Solving Combinatorial Optimization Problems using the Quantum Approximation Optimization Algorithm

THESIS

Nicolas J. Guerrero, 2Lt, USAF

AFIT-ENP-MS-20-M-098

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT-ENP-MS-20-M-098

SOLVING COMBINATORIAL OPTIMIZATION PROBLEMS USING THE QUANTUM APPROXIMATION OPTIMIZATION ALGORITHM

THESIS

Presented to the Faculty

Department of Engineering Physics

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

in Partial Fulfillment of the Requirements for the

Degree of Master of Science in Applied Physics

Nicolas J. Guerrero, B.S.

2Lt, USAF

March 2020

Abstract

The Quantum Approximation Optimization Algorithm (QAOA) is one of the most promising applications for noisy intermediate-scale quantum machines due to the low number of qubits required as well as the relatively low gate count. Much work has been done on QAOA regarding algorithm implementation and development; less has been done checking how these algorithms actually perform on a real quantum computer. Using the IBM Q Network, several instances of combinatorial optimization problems (the max cut problem and dominating set problem) were implemented into QAOA and analyzed. It was found that only the smallest toy max cut algorithms performed adequately: those that had at most 10 controlled swap gates. The dominating set problem did not work at all as it used many more controlled swap gates than the allowable number. Additionally, a sufficient condition for polynomial implementation in QAOA is shown that generalizes for all combinatorial optimization problems. Finally, further experiments using random circuits also demonstrated that the qubits have a natural tendency to decohere towards the ground state of the system during the lifetime of the algorithm. While unfortunate, these experiments demonstrate the need for better hardware in order for any sort of practical algorithm to be of use.

Contents

1 Introduction

1.1 History of Quantum Computing

1.1.1 Initial ideas: The 1980s

As with almost all new technology, the dawn of quantum computing started not with a bang, but rather a slight spark. By 1979 classical computers had yet to revolutionize industry, government, and everyday life the way they have in modern times. For example, the new Texas Instruments TI-99/4 microcomputer boasted an incredible 70kb of total memory while consuming 20W [1]. The lack of memory stands in striking contrast to modern computers, where even laptops can have upwards of hundreds of gigabytes of memory. However, the story of quantum computing actually begins with concerns for energy consumption, rather than the overall computing power. In what can be regarded as the first paper on quantum computing, Benioff proposed a quantum mechanical model of Turing machines in order to answer the problem: is it possible to construct a model of the computation process as an evolving closed conservative system? [2] Benioff was concerned with quantum computation not for its own sake, but rather for the applications it might provide to classical computation theory. He showed that for any Turing machine and N steps of the machine, there exists an initial stat $\psi(0)$ and Hamiltonian H such that the state

$$\psi(t) = \exp(-itH)\psi(0) \tag{1}$$

describes the Turing machine. That is, there exist times $t_1, t_2, \cdots, t_N$ such that $\psi(t_i)$ corresponds to the state of the Turing machine after the ith step. Additionally, The Hamiltonian and initial state can be made in such a way that the quantum system is stationary for an arbitrary amount of time around each t_i.

In Benioff's paper, many familiar ideas were introduced which would carry on to later liter-

ature in quantum computing. For example, Benioff was the first to restrict the quantum systems to finite dimensional spaces, therefore bypassing all difficulties associated with infinite dimensional spaces. Additionally, he matched symbols on a Turing tape with the spins of a quantum lattice. Although he generalized his discussion to a tape with $2s + 1$ symbols and a quantum system where each point has $2s$ possible spins (one symbol corresponds to the blank space). Benioff recognized that with this configuration, any string of length N could be represented with N quantum spins. Of course, it is now common to design quantum computers with $s = 1$, which corresponds to the binary alphabet.

The next milestone in quantum computing occurred in 1981. During a keynote address to the First Conference on the Physics of Computation at MIT, Feynman discussed how a physicist could possibly hope to model a physical system with a classical computer [3]. To start, the size of a computer used to describe a physical system could not increase exponentially with a linear increase in the size of the system. However, this is direct contradiction to a many particle system with quantum properties. The simplest example is also the most applicable to quantum computation: imagine a system with N particles whose spins interact in some manner. This system takes at least 2^N bits to describe as there are 2^N different configurations to take account of. Of course, if we add another particle into the mix, then the number of bits needed doubles, which implies that any classical computer used to simulate this system would increase exponentially in size with the number of particles.

In order to get around this apparent impossibility, Feynman introduced the basic idea behind a quantum computer. In order to simulate quantum systems, a physicist would run a quantum computer and tabulate the outputs. These statistics would then inform the researcher about the underlying nature of the quantum system. From his original talk, it is obvious that Feynman considered quantum computers to be more analogous to exotic experimental setups rather than a method for solving difficult classical problems. That is, a quantum computer would perform the

difficult simulations that a classical computer could not hope to replicate, and that these machines would be a small part of the overall experimental process.

The jump from experimental tools to the modern idea of quantum computation did not happen until 1985. At that point, Deutsch published a paper on a theoretical universal quantum computer [4] which would be able to not only model quantum systems efficiently, but also be able to do anything that a classical computer could do (at least in terms of ability, not efficiency). Years later, it would be shown that any quantum algorithm could be reduced in polynomial time to a universal quantum computation. Much like quantum algorithms correspond to universal quantum computations, classical computers correspond to standard Turing machines. A normal Turing machine involves a discrete alphabet and set of rules to write/rewrite this alphabet onto a tape. The quantum Turing machine has three tapes (instead of one): input, intermediate calculations, and final output. Additionally, where a Turing machine can write any element of a finite alphabet onto the tape, a quantum Turing machine can write any element of a Hilbert space onto the intermediate/final tape. For the classical Turing machine, the tape can be any element in a set of strings whereas a quantum Turing tape can be any element in a Hilbert space (usually a different Hilbert space than the 'alphabet' space, although this is not required). Finally, a classical Turing machine's transition function is described by a set of rules; while a quantum Turing machine's transition function is described by unitary matrices.

Deutsch also introduced much of the terminology and practices that are still in use today. He defined the 'computational basis states' to be the set of eigenvectors that span the Hilbert space of the intermediate/final tape. Additionally, he specifically changed the spectrum of the two state particles from $\{-1/2, 1/2\}$ (corresponding to spin down and up) to $\{0, 1\}$ (which corresponded to binary numbers). However, not everything in a universal quantum computer is recognizable to the modern quantum programmer. A Turing machine is said to halt when two states of the tape are identical after a non-trivial operation. This is impossible to implement in the universal

quantum computer as it directly contradicts the no-cloning theorem which had been proved back in 1982 [5]. As such, Deutsch set the following test to tell if a quantum algorithm had halted or not: set an extra qubit to $|0\rangle$, change this qubit to $|1\rangle$ whenever the algorithm has finished but otherwise do not interact with it, and periodically check this qubit to see if it is in the state $|1\rangle$. Note that this terminology varies from Deutsch's original paper, as the term 'qubit' had not yet been coined. Additionally, this extra stipulation to see if a quantum program has halted is not of practical use yet as all quantum algorithms in modern times are built with a predetermined set of gates. The analogous situation in classical computation would be: only running programs that perform single operations on a system and do not have for loops, while loops, and other such difficult commands which might cause a program to run forever.

1.1.2 Practical steps: The 1990s

The first major algorithm to demonstrate the potential of quantum computing was the Deutsch-Jozsa algorithm [6]. Published in 1992, this algorithm gave an answer to the question: Given a function $f : \{0, 1, 2, \cdots, 2N-1\} \to \{0, 1\}$ with the stipulation that

$$f(0) = f(1) = f(2) = \cdots = f(2N-1) \text{ or } |\{i : f(i) = 0\}| = N \tag{2}$$

can we efficiently determine if f is a constant function. For example, suppose $N = 2$ and $f(0) = f(1) = 0$ while $f(2) = 1$. Then f is clearly not a constant function and we are done. Of course, one could classically compute $N+1$ values of f and thereby exactly answer this question for any N. Deutsch and Jozsa provided a quantum algorithm which would require exactly 1 query in order to answer this problem. That is, they could take a function f, run their algorithm once, and know exactly whether f was a constant function or not. The drawback to their algorithm was that afterwards the user would have no idea if the function was identically zero or identically one. However, this was insignificant as the algorithm sped up the process from $N + 1$ operations in

the worst case to 1 operation, a vast improvement over classical computing. While the Deutsch-Jozsa algorithm provided the first example of a speedup for quantum computing versus classical computing, their overall approach did not drastically change anything in the nascent quantum computing field. This was primarily because they worked on a 'toy problem', or problem that had no real world significance. While interesting, the overall effect of their work would reverberate in classrooms rather than companies.

The algorithm that would blow quantum computing out of the back rooms of researchers and into prominence has to do with factoring integers. That is, given a very large integer, can a computer efficiently determine its prime factors? Current research would suggest no, although there is no definitive proof that factoring can be done in polynomial time or is an NP-complete problem. For example, in 2009 researchers factored an RSA-768 (768-bit number) into its two component primes, taking over two years on dozens of classical computers [7]. In fact, it is suspected that integer factoring belongs to NPI (NP-intermediate) although this has not been shown. Nevertheless, the RSA cryptography system directly relies on the difficulty of factoring these large numbers to provide security to everything from banking transactions to mobile texts.

Shor's algorithm is an algorithm to factor large integers in polynomial time and resources [8], something that has the potential to revolutionize almost all aspects of modern encryption. Of course, this may be slightly hyperbolic as the algorithm is not yet usable for actual factoring. For example, in order to factor an 2048-bit number (RSA-2048), it is estimated that around 4000 qubits would be needed as well as 100 million gates [9]. While just an estimate, this demonstrates the vast distance current quantum computers are from actually cracking modern RSA keys. While not applicable yet, Shor's algorithm does demonstrate that quantum computers are more efficient than classical computers (as long as factoring integers turns out to be impossible in polynomial time) while also providing an impetus for governments and industry to begin exploring quantum computers.

Another useful algorithm for practical considerations is Grover's algorithm. Published in 1997 [10], this algorithm does not provide an exponential speedup like Shor's algorithm. Rather, it provides a quadratic speed up when searching lists for a specific member. The problem can be stated thus: consider a list of N elements, only one of which satisfies some condition. How long does it take to find this element? For example, consider the set

$$S = \{1, 3, 9, 5, 22, 29, 31, -3, 5, 14\} \tag{3}$$

and find $s \in S$ such that 7 divides s. A classical computer would take $N = 10$ computations to find 14, as it is at the end of the list. More generally, for a list of length N, a classical computer will find the correct element on average after $N/2$ evaluations while $N - 1$ evaluations for the worst case scenario. Grover's algorithm takes the same list and finds the correct answer using only $\sqrt{N}$ evaluations of the function, a quadratic speedup over classical computers. To put this in perspective, if we had a list with a million elements, it would take on average 500 thousand evaluations to find the answer. Grover's algorithm would take the same list and find the correct answer in a thousand evaluations, 500 times better than a classical computer. This algorithm along with Shor's algorithm provided needed motivation for the new field of quantum computing.

The end of the 1990's brought a flurry of new experimental machines to the public eye. The most impressive such machine might have been the 2 qubit Nuclear Magnetic Resonance (NMR) machine created by researchers at the UC Berkley, Stanford, and MIT. They used this machine to run Grover's algorithm [11] and found agreement between predicted outputs and actual results.

1.1.3 Taking the leap: Post 20th century

The dawn of the new millennium brought a new focus to quantum computing and a realization that the whole field of study had to be standardized/fixed. At the time, different research groups

and universities had different diagrams and ways of expressing quantum circuits/algorithms. A major step towards this goal was the publishing of the first major textbook, Quantum Computation and Quantum Information by Michael Nielsen and Isacc Chuang [18]. A comprehensive look at everything a beginner might need to get into quantum computing, it has become the gold standard for universities teaching the subject. Of course, advancements in teaching were not the only structures added to the field. In 2000, David DiVincenzo introduced his five criteria for a successful quantum computer [12]. These five criteria are:

- A quantum computer must be scalable with well-defined qubits
- The ability to initialize the state of the qubits to a simple fiducial state (this is usually the state $|00\cdots0\rangle$ although it does not have to be)
- Long, relevant decoherence times
- A "universal" set of quantum gates (see section 2.3.1)
- A qubit-specific measurement capability

Additionally, two extra criteria were added for quantum communication purposes

- The ability to interconvert stationary and flying qubits
- The ability to faithfully transmit flying qubits between specified locations

These criteria provided a road map that anyone could follow if they wished to design and build a computer. While simple in theory to explain, no system has yet been created that performs well in all five computing criteria.

At this point in history, quantum computing began to experience a large jump not only in the number of experimental devices brought online, but also in the number of algorithms created and investigated. This led a little over a decade later to the IBM Q Network which provides access through the cloud to physical quantum computers.

1.2 The IBM Q Network

IBM is in a unique position in the quantum computing landscape as they provide their hardware and services to the general public free of charge. Started in 2017 and accessible from the cloud, the IBM Q Network allows users from all over the world to program IBM's quantum computers. People access the physical quantum computers as well as IBM's personal simulators through python commands. The commands that allow one to access the machines are known as 'qiskit', and include all the base gates as well as several special gates for quantum computation. Updated extremely often, qiskit is in version 11 as of the end of 2019.

The IBM quantum computers come in three flavors: public machines, private machines, and the simulator. The public computers include

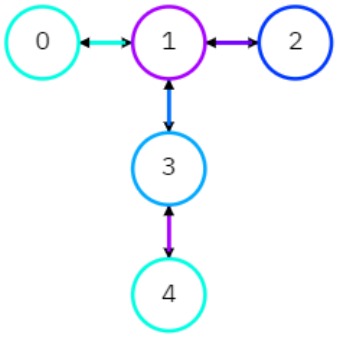


Figure 1: London: a 5 qubit quantum computer accessible to the public. It came online 13 September, 2019. The different colors correspond to node/edge the fidelity the day this was taken while the edges denote that qubits that can be entangled.

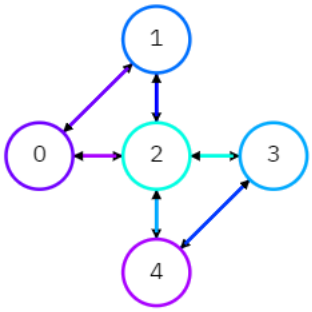


Figure 2: Yorktown: a 5 qubit quantum computer accessible to the public. It came online 24 January, 2017. The different colors correspond to node/edge the fidelity the day this was taken while the edges denote that qubits that can be entangled.

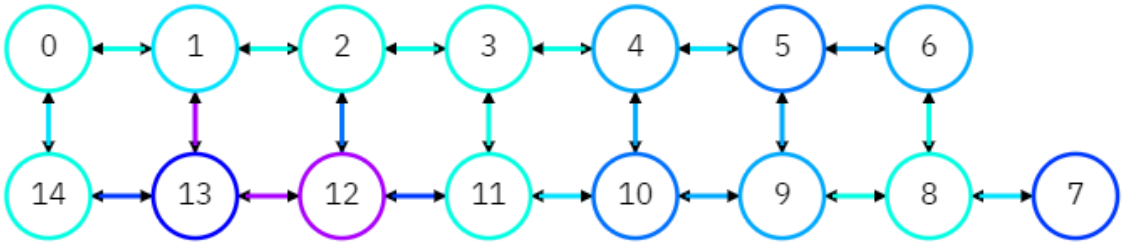


Figure 3: Melbourne: a 15 qubit quantum computer accessible to the public. It cam online 6 November, 2018. It used to have 16 qubits, but one has been permanently taken off line.

While these are the only public machines available, they give a good indication of the type of hardware that IBM makes ready for the public. The private machines are limited to

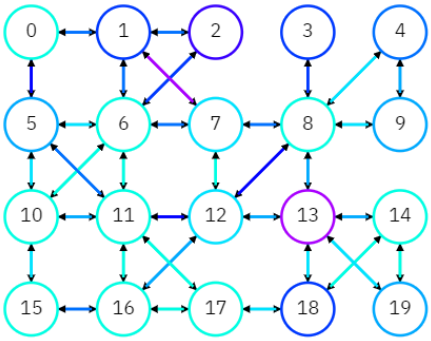


Figure 4: Tokyo: a 20 qubit quantum computer accessible for contracted access. It came online 10 May, 2018. Unfortunately, it has been offline since the beginning of October 2019.

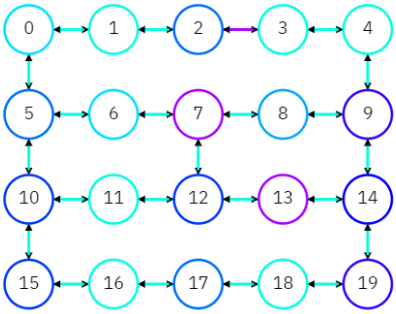


Figure 5: Poughkeepsie: a 20 qubit quantum computer accessible for contracted access. It came online 29 August, 2018. All experiments performed in this paper were ran on this quantum computer.

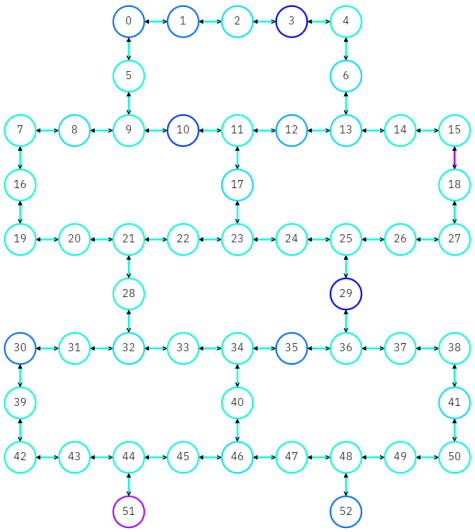


Figure 6: Rochester: a 53 qubit quantum computer accessible for contracted access. It came online 1 October, 2019. It is the largest quantum computer that IBM provides access to.

Of course, all of these quantum computers are paired nicely with the IBM simulator. This allows researchers to check there computations against expected results, which allows for better research and algorithms overall.

These machines allow for many different levels of control, some of which are so esoteric as to be understood by possibly a handful of researchers at any given time. However, the two main controls used in this paper are the number of shots performed as well as the optimization level for algorithm implementation. The former simply describes how many times an quantum computation will be run before sending results back to the user. Obviously, the more shots taken the better the statistics but at the cost of more time. The optimization level describes how qiskit matches logical qubits to physical qubits in the system. It comes in four levels: $0-3$. Levels 0 and 1 correspond to a low computational overhead while 2 and 3 correspond to a high computational overhead. Stated simply, 0 and 1 are what can be realistically found of implementation while 2 and 3 are useful for best case scenario testing.

2 Quantum Approximation Optimization Algorithm (QAOA)

The Quantum Approximation Optimization Algorithm (QAOA) is thought to be one of the best candidates for noisy intermediate-scale quantum (NISQ) computation due to the low number of qubits and gates generally needed. This algorithm takes a combinatorial optimization problem and attempts to approximate a solution to whatever level of accuracy is possible/desired. Generally, these problems will be NP-complete, else it would make more sense to implement it onto a classical computer. In this section, the algorithm itself is discussed, as well as the types of gates available for use and how they are implemented from the base gates available in the IBM Q Network.

2.1 Combinatorial Optimization Problems

Formally, combinatorial optimization problems are maximization or minimization problems with s input strings in some set S and m clauses (with $s \geq m$) [13]. Each clause takes as input a string and returns a value. The total cost function of a string is the sum over the m clauses. Symbolically, if the input string is denoted by z and clauses by C_k, then the total cost function is

$$C(z) = \sum_{k=1}^{m} C_k(z) \tag{4}$$

The goal is then to find $\overline{z} \in S$ such that $C(\overline{z}) \geq C(z)$ for all $z \in S$ (in the case of minimization, $C(\overline{z}) \leq C(z)$ for all $z \in S$). Note that $\overline{z}$ need not be unique.

In order to simplify the problem somewhat, restrictions will be put on the clauses and input strings. The clauses are restricted to the integers 0 and 1 while input strings will be restricted to the binary representations of the integers 0 through $2^n - 1$. That is, z can be written as

$z = z_0 z_1 ... z_{n-1}$ for $z_i \in \{0, 1\}$. Additionally, only maximization problems shall be investigated as minimization problems can be studied in the following manner: if $C_k(z)$ is a clause in a minimization problem, then the corresponding maximization clause is $C_k^{'}(z) = 1 - C_k(z)$. Then $\overline{z}$ for $C(z)$ is identical to $\overline{z}$ for $C^{'}(z)$ as

$$C^{'}(\overline{z}) = \sum_{k=0}^{m-1} C_k^{'}(\overline{z}) = \sum_{k=0}^{m-1}(1 - C_k(\overline{z})) = m - \sum_{k=0}^{m-1} C_k(\overline{z}) = m - C(\overline{z}) \tag{5}$$

and $C(z)$ is minimized at $\overline{z}$.

2.1.1 NP-complete problems

The goal of QAOA is to use a quantum computer to find $\overline{z}$ or z such that $C(z)$ approximates $C(\overline{z})$. As with any measure, what defines 'approximates' varies from problem to problem. Of course, some problems are better suited to approximation than others. For example, if you run a mapping algorithm that finds a route in 105km and the best route is 100km, then you will probably be ambivalent to a 5% discrepancy. However, you might consider a 5% discrepancy to be unacceptable for another application (yearly budget perhaps). As with everything in life, context matters. Using QAOA to approximate certain problems can be useful but should be tempered by reasonable expectations for what the results might show.

In general, these types of problems can be grouped into 'easy' or 'hard' problems for classical computers. For example, define $C_k(z)$ as 1 if $z = k$ and 0 otherwise. Then $\overline{z}$ is any element of S and the cost function is maximized at any input. This is an 'easy' problem . Unfortunately (or fortunately depending on your point of view), it is generally useless to implement easy problems using QAOA as there are already efficient algorithms that can solve the problem using classical computers. However, it is possible that an polynomial time problem has a much more efficient algorithm that can be implemented in QAOA. Some possible examples of easy problems that one might think could be put into QAOA are

- Calculating the greatest greatest common divisor between two integers (Euclids algorithm [14])
- Fast Fourier Transform (runs in [15])
- Primality checking (AKS in [16])

Unfortunately, QAOA is probably not useful for these problems (the next list of examples will illustrate why). On the other hand, there are lots of 'hard' problems which can be implemented into QAOA. Some of these are

- The max cut problem (MCP), see section 3
- The dominating set problem (DSP), see section 4
- The vertex cover problem
- The K-coloring problem

These problems all have something in common: they are well-known NP complete problems from graph theory. This is because QAOA assigns each qubit to a node in the graph, and then performs a computation based on the interactions between different nodes. As such, if someone wanted to use QAOA for a polynomial time problem, the best case example would be some problem in graph theory that can be solved in polynomial time and use QAOA to find the answer in a better polynomial time. One such example could be the minimum cut problem. The opposite of the MCP (which is NP complete), the minimum cut problem can be solved in $O(n^2)$ time on a classical computer. Using QAOA, it is possible that this is reducible down to $O(n)$ time. In fact, since the max cut and minimum cut problems would most likely be implemented in a similar fashion, it is not unreasonable that this might be the case. However, further research is needed in order to answer this question.

2.2 Symbolic Representation

Initially, it is instructive to discuss QAOA from a purely symbolic perspective. In general, QAOA is an algorithm that takes as inputs a set of m clauses, a set of p angles denoted by $\{\gamma_0, \gamma_1, ..., \gamma_{p-1}\}$, and a set of p angles denoted by $\{\beta_0, \beta_1, ..., \beta_{p-1}\}$. However, as increasing p simply increases the number of iterations of the initial algorithm, we will only consider $p = 1$ in this paper. For the sake of notation, denote $\gamma_0 = \gamma$ and $\beta_0 = \beta$. Define the matrix

$$U_C(\gamma) = \prod_{k=1}^{m} \exp(-i\gamma C_k) \tag{6}$$

where C_k is a $n \times n$ diagonal matrix with entries defined by

$$C_k(i,j) = \begin{cases} 1 & \text{if } i = j \text{ and } C_k(i-1) = 1 \\ 0 & \text{otherwise} \end{cases} \tag{7}$$

Note that we have $i - 1$ in the above definition as z is indexed from 0 to $2^n - 1$. For example, for $n = 2$ and the clause

$$C_1(z) = \begin{cases} 1 & \text{if } z \in \{1, 2\} \\ 0 & \text{if } z \in \{0, 3\} \end{cases} \tag{8}$$

the matrix is

$$C_1 = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \tag{9}$$

If we include the additional clause

$$C_2(z) = \begin{cases} 1 & \text{if } z \in \{1\} \\ 0 & \text{if } z \in \{0, 2, 3\} \end{cases} \tag{10}$$

to the example above, we can construct $U_C(\gamma)$:

$$U_C(\gamma) = \prod_{k=1}^{2} \exp(-i\gamma C_k) = \exp\left[-i\gamma \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}\right] \exp\left[-i\gamma \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}\right] \tag{11}$$

$$= \exp\left\{-i\gamma \left[\begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} + \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}\right]\right\} \tag{12}$$

$$= \exp\left[-i\gamma \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}\right] \tag{13}$$

If we return to the original cost function $C(z) = C_1(z) + C_2(z)$, we see that $C(0) = 0$, $C(1) = 2$, $C(2) = 1$, and $C(3) = 0$. This provides and alternate definition of $U_C(\gamma)$:

$$U_C(\gamma) = \exp(-i\gamma C) \tag{14}$$

where C is a diagonal matrix with entries

$$C(i,j) = \begin{cases} C(i-1) & \text{if } i = j \\ 0 & \text{otherwise} \end{cases} \tag{15}$$

In addition, since $C(z)$ has integer values we need only consider γ between 0 and 2π. This is evident as for any $\gamma_1 \notin [0, 2\pi]$ there exists $\gamma_0 \in [0, 2\pi]$ such that $\gamma_1 = \gamma_0 + 2\pi k$ (for some $k \in \mathbb{Z}$). Then we have

$$U_C(\gamma_1) = \exp(-i\gamma_1 C) = \exp[-i(\gamma_0 + 2\pi k)C] = \exp(-i\gamma_0 C)\exp(-i2\pi k C) \tag{16}$$

However, since C is a diagonal matrix with integer values, this simplifies to

$$= \exp(-i\gamma_0 C) \begin{pmatrix} e^{-i2\pi k C(0)} & 0 & \dots & 0 \\ 0 & e^{-i2\pi k C(1)} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & e^{-i2\pi k C(2^n-1)} \end{pmatrix} \tag{17}$$

$$= \exp(-i\gamma_0 C) \begin{pmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 \end{pmatrix} = \exp(-i\gamma_0 C) I = \exp(-i\gamma_0 C) = U_C(\gamma_0). \tag{18}$$

Now, define

$$B = \sum_{k=1}^{n} \left(\bigotimes_{i=1}^{k-1} I \otimes \sigma_x \otimes \bigotimes_{i=k+1}^{n} I \right) = \sigma_x \otimes I^{\otimes(n-1)} + I \otimes \sigma_x \otimes I^{\otimes(n-2)} + \cdots + I^{\otimes(n-1)} \otimes \sigma_x \tag{19}$$

(where $\sigma_x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$) and the matrix $U_B(\beta)$ as

$$U_B(\beta) = \exp(-i\beta B). \tag{20}$$

In their original paper [13], Farhi and Goldstone let β run from 0 to π, which implies an input domain of $(\gamma, \beta) \in [0, 2\pi] \times [0, \pi]$. However, this can be reduced to either $(\gamma, \beta) \in [0, 2\pi] \times [0, \frac{\pi}{2}]$ or $(\gamma, \beta) \in [0, \pi] \times [0, \pi]$ without loss of information.

To see this, consider the arbitrary angles $0 \leq \gamma_0 < 2\pi$ and $0 \leq \beta_0 < \pi$ and define the new angles

$$\gamma_1 = -\gamma_0 + 2\pi \tag{21}$$

$$\beta_1 = -\beta_0 + \pi \tag{22}$$

Note that we have simply reflected γ_0 across the point π and β_0 across the point $\frac{\pi}{2}$. Thus, if we can show that running QAOA with γ_1 and β_1 produces the same output as γ_0 and β_0 (by same output, we mean the same probabilities for measurement), then we could reduce either domain by a factor of a half and still have the full range of outputs. To this end

$$U_C(\gamma_1)U_B(\beta_1) = e^{-i\gamma_1 C}e^{-i\beta_1 B} = e^{-i(-\gamma_0+2\pi)C}e^{-i(-\beta_0+\pi)B}. \tag{23}$$

Since C is diagonal, we have

$$e^{-i(-\gamma_0+2\pi)C} = \begin{pmatrix} e^{-i(-\gamma_0+2\pi)C(0)} & 0 & \cdots & 0 \\ 0 & e^{i(-\gamma_0+2\pi)C(1)} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & e^{-i(-\gamma_0+2\pi)C(2^n-1)} \end{pmatrix} \tag{24}$$

$$= \begin{pmatrix} e^{i\gamma_0 C(0)} e^{-2\pi i C(0)} & 0 & \cdots & 0 \\ 0 & e^{i\gamma_0 C(1)}) e^{-2\pi i C(1)} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & e^{i\gamma_0 C(2^n-1)} e^{-2\pi i C(2^n-1)} \end{pmatrix} \tag{25}$$

But $C(z) \in \mathbb{Z}$, which implies $e^{-2\pi i C(z)} = 1$. Thus, the matrix above simplifies to

$$= \begin{pmatrix} e^{i\gamma_0 C(0)} & 0 & \cdots & 0 \\ 0 & e^{i\gamma_0 C(1)}) & 0 & 0 \\ \vdots & 0 & \ddots & \vdots \\ 0 & 0 & \cdots & e^{i\gamma_0 C(2^n-1)} \end{pmatrix} = e^{i\gamma_0 C}. \tag{26}$$

For the case involving $U_B(\beta_1)$ we must first prove all the eigenvalues of B are odd or even integers. Before proceeding, we define B in a recursive manner which shall be conducive to our future work. Define

$$B_1 = \sigma_x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \tag{27}$$

$$B_{n+1} = \begin{pmatrix} B_n & I_n \\ I_n & B_n \end{pmatrix} \tag{28}$$

where I_n is the identity matrix of size $2^n \times 2^n$. We proceed by induction to show this recursive definition is equivalent to the definition for B defined above (for n qubits). For $n = 1$, we have

$$B_1 = \sum_{k=1}^{1} \left(\bigotimes_{i=1}^{0} I \otimes \sigma_x \otimes \bigotimes_{i=2}^{1} I \right) = \sigma_x \tag{29}$$

and the definition holds. Now, suppose it is true for $n-1$ qubits. Then

$$B_n = \sum_{k=1}^{n} \left(\bigotimes_{i=1}^{k-1} I \otimes \sigma_x \otimes \bigotimes_{i=k+1}^{n} I \right) \tag{30}$$

$$= \sigma_x \otimes I_1 \otimes I_1 \otimes \cdots \otimes I_1 + I_1 \otimes \sigma_x \otimes I_1 \otimes I_1 \otimes \cdots \otimes I_1 + \cdots + I_1 \otimes I_1 \otimes \cdots \otimes I_1 \otimes \sigma_x \tag{31}$$

$$= \sigma_x \otimes I_1 \otimes I_1 \otimes \cdots \otimes I_1 + I_1 \otimes (\sigma_x \otimes I_1 \otimes I_1 \otimes \cdots \otimes I_1 + \cdots + I_1 \otimes \cdots \otimes I_1 \otimes \sigma_x) \tag{32}$$

$$= \sigma_x \otimes I_{n-1} + I_1 \otimes B_{n-1} = \begin{pmatrix} 0 & I_{n-1} \\ I_{n-1} & 0 \end{pmatrix} + \begin{pmatrix} B_{n-1} & 0 \\ 0 & B_{n-1} \end{pmatrix} = \begin{pmatrix} B_{n-1} & I_{n-1} \\ I_{n-1} & B_{n-1} \end{pmatrix}. \tag{33}$$

We conclude our definition holds for all n. Next, to show the eigenvalues are all odd or even integers, we again turn to induction. For B_1, it is obvious that the eigenvalues are ± 1. Now, suppose the eigenvalues for B_{n-1} are all odd or even integers. We can calculate the eigenvalues of B_n by solving

$$0 = |B_n - \lambda I_n| = \left| \begin{pmatrix} B_{n-1} & I_{n-1} \\ I_{n-1} & B_{n-1} \end{pmatrix} - \lambda I_n \right| = \begin{vmatrix} B_{n-1} - \lambda I_{n-1} & I_{n-1} \\ I_{n-1} & B_{n-1} - \lambda I_{n-1} \end{vmatrix}. \tag{34}$$

However, using the fact that for symmetric block matrices

$$\det \begin{pmatrix} A & B \\ B & A \end{pmatrix} = \det(A-B)\det(A+B), \tag{35}$$

we get

$$= \det(B_{n-1} - \lambda I_{n-1} - I_{n-1}) \det(B_{n-1} - \lambda I_{n-1} + I_{n-1}) \tag{36}$$

$$= \det(B_{n-1} - (\lambda+1) I_{n-1}) \det(B_{n-1} - (\lambda-1) I_{n-1}). \tag{37}$$

But both of these determinants are simply the characteristic polynomial for B_{n-1} with roots shifted ± 1. Thus, the eigenvalues of B_n are the eigenvalues of B_{n-1} with 1 added or subtracted. Since we assumed the eigenvalues of B_{n-1} were all odd or even integers, the eigenvalues of B_n must be all odd or even integers. This proves our proposition.

Having solved for the eigenvalues of B_n we are finally ready to work with $U_B(\beta_1)$. We have

$$U_B(\beta_1) = e^{-i\beta_1 B} = e^{-i(-\beta_0+\pi)B}. \tag{38}$$

Since B is diagonalizable (it is the tensor product of diagonalizable matrices), we know $B = UB_DU^{-1}$ where B_D is a matrix with the eigenvalues of B along the diagonal and zeros elsewhere and U is some unitary matrix. This implies

$$= Ue^{-i(-\beta_0+\pi)B_D}U^{-1} = Ue^{i\beta_0 B_D}e^{-i\pi B_D}U^{-1}. \tag{39}$$

However, since the eigenvalues of B are all odd or even integers, we know that $e^{-i\pi B_D}$ is simply the identity matrix times $(-1)^n$ (depending on wether the eigenvalues are odd or even respectively). Since this matrix commutes with everything, we are left with

$$= (-1)^n U e^{i\beta_0 B_D} U^{-1} = (-1)^n e^{i\beta_0 B}. \tag{40}$$

We may finally conclude that

$$U_B(\beta_1)U_C(\gamma_1) = (-1)^n e^{i\beta_0 B} e^{i\gamma_0 C}. \tag{41}$$

In our final measurement of the system, the $(-1)^n$ clearly has no effect as it is squared to 1. All that is left is $e^{i\beta_0 B}e^{i\gamma_0 C}$, which are clearly matrices $U_B(\beta_0)$ and $U_C(\gamma_0)$ with rotation in the opposite directions. That is, the final measurement will be equal whether we use the angles (γ_0, β_0) or (γ_1, β_1).

It is reasonable to ask whether $(\gamma, \beta) \in [0, 2\pi] \times [0, \frac{\pi}{2}]$ or $(\gamma, \beta) \in [0, \pi] \times [0, \pi]$ is a better domain in the long run. Later on, it will be shown that the former provides a better output space for measurements than the latter. However, before continuing the rest of the algorithm must be presented. The input state of the algorithm is $H^{\otimes n}|0\rangle^{\otimes n}$ where H is a standard Hadamard gate

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \tag{42}$$

Thus, running QAOA with a set of angles (γ, β) entails implementing the following matrices on a quantum computer:

$$|\psi\rangle = U_B(\beta)U_C(\gamma)H^{\otimes n}|0\rangle^{\otimes n} \tag{43}$$

and measuring each qubits output. If $p > 1$ was used, then this would be

$$|\psi\rangle = U_B(\beta_{p-1})U_C(\gamma_{p-1}) \cdots U_B(\beta_1)U_C(\gamma_1)U_B(\beta_0)U_C(\gamma_0)H^{\otimes n}|0\rangle^{\otimes n} \tag{44}$$

which demonstrates how increasing p simply cycles through the algorithm.

Now, $(\gamma, \beta) \in [0, 2\pi] \times [0, \frac{\pi}{2}]$ provides a better result as

$$P(|z\rangle)\Big|_{\gamma=0} = P(|z\rangle)\Big|_{\gamma=2\pi} = P(|z\rangle)\Big|_{\beta=0} = P(|z\rangle)\Big|_{\beta=\frac{\pi}{2}} = \frac{1}{2^n} \tag{45}$$

That is, around the edge of the input domain, all states are equally likely to be measured. In order to show this, we will proceed through all four cases. First, consider the case where $\gamma = 0$. Then

$$U_C(0) = \exp(-i0C) = \exp(0) = I \tag{46}$$

and we may ignore the U_C gates. Thus, the final state of the quantum computer (before measurement) is

$$U_B(\beta)H^{\otimes n}|0\rangle^{\otimes n} \tag{47}$$

However, we may also replace $U_B(\beta)$ by noting that

$$U_B(\beta) = \begin{pmatrix} \cos(\beta) & -i\sin(\beta) \\ -i\sin(\beta) & \cos(\beta) \end{pmatrix}^{\otimes n}. \tag{48}$$

In order to show this relationship, we will use the recursive definition from equation (28) and induction. Obviously, for $n = 1$, we have

$$U_B^1(\beta) = \exp(-i\beta B_1) = \exp\left[-i\beta \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}\right] \tag{49}$$

which we can easily calculate to see

$$= \begin{pmatrix} \cos(\beta) & -i\sin(\beta) \\ -i\sin(\beta) & \cos(\beta) \end{pmatrix} = \begin{pmatrix} \cos(\beta) & -i\sin(\beta) \\ -i\sin(\beta) & \cos(\beta) \end{pmatrix}^{\otimes 1} . \tag{50}$$

Now, assume the proposition is true for $n-1$ and consider $U_B^n(\beta)$ with the recursive definition established previously:

$$U_B^n(\beta) = \exp(-i\beta B_n) = \exp\left[-i\beta \begin{pmatrix} B_{n-1} & I_{n-1} \\ I_{n-1} & B_{n-1} \end{pmatrix}\right] \tag{51}$$

$$= \exp\left[-i\beta \begin{pmatrix} 0 & I_{n-1} \\ I_{n-1} & 0 \end{pmatrix} - i\beta \begin{pmatrix} B_{n-1} & 0 \\ 0 & B_{n-1} \end{pmatrix}\right] \tag{52}$$

$$= \exp\left[-i\beta(\sigma_x \otimes I_{n-1} + I_1 \otimes B_{n-1})\right]. \tag{53}$$

Next, note that the matrices in the exponential follow the format for the Kronecker sum [17]. Thus, we can simplify (53) to

$$= \exp\left(-i\beta\sigma_x\right) \otimes \exp(-i\beta B_{n-1}) = \begin{pmatrix} \cos(\beta) & -i\sin(\beta) \\ -i\sin(\beta) & \cos(\beta) \end{pmatrix} \otimes \exp(-i\beta B_{n-1}). \tag{54}$$

However, by our inductive hypothesis we know that

$$\exp(-i\beta B_{n-1}) = U_B^{n-1}(\beta) = \begin{pmatrix} \cos(\beta) & -i\sin(\beta) \\ -i\sin(\beta) & \cos(\beta) \end{pmatrix}^{\otimes(n-1)} . \tag{55}$$

Thus, (54) simplifies to

$$= \begin{pmatrix} \cos(\beta) & -i\sin(\beta) \\ -i\sin(\beta) & \cos(\beta) \end{pmatrix} \otimes \begin{pmatrix} \cos(\beta) & -i\sin(\beta) \\ -i\sin(\beta) & \cos(\beta) \end{pmatrix}^{\otimes(n-1)} = \begin{pmatrix} \cos(\beta) & -i\sin(\beta) \\ -i\sin(\beta) & \cos(\beta) \end{pmatrix}^{\otimes n} \tag{56}$$

which proves the proposition. Therefore, the final state of the system (47) becomes

$$\begin{pmatrix} \cos(\beta) & -i\sin(\beta) \\ -i\sin(\beta) & \cos(\beta) \end{pmatrix}^{\otimes n} H^{\otimes n}|0\rangle^{\otimes n} = \left[\begin{pmatrix} \cos(\beta) & -i\sin(\beta) \\ -i\sin(\beta) & \cos(\beta) \end{pmatrix} H|0\rangle \right]^{\otimes n}. \tag{57}$$

Since this state is factorable, we may consider the probability that any qubit is measured in state 0. That is, the probability that qubit k is in state 0 is

$$P(|q_k\rangle = |0\rangle) = \left| \langle 0| \begin{pmatrix} \cos(\beta) & -i\sin(\beta) \\ -i\sin(\beta) & \cos(\beta) \end{pmatrix} H|0\rangle \right|^2 \tag{58}$$

$$= \frac{1}{2} \left| \begin{pmatrix} 1 & 0 \end{pmatrix} \begin{pmatrix} \cos(\beta) & -i\sin(\beta) \\ -i\sin(\beta) & \cos(\beta) \end{pmatrix} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} \right|^2 = \frac{1}{2}. \tag{59}$$

Of course, this implies that the probability of measuring qubit k in state 1 is also $\frac{1}{2}$. Since the index k was arbitary, the probability of measuring any output is

$$P(|z\rangle)\Big|_{\gamma=0} = \left(\frac{1}{2}\right)^n = \frac{1}{2^n}. \tag{60}$$

Next, consider the case where $\gamma = 2\pi$. Since the eigenvalues of C are integers, this implies

$$U_C(2\pi) = \exp(-i2\pi C) = \begin{pmatrix} e^{-i2\pi C(0)} & 0 & \dots & 0 \\ 0 & e^{-i2\pi C(1)} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & e^{-i2\pi C(2^n-1)} \end{pmatrix} = I. \tag{61}$$

As this is the same as the $\gamma = 0$ case, we conclude

$$P(|z\rangle)\Big|_{\gamma=2\pi} = P(|z\rangle)\Big|_{\gamma=0} = \frac{1}{2^n} \tag{62}$$

For the third case, let $\beta = 0$. Then clearly

$$U_B(0) = \exp(-i0B) = \exp(0) = I \tag{63}$$

and we have

$$P(|z\rangle)\Big|_{\beta=0} = |\langle z|U_C(\gamma)H^{\otimes n}|0\rangle^{\otimes n}|^2. \tag{64}$$

We may replace the initial state in (64) with

$$H^{\otimes n}|0\rangle^{\otimes n} = \frac{1}{\sqrt{2^n}} \sum_{k=0}^{2^n-1} |k\rangle \tag{65}$$

to get

$$= \left| \langle z|U_C(\gamma) \frac{1}{\sqrt{2^n}} \sum_{k=0}^{2^n-1} |k\rangle \right|^2 = \frac{1}{2^n} \left| \sum_{k=0}^{2^n-1} \langle z|U_C(\gamma)|k\rangle \right|^2. \tag{66}$$

Now, consider just the inner-product of the above equation: $\langle z|U_C(\gamma)|k\rangle$. $\langle z|$ is a row vector of all 0s except for one 1 in the $(z+1)$th spot, $U_C(\gamma)$ is a diagonal matrix with entry $\exp(-i\gamma C(s))$ in the $(s+1)$th spot, and $|k\rangle$ is a column vector of all 0s except for one 1 in the kth entry. Put

together, this is

$$\langle z|U_C(\gamma)|k\rangle = \begin{pmatrix} 0 & \cdots & 1_{z+1} & \cdots & 0 \end{pmatrix} \begin{pmatrix} e^{-i\gamma C(0)} & 0 & \dots & 0 \\ 0 & e^{-i\gamma C(1)} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & e^{-i\gamma C(2^n-1)} \end{pmatrix} \begin{pmatrix} 0 \\ \vdots \\ 1_k \\ \vdots \\ 0 \end{pmatrix} \tag{67}$$

$$= \begin{pmatrix} 0 & \cdots & 1_{z+1} & \cdots & 0 \end{pmatrix} \begin{pmatrix} 0 \\ \vdots \\ e^{-i\gamma C(k-1)} \\ \vdots \\ 0 \end{pmatrix} = \delta_{k(z+1)} e^{-i\gamma C(k-1)}. \tag{68}$$

Substituting this into (66), we get

$$= \frac{1}{2^n} \left| \sum_{k=0}^{2^n-1} \delta_{k(z+1)} e^{-i\gamma C(k-1)} \right|^2 = \frac{1}{2^n} \left| e^{-i\gamma C(z)} \right|^2 = \frac{1}{2^n}. \tag{69}$$

We conclude

$$P(|z\rangle)\Big|_{\beta=0} = \frac{1}{2^n}. \tag{70}$$

For the final situation, let $\beta = \frac{\pi}{2}$. Then using equation (48), we know

$$\exp\left(-i\frac{\pi}{2}B\right) = \begin{pmatrix} \cos\left(\frac{\pi}{2}\right) & -i\sin\left(\frac{\pi}{2}\right) \\ -i\sin\left(\frac{\pi}{2}\right) & \cos\left(\frac{\pi}{2}\right) \end{pmatrix}^{\otimes n} = \begin{pmatrix} 0 & -i \\ -i & 0 \end{pmatrix}^{\otimes n} = (-i)^n \sigma_x^{\otimes n}. \tag{71}$$

Pluggin this into (44), we get

$$\exp\left(-i\frac{\pi}{2}B\right)\exp(-i\gamma C)H^{\otimes n}|0\rangle^{\otimes n} = (-i)^n\sigma_x^{\otimes n}\exp(-i\gamma C)H^{\otimes n}|0\rangle^{\otimes n}. \tag{72}$$

Then the probability of measuring any state is

$$P(|z\rangle)\Big|_{\beta=\frac{\pi}{2}} = |\langle z|(-i)^n\sigma_x^{\otimes n}\exp(-i\gamma C)H^{\otimes n}|0\rangle^{\otimes n}|^2 \tag{73}$$

$$= |(-i)^n|^2|\langle z|\sigma_x^{\otimes n}\exp(-i\gamma C)H^{\otimes n}|0\rangle^{\otimes n}|^2 = |\langle z|\sigma_x^{\otimes n}\exp(-i\gamma C)H^{\otimes n}|0\rangle^{\otimes n}|^2. \tag{74}$$

However, we know that $\sigma_x^{\otimes n}$ is an anti-diagonal matrix with all 1s along the anti-diagonal:

$$\sigma_x^{\otimes n} = \begin{pmatrix} 0 & \cdots & 0 & 1 \\ 0 & \cdots & 1 & 0 \\ \vdots & \iddots & \vdots & \vdots \\ 1 & \cdots & 0 & 0 \end{pmatrix}. \tag{75}$$

Then (74) in matrix form is

$$= \frac{1}{2^n}\left|\begin{pmatrix} 0 & \cdots & 1_{z+1} & \cdots & 0 \end{pmatrix}\begin{pmatrix} 0 & \cdots & 0 & 1 \\ 0 & \cdots & 1 & 0 \\ \vdots & \iddots & \vdots & \vdots \\ 1 & \cdots & 0 & 0 \end{pmatrix}\right.$$
$$\left.\begin{pmatrix} e^{-i\gamma C(0)} & 0 & \dots & 0 \\ 0 & e^{-i\gamma C(1)} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & e^{-i\gamma C(2^n-1)} \end{pmatrix}\begin{pmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{pmatrix}\right|^2. \tag{76}$$

This is easily computed to be

$$P(|z\rangle)\Big|_{\beta=\frac{\pi}{2}} = \frac{1}{2^n}|\exp(-i\gamma C(2^n-1-z)|^2 = \frac{1}{2^n}. \tag{77}$$

We conclude that for all four cases

$$P(|z\rangle)\Big|_{\gamma=0} = P(|z\rangle)\Big|_{\gamma=2\pi} = P(|z\rangle)\Big|_{\beta=0} = P(|z\rangle)\Big|_{\beta=\frac{\pi}{2}} = \frac{1}{2^n} \tag{78}$$

This is not the case for $(\gamma, \beta) \in [0, \pi] \times [0, \pi]$. We conclude $(\gamma, \beta) \in [0, 2\pi] \times [0, \pi/2]$ is the preferable input domain.

2.3 Algorithm Implementation

Theoretically, there is no limit on the size and scope of implementable unitary gates on quantum computers. One could design a control mechanism (a magnetic pulse for example) which acts upon n qubits that, when measured, creates a spike at the desired answer. However, reality is not so simple. Physical limitations in hardware create the need for a simpler set of gates that are easily identified and controlled. These are dependent on the quantum computer being used, but they must include some sort of entangling gate (acting upon at least two qubits) and some set of single qubit gates. This set of gates is known as the set of "base gates".

2.3.1 Base gates and common gates

The IBM quantum computers run off the following set of five unitary gates:

$$U_3(\theta, \phi, \lambda) = \begin{pmatrix} \cos(\theta/2) & -e^{i\lambda}\sin(\theta/2) \\ e^{i\phi}\sin(\theta/2) & e^{i\lambda+i\phi}\cos(\theta/2) \end{pmatrix} \tag{79}$$

$$U_2(\phi,\lambda) = U_3(\pi/2,\phi,\lambda) = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & -e^{i\lambda} \\ e^{i\phi} & e^{i\lambda+i\phi} \end{pmatrix} \tag{80}$$

$$U_1(\lambda) = U_3(0,0,\lambda) = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\lambda} \end{pmatrix} \tag{81}$$

$$I = U_3(0,0,0) = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \tag{82}$$

$$CS = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \tag{83}$$

Of note, the identity gate (82) is not a physical gate, but rather refers to a brief halt in the physical implementation of the algorithm. It is generally used for bench marking and testing purposes, and is rarely (if ever) used in an actual algorithm. Also, while it would appear that U_1 and U_2 are simply special cases of the U_3 gate, they both in fact have separate implementations in the physical machines. As such, they are given their own section in the base gates.

From these base gates, we can make all of the operations used in quantum computing. Some of the simpler examples include

$$\text{Hadamard: } H = U_3(\pi/2,0,\pi) = U_2(0,\pi) = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \tag{84}$$

$$\text{Swap gate: } X = U_3(\pi,0,\pi) = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \tag{85}$$

which have circuit diagrams of

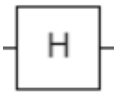


Figure 7: Single qubit Hadamard gate. It uses depend on the context

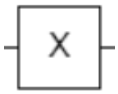
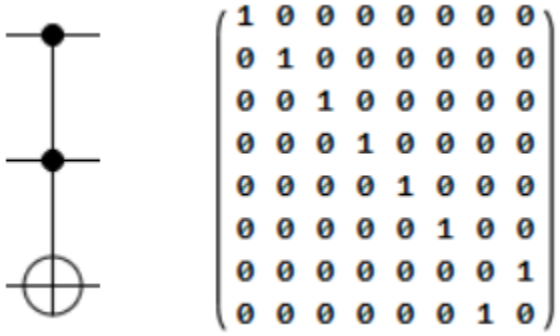


Figure 8: Single qubit swap gate. It flips bits from 1 to 0 and 0 to 1.

More complicated gates that will be used throughout this paper include the multi-control AND swap, multi-control OR swap gate, and controlled phase gate. The first two gates are multi-qubit gates while the last one is a 2-qubit gate. The first two gates are built out of multiple sets of smaller gates: tofolli gates and OR gates. A toffoli gate is comprised of 3-qubits: two control qubits and one target qubit. If both control qubits are 1, then the target qubit is flipped. It has a circuit diagram and matrix representation of



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

Figure 9: The toffoli gate. If both control qubits (black dots) are 1, then the target (crossed dot) is flipped.

Note that in the state $|z_2 z_1 z_0\rangle$, the z_0 (control) corresponds to the top qubit, z_1 (control) corresponds to the middle qubit, and z_3 (target) is the bottom qubit. The OR gate is also composed of two control qubits and one target qubit. If either control qubit is 1, then the target qubit is flipped. It looks like

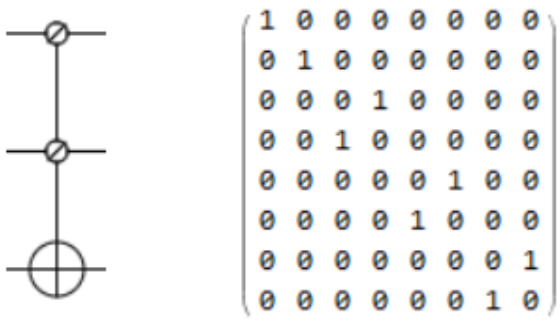


Figure 10: The OR gate. If either control qubits (diagonal lined dots) are 1, then the target (crossed dot) is flipped.

Now, the toffoli gate has an implementation in qiskit (see Appendix [1]) which allows the user to specify the control qubits and target qubit using a single command. This implementation takes 15 base gates, 6 CX's and 9 single qubit gates. However, no such option exists for the OR gate. It can be implemented as

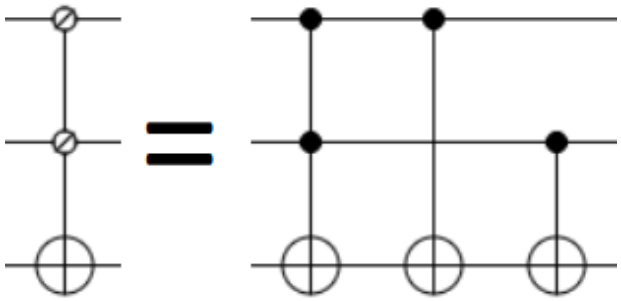


Figure 11: The OR gate is equivalent to a Toffoli gate and two CX gates which hit the target qubit

These gates can be extended to an arbitrary number of inputs in a polynomial number of resources. Perhaps the most naive way of doing this is simply chaining multiple of both types of gates using ancillary qubits set to $|0\rangle$. For example, to make 4-controlled AND gates and OR gates, simply perform the following sequence

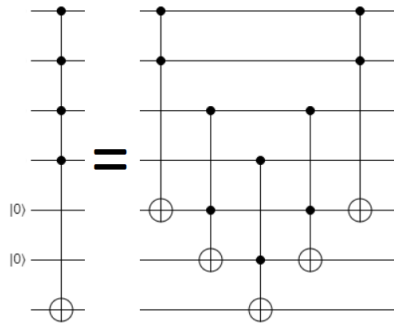


Figure 12: The large AND gate of n inputs consists of $2n-3$ Toffoli gates as well as $n-2$ ancillary qubits.

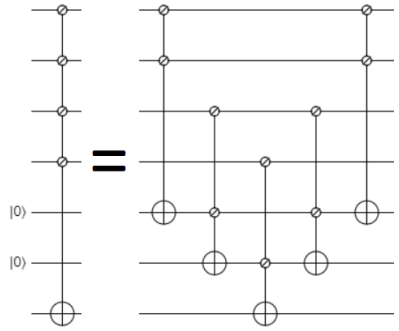


Figure 13: The large OR gate of n inputs consists of $2n-3$ OR gates as well as $n-2$ ancillary qubits.

In general, to implement an n-controlled AND or OR gate it takes $2n-3$ Toffoli/OR gates as well as $n-2$ ancillary qubits. Of course, this is a simple way of doing this and other ways exist depending on the number of ancillary qubits available as well as computing resources one is willing to use.

The controlled phase gate is a two qubit gate (a control qubit and target qubit) which adds a desired phase to the target qubit if the control qubit is $|1\rangle$. It looks like

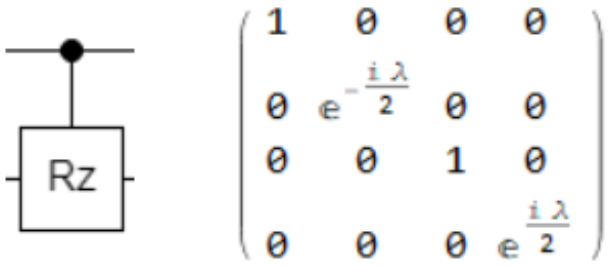


Figure 14: A controlled phase gate which adds a phase of λ to the bottom qubit if the top qubit is $|1\rangle$.

In Fig. 14, a phase of λ is added to the bottom qubit if the top qubit is in the excited state. This gate is implemented in qiskit using 6 gates, 2 CX's and 4 single qubit gates. Additionally, this gate can be expanded to include multiple control qubits. A simple way of doing this is by using an ancillary qubit set to $|0\rangle$ as well as two large AND gates. For example, to create a 3 controlled phase gate is is enough to implement

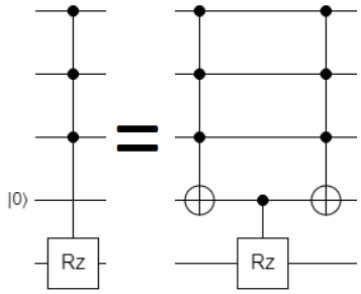


Figure 15: A multi-qubit controlled phase gate with 3 control qubits. This implementation uses 1 ancillary qubit (in addition to any that the large AND gates might use) and two 3 controlled AND gates.

Of course, this gate can be simplified slightly as many of the toffoli gates in the multi-controlled AND gates cancel. If we use the naive implementation shown in Fig. 12, then the following gates are all equivalent

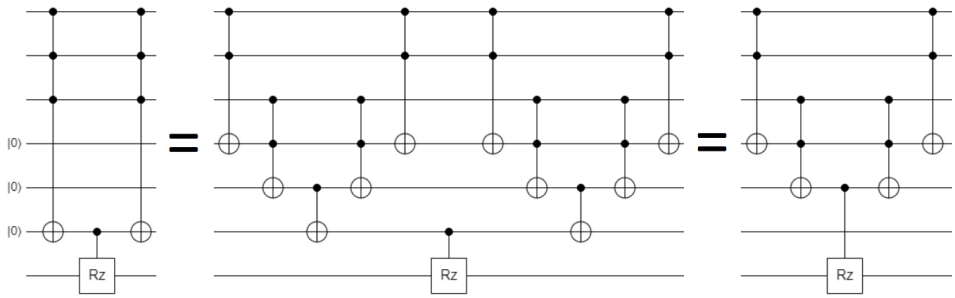


Figure 16: Using the naive implementation from Fig. 12, many toffoli gates cancel to give a simplified circuit. For a 3 controlled phase change, 4 toffoli gates cancel out which saves 24 CX gates by itself. In general, this setup takes 1 controlled R gate, $2(n-1)$ toffoli gates, and $n-1$ ancillary qubits.

Using this implementation, many of the toffoli gates cancel which give rise to a simplified multi-controlled phase gate. In qiskit, an arbitrary phase gate with n controls will take $30n-24$ gates and $n-1$ ancillary qubits. These gates split into $12n-10$ CX gates and $18n-14$ single qubit gates. As before, this implies that these circuits can be implemented in a linear number of gates, but the constant of 12 in the CX term is cause for concern. This idea is explored more carefully in sections 4 and 5.

The final gate that is applicable to this work is the inverted control gate. In essence, the inverted control gate takes any of the multi-qubit gates previously discussed and inverts the control such that $|0\rangle$ is te desired input instead of $|1\rangle$. This is easily done by surround the input qubit with swap gates, which has the desired effect.

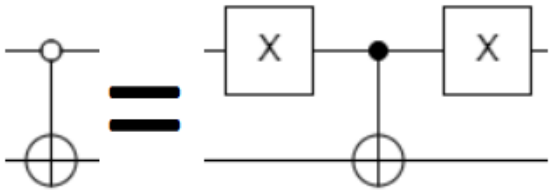


Figure 17: To invert the controls for any multi-qubit gate, simply surround the control with two swap gates (see equation (85) for details.

To denote the swapped AND control, a blank circle is used (see Fig. 17). The swapped OR

control is not used in this paper and so is not given a special symbol.

2.3.2 Sufficient condition for polynomial implementation of Clause

Theoretically, it is possible to implement any set of clauses into a QAOA algorithm. In actuality, practicality is reserved to those that can be implemented in polynomial time. For example, consider the set of clauses

$$C_i(z) = \begin{cases} 1 & \text{if } z = i \\ 0 & \text{if } z \neq i \end{cases} \tag{86}$$

Since this set of clauses has 2^n elements, it would take at least 2^n gates to implement in QAOA. That is, it is not practical for real world applications to consider these clauses. On the flip side, there does seem to be some connection between a large number of independent clauses and exact algorithms to find the optimal z, but that is not investigated in this paper. As is, we will need to define three properties of clauses. Then, if we have a sufficiently small number of clauses (ie, a polynomial number in n) whose properties are bounded, then the cost function can be implemented in polynomial time in QAOA. Note that the following is a proof of sufficiency, not necessity.

The three properties are:

$$P_0(C_i) = |\{z : C_i(z) = 1\}| \tag{87}$$

This property simply asks how many z activate the clause. If it is a small number, then we can implement it directly. The next property is the inverse

$$P_1(C_i) = |\{z : C_i(z) = 0\}| \tag{88}$$

Again, if there are only a "few" z such that $C_i(z) = 0$, then we can implement them easily in the circuit. The next two properties deal with a more abstract notion. Consider a scenario where someone gives you a clause $C_i(z) = 1$ (or $C_i(z) = 0$) for some string z. How are you to check their claim? Obviously, you can input the string into the clause and see if it outputs 1 (or 0). However, you may be able to get away with inputting a subset of the string to confirm it is 1 (or 0). For example, if

$$C_i(z) = \begin{cases} 1 & \text{if } z_0 = 1 \\ 0 & \text{if } z_0 = 0 \end{cases} \tag{89}$$

then you could confirm the $C_i(z) = 1$ by simply checking the z_0 digit. We define this abstract idea more formally below as the certificate number of a clause. Of course, it can sometimes be difficult to check this number. For example, when

$$C_i(z) = \begin{cases} 1 & \text{if } \sum_{i=0}^{2^n-1} z_i < \frac{3n}{4} \\ 0 & \text{otherwise} \end{cases} \tag{90}$$

then the certificate number is $\lceil \frac{n}{4} \rceil$ as you can check this many digits of z, see that they are 0, and you would confirm that the clause outputs 1. With these examples out of the way, here is the formal definition for the last properties:

- Define $A(C_i)$ as the set of z such that $C_i(z) = 1$
- Define $B(C_i)$ as the set of z such that $C_i(z) = 0$
- Define $PA(C_i, z)$ as the certificate number of $C_i(z) = 1$ for $z \in A(C_i)$
- Define $PB(C_i, z)$ as the certificate number of $C_i(z) = 0$ for $z \in B(C_i)$

$$\mathrm{P}_2(C_i) = \min\left(\max\left\{PA(C_i, z) : z \in A(C_i)\right\}, \max\left\{PB(C_i, z) : z \in B(C_i)\right\}\right) \quad (91)$$

Using these three properties, we can finally lay out the sufficiency conditions for a cost function to be implemented in QAOA. They are

- For $C(z) = \sum_{i=1}^{m-1} C_i(z)$, $m = O(P(m^a))$ for some $a > 0$. That is, the number of clauses grows polynomially.
- $\max\{\min\{P_0(C_i), P_1(C_i), P_2(C_i)\} : 0 \leq i \leq m-1\}$ is bounded above by some constant $M > 0$ as n gets large. That is, as n gets large, every clause will have a bound on at least one of the above properties.

The first condition is self-evident: if the number of clauses does not grow at most polynomially, then clearly the algorithm can not be implemented in polynomial time. For the rest of this proof, we shall only consider a single clause; that which creates the upper bound on condition two. That is i such that

$$\min\{P_0(C_i), P_1(C_i), P_2(C_i)\} = \max\{\min\{P_0(C_i), P_1(C_i), P_2(C_i)\} : 0 \leq i \leq m-1\} \quad (92)$$

For the sake of brevity, we shall call this clause $C_i(z) = T(z)$ (we forgo calling it $C(z)$ as this already has another meaning throughout this paper). Thus, we must show that

$$\min\{P_0(T), P_1(T), P_2(T)\} < M \quad (93)$$

for some $M > 0$ is sufficient to implement $T(z)$ in QAOA in a polynomial number of gates. To this end, suppose that the above condition holds. We shall consider the three cases separately. First, suppose that

$$P_0(T) = \min\{P_0(C_i), P_1(C_i), P_2(C_i)\} < M \tag{94}$$

Then there are at most M strings z such that $T(z) = 1$. Thus, we can rewrite

$$T(z) = \sum_{i=0}^{M-1} T_i(z) \tag{95}$$

where

$$T_i(z) = \begin{cases} 1 & \text{if } z \text{ is the } i\text{th element of } \{z : T(z) = 1\} \\ 0 & \text{otherwise} \end{cases} \tag{96}$$

Since there are at most M subclauses of $T(z)$, are problem is simplified to showing that there exists a way to implement a specific z in a polynomial number of gates. Note that we must implement z in a polynomial number of gates regardless of how long (how many digits) it is. To this end, let z be a binary string

$$z = z_0 z_1 \dots z_{n-1} \tag{97}$$

with n digits where $z_i \in \{0, 1\}$. To start, define

$$U = \{i : z_i = 1\} \text{ and } D = \{i : z_i = 0\} \tag{98}$$

in z. The first step to implementing the clause is to perform a n-controlled phase gate using the n main qubits as inputs. If the setup from Fig. 12 is utilized, then $2(n-1)$ toffoli gates and $n-1$ ancilla qubits are used to perform the controlled R phase change of $-\gamma$. Since toffoli gates and controlled R gates can be implemented in polynomial time, this whole gate can be implemented in polynomial time. See [18] and [19] for more information and proofs of the above statistics.

The next step is to hit every qubit in the circuit with σ_x^q where $q = 0$ if $z_q = 1$ while $q = 1$ if $z_q = 0$. It is instructive to present an example: suppose $z = 010$. Then the circuit would look like

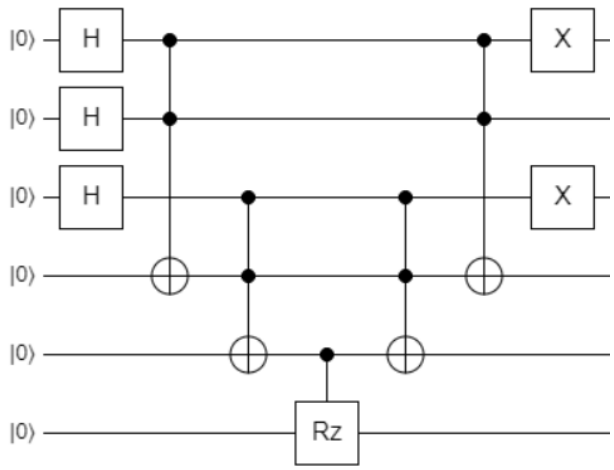


Figure 18: The full circuit to add a phase change for $z = 010$

If we denote the initial state as $|\psi_0\rangle$, the state after the Hadamards as $|\psi_1\rangle$, the state after the n-controlled phase change as $|\psi_2\rangle$, and the final state as $|\psi_3\rangle$, then we get

$$|\psi_1\rangle = H^{\otimes 3}|\psi_0\rangle = \frac{1}{2\sqrt{2}}\left(|000\rangle + |001\rangle + |010\rangle + \cdots + |111\rangle\right) \tag{99}$$

$$|\psi_2\rangle = T_n|\psi_1\rangle = \frac{1}{2\sqrt{2}}\left(|000\rangle + |001\rangle + |010\rangle + \cdots + e^{-i\gamma}|111\rangle\right) \tag{100}$$

$$|\psi_3\rangle = (\sigma_x \otimes I \otimes \sigma_x)|\psi_2\rangle = \frac{1}{2\sqrt{2}}\left(|000\rangle + |001\rangle + e^{-i\gamma}|010\rangle + \cdots + |111\rangle\right) \tag{101}$$

As we had hoped, there is a phase change on state $|010\rangle$ and this state alone. Now, let us count the number of gates we have used. For an n digit string, we have at most

- Toffoli gates: $2(n-1)$
- Hadamard gates: n
- Swap gates: n
- Controlled R gates: 1
- Ancillary qubits n

Thus, we can implement a single binary string $z = z_0 z_1 \dots z_{n-1}$ in $O(n)$ gates and n ancillary qubits. Since these are both polynomial, we are done. If we are implementing QAOA using an IBM Q system, then toffoli gates are implemented with 15 base gates and controlled R gates are implemented with 4 base gates (see Appendix 1). Thus, the total number of gates is $36n - 26$ gates. Although this is linear in n it is still not ideal as the coefficient in front of the leading term is 36.

The next case is similar to the first case except that here

$$P_1(T) = \min\{P_0(C_i), P_1(C_i), P_2(C_i)\} < M \tag{102}$$

However, we can proceed in the same manner as before. That is, if there are at most M strings z such that $T(z) = 0$. Then as before, we can split $T(z)$ into M subclauses, each of which output 0 if z is input, and 1 otherwise. In fact, the algorithm for implementing this subclause is exactly the same as the previous case, the only difference being we rotate by γ instead of $-\gamma$. For example, if we implement this algorithm with $z = 010$ being the only string which outputs zero, then the final state would be

$$|\psi\rangle = \frac{1}{2\sqrt{2}} \left(|000\rangle + |001\rangle + e^{i\gamma}|010\rangle + \dots + |111\rangle \right) \tag{103}$$

However, we pull out an overall global phase of γ to get

$$|\psi_{\text{Final}}\rangle = \frac{e^{i\gamma}}{2\sqrt{2}}\left(e^{-i\gamma}|000\rangle + e^{-i\gamma}|001\rangle + |010\rangle + \cdots + e^{-i\gamma}|111\rangle\right) \tag{104}$$

Since this global phase factor is irrelevant once the system is measured, we conclude we have succeeded in implementing a subclause which is 0 when $z = 010$ and 1 otherwise.

The final case is also the most difficult to prove. Suppose that

$$P_2(T) = \min\{P_0(C_i), P_1(C_i), P_2(C_i)\} < M \tag{105}$$

where $P_2(T)$ is defined as by equation (91). Now, suppose that

$$P_2(T) = \max\{PA(T, z) : z \in A(T)\} \leq M \tag{106}$$

That is, for all strings z such that $T(z) = 1$, we have at most M digits of z to check to confirm that $T(z) = 1$. As we are showing that it is possible to implement this clause in QAOA, we are able to assume perfect knowledge of which substrings (at length most M) correspond to outputs of 1. In actuality, this could be difficult to recover, but is not barrier to the overall algorithm. Let us count the ways we can select M substrings from $z = z_0 z_1 \ldots z_{n-1}$. Clearly, there are n choose M, or $\binom{n}{M}$ ways to pick from $\{z_0, z_1, \ldots, z_{n-1}\}$ to create a substring. The algorithm then splits $T(z)$ up into $\binom{n}{M}$ subclauses and implements each of them using the method from the case P_0 above. This can be implemented in a polynomial number of gates as

$$\binom{n}{M} = \frac{n!}{M!(n-m)!} = \frac{n(n-1)(n-2)\cdots(n-M+1)}{M!} = O(n^M) \tag{107}$$

For a non-trivial example of this behavior, see the DSP (section 4).

We finish this proof with the case when

$$P_2(T) = \max\{PB(T, z) : z \in B(T)\} \leq M \tag{108}$$

This implies we have to check at most M digits of $z \in B(T)$ to confirm that $T(z) = 0$. The logic flows in the same manner as before: there are $\binom{n}{M}$ subclauses that must be implemented, where the subclause in this case is the same as in $P_1(T)$. Again, the resources required go as $O(n^M)$.

3 Max Cut Problem (MCP)

The MCP is a well known NP-complete question regarding graph labeling [20]. Take any graph and color each node either red or white. Then, an edge is worth a point if it connects two nodes with different colors.

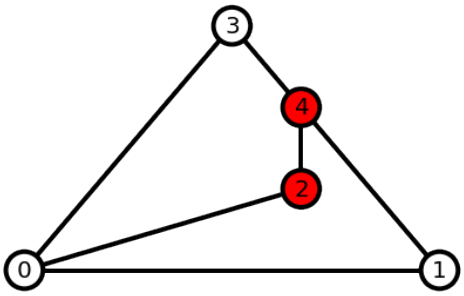


Figure 19: Coloring nodes 2 and 4 red results in a score of three

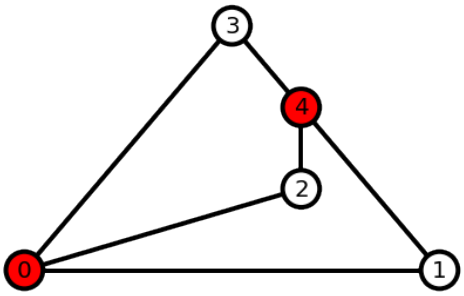


Figure 20: Coloring nodes 0 and 4 red results in a score of six

For example, in Figure 19 coloring nodes 2 and 4 results in a score of three as edges $(0, 2)$, $(1, 4)$, and $(3, 4)$ connect different colored nodes. However, in Figure 20 nodes 0 and 4 are colored, which leads to a score of six since every edge connects two different colored nodes.

3.1 Cost Function Definition

The MCP create very simple clauses in the cost function. There are exactly as many clauses as there are edges, and the clause corresponding to the edge between node i and node j is defined as

$$C_{(i,j)}(z) = \begin{cases} 1 & \text{if } z_i \neq z_j \\ 0 & \text{if } z_i = z_j \end{cases} \tag{109}$$

It is useful to examine these definitions through an example. Consider the following graph of four nodes:

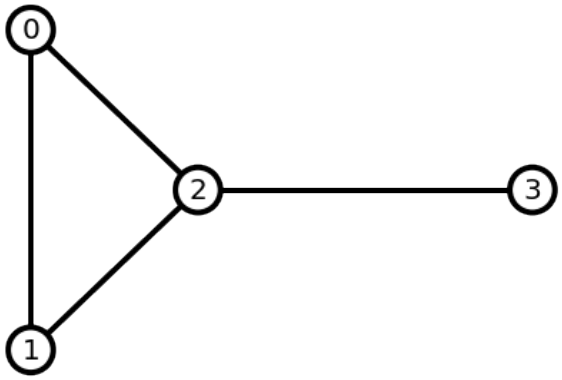


Figure 21: The standard 4-node example graph used throughout this paper

The nodes have been labeled 0 through 3 in accordance with computer science notation. Writing out all possible z's, the following table is constructed:

Table 1: The full set of clauses for the standard example implemented for the MCP.

z	$z_0z_1z_2z_3$	$(0,1)$	$(0,2)$	$(1,2)$	$(2,3)$	$C(z)$
0	0000	0	0	0	0	0
1	0001	0	0	0	1	1
2	0010	0	1	1	1	3
3	0011	0	1	1	0	2
4	0100	1	0	1	0	2
5	0101	1	0	1	1	3
6	0110	1	1	0	1	3
7	0111	1	1	0	0	2
8	1000	1	1	0	0	2
9	1001	1	1	0	1	3
10	1010	1	0	1	1	3
11	1011	1	0	1	0	2
12	1100	0	1	1	0	2
13	1101	0	1	1	1	3
14	1110	0	0	0	1	1
15	1111	0	0	0	0	0

From Table 1, it is obvious that the strings $z = 0010, 0101, 0110, 1001, 1010, 1101$ maximize the cost function. One of these strings corresponds to

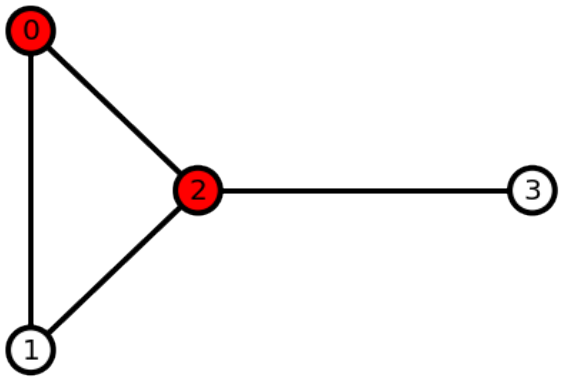


Figure 22: $z = 1010$ corresponding to coloring node 0 and node 2. This gives a score $C(1010) = 3$.

3.2 Algorithm Implementation and Analysis

As was previously discussed in equation (44), running QAOA (at $p = 1$) on n qubits corresponds to implementing the following unitary matrices

$$U_B(\beta)U_C(\gamma)H^{\otimes n}|0\rangle^{\otimes n}. \tag{110}$$

where γ and β are parameters of the algorithm. In this section, we will show each step on the standard example. This will include both a circuit diagram as well as the ket analysis.

Step 1: Clearly, the first step for actual implementation is to hit each z-qubit with a Hadamard gate (84):

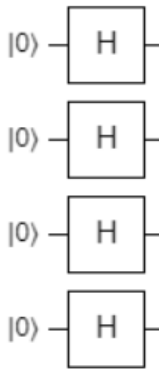


Figure 23: The first step of implementing the standard example. Additionally, the qubit on top is qubit 0, the one below it is qubit 1, and the pattern continues to qubit 4 on the bottom.

In the standard example, this is equivalent to

$$|\psi_0\rangle = |0_0 0_1 0_2 0_3\rangle \tag{111}$$

Note that the subscripts denote which position the qubit is in the circuit above. If no subscript is provided, it is assumed that 0 starts on the left and goes up as the digits progress. That is,

$|0100\rangle = |0_0 1_1 0_2 0_3\rangle$. Then

$$|\psi_1\rangle = H^{\otimes 4}|\psi_0\rangle = \frac{1}{4}\left(|0000\rangle + |0001\rangle + |0010\rangle + \cdots + |1110\rangle + |1111\rangle\right) \tag{112}$$

Step 2: The second step is run the following circuit on all pairs of qubits corresponding to edges:

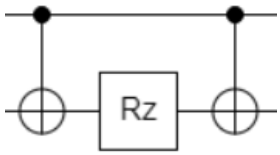


Figure 24: Each edge of the original graph corresponds to a gate of this nature. The top line corresponds to qubit i and the bottom line corresponds to qubit j, where i and j are nodes connected by an edge in the graph

where $R_z = U_1(-\gamma)$ (see equation (6)). If we call the upper qubit 0 and the bottom qubit 1, then the matrix form of this gate is

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \left[\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}_0 \otimes \begin{pmatrix} 1 & 0 \\ 0 & e^{-i\gamma} \end{pmatrix}_1 \right] \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \tag{113}$$

$$= \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & e^{-i\gamma} & 0 & 0 \\ 0 & 0 & e^{-i\gamma} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{114}$$

where the subscripts on the matrices simply refers to which qubit they act upon. See [21] for more details. Of course, one might as well ask whether the following gate might produce a different result.

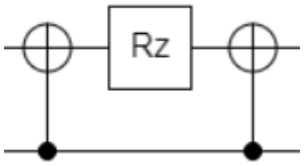


Figure 25: The same gate as Fig. 24 but flipped upside down

That is, does flipping the controls of the gate alter the final state. The answer is no as the matrices that correspond to this ”flipped” gate are

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \left[\begin{pmatrix} 1 & 0 \\ 0 & e^{-i\gamma} \end{pmatrix}_0 \otimes \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}_1 \right] \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \tag{115}$$

$$= \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & e^{-i\gamma} & 0 & 0 \\ 0 & 0 & e^{-i\gamma} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{116}$$

These are the exact same as the original gate we described in Fig. 24. If we consider the standard example, then the circuit looks like

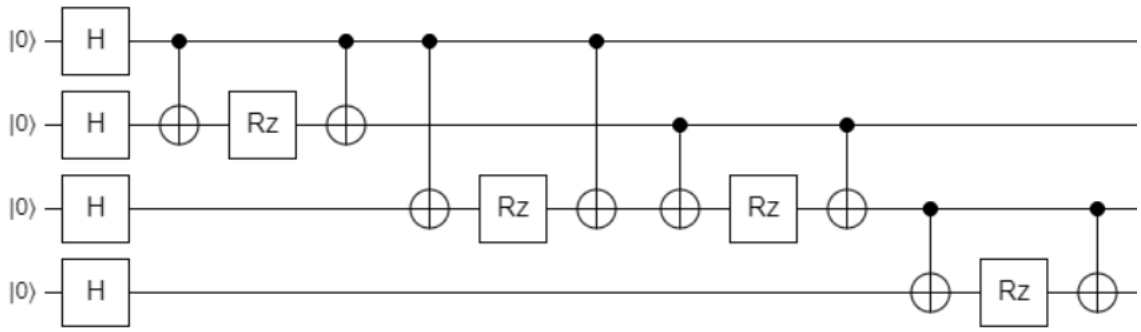


Figure 26: The second step of implementing the standard example. Each edge in Fig. 21 corresponds to a pair of controlled NOT gates and $U_1(-\gamma)$ gates

To demonstrate the effect of this gate, it is simplest to consider the matrix analysis on a two-qubit superposition. That is

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & e^{-i\gamma} & 0 & 0 \\ 0 & 0 & e^{-i\gamma} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \frac{1}{2} \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} = \frac{1}{2} \begin{pmatrix} 1 \\ e^{-i\gamma} \\ e^{-i\gamma} \\ 1 \end{pmatrix} = \frac{1}{2} \left(|00\rangle + e^{-i\gamma}|01\rangle + e^{-i\gamma}|10\rangle + |11\rangle \right) \tag{117}$$

From this, it is obvious that this gate adds a phase change of $-\gamma$ if the state of the two qubits are different. If we consider the standard example, then the ket becomes

$$|\psi_2\rangle = \frac{1}{4} \left(|0000\rangle + e^{-i\gamma}|0001\rangle + e^{-3i\gamma}|0010\rangle + \cdots + e^{-i\gamma}|1110\rangle + |1111\rangle \right) \tag{118}$$

Step 3: The third step is to implement the $-\beta$ rotation gates described in equation (48). These are single qubit U_3 gates described by the matrix

$$U_B(\beta) = U_3\left(2\beta, \frac{3\pi}{2}, \frac{\pi}{2}\right) = \begin{pmatrix} \cos(\beta) & -i\sin(\beta) \\ -i\sin(\beta) & \cos(\beta) \end{pmatrix} \tag{119}$$

Hitting each qubit with these gates gives

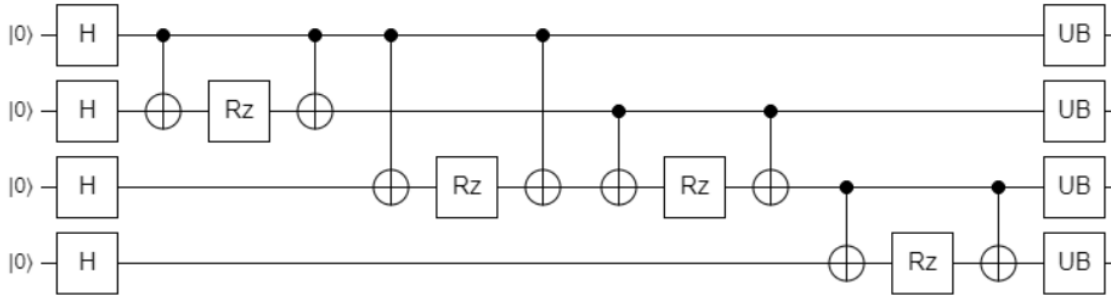


Figure 27: The third step of implementing the standard example. Each qubit is hit with a $-\beta$ rotation gate. This gate is a described by equation (48)

If we wanted to run QAOA for $p >$, then we would return to step $2-3$ for $p-1$ times, each time with a different γ and β angle. Of course, this is useful for actual problem implementation, but at this point does not provide further insight into how the algorithm performs. It is less useful to describe how the overall ket changes in the standard example. This is because the purpose of the β rotation is to reduce the order created be the previous steps. In fact, this sort of reduction is not even necessary. There are ongoing studies into whether other types of rotation gates might produce further results. See [22] and [23] for further details.

Step 4: The final step is to measure all the qubits and record the outputs. Repeat the whole algorithm enough times to get an accurate description of the underlying probability distribution function (PDF) which will be some function of n. With this step, we can describe the entire circuit for the standard example. It is

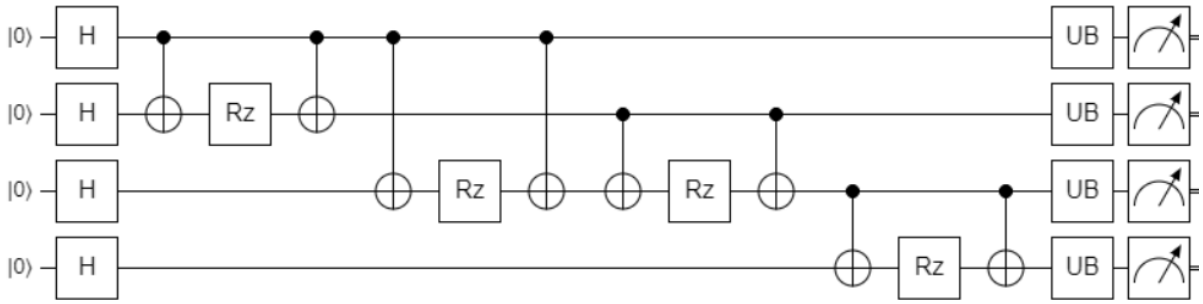


Figure 28: The final step of implementing the standard example. Measure each qubit and record the output. Repeat the whole algorithm enough times to get an accurate description of the PDF of the results.

For a simple example such as this, we can explicitly compute the PDF at any γ and β we like. For example, at $\beta = \gamma = \frac{2\pi}{5}$ we get

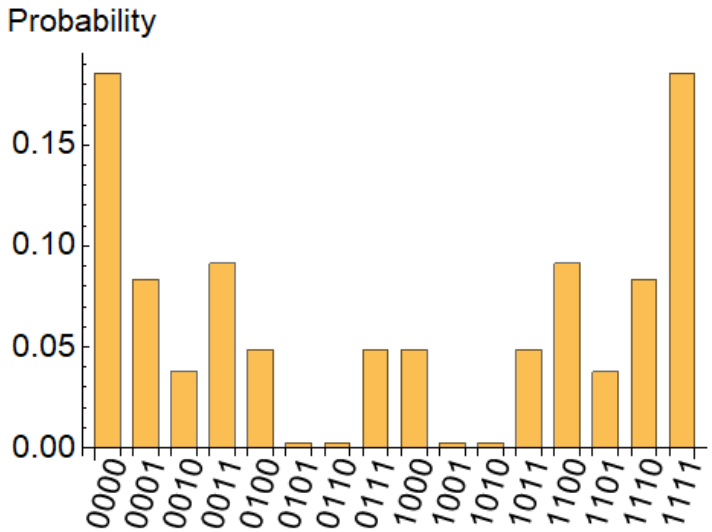


Figure 29: The PDF generated when setting $\beta = \gamma = \frac{2\pi}{5}$ for $p = 1$. Note that the symmetry arises since swapping node colors does not effect whether or not an edge is connected to two different colored nodes.

One interesting aspect of this graph is that it is symmetric about the center. This is an artifact of the MCP and not QAOA in general. It arises as swapping colors (and hence 0s and 1s in z does not change whether an edge is connected to two different colored nodes.

There are multiple ways to utilize QAOA, but perhaps the most basic approach simple runs the algorithm for a grid of γ and β angles. Recalling that equation (44) defines

$$|\psi\rangle = U_B(\beta)U_C(\gamma)H^{\otimes n}|0\rangle^{\otimes n} \tag{120}$$

we can consider the expectation value of the cost function evaluated at (γ, β) to be

$$F(\gamma, \beta) = \langle\psi|C|\psi\rangle \tag{121}$$

For $p = 1$, this is a function in two variables. For the standard example, the plot over γ and β is

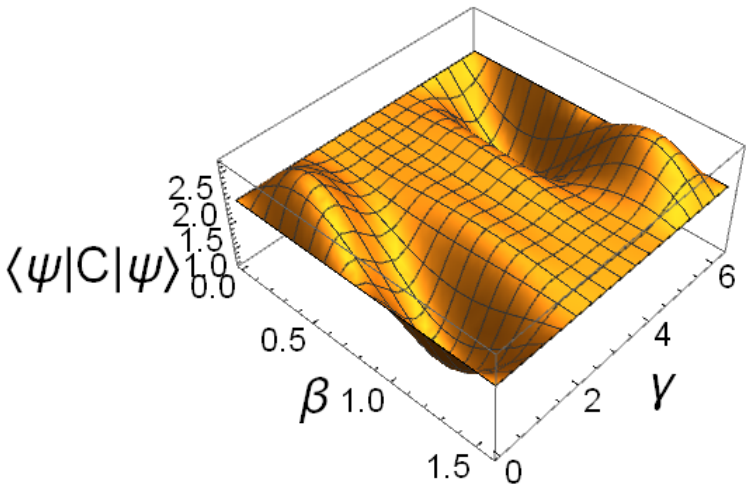


Figure 30: The expectation value of the cost function evaluated at γ and β for $p = 1$ while implementing the standard example.

Fig. 30 can be created using a quantum computer and classical computer in conjunction with each other. The algorithm is

- **Step 1:** Subdivide $(\gamma, \beta) \to [0, 2\pi] \times [0, \pi/2]$ into as fine a grid as desired.
- **Step 2:** At each point, run the quantum computer, getting a string z as output.

- **Step 3:** Evalute $C(z)$ to get a cost function output.
- **Step 4:** Repeat steps $2-3$ until the expectation value of C at (γ, β) is established.
- **Step 5:** Repeat steps $2-4$ with all points in the grid.

Once this plot is created, one can compute the gradient and find a local maximization point. Here, the expectation value of $C(z)$ is higher than the surrounding angles. Run QAOA for this point (or use the statistics already created for the plot) and compute $C(z)$ for the measured outputs. As this is a local maximum, it is expected that $C(z)$ generated from these angles is greater than

$$\sum_{i=0}^{2^n-1} \frac{C(i)}{2^n} \tag{122}$$

(the average cost function output over all input strings). While this has been shown to be true for $p = 1$, it is in fact an open problem whether this holds for $p > 1$ although numerical examples would seem to suggest this.

3.3 Algorithm Analysis

It is easy to bound the number of gates required to run max cut on an arbitrary graph. Consider a graph with n vertices which has at most $\binom{n}{2} = \frac{n(n-1)}{2}$ edges. Each step of the implementation corresponds to a certain number of gates. These correspond in turn to a certain number of base gates in qiskit. These gates are

- The first step is to hit each qubit with a hadamard gate, which corresponds to n single qubit gates.
- The second step is to implement the gate shown in Fig. 25. These correspond to 2 CX gates and 1 single qubit gate. However, these numbers correspond to the number of edges. As such, there are at most $n(n-1) = n^2 - n$ CX's and $n^2 - n$ single qubit gates.

- The final step before measurement is the β rotation gates. This corresponds to n single qubit gates.

Adding these respective gates together gives the final numbers

- There are at most $n^2 - n$ CX's used in the algorithm.
- There are at most $n^2 + n$ single qubit gates used in the algorithm.
- The quantum computer needs n qubits.

Overall, the algorithm takes at most $O(n^2)$ gates to implement. However, as most graphs will not have $\frac{n(n-1)}{2}$ edges (only complete graphs have this) a more illuminating bound is given by $O(e)$ CX's and single qubit gates. It is better to look at the number of edges in a given graph and from there get bounds than look at the number of vertices to derive bounds.

3.4 Running Max Cut

The main result presented here is an in-depth dive into how max cut actually runs on QAOA. However, in order to accurately measure how well the quantum computer runs, we need to define a metric which can discriminate between "good" and "bad" runs. Suppose an algorithm is run on an actual quantum computer until a reasonable PDF is established. Call this PDF T (for 'test'). In the background, there is another PDF that corresponds to what the quantum computer would output in ideal world, one free of noise with perfect gates. Call this PDF E (for 'exact'). We define the similarity between these two distributions as one minus the total variational distance. That is

$$D(E,T) = 1 - \frac{1}{2}\sum |E_i - T_i| \tag{123}$$

The sum in this case is taken over all elements of the respective PDFs. Basically, this metric is just taking the absolute value between the actual histogram produced by the quantum computer and the expected histogram. It is always in $[0, 1]$, with numbers closer to 1 signifying better runs. Consider the example PDF given in Fig. 31.

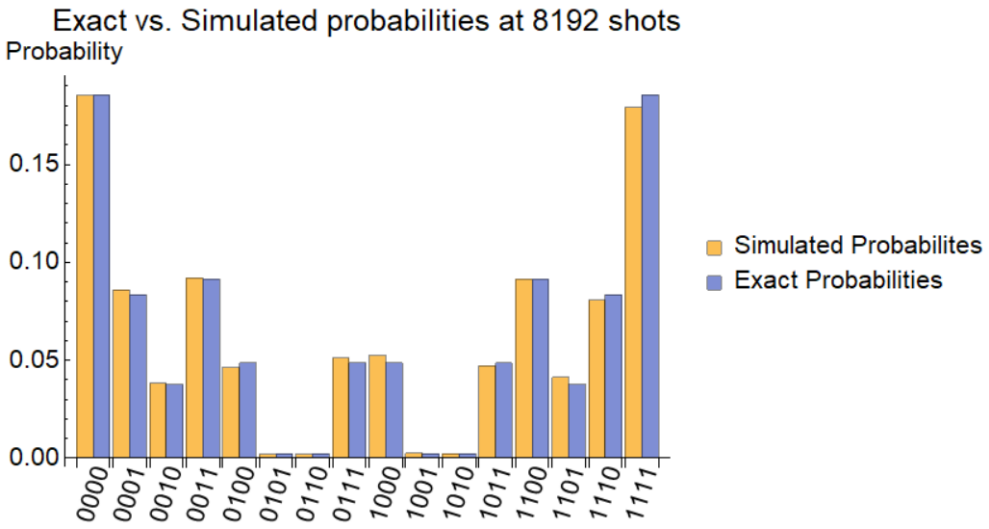


Figure 31: The exact probabilities versus the simulated probabilities. As expected, they agree heavily with each other.

In Fig. 31, the MCP on the standard example (Fig. 21) was ran in QAOA at $\gamma = \beta = 2\pi/5$. For these probability distributions, we get that a similarity of 0.986. However, context is needed in order to describe whether or not this is actually a good score (hint: it is).

Let us now put some perspective on$D(E, T)$. Consider two random PDFs, E and T of n entries. A random PDF of is generated by the following algorithm

- Set $E_n^{'} = \{e_1, e_2, ..., e_n, \}$ where e_i is picked uniformly from $[0, 1]$
- Define $\chi = \sum_{i=1}^{n} e_i$
- Set $E_n = \frac{E_n^{'}}{\chi} = \left\{ \frac{e_1}{\chi}, \frac{e_2}{\chi}, ..., \frac{e_n}{\chi} \right\}$

(and let T_n be defined in a similar manner). Now, define S as the distribution of $D(E,T)$ for random PDFs E and T. For example, for $2^3 = 8$ entries (corresponding to 3 qubits) the distribution looks like

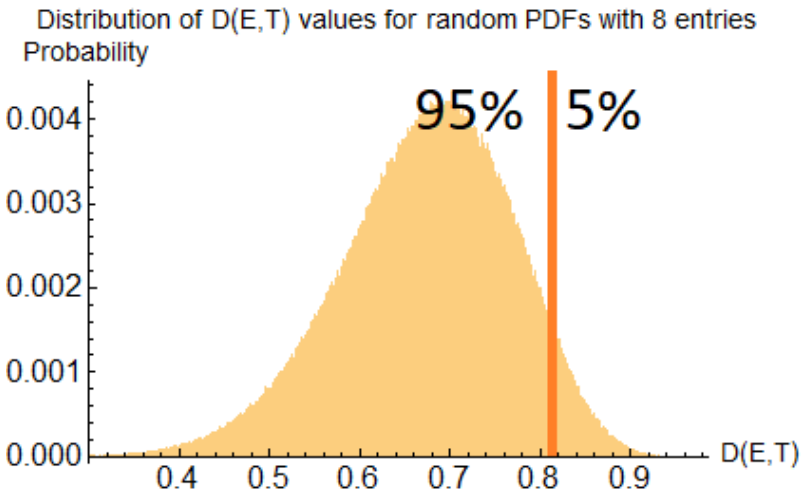


Figure 32: The distribution of $D(E,T)$ for random PDFs with $2^3 = 8$ entries. The line denotes the 95th percentile mark and defines an acceptable run: if $D(E,T) > 0.817$ then it is in the top 5% of runs and is therefore a good run. If it is less than this, then it is a bad run.

The orange line corresponds to the 95th percentile mark, or the point where the area under the curve (integrating from left to right) hits $.95$. This mark is chosen to signify a good run. That is, if $D(E,T) > 0.817$ (in the case of 8 entries), then E and T are close to being the same PDF. If $D(E,T) < 0.817$, then E and T are not close to being the same PDF. Of course, it could be argued that the fault tolerance is too high, and that many correlated distributions could be thrown out in this manner. Unfortunately, something that describes if PDFs are correlated or not will always require some sort of choice, and the 95th percentile was chosen for this delimiting point. A table with these values as well as the values for the 90th percentile are presented below:

Table 2: Cutoff points at the 90th and 95th percentile for $2-10$ qubits.

Qubits	90th percentile	95th percentile
2	0.850	0.884
3	0.790	0.817
4	0.752	0.773
5	0.726	0.742
6	0.708	0.720
7	0.696	0.704
8	0.687	0.693
9	0.680	0.685
10	0.676	0.679

Obviously, the values for the 90th percentile are lower than the values for the 95th percentile. However, they both converge towards $2/3$, and for arrays with a large number of qubits it would be beneficial to provide an estimate on how these cutoff points behave. We say this as these values were computed numerically and would not scale to large number of qubits. With this table, we can finally say that the PDFs in Fig. 31 are correlated to each other as $0.986 > 0.773$ (0.773 is the cutoff point for 4 qubits).

This experiment consisted of running the max cut algorithm defined above on IBM's quantum computers for a variety of graphs. First, a graph was chosen for which to implement QAOA. Next, the input space was defined to be

$$(\gamma, \beta) = 2\pi\{0, .1, .2, \cdots .9, 1\} \times \pi/2\{0, .1, .2, \cdots .9, 1\} \tag{124}$$

for a total of 121 combinations of input angles. Next, for each pair of angles (γ, β), the algorithm was ran $2^8 = 8192$ times on the IBM Poughkeepsie machine at optimization level 3. Once the results had been collected, the same algorithm was ran 8192 times on the IBM simulator. $D_{(\gamma,\beta)}(A, E)$ is calculated for the particular set of angles (γ, β). Then $D(E, T)$ is defined as

$$D(E,T) = \frac{1}{121} \sum_{(\gamma,\beta)} D_{(\gamma,\beta)}(E,T) \tag{125}$$

This final score determines how well QAOA performed for that graph. If this score puts the correlation between the actual data and the simulated data in the 95th percentile, then QAOA is considered to have succeeded on that graph.

In total, 26 different graphs were considered in this experiment. All graphs of 5 edges or less were checked as well as four graphs with 6 edges. These graphs are detailed in Appendix 2. The experimental data is produced in Fig. 33.

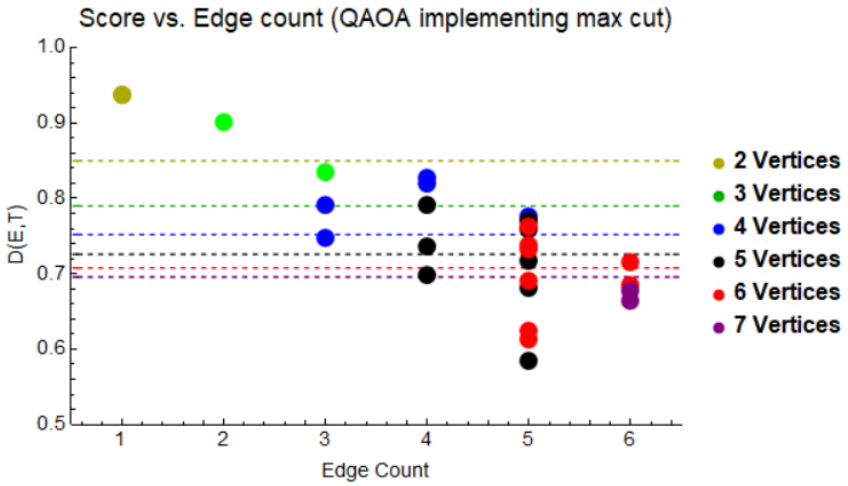


Figure 33: $D(E,T)$ for all graphs of 5 edges or less and four graphs with 6 edges. The colors of the dots correspond to the same colored lines. That is, if a dot is above the line of the same color, then QAOA succeeded for that graph. Otherwise, we declare that the graph failed to be implemented well on the quantum computer.

There are several things of interest in this graph. First, each dot corresponds to the line of the same color. If the dot is above that line, then the run succeeded and the correlation between the actual data and the simulated data is high. Otherwise there low to no correlation between the actual data and simulated data and the quantum computer failed to implement QAOA well. Additionally, there seems to be no obvious correlation between the number of vertices and the overall

score. That is, the edge count is a much better predictor than vertex number. This is somewhat surprising, as vertex number corresponds directly to qubits while edge count only corresponds to algorithm implementation. Overall, it would seem that max cut is only implementable if there are 5 or less edges, although 6 might also work in the best case scenario. Either way, QAOA is only useful for toy problems of graphs with less than 6 edges.

This section is concluded with a proof that $D(E,T)$ converges almost surely to $\frac{2}{3}$ for random PDFs (like what might be expected from noisy qubits). First, consider the related function

$$\delta(E,T) = 1 - D(E,T) = \frac{1}{2}\sum_{i=1}^{n} |E_i - T_i| \tag{126}$$

Now, define something very similar to the random PDFs except that the entries need not sum to 1. That is

$$P_n = \left\{\frac{2a_1}{n}, \frac{2a_2}{n}, \cdots, \frac{2a_n}{n}\right\} \text{ and } Q_n = \left\{\frac{2b_1}{n}, \frac{2b_2}{n}, \cdots, \frac{2b_n}{n}\right\} \tag{127}$$

where a_i and b_i are picked uniformly from $[0,1]$. The expected value of $\delta(P_n, Q_n)$ is

$$E[\delta(P_n,Q_n)] = \frac{n}{2}\int_0^1 \int_0^1 \frac{2}{n}|x-y|dxdy = \frac{1}{3} \tag{128}$$

By the strong law of large numbers we are assured that the average of $\delta(P_n,Q_n)$ approaches $\frac{1}{3}$ as n increases. Of course, P_n and Q_n are not the same as the random PDFs we defined above. However, they behave increasingly the same as n increases. This is because the expected value of summing P_n is

$$E\left[\sum_{i=1}^{n} P_n(i)\right] = n\int_0^1 \frac{2}{n}xdx = 1 \tag{129}$$

Thus, as n increases to infinity we are also assured that

$$E[\delta(E,T)] = E[\delta(P_n, Q_n)] = \frac{1}{3} \tag{130}$$

and hence the average of $\delta(E,T)$ converges to $\frac{1}{3}$ as well. Thus, if E and T are two random PDFs of sufficiently large number of entries, we are assured to be within some close ϵ of $\frac{1}{3}$.

4 Dominating Set Problem (DSP)

Imagine a set of targets that require surveillance. Between certain targets are lines of site which allow the surveillance apparatus to view more than one target at a time. For example, we might use drones to watch over a mountainous region for movement with lines of site up a valley. The surveillance targets can be thought of as nodes of a graph and the lines of site can be thought of as edges connected the nodes. Then the optimization problem is what is the least number of drones required to survey the entire network.

4.1 Cost Function Definition

The DSP can be implemented by utilizing the following clauses for an n node network. Note that T_k and D_k are used for the clauses instead of C_k in order to differentiate between the types of measures used. For input $z = z_0 z_1 ... z_{n-1}$, the first clause measures the number of targets surveyed:

$$T_k(z) = \begin{cases} 1 & \text{if the } k\text{th nodes is connected to some } i\text{th node where } z_i = 1 \\ 0 & \text{if otherwise} \end{cases} \tag{131}$$

For the purposes of this definition, every node is connected to itself. The cost function is larger the more nodes are surveyed in the network. The second set of clauses measure the number of

drones used:

$$D_k(z) = \begin{cases} 1 & \text{if } z_k = 0 \\ 0 & \text{if } z_k = 1 \end{cases} \tag{132}$$

4.1.1 Relation between DSP and clause definitions

It is natural to wonder whether this set of clauses actually corresponds to the DSP. Recall that the DSP is defined as finding the least number of drones used to survey all targets (nodes). In fact, the list of clauses defined above does not correspond exactly to the dominating set problem, it requires a further algorithm to output a solution to the DSP. Luckily, this algorithm runs in polynomial time on the number of nodes on the graph and therefore does not impact QAOA overall. Suppose $\overline{z} = z_0 z_1 ... z_{n-1}$ is found in some manner (classical computer via brute force, QAOA, or some other algorithm). First, place a drone on the ith node if $z_i = 1$ and do not place a drone otherwise. Next, use a classical computer to check every node and see if it is under surveillance (recall that a drone on the ith node implies that node is under surveillance). A trivial upper bound on this step is n^2 operations as each of n nodes can have at most n edges (we count self-surveillance in this step). Finally, if some ith nodes is not under surveillance, place a drone on that node.

Now we will show that if $\overline{z}$ is given, then the above algorithm corresponds to a solution to the DSP. First, note that if placing drones according to $\overline{z}$ does not survey some nodes, then these nodes can only be connected to other nodes which themselves are surveyed but do not have a drone on them.

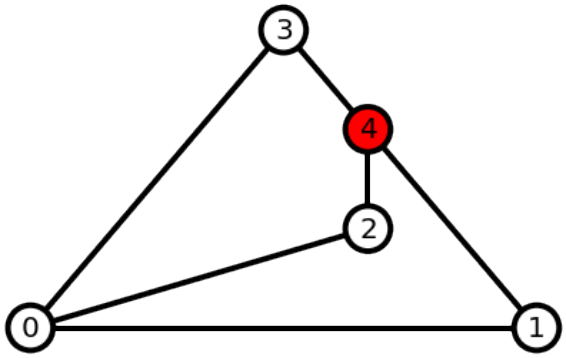


Figure 34: $\overline{z} = 00001$ corresponding to a drone on node 4 while nodes 0 is not under surveillance but nodes $1, 2, 3$ are

For example, in Figure 34, node 0 is not surveyed, and every node it is attached to is surveyed. Consider a general case where $\overline{z}$ has been found and drones have been placed on the graph accordingly. For ease of notation, let node 0 be a node that is not surveyed and let A be the set of nodes connected to node 0 which are surveyed by other nodes. Finally, let there be m nodes in A (note that m can be zero, which corresponds to node 0 being isolated). Without loss of generality, let these m nodes correspond to nodes 1 through m. Then

$$C(\overline{z}) = D_0(\overline{z}) + T_0(\overline{z}) + \sum_{i=1}^{m} (D_i(\overline{z}) + T_i(\overline{z})) + \sum_{i=m+1}^{n-1} (D_i(\overline{z}) + T_i(\overline{z})) \tag{133}$$

$$= 1 + 0 + \sum_{i=1}^{m} (1 + 1) + \sum_{i=m+1}^{n-1} (D_i(\overline{z}) + T_i(\overline{z})) = 2m + 1 + \sum_{i=m+1}^{n-1} (D_i(\overline{z}) + T_i(\overline{z})) \tag{134}$$

Now, define $z^{'}$ to be $1z_1z_2...z_{n-1}$ where $z_1, z_2, ...z_{n-1}$ are defined by the binary digits in $\overline{z}$. Note that $z_1 = z_2 = ... = z_m = 0$ for both $\overline{z}$ and $z^{'}$. For Figure 34 with $\overline{z} = 00001$, the corresponding $z^{'}$ is 10001. Replacing $\overline{z}$ in 133 with $z^{'}$, we get

$$C(z^{'}) = D_0(z^{'}) + T_0(z^{'}) + \sum_{i=1}^{m} \Big(D_i(z^{'}) + T_i(z^{'}) \Big) + \sum_{i=m+1}^{n-1} \Big(D_i(z^{'}) + T_i(z^{'}) \Big) \tag{135}$$

$$= 0 + 1 + \sum_{i=1}^{m} (1+1) + \sum_{i=m+1}^{n-1} \Big(D_i(z^{'}) + T_i(z^{'}) \Big) = 2m + 1 + \sum_{i=m+1}^{n-1} \Big(D_i(z^{'}) + T_i(z^{'}) \Big). \tag{136}$$

However, for nodes 1 through $n-1$, $\overline{z}$ and $z^{'}$ correspond to the same nodes with drones on them and the same set of nodes covered by drones (that is, excluding node 0). Thus, $D_i(z^{'}) = D_i(\overline{z})$ and $T_i(z^{'}) = T_i(\overline{z})$. Then (136) becomes

$$= 2m + 1 + \sum_{i=m+1}^{n-1} (D_i(\overline{z}) + T_i(\overline{z})) = C(\overline{z}). \tag{137}$$

Thus, $C(\overline{z}) = C(z^{'})$ and $\overline{z}$ is not unique.

To complete this section of the proof, suppose that drones are placed according to $\overline{z}$ and that a node that is not surveyed is connected to a different node that is not surveyed. If we define $z^{'}$ to be the equal to $\overline{z}$ except for a drone on either of these nodes, then clearly $C(z^{'}) = C(\overline{z}) + 1$ as we can cover two nodes with one drone. But this implies $C(z^{'}) > C(\overline{z})$, a contradiction. We conclude that if drones placed according to $\overline{z}$ does not survey some nodes, then modifying $\overline{z}$ by placing drones on those nodes does not decrease $C(\overline{z})$.

Finally, suppose we have a graph with a solution to the DSP of $z^{'}$ (that is, place drones according to $z^{'}$) such that $C(z^{'}) \neq C(\overline{z})$. From the previous section, without loss of generality we may take $\overline{z}$ to be the string wich maximizes the cost function and surveys all nodes in the graph. Now, if $C(z^{'}) < C(\overline{z})$, then $z^{'}$ has more 1s in its binary expansion than $\overline{z}$. This is because both $z^{'}$ and $\overline{z}$ survey all nodes, which implies $\overline{z}$ users fewer drones than $z^{'}$. However, this is a

contradiction as we defined $z^{'}$ to be a solution to the DSP, which means it uses the minimum number of drones to survey the entire graph. Thus, $C(z^{'}) \not< C(\overline{z})$. The other possibility, that $C(z^{'}) > C(\overline{z})$, is equally impossible as we defined $\overline{z}$ to be a string which maximizes the cost function. Having exhausted all cases, we conclude $C(z^{'}) = C(\overline{z})$. Thus, finding a string which maximizes $C(z)$ is equivalent to finding a solution to the DSP.

4.2 Standard Example

It is useful to examine these definitions through an example. Consider the following graph of four nodes:

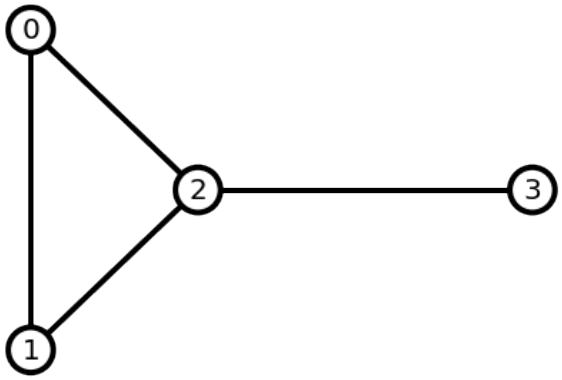


Figure 35: The standard 4-node example graph used throughout this paper

The nodes have been labeled 0 through 3 in accordance with computer science notation. Writing out all possible z's, the following table is constructed:

Table 3: The full set of clauses for the standard example implemented for the DSP

z	$z_0z_1z_2z_3$	T_0	T_1	T_2	T_3	D_0	D_1	D_2	D_3	$C(z)$
0	0000	0	0	0	0	1	1	1	1	4
1	0001	0	0	1	1	1	1	1	0	5
2	0010	1	1	1	1	1	1	0	1	7
3	0011	1	1	1	1	1	1	0	0	6
4	0100	1	1	1	0	1	0	1	1	6
5	0101	1	1	1	1	1	0	1	0	6
6	0110	1	1	1	1	1	0	0	1	6
7	0111	1	1	1	1	1	0	0	0	5
8	1000	1	1	1	0	0	1	1	1	6
9	1001	1	1	1	1	0	1	1	0	6
10	1010	1	1	1	1	0	1	0	1	6
11	1011	1	1	1	1	0	1	0	0	5
12	1100	1	1	1	0	0	0	1	1	5
13	1101	1	1	1	1	0	0	1	0	5
14	1110	1	1	1	1	0	0	0	1	5
15	1111	1	1	1	1	0	0	0	0	4

Table 3 enumerates every possible $D_k(z)$, $T_k(z)$ and $C(z)$ for $z \in \{0, 1, ..., 2^4 - 1\}$. With 4 nodes, this corresponds to $2^4 = 16$ states. However, as the number of nodes increases, the number of possible states increases exponentially, which implies writing out the full table is only useful for small graphs. From Table 3, it is clear that $\overline{z} = 2$ (or $\overline{z} = 0010$ in binary), which corresponds to a drone on node 2. Of course, this was obvious at a glance at Figure 35, where node 2 is connected to all other nodes in the graph.

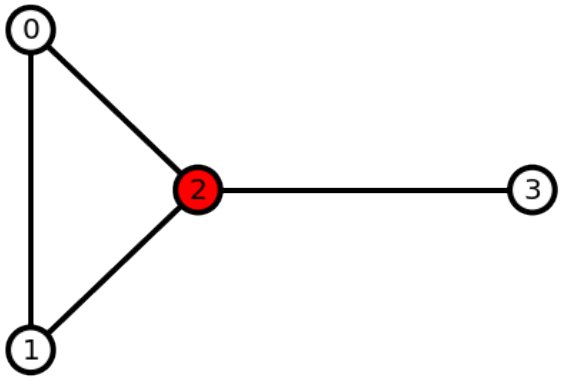


Figure 36: $C(\overline{z}) = C(0010) = 7$ corresponding to a drone on node 2

4.3 Algorithm Implementation

Unlike the MCP, the implementation of the DSP requires an extra ancillary qubit. For the sake of clarity, non-ancillary qubits shall henceforth be known as z-qubits.

Step 1: the first step for actual implementation is to hit each z-qubit with a Hadamard gate (84):

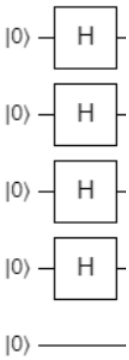


Figure 37: The first step of implementing the standard example. The extra ancillary qubit is used later in the algorithm to introduce the $-\gamma$ rotations. As before, the qubit on top is qubit 0, the one below it is qubit 1, and the pattern continues to qubit 4 on the bottom.

In the standard example, this is equivalent to

$$|\psi_1\rangle = H^{\otimes 4} \otimes I|\psi_0\rangle = \frac{1}{4}\left(|0000\rangle + |0001\rangle + |0010\rangle + \cdots + |1110\rangle + |1111\rangle\right) \otimes |0_4\rangle \quad (138)$$

Step 2: The next step in the implementation of QAOA is the D_k clauses (132). Reiterating, these clauses are defined as

$$D_k(z) = \begin{cases} 1 & \text{if } z_k = 0 \\ 0 & \text{if } z_k = 1 \end{cases} \quad (139)$$

and are implemented as

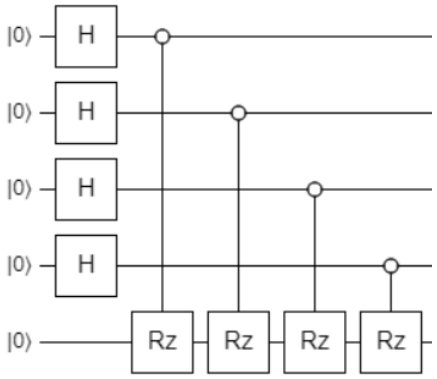


Figure 38: The second step implements the D_k clauses. These are inverted-controlled gates that depend on γ. This is simply the $U_1(\gamma)$ gate (81).

Step 3: The third step is to implement the T_k clauses (131). These clauses are defined as

$$T_k(z) = \begin{cases} 1 & \text{if the } k\text{th nodes is connected to some } i\text{th node where } z_i = 1 \\ 0 & \text{if otherwise} \end{cases} \quad (140)$$

Although complicated to define, they are in fact quite easy to implement. For some node k, let T be the set of nodes such that $i \in T$ implies node i covers node k. Note that $k \in T$. Now, run an

OR gate (see Fig. 13) with inputs at qubits in T and outputs a phase change of $-\gamma$. For example, in the standard example we have

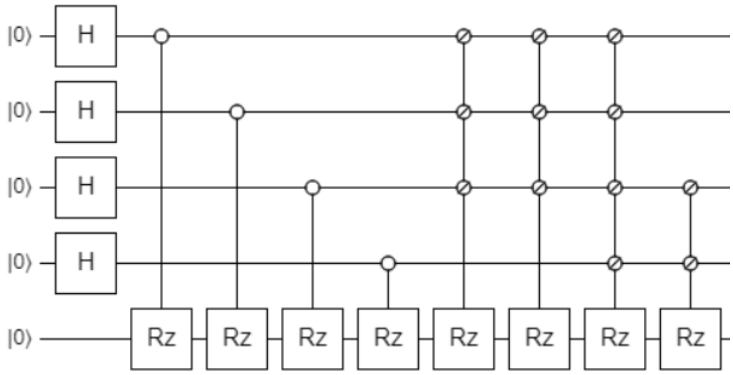


Figure 39: The third step implements the T_k clauses. These are controlled-or gates that phase the state by $-\gamma$.

Steps 2 and 3 have the effect of adding a phase of $-\gamma$ for each of the $2n$ clauses. For the standard example, this is

$$|\psi_3\rangle = \frac{1}{4}\left(e^{-4i\gamma}|0000\rangle + e^{-5i\gamma}|0001\rangle + e^{-7i\gamma}|0010\rangle + \cdots + e^{-4i\gamma}|1111\rangle\right) \tag{141}$$

Step 4: The fourth step is to implement the $-\beta$ rotations in the algorithm the same way as step 3 did for the max cut algorithm. That is, implement

$$U_B(\beta) = U_3\left(2\beta, \frac{3\pi}{2}, \frac{\pi}{2}\right) = \begin{pmatrix} \cos(\beta) & -i\sin(\beta) \\ -i\sin(\beta) & \cos(\beta) \end{pmatrix} \tag{142}$$

for all z-qubits. For the standard example, this looks like

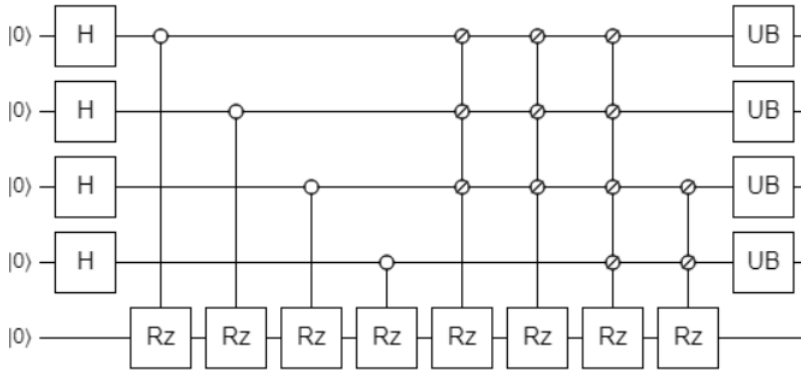


Figure 40: The second to last step in implementing the DSP problem rotates each z-qubit an angle $-\beta$ around the z-axis.

As discussed previously, if we wanted to run QAOA for $p > 1$, then we would repeat steps $2 - 4$ for different γ and β angles.

Step 5: The final step of the algorithm is measuring the z-qubits. This gives a final circuit (for $p = 1$) of

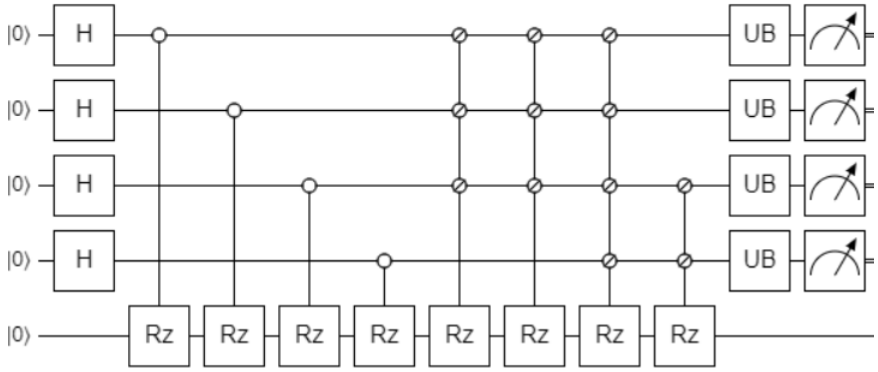


Figure 41: The full circuit implementing the DSP for $p = 1$.

Implementing this circuit for $\gamma = \beta = \frac{\pi}{5}$ give the following histogram

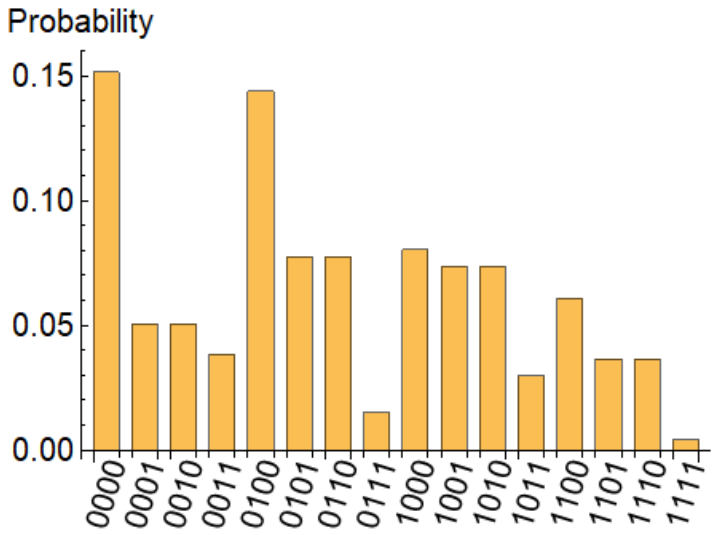


Figure 42: The histogram generated for the standard example with $\gamma = \beta = \frac{\pi}{5}$.

Contrasting the max cut algorithm (see Fig. 29), this histogram is not symmetric about the states $|0111\rangle, |1000\rangle$. This is because the DSP does not exhibit the same symmetry as the MCP. That is, switching which nodes you choose with those you do not choose will not result in another solution (as it does for the MCP).

4.4 Algorithm Analysis

It is possible to bound the number of gates used to implement the DSP on an arbitrary graph using QAOA. We will then break these gates further into the base gates used by qiskit. Consider a graph with n vertices. Let us list out the gates used in each step:

- The algorithm starts with n Hadamard gates.
- The D_k clauses are inverted controlled phase gates. This uses $2n$ swap gates as well as n controlled phase gates. Each of these takes 4 single qubit gates and 2 CX gates (see Appendix 1 for more details) for a total of $6n$ single qubit gates and $2n$ CX gates. Additionally, it takes 1 ancillary qubit which will not change throughout the algorithm.

- The T_k clauses are more difficult to analyze as they are different from graph to graph. However, it is obvious that a node connected to i nodes will take more gates to implement than a node connected to $i-1$ nodes (this can also be shown by induction). This implies that at most each clause will be an n controlled OR gate. This simplifies down to a maximum of $16n^2 - 14n$ CX's and $18n^2 - 14n$ single qubit gates. Also, the algorithm takes n ancillary qubits to run.

- The final step is implementing the β rotation gates. These are single qubit gates, which add n gates to the total.

Adding all the gate counts together, we find that the gate counts can be bounded by

- There are at most $18n^2 - 6n$ single qubit gates.

- There are at most $16n^2 - 12n$ CX gates.

- The algorithm uses at most n ancillary qubits. With the z-qubits, this equates to a total of $2n$ qubits.

The most interesting aspect of this analysis is that the DSP goes as $16n^2$ in the number of CX gates while the MCP goes as n^2 CX gates. That is, the max cut algorithm is easier to implement into QAOA as it uses at least 16 times less than the DSP.

4.5 Running the DSP

The purpose of the next experiment is to demonstrate that quantum computers do not succeed for large numbers of CX gates. Unlike the max cut experiment outlined in the previous section, only one graph was implemented in QAOA for the DSP.

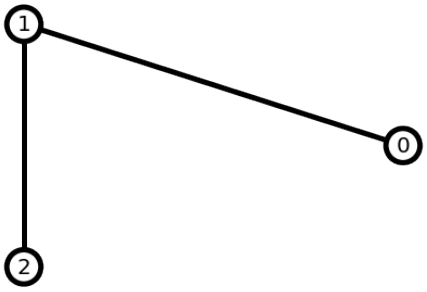


Figure 43: The path graph of 3 vertices was the only graph implemented for the DSP. This experiment is simply to show that the DSP does not work for large numbers of CX gates.

This graph is implemented with the circuit

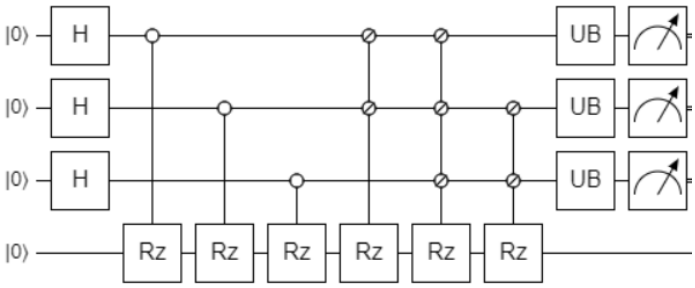


Figure 44: The implementation of the graph in Fig. 43 in QAOA. Using the circuits already described in this paper, this leads to 70 CX gates and 105 single qubit gates.

Using this circuit diagram as well as the naive implementations already described in this paper, it is easy to show that this will take 70 CX gates, 105 single qubit gates, and 6 qubits in total.

While the number of graphs implemented might be lesser, the method of implementation is exactly the same. The input space was defined to be

$$(\gamma, \beta) = 2\pi\{0, .1, .2, \cdots .9, 1\} \times \pi/2\{0, .1, .2, \cdots .9, 1\} \tag{143}$$

Next, for each pair of angles (γ, β), the algorithm was ran $2^8 = 8192$ times on the IBM Poughkeepsie machine at optimization level 3. Once the results had been collected, the same algorithm

was ran 8192 times on the IBM simulator. $D_{(\gamma,\beta)}(E,T)$ is calculated for the particular set of angles (γ,β). Then $D(E,T)$ is defined as

$$D(E,T) = \frac{1}{121}\sum_{(\gamma,\beta)} D_{(\gamma,\beta)}(E,T) \tag{144}$$

After all 8192 runs concluded, $D(E,T) = 0.742$. Compared to the cutoff of 0.817, we can confidently declare the the quantum computer failed for the DSP on this graph. It is interesting to add this result to Fig. 33, modifying the x-axis from edge count to CX count (since the number of CX gates is equal to $2e$ for max cut, each input is simply doubled). This gives a graph

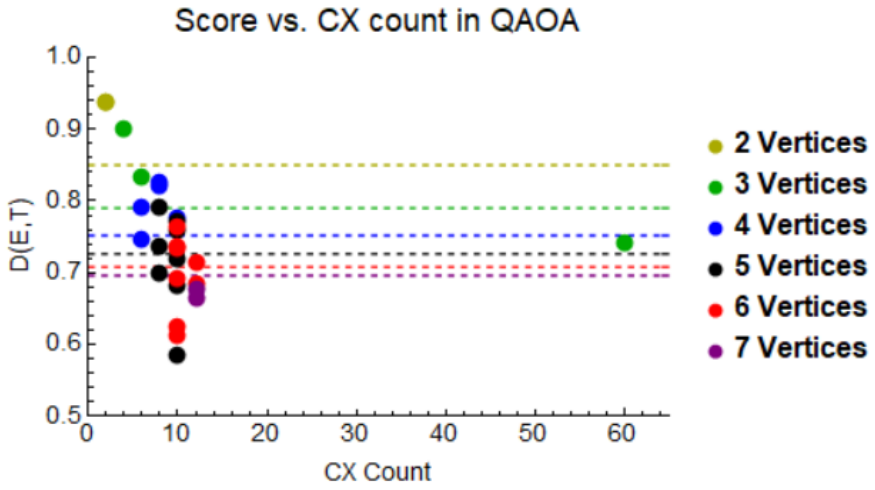


Figure 45: This graph is the result of stretching the x-axis in Fig. 33 by a factor of 2. Then $D(E,T)$ is a function of the number of CX gates. However, there is a clear gap between the max cut runs (group on the left) and the DSP run (single green dot on the right. This shows the vast gulf between running a simple max cut algorithm successfully and running a simple DSP algorithm successfully.

The first thing to note about this graph is that the algorithms begin to fail around the $10-12$ CX mark. That is, the quantum computers begin to fail at about 20% of what is needed to run this very simple DSP example. The second thing to note how much better suited the MCP is to QAOA the the DSP is. While both are combinatorial optimization problems, the MCP is the

clear favorite when it comes to QAOA. That is, future work must be put into recognizing and developing problems that would fit similarly well with QAOA, while avoiding those that require a larger number of implementation gates.

5 Difficulties Running the Quantum Computer

Unfortunately, it is impossible to run anything other than the smallest quantum computations at this time. Without further improvements in hardware, this will continue to be the case for the foreseeable future. However, it is possible to catalogue and study how these algorithms fail. The reasons are numerous: poor initial states, bad gate implementation, cross-talk, and state-decoherence. Of course, the term decoherence is also used to describe the general case where an algorithm does not work as intended. However, we use it here to describe the drift of the qubit around the bloch sphere while the algorithm is performed.

In fact, the qubit does not drift uniformly. It more more likely to drift towards the ground state than the excited state. This can be shown with the following experiment: Construct a circuit on three qubits in the following manner

- **Step 1:** Initialize the circuit with three Hadamard gates (42)
- **Step 2:** Choose one of the following six sub-circuits

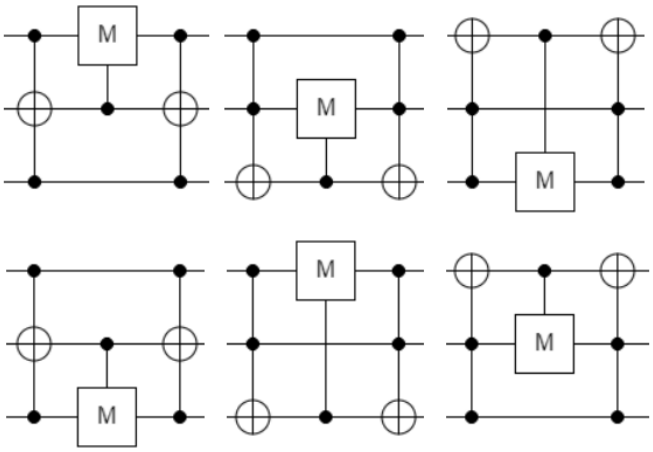


Figure 46: Two qubits are chosen as control qubits while one qubits is the target qubit. The target qubit then performs a controlled R gate (see Fig. 14) where the angle is chosen uniformly from $[0, 4\pi]$ and the target is one of the the other qubits

Here, all possibilities where two qubits are chosen as control qubits while one qubits is the target qubit. The target qubit then performs a controlled R gate (see Fig. 14) where the angle is chosen uniformly from $[0, 4\pi]$. This is denoted by M in Fig. 46.

- **Step 3:** Repeat the previous step 32 times
- **Step 4:** Run circuit 1024 times and tabulate outputs.
- **Step 5:** Repeat steps $1 - 4$ 1024 times and tabulate outputs.

The basic idea is that if the quantum computer behaved as expected, then $|0\rangle$ and $|1\rangle$ would be measured at roughly the same rate. That is, as the circuit was purely random, there is the same probability to measure both outputs. However, the data does not agree with this hypothesis:

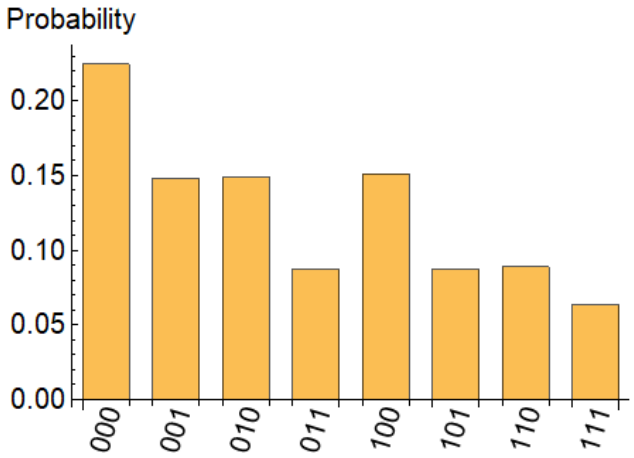


Figure 47: 1024 random circuits are each run 1024 times. If everything worked perfectly, a uniform distribution would be expected as the output. However, it is clear that the system has a bias towards $|0\rangle$ overall.

That is, 61.2% of the qubits were measured as $|0\rangle$ while only 38.8% were measured as $|1\rangle$. Of course, it would be possible to perform a p-test on the hypothesis that the qubits measure $|0\rangle$ 50% time (or $49\% - 51\%$ time). However, this hypothesis can be rejected with with a very high degree of certainty. In fact, the probability that the probability is in fact between $49\% - 51\%$ given that we measured $3 \cdot 2^{20}$ qubits and found 61.2% in the $|0\rangle$ state is given by

$$p = \frac{\Gamma(n+2)\left(B_{x+\frac{1}{2}}(a+1,-a+n+1) - B_{\frac{1}{2}-x}(a+1,-a+n+1)\right)}{a!(n-a)!} \tag{145}$$

Here, B is the incomplete beta function, $x = .01$, $a = 1925438$ (the number of times $|0\rangle$ was measured), and $n = 3 \cdot 2^{20}$. This number is equal to $10^{-1.91027 \cdot 10^7}$. That is, if written out in decimal form, this number would have around 10 million zeros before any other digit appears. As such, it is easy to dismiss the hypothesis that $.49 < P(|0\rangle) < .51$ as false.

This test clearly demonstrates the unreliable nature of large scale quantum computation. For larger circuits with lots of entangled qubits, one can expect decoherence towards the ground state. Of course, if this happens mid-circuit then the following computations are completely

worthless. It would seem at this time that only better hardware can resolve this problem.

6 Results, Future Work, and Final Thoughts

6.1 Results

The experiments presented in this paper represent two different strains of though. On one hand, using QAOA to approximate solutions to instances of the MCP and DSP display the potential pitfalls that quantum algorithms might fact. These experiments define a boundary past which quantum computers are not yet able to perform. From

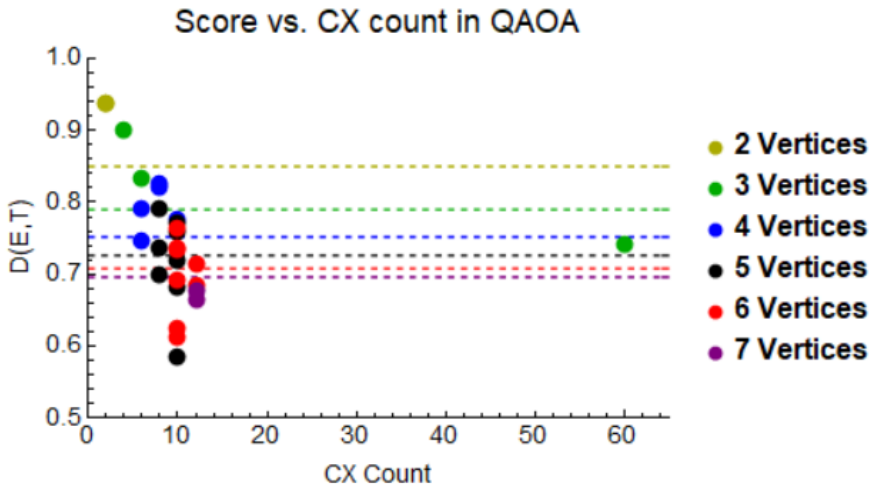


Figure 48: This graph displays the point at which the quantum computer stops working. That is, the quantum computer seems to have a hard time with anything over 10 CX gates.

it is apparent that the cut off point is around 10 CX gates. Of course, this is not an exact cut off, as several graphs with 10 CX gates as well as one graph with 12 CX's perform well under QAOA. However, these results do show conclusively that something on the order of 60 CX's will not perform well. That is, only the smallest algorithms are currently feasible on quantum computers, and further research into hardware development is needed in order to overcome this obstacle.

The other experiment looked at how quantum algorithms fail instead of the point where they fail. In order to explore this, 1024 random circuits (see section 5 for a more in-depth definition) were run 1024 times and the results were tabulated. In the end, over 60% of the outputs measured $|0\rangle$. This demonstrates that one problem with the IBM quantum computers is that they degrade back to the ground state while the algorithm is still running. Again, this seems to be something that only better hardware can fix.

6.2 Future Work

Many of the ideas explored in this paper deserve further development. Three possible avenues of investigation are the metric used to define the similarity between two PDFs, expanding Fig. 48 to other algorithms apart from QAOA, and studying different forms of error and how they impact the overall score between the expected PDF and actual PDF.

There is one main issue with the metric used to define the similarity between two PDFs used in this paper. A run was defined as good if $D(E, T)$ was in the 95th percentile of random runs. However, this is an issue as it assumed that in the worst case scenario the output PDF looks like a random PDF. This is not the case as can be seen in Fig. . Here, a system that has fully decohered is strongly tilted towards $|0\rangle$. This means that is is feasible for a large scale quantum algorithm to fully decohere but still register as a good run using the current metric. In order to fix this problem, one would need to normalize the PDF used to generate the distribution used to find the 95th percentile. That is, the way a system decoheres would have to be taken into account, which would create a much more robust metric that can be used for large numbers of qubits.

It would be instructive to add more algorithms to Fig. 48. In actuality, this graph should be a three dimensional plot which is a function of the number for CX gates and the number of qubits. This would give a much clearer picture of the cause for bad runs: more CX gates or more qubits. Hopefully adding different algorithms would create useful data which could

then be used to predict whether or not an algorithm will be successful without implementing it into the quantum computer. It would also allow circuit designers a hard upper bound on the resources they can use if they want a successful run. Realistically, it is probable that adding more algorithms would show that there are other factors that heavily into algorithm success. However, more study is needed to determine whether or not this is the case.

The most broad future avenue for research is to analyse different forms of error that effect quantum computations. Obviously, in this paper the degradation of the qubits back to the ground state was observed. However, lots of other types of error exists. Phase drift describes how a qubit rotates around the z-axis even when no gate acts upon it. Errors in the initial state obviously greatly effect the algorithm further down the line. Additionally, it is important to study how cross-talk (qubits interacting with qubits due to physical proximity on the chip rather than through gates) effects the overall drift in states as well how quickly these rotations occur. Obviously, this is an in-depth subject that requires many more years of study before using the knowledge to produce error free quantum algorithms.

6.3 Final Thoughts

This project changed forms many times throughout the months. Initially, I had hoped to be able to run QAOA for a large graph using the IBM Q quantum computers and then say something about the results. It quickly became apparent that this was infeasible as the DSP failed for even the simplest of graphs (specifically, the graph in Fig. 43). Many attempts were made to salvage the runs, either by simplifying the code to performing post-run analysis on the results. Unfortunately, nothing I did was able to improve the results. I then turned to the MCP, which allowed me to get a positive result. Of course, this also gave me an idea about the limitations of the quantum computers and about where algorithms began to break down. Finally, I performed the random circuit experiments and found that to my surprise, the qubits were decohering down to

the ground state at by a large margin. I had expected the decoherence to be uniform, which would have been reflected in a $50-50$ split between $|0\rangle$ and $|1\rangle$ begin measured in the random circuit. Of course, while unfortunate for overall algorithm development, this decoherence provides exciting opportunities for further research in the field.

References

[1] Texas Instruments. Texas Instruments Introduces the TI-99/4 Home Computer. Texas Instruments Introduces the TI-99/4 Home Computer, Texas Instruments, 1979, www.ti994.com/1979/brochures/.

[2] P. Benioff (1980), *The computer as a physical system: A microscopic quantum mechanical Hamiltonian model of computers as represented by Turing machines*, J. Statist. Phys., 22, pp. 563-591.

[3] R. Feynman (1981), *Simulating physics with computers*, International Journal of Theoretical Physics

[4] D. Deutsch (1985), *Quantum theory, the Church-Turing principle and the universal quantum computer*, Proceedings of the Royal Society of London, A400, pp. 97-117.

[5] W. Wooters and W. Zurek (1982), *A single quantum cannot be cloned*, Nature, 299, pp. 802-803.

[6] D. Deutsch and R. Jozsa (1992), *Rapid solution of problems by quantum computation*, Proceedings Royal Society of London, A400, pp. 73-90.

[7] T. Kleinjung, K. Aoki, J. Franke, A. Lenstra, E. Thome, J. Bos, P. Gaudry, A. Kruppa, P.

Montgomery, D. A. Osvik, H. te Riele, A. Timofeev, and P. Zimmermann (2010), *Factorization of a 768-bit rsa modulus*, Cryptology ePrint Archive, Report 2010/006

[8] P. Shor (1997), *Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer*, SIAM J. Comput. 26, 1484.

[9] T. Moses (2009), *Quantum Computing and Cryptography*, ETSI White Papers No. 8.

[10] L. Grover (1997), *A Fast Quantum Mechanical Algorithm for Database Search*, quantph/9605043; Phys. Rev. Lett. 78, 325.

[11] I. Chuang, N. Gershenfeld, and M. Kubinec (1998), *Experimental Implementation of Fast Quantum Searching*, Phys. Rev. Lett. 80, 3408

[12] D. DiVincenzo (2000), *The physical implementation of quantum computation*, fortschritte Der Physik-progress Phys. 48(9- 11), 771.

[13] E. Farhi, J. Goldstone, and S. Gutmann (2014), *A quantum approximate optimization algorithm*, Preprint at https://arxiv.org/abs/1411.4028

[14] D. Knuth (1997), *The Art of Computer Programming, Volume 2: Seminumerical Algorithms*, ed 3. AddisonWesley. ISBN 0-201-89684-2.

[15] Winograd and Shmuel (1978), *On computing the discrete Fourier transform*, Mathematics of Computation. 32 (141): 175199.

[16] M. Agrawal, N. Kayal;N. Saxena (2004), *PRIMES is in P*, Annals of Mathematics. 160 (2): 781793.

[17] R. Horn and C. Johnson (1994), *Topics in Matrix Analysis*, Cambridge, England: Cambridge University Press, p. 208.

[18] M. Nielsen and I. Chuang (2010), *Quantum Computation and Quantum Information*, Cambridge University Press.

[19] A. Barenco, C. Bennet, R. Cleve, D. DiVincenzo, N. Margolus, P. Shor, T. Sleator, J. Smolin, and H. Weinfurter (1995), *Elementary gates for quantum computation*, Phys. Rev.

[20] M. Garey and D. Johnson (1979), *Computers and Intractability: A Guide to the Theory of NP-Completeness*, New York

[21] Q. Wang and T. Abdullah (2018), *An Introduction to Quantum Optimization Approximation Algorithm*

[22] S. Hadfield, Z. Wang, B. OGorman, E. G. Rieffel, D. Venturelli, and R. Biswas (2017), *From the quantum approximate optimization algorithm to a quantum alternating operator ansatz*, arXiv:1709.03489.

[23] J. Cook, S. Eidenbenz, and A. Brtschi (2019), *The Quantum Alternating Operator Ansatz on Max-k Vertex Cover*, arXiv:1910.13483

7 Appendix

7.1 A1: Controlled phase and toffoli gate implementation in qiskit

```
q = QuantumRegister(2, "q")
rule = [
    (U1Gate(self.params[0] / 2), [q[1]], []),
    (CnotGate(), [q[0], q[1]], []),
    (U1Gate(-self.params[0] / 2), [q[1]], []),
    (CnotGate(), [q[0], q[1]], [])
]
```

Figure 49: Qiskit's implementation of the controlled phase gate.

```
q = QuantumRegister(3, "q")
rule = [
    (HGate(), [q[2]], []),
    (CnotGate(), [q[1], q[2]], []),
    (TdgGate(), [q[2]], []),
    (CnotGate(), [q[0], q[2]], []),
    (TGate(), [q[2]], []),
    (CnotGate(), [q[1], q[2]], []),
    (TdgGate(), [q[2]], []),
    (CnotGate(), [q[0], q[2]], []),
    (TGate(), [q[1]], []),
    (TGate(), [q[2]], []),
    (HGate(), [q[2]], []),
    (CnotGate(), [q[0], q[1]], []),
    (TGate(), [q[0]], []),
    (TdgGate(), [q[1]], []),
    (CnotGate(), [q[0], q[1]], [])
]
```

Figure 50: Qiskit's implementation of the toffoli gate.

7.2 A2: Graphs used for MCP in QAOA

Figure 51: $D(E,T) = 0.938$

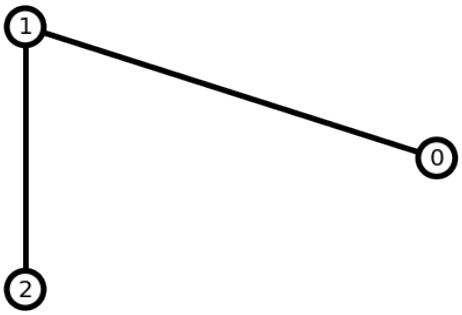


Figure 52: $D(E,T) = 0.901$

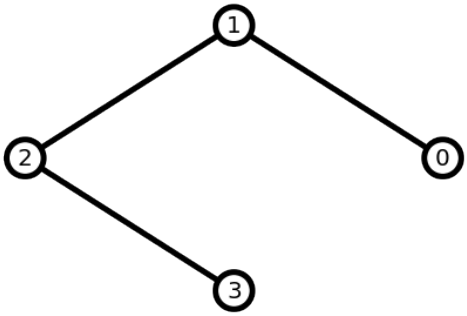


Figure 53: $D(E,T) = 0.792$

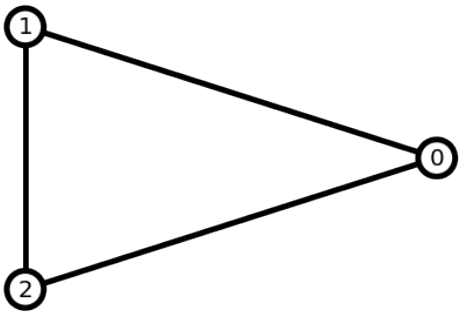


Figure 54: $D(E,T) = 0.835$

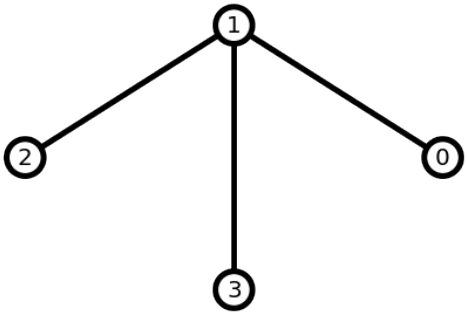


Figure 55: $D(E,T) = 0.773$

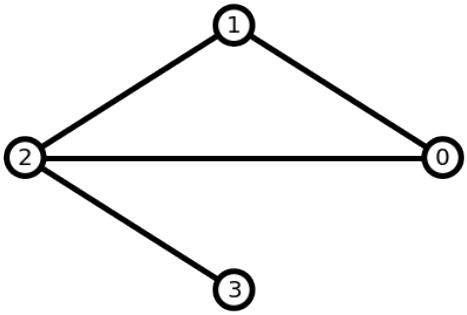


Figure 56: $D(E,T) = 0.820$

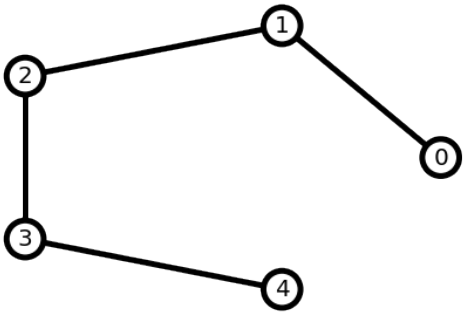


Figure 57: $D(E,T) = 0.792$

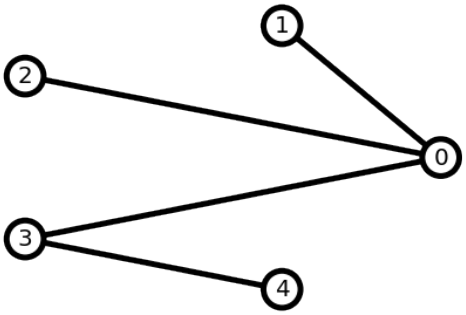


Figure 58: $D(E,T) = 0.699$

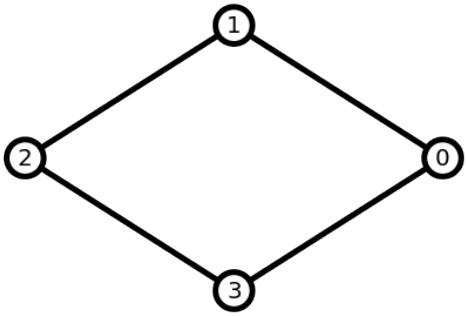


Figure 59: $D(E,T) = 0.827$

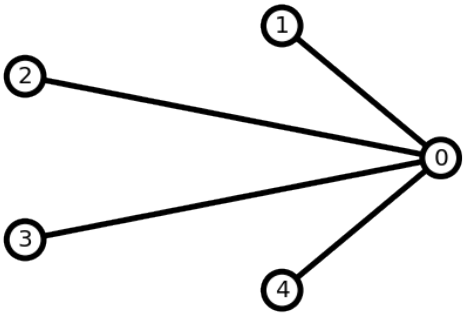


Figure 60: $D(E,T) = 0.736$

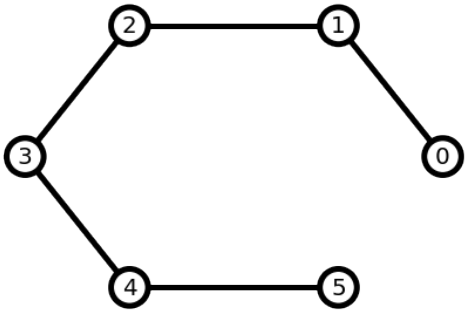


Figure 61: $D(E,T) = 0.733$

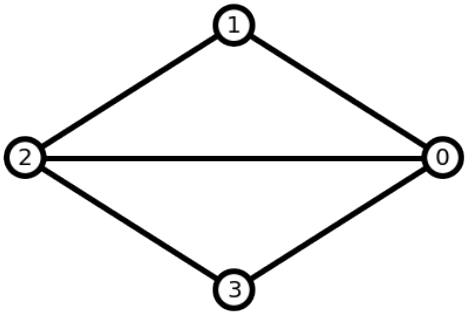


Figure 62: $D(E,T) = 0.777$

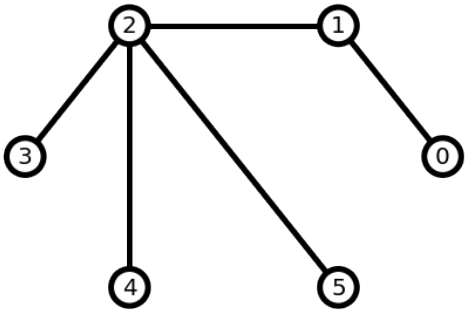


Figure 63: $D(E,T) = 0.691$

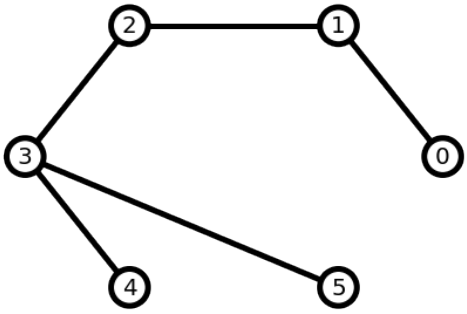


Figure 64: $D(E,T) = 0.737$

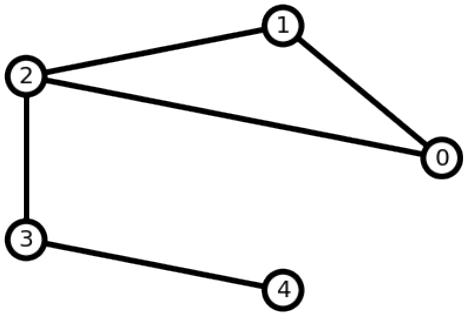


Figure 65: $D(E,T) = 0.759$

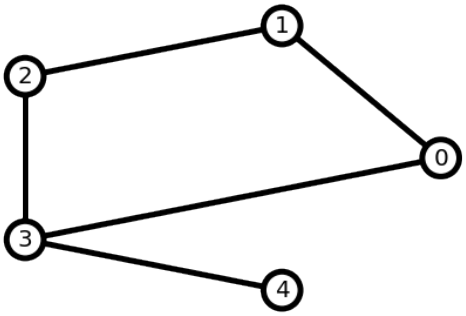


Figure 66: $D(E,T) = 0.771$

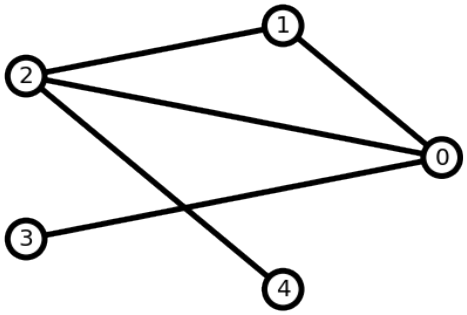


Figure 67: $D(E,T) = 0.704$

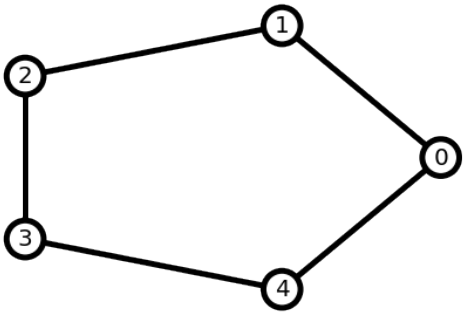


Figure 68: $D(E,T) = 0.763$

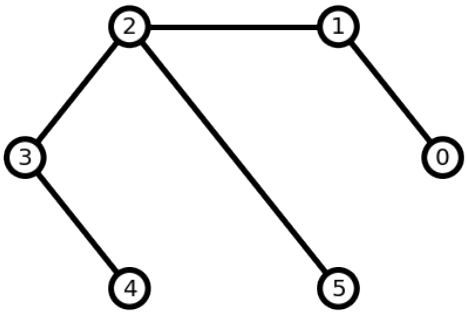


Figure 69: $D(E,T) = 0.763$

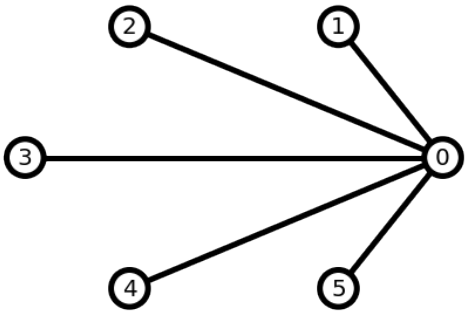


Figure 70: $D(E,T) = 0.625$

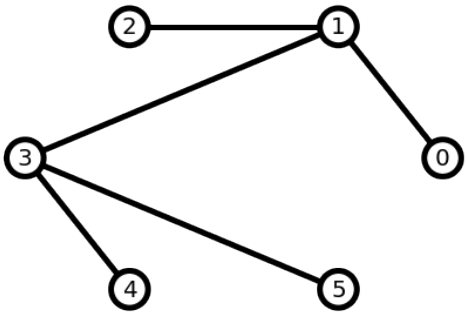


Figure 71: $D(E,T) = 0.647$

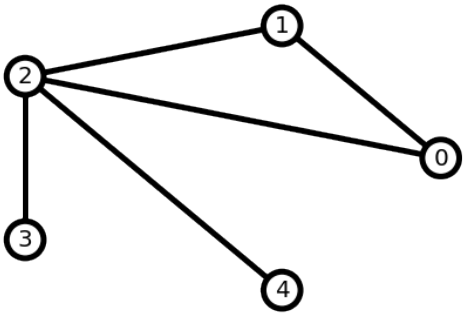


Figure 72: $D(E,T) = 0.718$

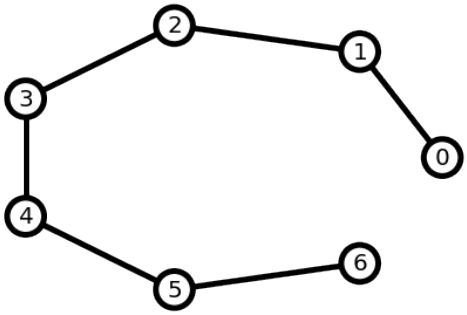


Figure 73: $D(E,T) = 0.664$

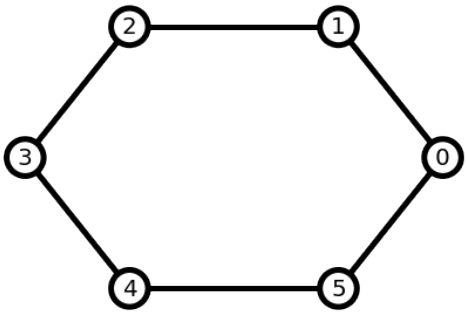


Figure 74: $D(E,T) = 0.685$

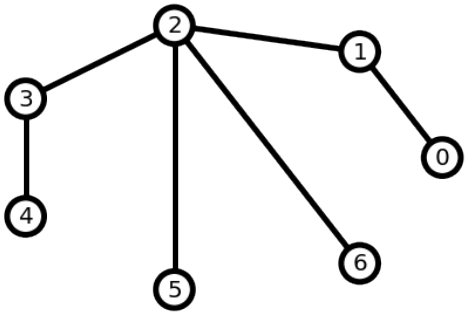


Figure 75: $D(E,T) = 0.677$

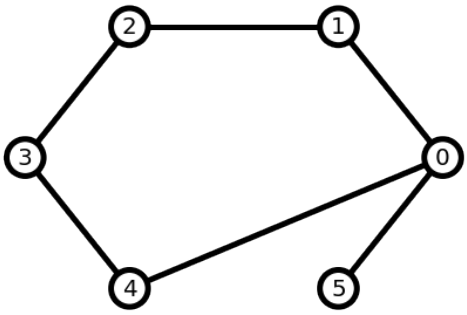


Figure 76: $D(E,T) = 0.715$

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704–0188

1. REPORT DATE *(DD–MM–YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From — To)*
26–03–2020	Master's Thesis	August 2018 — Mar 2020

4. TITLE AND SUBTITLE

SOLVING COMBINATORIAL OPTIMIZATION PROBLEMS USING THE QUANTUM APPROXIMATION OPTIMIZATION ALGORITHM

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Guerrero, Nicolas Joseph, 2Lt, USAF

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 Hobson Way
WPAFB OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT-ENP-MS-19-M-098

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Air Force Research Laboratory
Quantum Information Sciences
26 Electronic Parkway
Rome NY, 13441-4514

10. SPONSOR/MONITOR'S ACRONYM(S)

AFRL/RITQ

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

DISTRIBUTION STATEMENT A:
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

The Quantum Approximation Optimization Algorithm (QAOA) is one of the most promising applications for noisy intermediate-scale quantum machines due to the low number of qubits required as well as the relatively low gate count. Much work has been done on QAOA regarding algorithm implementation and development; less has been done checking how these algorithms actually perform on a real quantum computer. Using the IBM Q Network, several instances of combinatorial optimization problems (the max cut problem and dominating set problem) were implemented into QAOA and analyzed. It was found that only the smallest toy max cut algorithms performed adequately: those that had at most 10 controlled swap gates. The dominating set problem did not work at all as it used many more controlled swap gates than the allowable number. Additionally, a sufficient condition for polynomial implementation in QAOA is shown that generalizes for all combinatorial optimization problems. Finally, further experiments using random circuits also demonstrated that the qubits have a natural tendency to decohere towards the ground state of the system during the lifetime of the algorithm. While unfortunate, these experiments demonstrate the need for better hardware in order for any sort of practical algorithm to be of use.

15. SUBJECT TERMS

Quantum Computing, Algorithm Testing

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Dr. David E. Weeks, AFIT/ENP
U	U	U	U	87	19b. TELEPHONE NUMBER *(include area code)* (937) 255-3636, x4561; David.Weeks@afit.edu