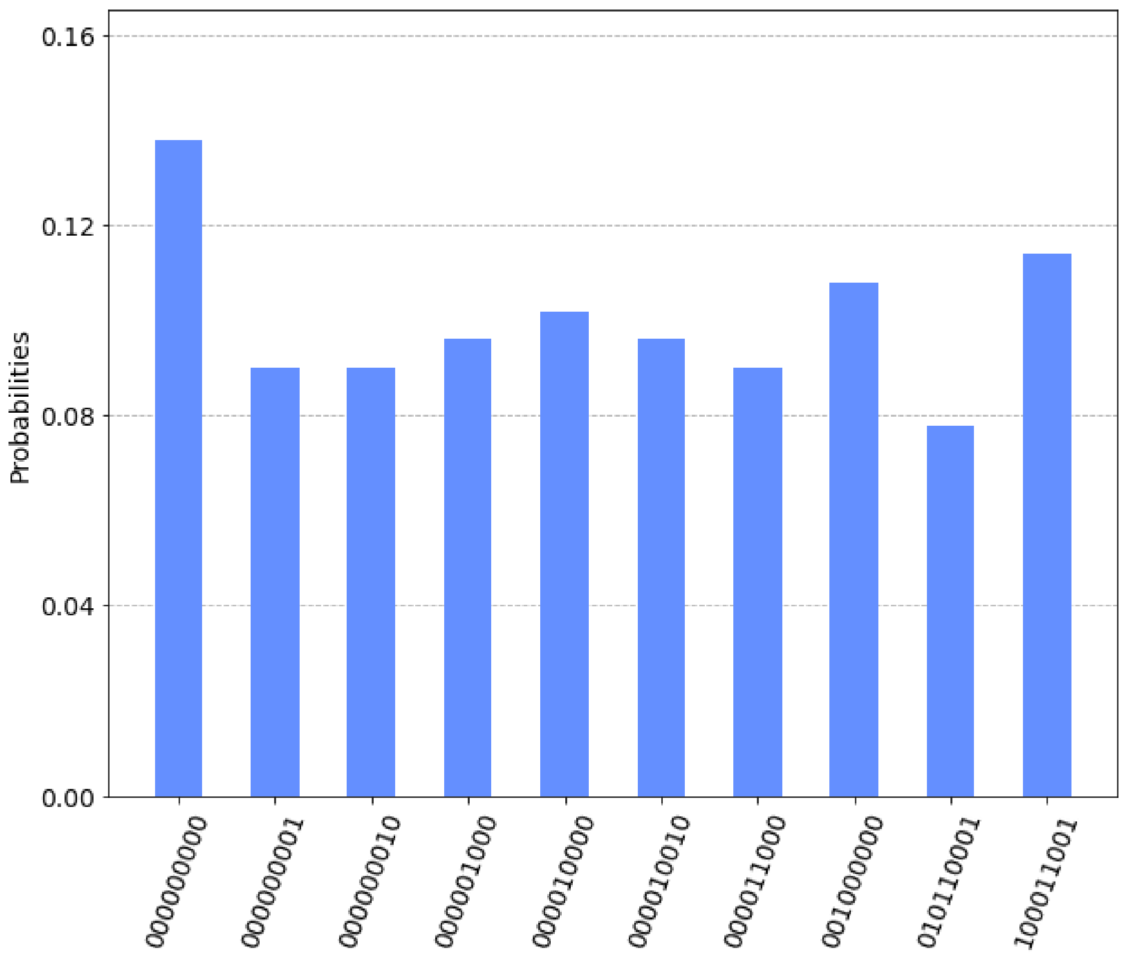


Figure 6. – Top 10 high probable solutions on IBMQ-Melbourne

The above figure showcases the results when we ran the optimized circuit to the IBMQ-Melbourne real quantum machine. We couldn't perform the entire algorithm on the machine due to the high latency in the queue where jobs (circuits to be run) are. QAOA builds and run circuits multiple times in order to find the optimal parameters. So the optimizing step of the algorithm is run on a simulator and the optimal circuit is run on the real quantum machine. This means that the time is more or less the same as above.

This machine was the only machine available on IBMQ that we could run this example since it has 15 qubits. Furthermore, we couldn't run a bigger graph since the next number of qubits required is 16 if n = 4. This machine, however, is quite faulty and compare it to other machines is the machine with the highest error rate.

And checking the results above it is clear that the above statement is true. We can see that there is no correct solution above the top probable bitstrings. There only two solutions that are close to the correct solutions, but they violate two constraints. The last two violate the repeated city and two the cities at a timestep constraint. The other results only one or no city is visited which is false.

Chapter 7 Algorithm 2 for Travelling Salesman Problem:

7.1 Intro-Description

In this chapter we describe and present an algorithm stated in this paper [22]. The algorithm uses the Quantum Alternating Optimization Ansatz variation to solve the Travelling Salesman Problem. This algorithm requires a lot of pre-processing before proceeding using the QAOA steps mentioned in the chapter 6.

First, the encoding of the solution is different and more efficient than the algorithm in chapter 7. Instead of n^2 this algorithm uses n(n-1) / 2 bits to encode the solution of the problem. Again, we considered a graph where edges are symmetric so we considered A -> B and B-> A as one edge.

Consider the following graph G = {V, E} where V={v_1, v_2, …, v_n} and E = {a_1, a_2, …, a_g} then a solution have this form

$$x = e_1 e_2 \dots e_{n(n-1)/2}$$

where e_k denotes the edge than includes the nodes i-j, 1<= i <= n, i+1<= j <= n (taking these as a nested for loop). $e_k = 1$ if the corresponding edge is included in the solution, else $e_k = 0$,

In addition, there is another vector that uses n(n-1) /2 length is used for the weights/cost of each edge that belongs to E (if such edge does not exist then the corresponding weight is assigned a big value).

$$\vec{w} = \{w_{e1}, w_{e2}, \dots, w_{en}\}$$

Since this algorithm uses the Alternating Ansatz which traverses in a sub-space of all possible states of the quantum system and which consists of subset of the feasible solutions of the corresponding problem, it is clear that the mixer Hamiltonian H_B and the initial state $|s\rangle$ will be different that the constant parts of the standard QAOA. Of course, the cost Hamiltonian will be different since there is different solution encoding but also it will be much simpler since the constraints of the feasibility of a solution is described by the mixer Hamiltonian and they don’t have to be penalized in the cost Hamiltonian.

7.2 Cost Hamiltonian-H_C

As stated, before H_C is easier to implement and simply is given by the below equation.

$$H_C = \sum_{x=0}^{2^{n(n-1)/2}-1} \vec{x} \cdot \vec{w} |x\rangle\langle x|$$

The cost function for a bitstring is given by

$$C(x) = \sum_{i=0}^{n(n-1)/2-1} x[i]\,w[i]$$

Then the mapping to the weighted sum of Pauli Z matrices will give the

$$C(x) = \left(\frac{1}{2}\right) \sum_{i=0}^{\frac{n(n-1)}{2}-1} -z[i]\,w[i]$$

One can also use the Docplex to yield this representation. This can be easily implemented using the way introduced in the QAOA algorithm as a layer of Rz gates in each qubit with a weight in the parameter (which will be a gamma value of the gamma vector) times the minus half of the corresponding cost to edge the qubit represents.

This cost function sums the cost edges that the bitstring indicates that they exist or not by the corresponding bit. But to work the bitstring must be a validate solution.

7.3 Validate Oracle

Previously, we said that for every NP problem there is an efficient method-oracle that checks for the validity of a solution. In this section the corresponding oracle for the Travelling Salesman Problem will be presented.

First, the constraint function for Hamiltonian cycles can be viewed as follows

$$f(x) = \sum_i \overrightarrow{c_i} * x_i$$

Where $\overrightarrow{c_i}$ is the coefficient vector of the i^th index of the bitstring which corresponds to an edge. This coefficient vector has a length of n (the number of vertices) and each position corresponds to a vertex. This vector has all its elements to 0 except in the positions where are the vertices that form the edge of the x_i and the value is 1.

Consider this, the constraint function for Hamiltonian Cycles and the fact that each city must be visited once in the traversal then easily one check for the validity of Hamiltonian cycle using the below equality:

$$\sum_i \overrightarrow{c_i} * x_i = \begin{pmatrix} 2 \\ \vdots \\ 2 \end{pmatrix}$$

The above is the sum of n(n-1)/ terms of n-length vectors which produce a vector of n-length final vector. This vector's terms must be 2 because a vertex to be visited only once must appear in two edges, or else the tour would stop in a node and could not visit other nodes (except the initial node which starts and ends with it which requires also two appearances). This equality, although is a necessary condition but not sufficient condition, because it doesn't consider closed routes (independent from each other) that occur from 3 nodes in the graph.

For instance, consider the below example,

V = {0,1,2,3,4,5},
x = {110001000000111} and
$\vec{c_0}$ = [1,1,0,0,0,0], $\vec{c_1}$ = [1,0,1,0,0,0] etc.

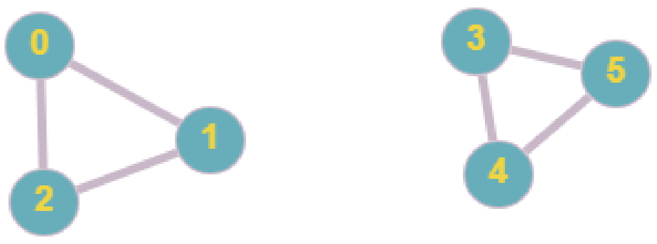


Figure 8.1 – A graph example with disconnected cycles

The above equality will hold correct in this example, but this is clearly not a Hamiltonian cycle since in any given starting vertex we cannot visiting all nodes exactly once and come back to the start (we visit once only nodes that are in the closed cycle the starting vertex belongs).

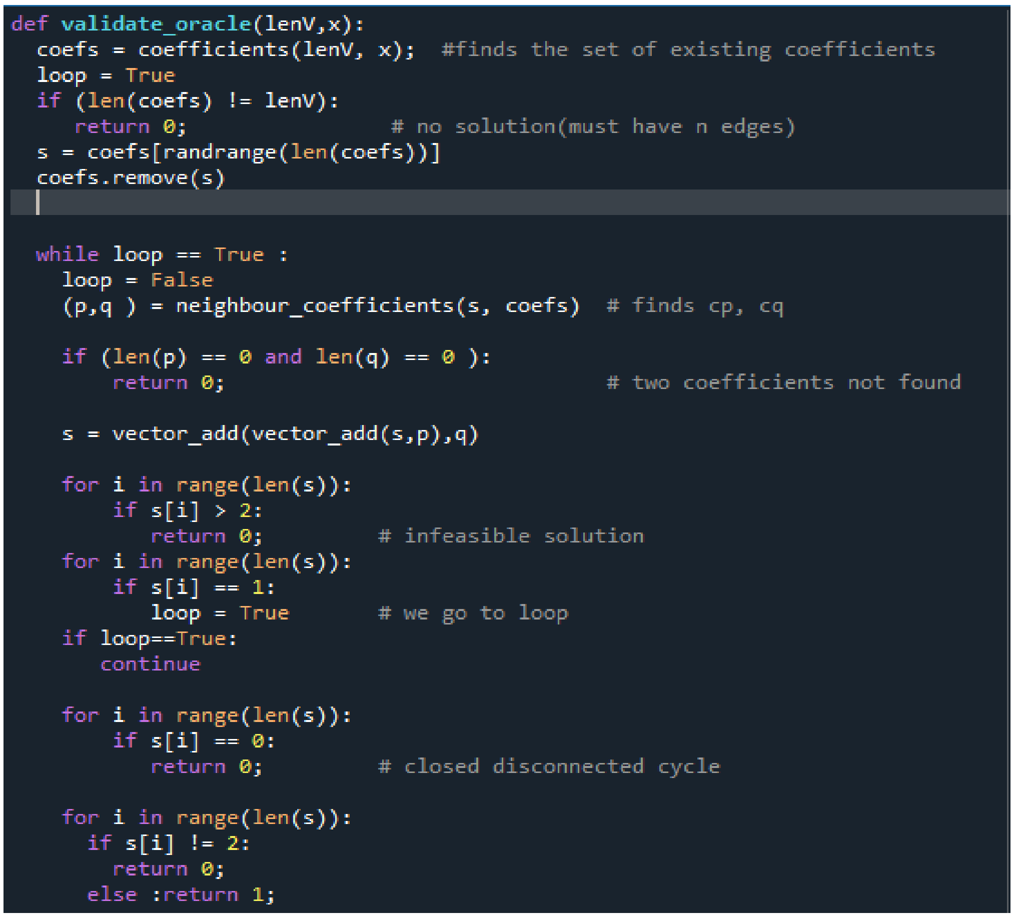
Now the full validate oracle for TSP that uses this equality and does some extra checks can be shown.

Validate Oracle for TSP (input: a bitstring x, output: 1 if x is Hamiltonian cycle, else 0)

1. Start with the set I of all coefficients $\vec{c_i}$ where $x_i = 1$ (if |I| != |V| **return** 0)
2. Arbitrarily choose a coefficient from I, $\vec{c_k}$ and initialize it in the vector $\overrightarrow{\text{Sol}}$
3. Enter a Loop. Based on the positions of 1 in $\overrightarrow{\text{Sol}}$ choose from I two coefficients, $\vec{c_p}$ and $\vec{c_q}$ whose a position of 1 is same as in $\overrightarrow{\text{Sol}}$

4. Compute $\overrightarrow{Sol} = \overrightarrow{Sol} + \vec{c_p} + \vec{c_q}$ and $I = I - \vec{c_p} - \vec{c_q}$
5. Make the following checks:
   a. If $\overrightarrow{Sol}$ has elements with value greater than 2 then **return** 0
   b. Else If $\overrightarrow{Sol}$ has elements with value equal to 1 then **Go to step** 3.
   c. Else If $\overrightarrow{Sol}$ has elements with value equal to 0 then **return** 0 (disconnected closed path exists)
   d. Else If $\overrightarrow{Sol}$ has elements with value equal to 2 then **return** 1 (feasible solution)

The implementation in Python below:



```python
def validate_oracle(lenV,x):
  coefs = coefficients(lenV, x);  #finds the set of existing coefficients
  loop = True
  if (len(coefs) != lenV):
     return 0;                # no solution(must have n edges)
  s = coefs[randrange(len(coefs))]
  coefs.remove(s)
  

  while loop == True :
    loop = False
    (p,q ) = neighbour_coefficients(s, coefs)  # finds cp, cq

    if (len(p) == 0 and len(q) == 0 ):
        return 0;                              # two coefficients not found

    s = vector_add(vector_add(s,p),q)

    for i in range(len(s)):
        if s[i] > 2:
           return 0;         # infeasible solution
    for i in range(len(s)):
        if s[i] == 1:
           loop = True       # we go to loop
    if loop==True:
       continue

    for i in range(len(s)):
        if s[i] == 0:
           return 0;         # closed disconnected cycle

    for i in range(len(s)):
      if s[i] != 2:
        return 0;
      else :return 1;
```

Figure 8.2 – Validate Oracle in Python

It is clear that the algorithm is efficient since the steps are linear to the length of the set I which is n(n-1)/2, so $O(n^2)$.

7.4 Mixer Hamiltonian (H_B)

Since the there is an oracle that can determine the feasibility of a solution (that meets the constraints of Hamiltonian Cycle), now the mixer Hamiltonian which traverses between feasible states can be build.

7.4.1 Initial state

In the QAOA chapter, when talking about the Alternating Ansatz, there were two approaches on how you define the initial state. The first is to use the quantum version of the validate oracle in the superposition of all states to get the superposition of all the feasible solutions. But this approach is non-trivial and the quantum oracle of TSP is not that easy to implement, since there is no clear pattern between feasible solutions in contrast with the Deutsch-Joza algorithm use of oracle. The second approach (the one that we will use) is to find a feasible solution and then use it to the mixer Hamiltonian to traverse to neighbouring solutions.

In general, from one feasible solution we can determine one other feasible solution. Because for a bitstring *x* given a vertex set V = {A, B, C, D, E, F} we are given a set of edges E = {e_1, e_2, e_3, … , e_k} depending the 1s in *x*. If we change the order of two vertices in the V and leave *x* intact then we will get two different edges and leave other 2, and still have a feasible different solution. And this can be done also by flipping bits in the initial example so that the Hamming distance between the bitstrings is 4.

7.4.2 H_B Intuition

Considered the above, the mixer Hamiltonian H_B can be treated as graph whose nodes are consist of possible solutions/bitstrings. More specifically the nodes are the feasible solution(s) that exist in the superposition of the initial state. If initial state is one feasible solution then the nodes are itself and the other bitstrings generated using the hamming distance of 4. Edges exist between 2 nodes *x, x'* if *x'* is generated from flipping 4 bits of *x*.

The evolution of H_B (exp(-i*H_{B*} t)) is a continuous random walk on H_B where the walker can reach all the connected nodes that belong to the subgraph that the initial node(s) are. Thus, is a walk over feasible solutions excluding the infeasible.

7.4.3 H_B Construction

The algorithm for how the mixer Hamiltonian is built using the above intuition is the following:

<u>Construct H</u>

```
For each $x$ in the superposition of feasible solutions (or a feasible solution)
 For each $x'$ in the set { $x'$ | Hamming Distance($x, x'$) = 4} // $x'$ = flip_4bits($x$)
    If  validate_oracle($x'$) == True/1 then
        $H_B[x, x'] = H_B[x', x] = 1$  // $|x\rangle\langle x'| + |x'\rangle\langle x|$
End Fors.
```

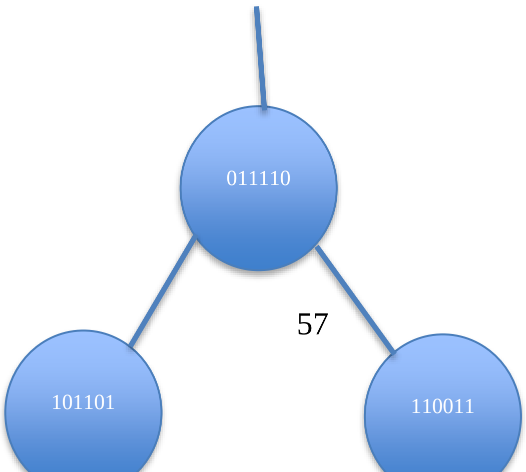
The above algorithm generates the mixer Hamiltonian that acts as an adjacency matrix of the graph consist of possible solutions.

Each row of that Hamiltonian is computed in an efficient way as the length of the set of the Hamming distance bitstrings is bounded by the number of combinations C_n^4 which is $O(n^4)$ -> [$C_n^4 = \frac{n!}{4!(n-4)!} = \frac{n(n-1)(n-2)(n-3)(n-4)!}{24(n-4)!} = O(n^4)$]

Also, since the validate function is also efficient the whole H_B is also efficiently generated.

It is important to notice that going with the approach of finding only one feasible solution it much more efficient for constructing since for that specific problem the number of feasible solutions are O(n!). But with this approach there is possibility to leave behind some feasible solutions. A trade-off between accuracy and efficiency.

For example, in this TSP encoding, for a graph instance of length(V) = 4, if we apply the second approach and find a feasible solution and generate the H_B we will get the following traversal between the solutions (the initial feasible solution is 011110).It is clear that the evolution of the Hamiltonian will be a quantum walk over the graph below



7.4.4 Evolution of mixer Hamiltonian H_B

Here the evolution of the mixer Hamiltonian is not that simple as it is in the standard QAOA scheme where we apply an Rx(θ) gate in each qubit. In the case of the mixer Hamiltonian as mentioned above in the Alternating Ansatz approach the things get more complicated. In general, there not all Hamiltonian are efficiently simulated. This paper here [2] hold more detailed information on how to decompose the Hamiltonian Matrix into a polynomial to n (number of qubits) depth.

In general, if the Hamiltonian is row-sparse, row-efficiently computable (which as stated before is) then it can be written into a sum of H_m Hamiltonians which they in turn can be simulated using the Trotter Evolution we mentioned above. With this way the whole Hamiltonian is simulated.

The idea is that if the Hamiltonian can be treated as an adjacency matrix(which in our case is as we mentioned above) then the expansion of the Hamiltonian into a sum of other Hamiltonians H_m can be thought as the edge colouring in the graph the adjacency matrix corresponds to. So, a Hamiltonian H_m corresponds to the sub-graph formed by edges of the color m. It is clear that in the case where the initial state is only a feasible solution and all edges in the graph have as a node the specific solution (like a star graph) the coloring should be different in every edge. So, the algorithm of coloring the edges does not have to run, and we just set each H_m with the m^{th} 1 element in the mixer Hamiltonian.

After we have the mixer Hamiltonian expanded into these Hamiltonians $H_{m,}$ Qiskit can make them written as weighted sum of Pauli matrices and then using the PauliTrotterEvolution method it uses Trotter-Suzuki evolution to simulate the evolution of the mixer Hamiltonian.

7.5 Circuit

Because the circuit created for this algorithm is quite huge to show it at once like the previous algorithm, we show the distinct and important part of the circuit since most of the circuit follows the same pattern due to the Trotter-Suzuki Expansion of the mixer Hamiltonian.

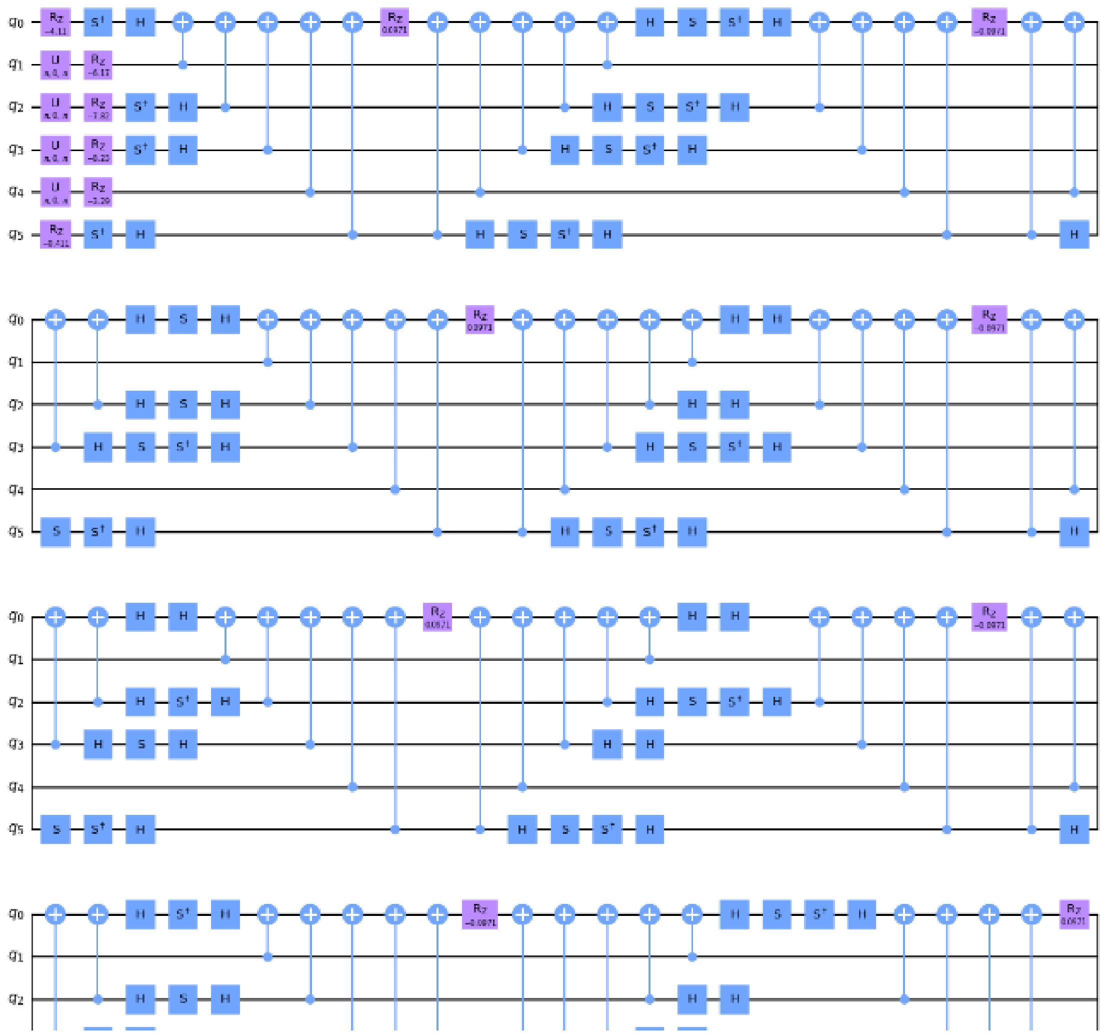


Figure 8.3 – A snip from the lengthy circuit of the algorithm

We can see in the start in some qubits the gate U(π,0,π). This gate is basically the X gate(as we mentioned it before). So The start of the circuit essentially creates the initial state 011110 which is the first feasible solution found. Then all qubits are followed by an Rz gate with parameter the gamma times the minus half of the corresponding weight on the edge that each qubit corresponds. After that what is left is the evolution of the mixer Hamiltonian. This was by done with the help of Qiskit's functions which transform the mixer Hamiltonian matrix and produce its evolution using the Trotter-Suzuki expansion.

7.6 Experiments

Now that evolution of the two Hamiltonians (cost and mixer) and the initial state we have everything we need follow the steps as stated in the QAOA chapter and begin to run the experiments.

<u>Experiment 1</u>

Suppose the graph G = {0,1,2,3} with edges E= {(0,1,10),(0,2,15),(0,3,19),(1,2,20),(3,2,1),(1,3,8), (2,3,20) } as can be seen below

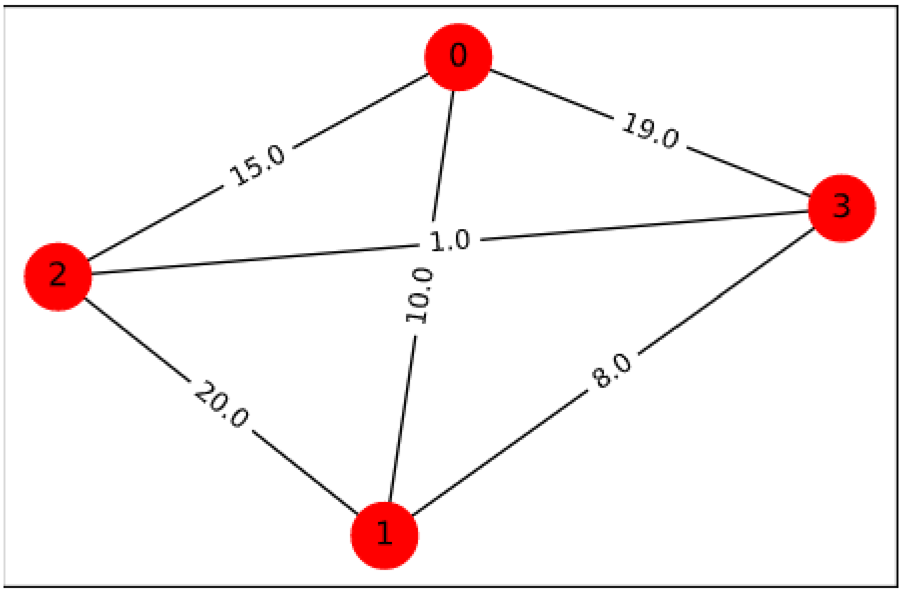


Figure 8.4 - Graph instance with 4 nodes

This is a fully connected undirected graph, so it has (n-1)! / 2 = 3 Hamiltonian Cycles which are the only feasible solutions. Using the second approach, that finds a feasible solution and from that it generates other feasible solutions, we get all of the three feasible solutions (the graph of the solutions is the same as presented above)

We expect that the algorithm (at least in the simulator) we first traverse only in the subspace of the feasible solutions {011110(52.0), 101101(50.0), 110011(34.0)} and then that it will find the correct solution with the lowest cost-eigenvalue which in that case is x = 110011.

Unfortunately, when try to run the algorithm in the real quantum machines in the IBM Q an error has occurred that length of the circuit is too large to be run. Mainly it is because of the Qiskit's method PauliTrotterEvolution which uses the Trotter-Suzuki expansion to simulate the evolution of the mixer Hamiltonian.

So, this experiment and the next one will only be tested on a simulator to test if the algorithm worked as we expect.

In general, in every experiment in this thesis there will be several parameters that must be set. First of all is the value of p in the QAOA algorithm which will increase the depth of the circuit and also the accuracy of the algorithm theoretically (the accuracy will be the probability of the correct solution after multiple shots of the optimized circuit. Other parameters are the optimizer that will be used and its according parameters. For each optimizer there are different hyperparameters available. But all of the have the maximum numbers of iterations the optimizer is run. For reasons of comparison, we will run the experiments with a fixed parameter set, except when we want to test the effects of the different parameter set. As an optimizer we will work with COBYLA as we find it the best in both accuracy and efficiency.

Below are the results in the form of probabilities. After the optimizer found the optimal parameters gamma and beta we run again the circuit with that parameters and we performed a measurement in the computational basis to yield a state/bitstring 2048 times. And found the probabilities of each state yield as the number of occurrences / 2048. We perform the experiment first with few maximum optimizing steps (20) and with more maximum optimizing steps (1000) to see the differences.

Below are the results:

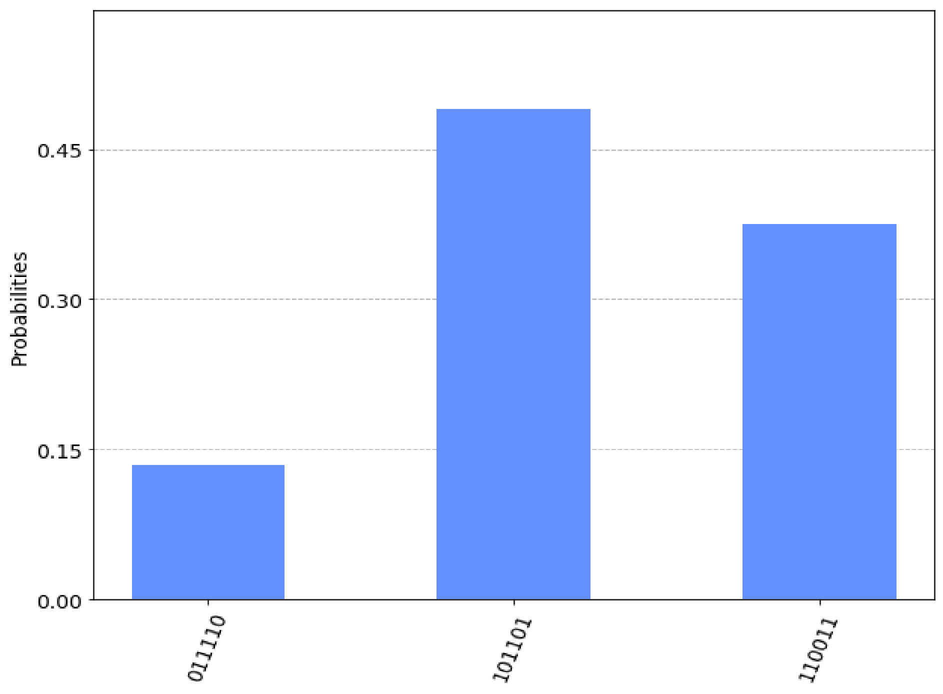


Figure 8.4 – Results with 20 optmizing steps, runtime = 3.91 secs

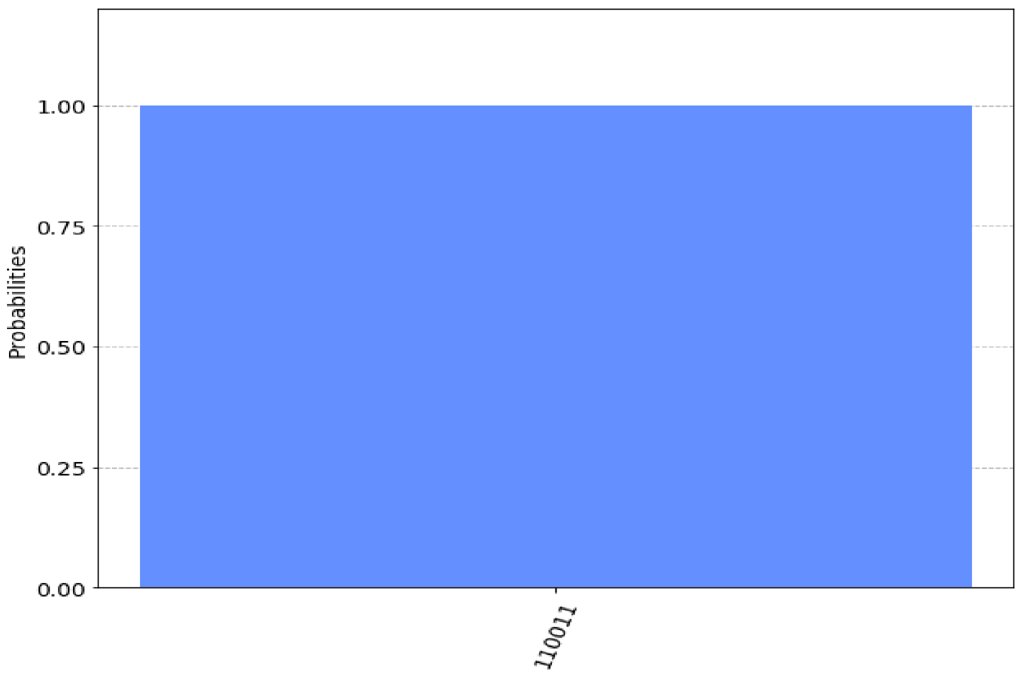


Figure 8.5 Experiment 1-Result with 1000 optimizing steps, 16.96 secs

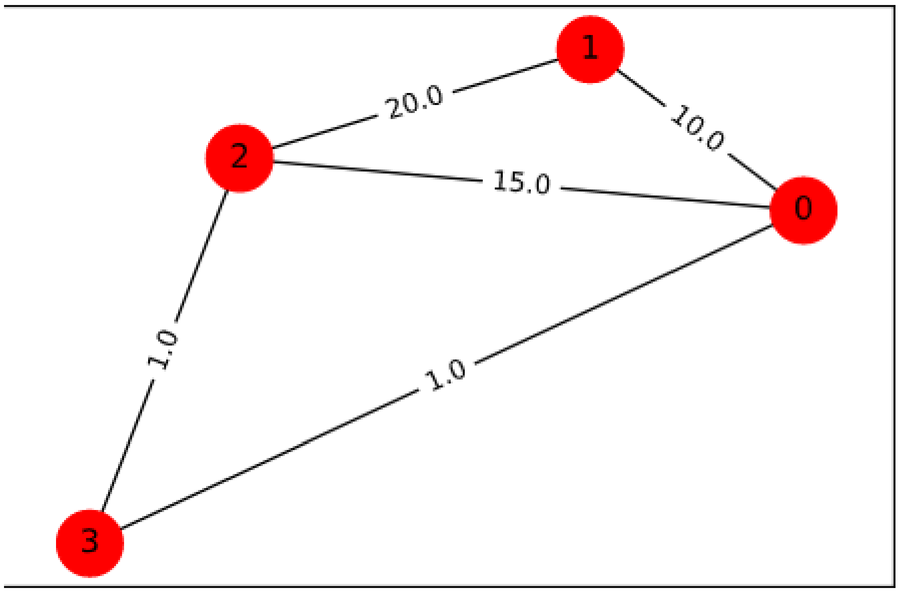
As we can see when the algorithm runs for only few classical optimizing steps, the highest probability bitstring is not the optimal solution but the sub-optimal, but still the probability of yielding the optimal solution (110011) is high.

Then, when the algorithm is run with more optimizing steps the simulator yields the optimal solution with certainty. For this there is time cost since it requires 16.96 seconds to run in contrast with the 3.91 secs for the 20 steps. But if someone wants precision then it must pay that cost.

These results showcase the importance of the classical optimizing process in the QAOA algorithm since it has the role of finding the optimal parameters in the circuit. And in general, is quite difficult to know the plane of the function in order for the optimizer to “know where to search”. Of course, when using grid search this problem can be somehow be overcomed. That’s why I believe that the optimizer struggles in the previous algorithm and the expensive grid search needed. In this algorithm, the expectation function is maybe simpler and smoother as the algorithm traverses in a small sub-space of the 2^n, and the optimizer is able to find the optimal parameters so there is no need of trying the grid search.

Experiment 2nd

Suppose the graph G = {0,1,2,3} with edges E= {(0,1,10.0), (0,2,15.0), (0,3,1.0), (1,2,20.0), (2,3,1.0) } as can be seen below :



In the second experiment the graph will be quite different as can be seen below. The major difference is that the graph is not fully connected, and one edge is missing (the edge between node 1 and 3). This has as subsequence the fact that the weight vector will have an element with a value that is larger than the maximum on the existing weights.

This will produce a higher cost function for the non-possible solution. That means the only solution to this instance is only one and it is the bitstring 101101.

We expect that with high probability the algorithm will yield the bitstring 101101.

In this experiment run the algorithm for bigger p because with low p the results were not the best. So, we increase the p to to yield better accuracy just like in theory.

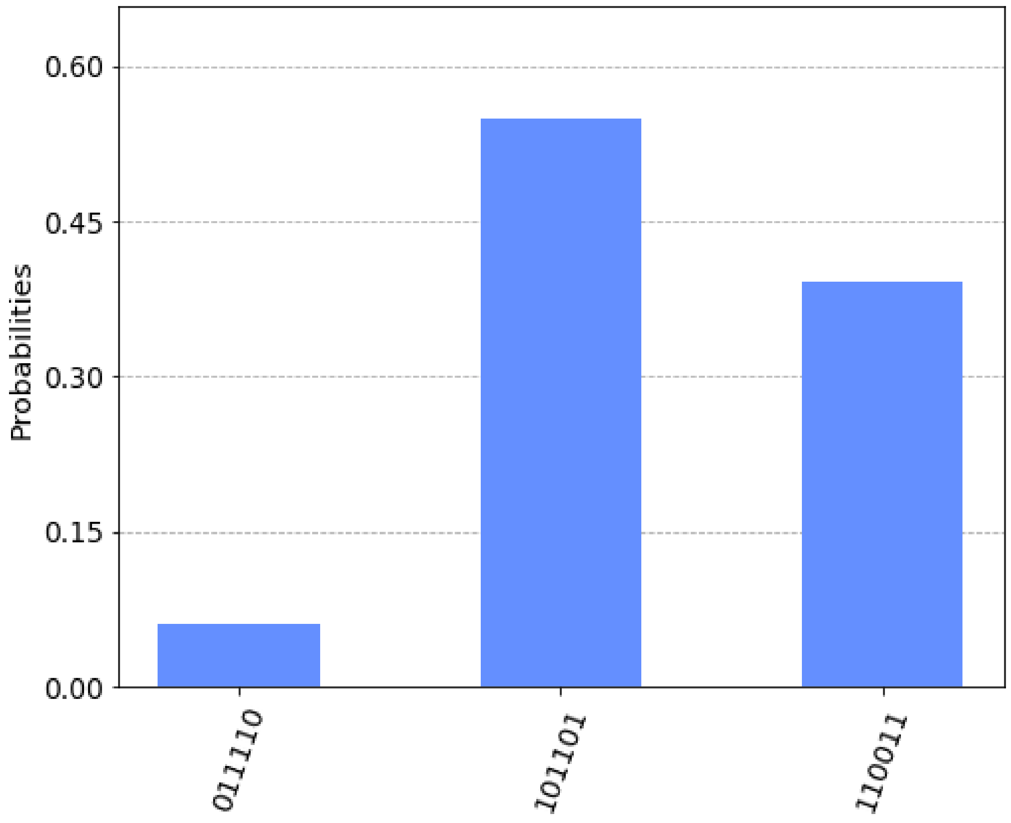


Figure 8.6 – p=2, weight penalty = 40 , runtime = 4.30 secs

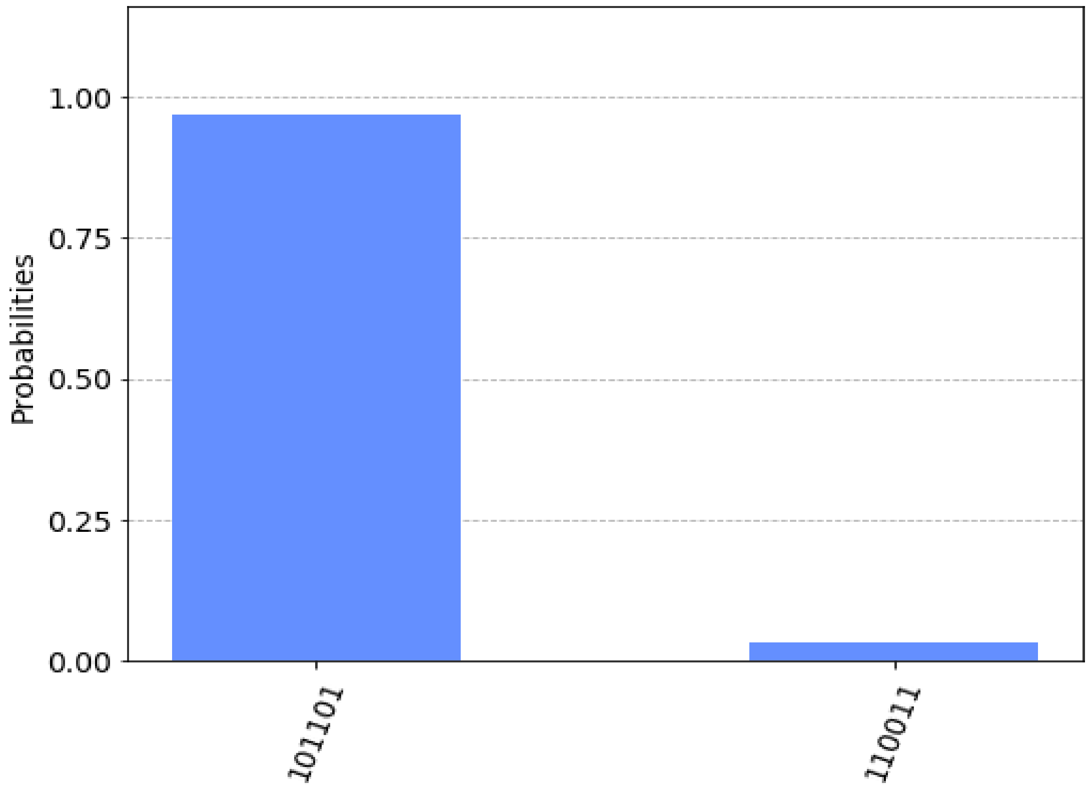


Figure 8.7 – p=20, weight penalty = 40, runtime = 287 secs(4.78 mins)

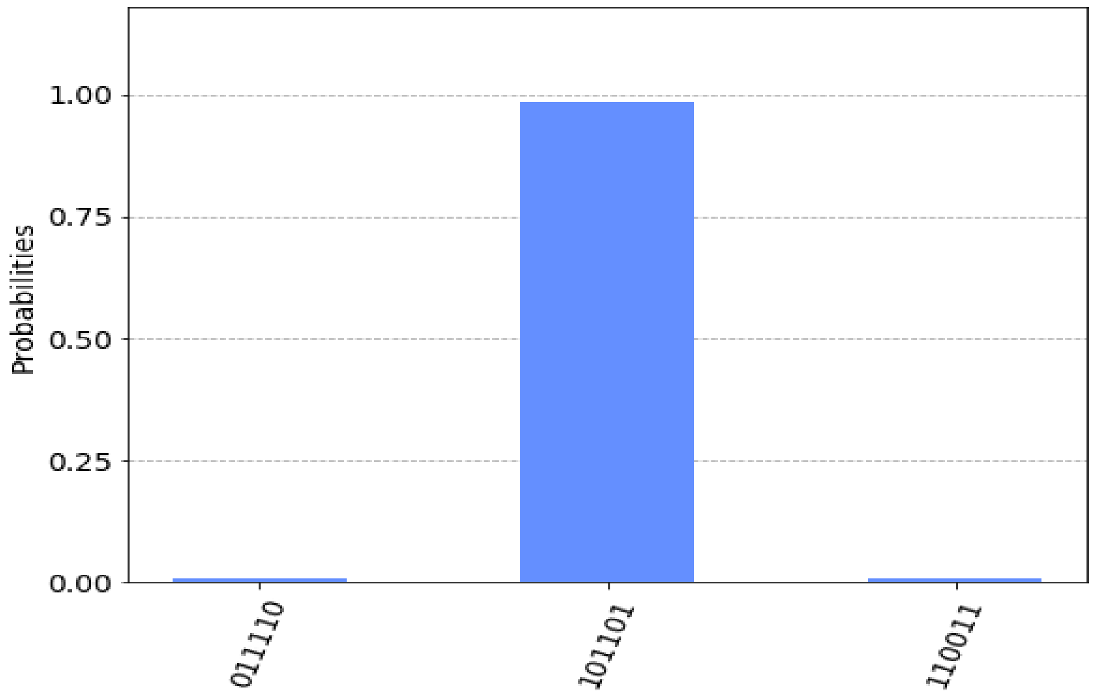


Figure 8.8 – p=20, weight penalty = 5000, runtime more or less the same as above

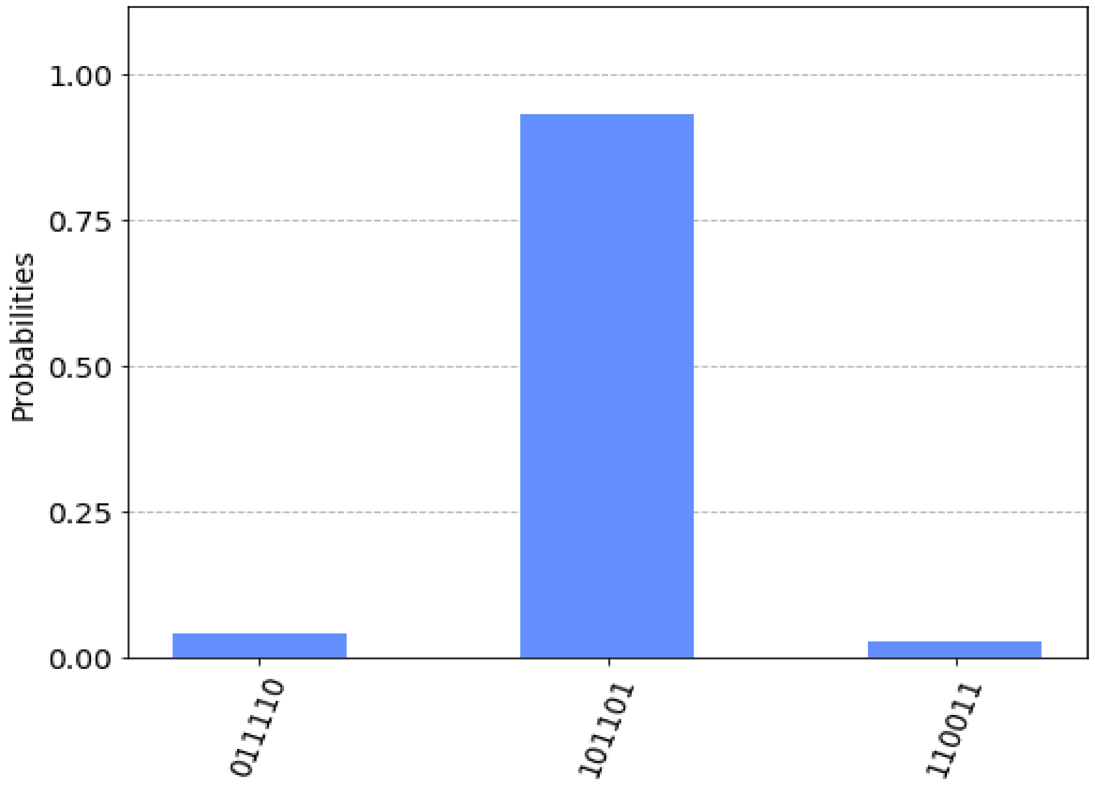


Figure 8.9 – p=35, weight penalty = 5000 – runtime = 683 secs (11.38 mins)

From the above results some things arise. First, the algorithm yields with the highest probability, the desired solution. But still there is high chance for other solution to be yield.

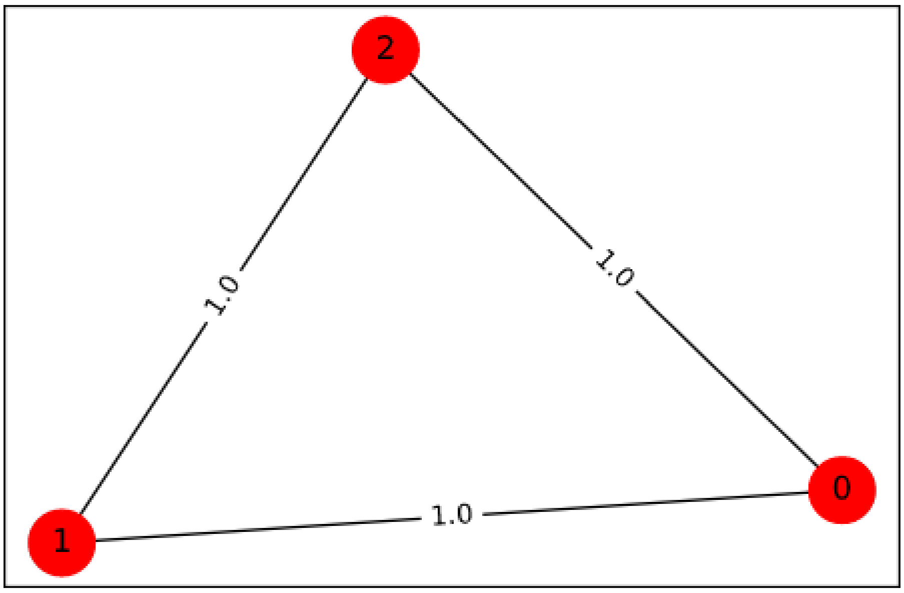
We run the algorithm with higher p(20) and the results were more desirable the probability was close to one. In addition to that , we change the penalty for the missing edge with a higher value and the results were slightly better the probability got almost to 1.

We even tried it with larger p(35) as shown in figure 8.9and the results were surprisingly worse. My intuition is that the optimizer for that problem is facing more difficulty to find the parameters in a higher dimensional parameter space. In that case, more trials in the optimizer and some changes in its parameters could possibly yield better results.

We can see also that the time is growing with larger p. For p = 35 we see the algorithm requires 11.38 minutes to be ran. We see that for a small number of qubits and relatively small p the algorithm struggles. For far more qubits and far bigger p, I don’t think that the simulation could be done by a simulator.

Experiment 3rd

Suppose the graph G = {0,1,2,3} with edges E= {(0,1,1.0), (0,2,1.0), (1,2,1.0)} as can be seen below :



From that instance it is clear that the only solution to the problem is one. In this encoding since there is no order but only the edges in the solution then the solution we expect the algorithm to yield is only one. Specifically, we expect the solution where all edges are included, the 111.

The qubits required for this qubit are n(n-1)/2 = 3*2/2 = 3. This enables us to run the experiment in more than one machine in the IBM Q to catch the differences on the accuracy of the machines. We ran the algorithm in the simulator, the Santiago machine (5 qubits), and the Melbourne (15 qubits). Below are the probabilities of the bitstrings for each experiment.

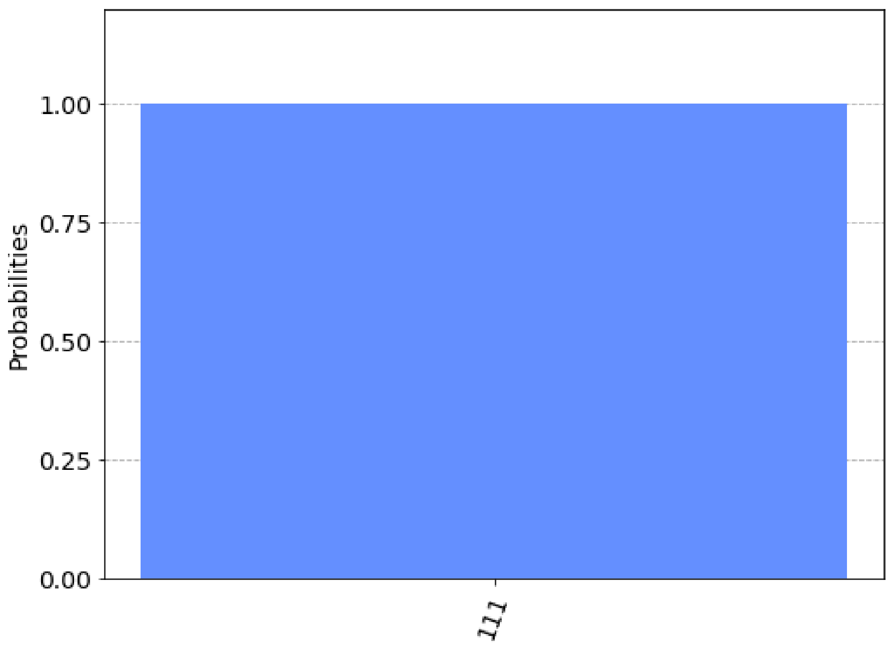


Figure 8.11 – simulator results- runtime = 2.19 secs

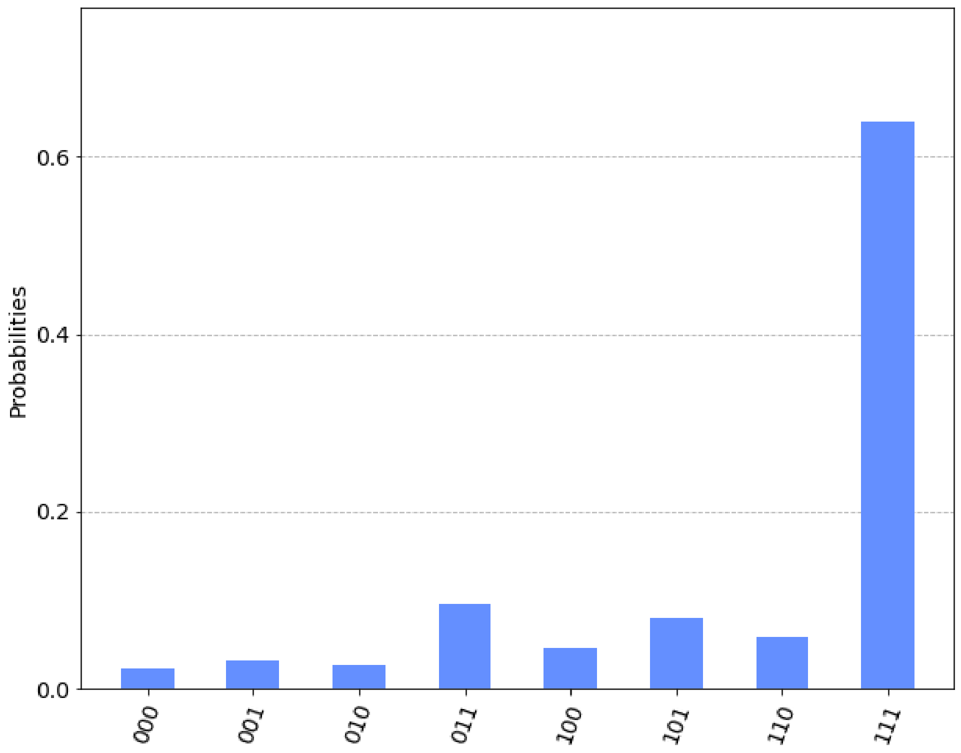


Figure 8.12 – results on IBM Q Santiago quantum machine(5 qubits)

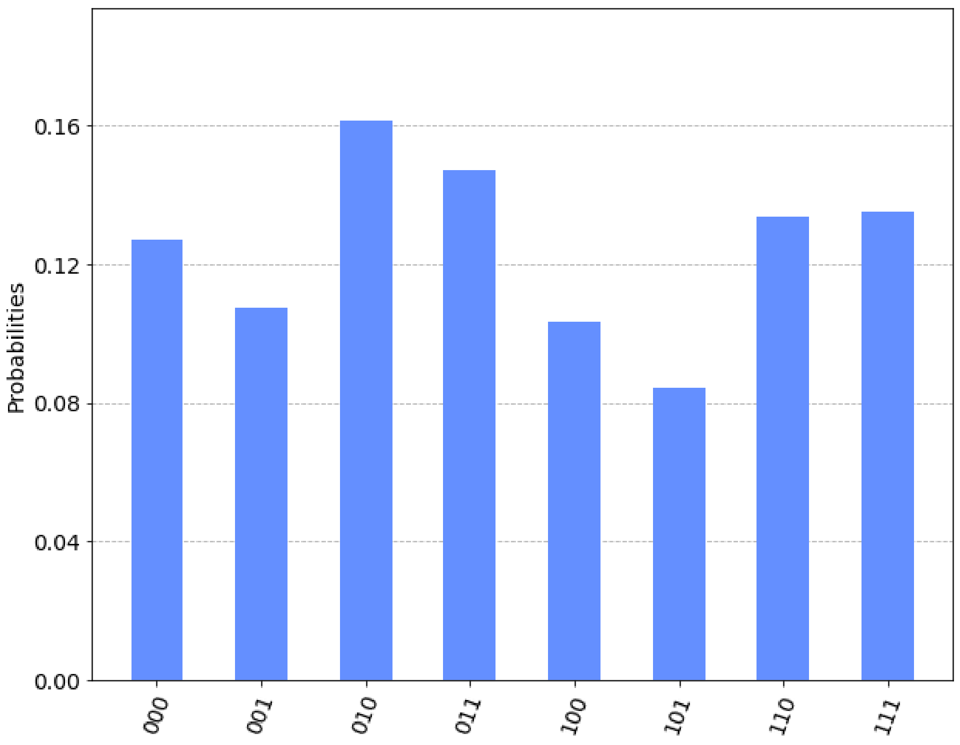


Figure 8.13 – results on IBM Q Melbourne quantum machine(15 qubits)

The simulator works as expected and the algorithm yields with certain probability the correct feasible solution.

However, the interest is on the real quantum machines. We can see that in both the machines the results are not ideal, which is something we expected. These machines are not ideal because they suffer for a big problem in Quantum Computing, Quantum Decoherence. This is a problem that arises from the noise and the interaction with the outer environment of the quantum system. Each machine has different error rates, and the IBMQ-Melbourne (15 qubits) has the most error rate, something that is clear from the above results. A challenge in the Quantum Computing is to mitigate these error rates to have more fault tolerant and thus more practical Machines.

8

Conclusions

Concluding this thesis, we first need to say that quantum computing is a very beautiful and promising field that it seems to have the power to alter the course of Computation with huge impact to humanity. Everything that uses computing today will be in somehow redefined and new fields will arise mostly in the areas such as Physics and Chemistry. The fact that I was engaged with this new fascinating way of computing was a wonderful experience. Furthermore, I think that most of the targets I set for my thesis were completed despite the many difficulties we faced throughout the process.

I started to get familiar and learn about Quantum Computation from its very bottom. Then to study some of the basic quantum algorithms to see the computation in action. For this it was necessary to see what Quantum Programming tools where available so I could practice them. The tool was IBM's Qiskit which for a beginner is one of the best tools available mainly due to its relative simplicity and the variety of support for quantum algorithms. Then, the focus was to the quantum approximation algorithm which is a more universal quantum approach for NP problems. The problem of focus was the Travelling Saleman in which I learned more extensively about and especially its QAOA algorithms. Finally, the direction was on the implementation these algorithms and their experimentation on both quantum simulators real quantum machines to see what their current level is.

In general, in my opinion, the programming on Quantum Computers is really challenging thing. In my opinion , this mainly because of the difficulty of creating and understanding the algorithms and the algorithmic approach of solving a problem. It is very different from the way a computer scientist is familiar, but it is a promising way which with hard and long-term work can be learned. The software support, I believe it is extremely good and it is constantly expanded. The only problem of it, is that the community of Quantum Practitioner is not as large as the computer science, but it is getting bigger. The hardware, however, is not in very good state, at least the hardware which is openly available is very limited and very error prone. This I think it is clear through the results shown above. Something that affects the impact of errors in the results are the length of the circuits and the circuits as in general. The results of the first algorithm on the IBMQ-Melbourne Machine were better comparing to the results of the second algorithm in the same machine despite the fact both results were different than the simulators. That is the length of the circuit of the second algorithm is much bigger that the algorithm 1.

8.1 Challenges and limitations

During the completion of this thesis, we encountered many limitation and challenges.

My first obstacle was the absence of previous experience with quantum computing. Quantum computing is not a course that is offered in the standard curriculum or any of the elective courses at the University of Cyprus. The only source were some books and the Internet, that’s why the steps were slow and small. Over the half of the duration of this thesis included the process of learning what is needed to go.

A highly correlated challenge was the physics of Quantum Computing, mainly Quantum Physics. Quantum Physics not only is the basic core of the primers of Quantum Computing but also played an important role to this thesis though the evolutions of Hamiltonian and Adiabatic Quantum Computation. And Quantum Physics is not something that can be learn and experience in short period of time. Quantum Physics is much bigger and challenging field that there are lot of more to be learned.

There were a lot of challenges when implementing the algorithms in Qiskit and especially when I ran the first algorithm with built in classical optimizers and the results were not the expected, while for other problems and algorithms were. And then when the grid search approach used there were other problems. I could not run the algorithm with bigger p because of my limited hardware resources, and the run-time for smaller and thus much more accurate function evaluation was extremely large.

Another limitation had to do with the hardware that were available from IBM. There were a lot of problems with the hardware of IBM Q. First, the system (because it is one system) that could run a practical example of the two algorithms is fully error-proned and the results prove that. In addition to that, the only example of the second algorithm that could run in the specific system, could not run because of the large depth and size of the circuit (mainly because of the evolution of the mixing Hamiltonian). That’s why for the practical experiments we only worked with the simulator which is also limited because couldn’t simulate large instances as the runtime was extremely large. But then again “nature isn’t classical”. But the systems with fewer qubits are better when it comes to errors so at his level, I think is much easier to experiment with these machines.

An additional limitation was getting familiar with the Qiskit framework. While Qiskit is a very rich framework and maybe the most famous one, still there isn’t anything available out there except the documentation, which is not that crystal clear. While qiskit is rich, there a lot of things that other frameworks are providing with more elegant way.

In general Quantum Computing is a field that needs work mainly because there isn’t so much work available. To harness the powerful physical laws of nature to create computation is not that easy as it is the classical computers. I think that is easier for a physicist to engage with Quantum Computing than a computer scientist, at least when it comes to theoretical part of the quantum computation and the quantum algorithms and excluding the implementation. Nevertheless, it is a great experience that can get you strongly familiar with important knowledge, such as Quantum Mechanics and Linear algebra.

8.2 Future Work

Because Quantum Computing is such a primitive field with huge potential, there are myriad of things that can be done in the future. Quantum Computing has applications and implications in many other fields. Security and Machine Learning are fields that trending now for the Quantum Supremacy. There also a lot of promises(and in the near future) on Finance mainly because of the optimization algorithms like the QAOA and in the Chemistry mainly due to the better simulation of molecules and algorithms like VQE(Variational Quantum Eigen solver) the cousin of QAOA. Promises exist also in other sub-fileds of Artificial Intelligence in problems which are hard for this domain, such as planning search algorithms, Integer Programming and Constraint Satisfaction Problems. Depending on what the future work be, it might require also a different programming tool. For example, if the work is on Quantum Machine Learning, then Pennylane will be more ideal choice.

When it comes directly to this thesis and the QAOA, also there a lot of things that can be considered as future work.

There is a lot of work that can be done when it comes to the parameter tuning in the optimizing step. Unfortunately, we only performed grid search in the space $[0,2\pi]$ for gamma and $[0,\pi]$ for beta, for step size 0.1 and 0.05 and $p = 1$. For $p = 2$, python was crushing. I think the reason is due to memory required to store the size of the grid holding such big number of float(8 bytes) numbers. At least this I think was the bottleneck in my hardware. Also, there is work to be done with the built in optimizers. To try different optimizers for example with more hyperparameter tuning. In general, there is work to be done for the classical optimizing strategy of the technique. Maybe a different tool produces different results and has different optimizers.

In addition to that there is a lot of work when it comes to other problems that QAOA can be applied. There are a lot of problems that even the QAOA^{+}(Alternating) can be applied in which the mixer Hamiltonian may have different structure. For example, Constraint Satisfaction Problems(CSP) such as N-Queen problem, can be adjusted so that is applicable to the algorithm. In general, the procedure will be the same despite some minor changes in the encoding of the solutions and the mixer Hamiltonian.

Finally, the only sure thing is that the hardware of Quantum Computing will grow and improved in the next years, so there will be a lot of work on experimenting bigger problems on actual quantum machines.

Bibliography

[1] Abdulkarim, H., Alshammari, I., (2015). Comparison of Algorithms for Solving Traveling Salesman Problem. *International Journal of Engineering and Advanced Technology* 4(6).

[2] Abraham, H. (2019). Qiskit: An Open-source Framework for Quantum Computing.

[3] Aharonov D., Ta-Shma A.(2003). Adiabatic Quantum State Generation and Statistical Zero Knowledge. arXiv.0301023v2 [quant-ph]

[4] Ambainis A., Balodis K. , Iraids J., Kokainis M., Prusis k., Vihrovs J., 2018, Quantum Speedups for Exponential-time Dynamic Programming Algorithms, arXiv.1807.05209 [quant-ph]

[5] Andrew W. Cross, Lev S. Bishop, John A. Smolin, & Jay M. Gambetta. (2017). Open Quantum Assembly Language. arXiv: 1707.03429v2[quant-ph]

[6] Behera B., Panigrahi P., Satyajit S., Srinivasan K, Srinivasan K.(2018). Efiicient quantum algorithm for solving travelling salesman problem: An IBM quantum experience. arXiv:1805.10928v1 [quant-ph]

[7] Benioff P.(1980). The computer as a physical system: A microscopic quantum mechanical Hamiltonian model of computing as represented in Turing Machines. https://www.dwavesys.com/sites/default/files/styles/Benioff%20--%20Quantum%20Mechanical%20Models%20of%20Turing%20Machines.pdf

[8] Bergamaschi T.(2020). Quantum Approximate Optimization Algorithms on the "Travelling Salesman Problem". https://medium.com/mit-6-s089-intro-to-quantum-computing/quantum-approximate-optimization-algorithms-on-the-traveling-salesman-problem-703b8aee6624

[9] Bernstein E., Vazirani U(1997). Quantum Complexity Theory. Retrieved from https://people.eecs.berkeley.edu/~vazirani/pubs/bv.pdf

[10] Biswas R., Hadfield S., O'Garman B., Rieffel E., Wang Z., Venturelli D.(2019). From the Quantum Approximate Optimization Algorithm to Quantum Alternating Operator Ansatz. arXiv:1709.03489v2 [quant-ph]

[11] Choi S., Lukin M., Pischler H., Wang S., Zhou L., Quantum Approximate Optimization Algorithm: Performance, Mechanism, Implementation on Near-Term Devices. arXiv:1812.01041v2 [quant-ph].

[12] Chuang I., Nielsen M. 2000, Quantum Computation and Quantum Information, Book.

[13] Deutsch D. and Jozsa R. (1992). "Rapid solutions of problems by quantum computation". Proceedings of the Royal Society of London A. 439: 553. Bibcode:1992RSPSA.439..553D. DOI:10.1098/rspa.1992.0167.

[14] Don Monroe. 2019. Closing in on quantum error correction. Commun. ACM 62, 10 (October 2019), 11–13. DOI:https://doi.org/10.1145/3355371

[15] Durr C., Hoyer P. (1999). A quantum algorithm for finding the minimum. arXiv: 9607014v2 [quant-ph].

[16] Farhi E., Goldstone, 2014, A Quantum Approximate Optimisation Algorithm, arXiv.1411.4028v1[quant-ph]

[17] Farhi E., Harrow A. (2019). Quantum Supremacy though the Quantum Approximate Algorithm. arXiv:1602.07674v2

[18] Feynman R. (2013), The Feynman Lectures on Physics, Volume III. Link by https://www.feynmanlectures.caltech.edu/III_toc.html

[19] Feynman, R.P(1982). Simulating physics with computers. Int J Theor Phys 21, 467–488 (1982). https://doi.org/10.1007/BF02650179

[20] Grover, L. (1996). A Fast Quantum Mechanical Algorithm for Database Search. In Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing (pp. 212–219). Association for Computing Machinery.

[21] Liu Z., Marsh S., Ruan Y., Wang J., Xue X. (2020). Quantum Approximate Algorithm for NP optimization problems with constraints. arXiv:2002.00943v1 [quant-ph]

[22] Liu Z., Marsh S., Ruan Y., Wang J., Xue X. (2020). The Quantum Approximate Algorithm for Solving Travelling Salesman Problem. Retrieved from CMC.doi:10.32604/cmc.2020.010001

[23] Manucci M., Yanofsky N., 2008, *Quantum Computing for Computer Science*, Book.

[24] Maylett D., Montanaro A., Linden N., (2016). Quantum Speedups for Traveling Salesman Problem for bounded-degree graphs, arXiv.1612.062003 [quant-ph]

[25] Shor, P. (1999). Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer SIAM Review, 41(2), 303-332.

[26] Sloss A., Gustafson S.(2019). 2019 Evolutionary Algorithms Review. arXiv:1906.08870v1

[27] Bloch Sphere Representation Picture. Available: http://www.sharetechnote.com/html/QC/QuantumComputing_BlochSphere.html

[28] IBM Quantum. Available: https://www.ibm.com/quantum-computing/.

[29] IBM Quantum Documentation. Available: https://quantum-computing.ibm.com/docs/

[30] Matplotlib Documentation. Available: https://matplotlib.org/3.3.3/contents.html

[31] NumPy Documentation. Available: https://numpy.org/doc/stable/

[32] Qiskit Documentation. Available: https://qiskit.org/documentation/

[33] Qiskit Textbook. Available: https://qiskit.org/textbook/preface.html

[34] Volkswagen Quantum Routing. Available: https://www.volkswagen-newsroom.com/en/press-releases/volkswagen-optimizes-traffic-flow-with-quantum-computers-5507.