Analysis of the Quantum Adiabatic Algorithm and Quantum Approximation Optimization Algorithm

by
Nicolas Coloma-Cook

Professor Frederick Strauch, Advisor

A thesis submitted in partial fulfillment
of the requirements for the
Degree of Bachelor of Arts with Honors
in Physics

WILLIAMS COLLEGE
Williamstown, Massachusetts
May 14, 2021

Abstract

Quantum algorithms created for Noisy Intermediate Scale Quantum (NISQ) devices are the building blocks of realizable algorithms on quantum hardware that will reshape our future in the next decades. In this thesis, we attempt to construct the Quantum Approximate Optimization Algorithm (QAOA) and implement it using the TensorFlow Quantum API. We test this implementation a on a 6-node, 3-regular problem instance of the MaxCut problem, bench-marking performance using the well researched approximation ratio metric[1]. We then construct the Quantum Adiabatic Algorithm (QAA) applied to MaxCut and highlight the crucial differences between QAOA and QAA. We implement a classical machine learning optimization method for QAOA on MaxCut and discuss the growing area of machine learning techniques applied to NISQ algorithmic frameworks.

Executive Summary

Quantum algorithms for the NISQ era quantum devices we have today are the most promising for a clear computational advantage on certain problem instances over any classical algorithm. The NISQ computational frameworks best suited to achieve this advantage is a fascinating area of active research that is the main topic of this thesis.

In this thesis, we first introduce the Quantum Approximation Optimization Algorithm (QAOA), diving into its components including specific quantum gates, optimization parameters, etc. We then apply QAOA on the Maximum Cut problem, directly constructing the driving and cost Hamiltonians $(\hat{H}_D, \hat{H}_C)$ from fundamental 1-qubit and 2-qubit quantum gates that are specific to this highly researched combinatorial optimization problem [2].

After QAOA is explained, we simulate this algorithm on a 6-node, 3-regular graph using the TensorFlow Quantum API [3]. TensorFlow Quantum provides machine learning methods to optimize the QAOA $(\boldsymbol{\beta}, \boldsymbol{\gamma})$ parameters in hopes of finding an optimal answer, or n-bit string, to the MaxCut problem. We simulate this graph instance with the Adam optimizer and find that this successfully results in the optimal bit string value $(\boldsymbol{z}^* = \boldsymbol{z_{opt}})$ with its corresponding cut value $(\Phi^* = \Phi_{opt})$ for this Maxcut instance.

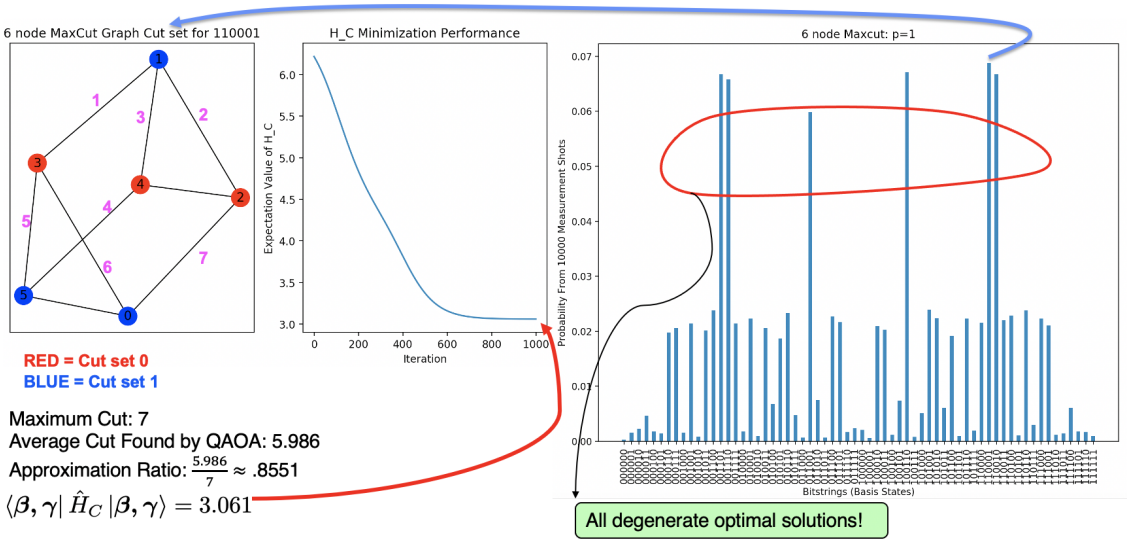

Figure 1: Results using TensorFlow Quantum on a $p = 1$ 6-node, 3-regular graph instance

We then introduce the Quantum Adiabatic Algorithm (QAA), which provides an alternate, continuous quantum computational framework for quantum system evolution. Using the adiabatic theorem, we construct the algorithm of QAA applied to the MaxCut problem with a step-by-step breakdown.

Algorithm 1 QAA on MaxCut

Require: $\hat{H}_D, \hat{H}_C, dt$ (time step), $\mathcal{G}(\mathcal{V}, \mathcal{E}), N_1$ (time step parameter)
  $|\psi\rangle \leftarrow H^{\otimes n} |0\rangle^{\otimes n}$
  $dt \leftarrow \frac{T}{N_1}$
  $t \leftarrow 0$
  for $k \in [0 : T]$ **do**
    $t \leftarrow t + \frac{dt}{2}$
    $\hat{H} \leftarrow (1 - s(t))\hat{H}_D + s(t)\hat{H}_C$
    $t \leftarrow t + \frac{dt}{2}$
    $|\psi\rangle \leftarrow e^{-i\hat{H}dt} |\psi\rangle$
    $E_{avg} \leftarrow \langle \hat{H} \rangle$
  end for
  return $|\psi\rangle$

We then simulate this QAA on MaxCut algorithm on another 6-node, 3-regular graph instance. We find that this framework also yields the optimal bit string value ($\boldsymbol{z^* = z_{opt}}$) with its corresponding cut value ($\Phi^* = \Phi_{opt}$) for this Maxcut instance.

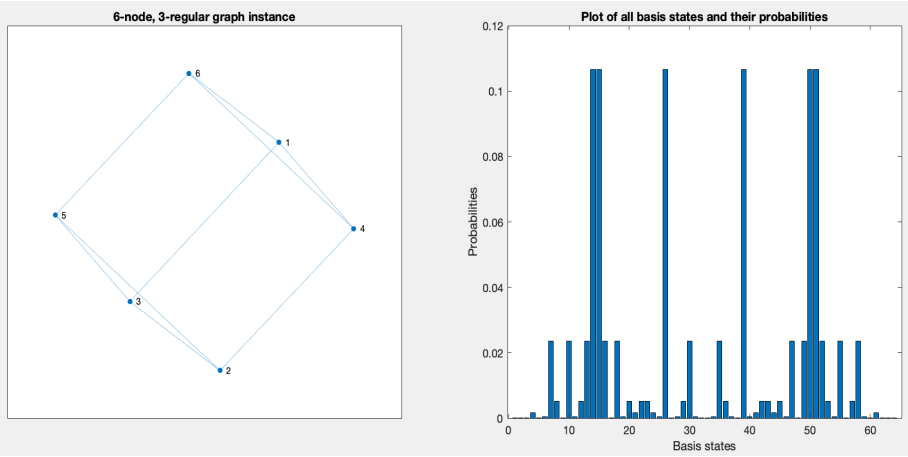

Figure 2: QAA on MaxCut MATLAB Simulation Results for an input 6-node, 3-regular graph instance.

The nuances between QAA and QAOA are then analyzed as we directly show that QAOA approaches the continuous QAA framework as the number of QAOA variational parameters get large ($p \to \infty$).

An efficiency analysis of QAA versus QAOA is also explored as we highlight the hardware barriers for the MaxCut implementation on NISQ devices. We conclude that as the number of nodes (n) and edges (m) increase for an input $\mathcal{G}(\mathcal{V}, \mathcal{E})$ graph instance in QAOA, the total number of physical gates scale up as well, leading to a larger depth circuit needed on NISQ devices. This results in a barrier for implementation on true NISQ devices as the total number of quantum gates is bounded by the decoherence time [4] of the quantum system.

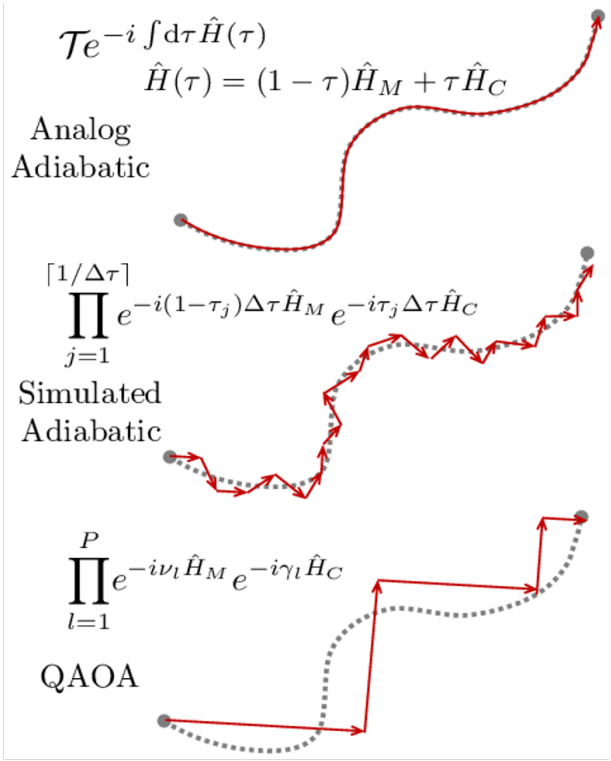

Figure 3: QAOA and QAA $\lim_{p\to\infty}$ relation, where τ is our dt [5]

In the conclusion, we highlight the further research to be done for the MaxCut problem using different variations of input $\mathcal{G}(\mathcal{V}, \mathcal{E})$ graphs. We also discuss a hybrid discrete-continuous algorithmic approach (AQAOA) that could also be applied to MaxCut. Lastly, we discuss the alternative outer loop classical optimizers one can implement for QAOA in TensorFlow Quantum.

Acknowledgments

The unbelievable support from a combination of family, friends, professors, and other mentors made this thesis possible. First, I'd like to thank my mom and dad for their constant support through the ups and downs not only during this thesis but for my whole journey through physics. Thanks for always telling me to pursue what I love no matter what gets in the way.

Huge thanks to Declan Smith and Brendan Hall for the great all night (sometimes into the early morning) study sessions for 411 and 402, not sure how I would've survived those classes without you guys. Can't wait for Brendan to host us in Arizona, thanks so much Brendan for offering up your home to us.

Thanks to Nick Patino as well for being fantastic to work with on labs and problems sets, but more importantly for being a great friend I can turn to with anything. It's been a great four in the Purple Valley with you and I know you're going to kill it in Colorado!

I would also like to thank Professor Ronellenfitsch for being my second reader and taking the time to read over the in-progress draft of my thesis. Thank you to Mike O'Keefe and Kevin Obenland as well for their fantastic guidance and patience over my junior summer. Without this experience and their clear explanations to quantum computational concepts this thesis would not be possible.

Lastly, a huge thank you to Professor Strauch for his constant support through this rollercoaster of a year due to COVID-19. I greatly appreciate his willingness to take the time to explain (and many times re-explain) key quantum computational concepts and more importantly spark my fascination in this field. Without his guidance I undoubtedly would not love the exciting field of quantum computation as much as I do today and I will always be thankful for this fanstastic introduction he gave me.

Contents

List of Figures

Chapter 1

Introduction

1.1 History of Quantum Computation

Since the advent of the computer over 80 years ago, the computational power of this device over time has been a technological marvel. Its compute power has increased exponentially, specifically doubling every two years according to Moore's law [6], paving way for the explosion of technological innovation we've seen as a result of its genesis. The transistor is the fundamental building block of computers and the more of these semiconductor devices packed into a computer, the more computational power. From the Apple II computer containing 10^3 transistors in 1970 to the iPhone 12 housing 10^8 transistors in peoples' pockets, it seems that that computers are on a path of unbounded computation. But as transistors get smaller and smaller, the peculiar properties of quantum mechanics have an increasing impact on these semiconductors via the phenomenon of quantum tunneling. It seems as though for computational power to continue to accelerate, these quantum mechanical effects must be not only taken into account, but become the center of the computer architectures of tomorrow.

Quantum computation harnesses the truly perplexing fundamental properties of quantum mechanics to break through this barrier of computational power, allowing for continual technological spurs that will reshape our world. The properties of entanglement, superposition, and quantization present in quantum systems can be utilized to construct a device capable of much more compute power than the classical computers of today.

There have been several theoretical quantum algorithms (Shor's algorithm, Grover's algorithm, etc.), that have shown to provide a computational speed up, or quantum advantage, relative to the best classical algorithms on certain problem instances [7]. Although this seems extremely promising, the largest obstacle from experimental realization of these algorithms is the imperfect quantum hardware we have today. Many of the quantum algorithms formulated decades ago were assuming access to a fully fault-tolerant quantum computer containing millions of physical qubits that can efficiently manipulate an input quantum state. The best quantum hardware today is orders of magnitude less efficient than these idealized devices, containing between 50-150 physical qubits [4]. Along with the small amount of physical

qubits, there is error propagation throughout this quantum system, in the form of dephasing and decoherence, which severely impacts the performance of today's quantum devices. Obviously, this limitation must be accounted for in the construction of quantum processing units (QPUs) in the form of error correction protocols and thus adds several layers of complexity to this engineering problem.

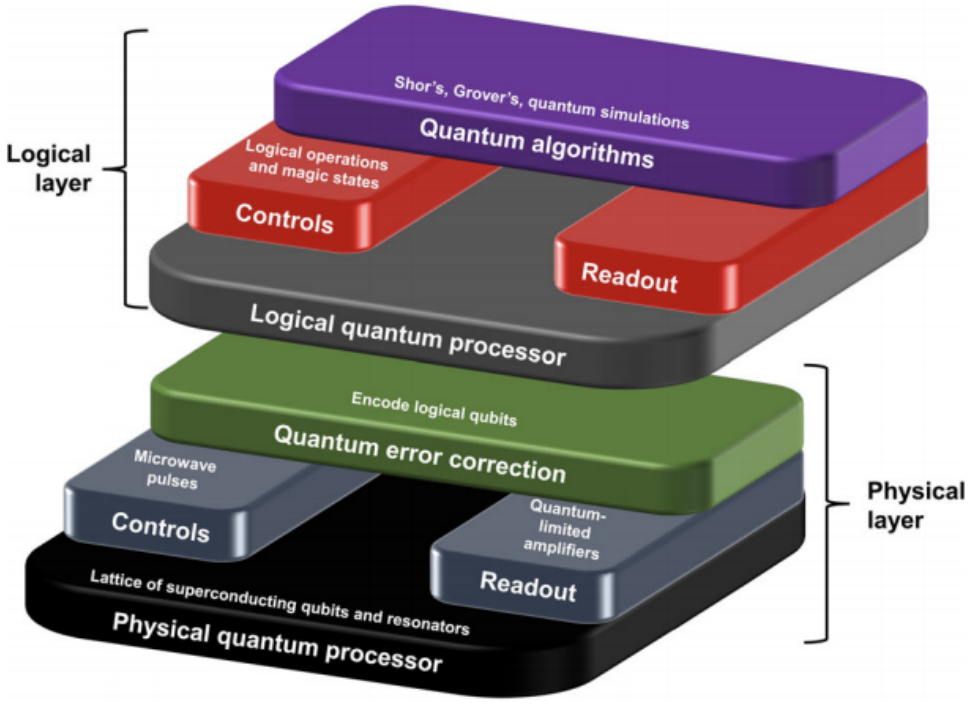

Figure 1.1: Abstraction Layers of an idealized, fault-tolerant Quantum Processing Unit (QPU)[8]. These ideal quantum devices would contain $10^5 - 10^7$ logical qubits in hopes of running quantum algorithms (Shor's, Grover's, etc.) with proven quantum advantages.

Similar to the added complexities on the hardware side, there must be quantum algorithms developed that are robust to the several different types of errors that can occur in these error-prone QPUs. The most promising formulations of algorithms for these devices are the recently developed Variational Quantum Eigensolver [9] (VQE) and the Quantum Approximation Optimization Algorithm (QAOA) [1]. This period in the development of quantum computation has then been coined the "Noisy-Intermediate-Scale-Quantum" (NISQ) era [4] and we specifically describe the criterion for NISQ era algorithms in Chapter 2. We will also analyze QAOA on a well known optimization problem in Chapter 3.

1.2 Combinatorial Optimization

Combinatorial optimization has had amazing success in finding exact solutions to crucial research problems and has been applied to a whole host of areas, from GPS routing[10] to

macroeconomic modelling during financial crises[11].

It is implemented in several fundamental classical algorithms used to find interesting quantities like shortest paths and optimal scheduling. Put simply, given some set of finite elements, combinatorial optimization methods attempt to find an optimal element that satisfies a specific problem of interest. There are many instances in which an exact solution can be found, but when applied to NP-hard and NP-complete problems, this method aims to find the best **approximate** solution.

There is ongoing research in developing combinatorial optimization methods to NP-hard problems ranging from Max-2Sat, the Travelling Salesman problem, etc[12]. QAOA, the NISQ era algorithm described in Chapter 3, is almost always applied to combinatorial optimization problems, specifically the Maximum Cut (MaxCut) problem[1]. This is due to the already existing body of research and results from approaching the MaxCut problem with classical algorithm techniques[2]. The use of QAOA on MaxCut will be explored in Chapter 3, so it is important to introduce the classical objective function used for this problem as a foundation for this analysis.

1.3 The MaxCut Problem

1.3.1 Graph Theory Fundamentals

Assume there is an input graph instance $\mathcal{G}(\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the set of all vertices in $\mathcal{G}$ and $\mathcal{E}$ the set of all its edges. Formally, for some vertex v_i and some edge e_{jk},

$$v_i \in \mathcal{V} \ \ \forall \ v_i, \tag{1.1}$$

$$e_{jk} = (v_j, v_k) \in \mathcal{E} \ \ \forall \ e_{jk}, \tag{1.2}$$

where v_j, v_k are nodes sharing an edge. An edge weight can be assigned to each edge e_{jk} and is denoted as,

$$w_{jk} = \text{edge weight between } v_j, v_k, \quad w_{jk} \in \mathbb{R}. \tag{1.3}$$

Graphs can also be directed, meaning that some path in which one vertex v_j is connected to another vertex v_k is unique. In other words, there is at least on edge, e_{jk}, in $\mathcal{G}(\mathcal{V}, \mathcal{E})$ that can be traverse in only one way. A path is defined with a starting vertex (v_j) and contains all of the vertices passed to end up at a final vertex (v_k). This is denoted as,

$$v_j \to v_l \to v_m \cdots \to v_k. \tag{1.4}$$

For any MaxCut algorithm explored, the input graph, $\mathcal{G}(\mathcal{V}, \mathcal{E})$, is chosen to be undirected. This is an important detail as it greatly reduces the set of all possible graphs which can be sampled from.

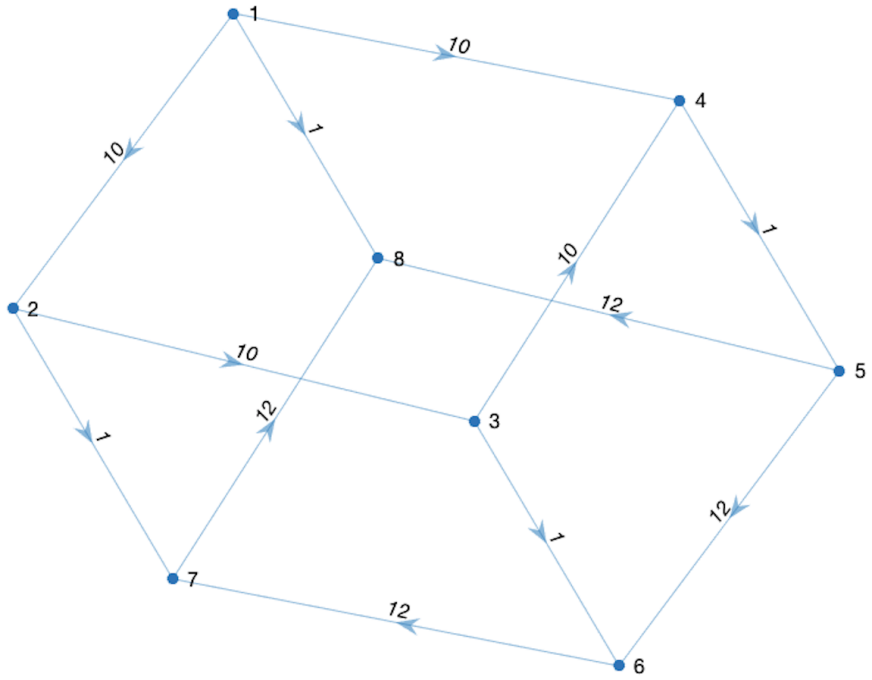

Figure 1.2: Example of a weighted, 10-node, directed graph, $\mathcal{G}(\mathcal{V}, \mathcal{E})$. The arrows indicate the unique path each vertex must take to another. For example, the allowed paths to traverse from v_2 (vertex 2) to v_8 (vertex 8) are $\{v_2 \to v_3 \to v_4 \to v_5 \to v_8,\ v_2 \to v_7 \to v_8, v_2 \to v_3 \to v_6 \to v_7 \to v_8, v_2 \to v_3 \to v_4 \to v_5 \to v_6 \to v_7 \to v_8\}$.

These components describe the most general structure of some input graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ as there are no imposed restrictions on graph structure (besides the assumption of an unweighted graph).

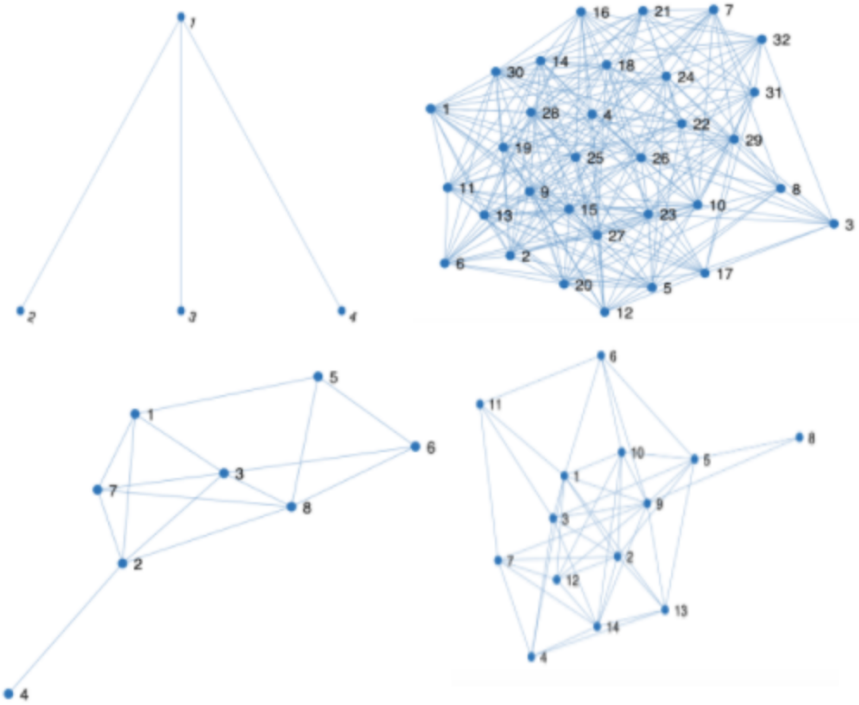

Figure 1.3: Instances of undirected input MaxCut graphs, $\mathcal{G}(\mathcal{V}, \mathcal{E})$, for varying vertex ($\mathcal{V}$) and edge sets ($\mathcal{E}$).

1.3.2 The MaxCut Cost Function

Suppose there is a graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ that is the input MaxCut graph you want to solve. The optimal answer to MaxCut is two disjoint subsets of nodes, or cut-set configurations, $S, \bar{S} \subseteq V$, that maximize the the total value of the edges, e_{jk}, crossing between these two subsets. In other words, out of all the possible ways to split up the nodes in $\mathcal{G}(\mathcal{V}, \mathcal{E})$ into two different "buckets", the configuration in which the edge weights crossing between $(S, \bar{S})$ sum to the maximum possible value is the optimal answer.

To compare different $(S, \bar{S})$ cut-set configurations, there must be a metric to quantify how optimal each of these configurations are. First, we need to start with some partition of the nodes between $(S, \bar{S})$ in $\mathcal{G}(\mathcal{V}, \mathcal{E})$.

This partition can initially be represented as an n-bit string, $\boldsymbol{z}$,

$$\boldsymbol{z} = z_1 ... z_n, \quad z_i \in \{0, 1\}, \tag{1.5}$$

where z_i denotes the subset the i^{th} node belongs to. Let's now define a function $g(z_i)$ that maps each 0 in $\boldsymbol{z}$, which represents all of the nodes in cut-set S, to -1,

$$g(z_i) = 2z_i - 1. \tag{1.6}$$

This is a necessary mapping for the cost function and will become clear as to why it is used in the explanation of Eq. (1.9). Using Eq. (1.6), we can construct an edge set, $\mathcal{E}_{bin}$, that contains the binary values of all the nodes sharing an edge in the input $\mathcal{G}(\mathcal{V}, \mathcal{E})$,

$$e_{bin,jk} = (z_j, z_k) \in \mathcal{E}_{bin} \;\; \forall \; e_{bin,jk}. \tag{1.7}$$

With this notation, a cost function can be constructed that is a function of $\boldsymbol{z}$,

$$C(\boldsymbol{z}) = \frac{1}{2} \sum_{(z_j,z_k)\in\mathcal{E}_{bin}} w_{jk}(1 - g(z_j)g(z_k)), \;\; \forall \; (z_j, z_k) \in \mathcal{E}_{bin}. \tag{1.8}$$

In other words, given the binary edge set ($\mathcal{E}_{bin}$) of some input graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$, $C(\boldsymbol{z})$ iterates through all of the edges in $\mathcal{E}_{bin}$ and adds the edge weight (w_{jk}) if the nodes have different values. From a graphical view, $C(\boldsymbol{z})$ adds the edge weight if the nodes are in different cut-sets and doesn't add anything if the nodes are in the same cut-set (both either in S or $\bar{S}$).

With this cost function construction, this allows us to define a "cut value", $\Phi_{\boldsymbol{z}}$, for each $\boldsymbol{z}$, which is just defined as the resulting value of $C(\boldsymbol{z})$. It's important to note that each $\boldsymbol{z}$ formulated from some undirected graph has doubly degenerate solutions, specifically,

$$C(\boldsymbol{z}) = \Phi_{\boldsymbol{z}} = \Phi_{\sim\boldsymbol{z}} = C(\boldsymbol{\sim z}), \tag{1.9}$$

where $\sim \boldsymbol{z}$ is the bitwise complement of $\boldsymbol{z}$, for example,

$$z = 001101011 \;\; \rightarrow \sim z = 110010100. \tag{1.10}$$

For some $\boldsymbol{z} \in \{0, 1\}^n$, there are 2^n possible unique n-bit strings and this cost function (Eq. (1.8)) is evaluated for every possible unique string. Graphically, each of these n-bit strings correspond to a unique (S, $\bar{S}$) cut-set configuration. The n-bit string that results in the largest (optimal) cut value ($\Phi_{\boldsymbol{z}}$), is then the optimal bit string, $\boldsymbol{z_{opt}}$. The exponential scaling of possible unique n-bit strings with n causes the MaxCut problem to be NP-hard. Due to this, as $n \rightarrow \infty$ the enumeration of all possible n-bit strings becomes computationally intractable and an **approximate** optimal n-bit string with its corresponding cut value is searched for.

There have been several variations of MaxCut instances explored with different classical algorithms [2] that have benchmarked the success of their methods by the widely used approximation ratio. This is an approximation to the optimal cut value, Φ^*, that is found within some error, ϵ. The n-bit string that results in this approximately optimal cut value, Φ^*, is then the n-bit string used as the solution, $\boldsymbol{z^*}$. Mathematically, the approximation ratio is,

$$r = \frac{\Phi^*}{\Phi_{opt}}, \;\; r \in [0, 1]. \tag{1.11}$$

Putting all of these pieces together, the goal of MaxCut is to satisfy the maximization function,

$$\max\{C(\boldsymbol{z}) = \frac{1}{2} \sum_{(z_j,z_k)\in\mathcal{E}_{bin}} w_{jk}(1 - g(z_j)g(z_k)) \ \forall\ (z_j, z_k) \in \mathcal{E}_{bin}\}, \tag{1.12}$$

$$\text{subject to all unique } n\text{-bit strings, } \boldsymbol{z}. \tag{1.13}$$

This results in Φ^* with its corresponding $\boldsymbol{z^*}$. For exactly solvable problems, the goal of MaxCut is to have $\Phi^* = \Phi_{opt}$, $\boldsymbol{z^* = z_{opt}}$.

When exploring the MaxCut problem, there are usually constraints applied to the input graph, $\mathcal{G}(\mathcal{V}, \mathcal{E})$, which truncates the set of all possible graphs one can sample. Some of the most popular limitations imposed involve,

1. Number of vertices
2. Number of edges between nodes
3. Allowed values of the edge weights, w_{jk}
4. Connectivity
   (a) Whether a vertex v_i is able to traverse $\mathcal{G}(\mathcal{V}, \mathcal{E})$ and reach all other vertices in the set $\mathcal{V}$.
   (b) The average number of edges connecting to some particular vertex v_i.

In Chapter 3 we discuss the specific constraints imposed on the graph instances used in the performance analysis of QAOA on MaxCut.

1.3.3 MaxCut Cost Function Values for an Example $\mathcal{G}(\mathcal{V}, \mathcal{E})$

Assume that we have a 5-node input graph, $\mathcal{G}(\mathcal{V}, \mathcal{E})$, that we would like to find the MaxCut for. Also assume that this is an unweighted graph,

$$w_{jk} = 1 \ \forall\ v_j, v_k \in \mathcal{V}. \tag{1.14}$$

Let's take as given that the vertex set $\mathcal{V}$ of the graph is,

$$\mathcal{V} = \{v_0, v_1, v_2, v_3, v_4\}, \tag{1.15}$$

and that it's edge sets $(\mathcal{E}, \mathcal{E}_{bin})$ are,

$$\mathcal{E} = \{(v_0, v_1), (v_0, v_2), (v_1, v_2), (v_2, v_3), (v_3, v_4), (v_4, v_1)\}, \tag{1.16}$$

$$\mathcal{E}_{bin} = \{(z_0, z_1), (z_0, z_2), (z_1, z_2), (z_2, z_3), (z_3, z_4), (z_4, z_1)\}. \tag{1.17}$$

In this example, our graph instance was chosen from the subspace of n-node, unweighted graphs, which fits our specific constraints imposed. Using the information contained in $\mathcal{V}$ and $\mathcal{E}_{bin}$, the general cost function for this cut-set configuration is,

$$C(\boldsymbol{z}) = \frac{1}{2}\{[1 - g(z_0)g(z_1)] + [1 - g(z_0)g(z_2)] + [1 - g(z_1)g(z_2)] \\ + [1 - g(z_2)g(z_3)] + [1 - g(z_3g(z_4)] + [1 - g(z_4)g(z_1)]\}. \quad (1.18)$$

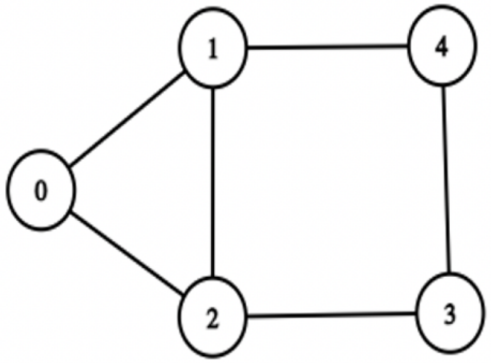

Figure 1.4: Input 5-node, unweighted $\mathcal{G}(\mathcal{V}, \mathcal{E})$ instance. Since all of the edge weights are 1, they were omitted in this depiction.

Our first step is to enumerate all of the cut-set configurations, or all the possible ways these nodes can be split up between $(S, \bar{S})$. This results in a set of all possible unique $\boldsymbol{z}$ values. We then pass in each $\boldsymbol{z}$ value into Eq. 1.8, resulting in a cut value, $C(\boldsymbol{z}) = \Phi_{\boldsymbol{z}}$. After all the cut values are found, the maximum value is chosen and this is the Φ^* value. Example $\Phi_{\boldsymbol{z}}$ values for each $(S, \bar{S})$ configuration in Fig. 1.6 are worked through below.

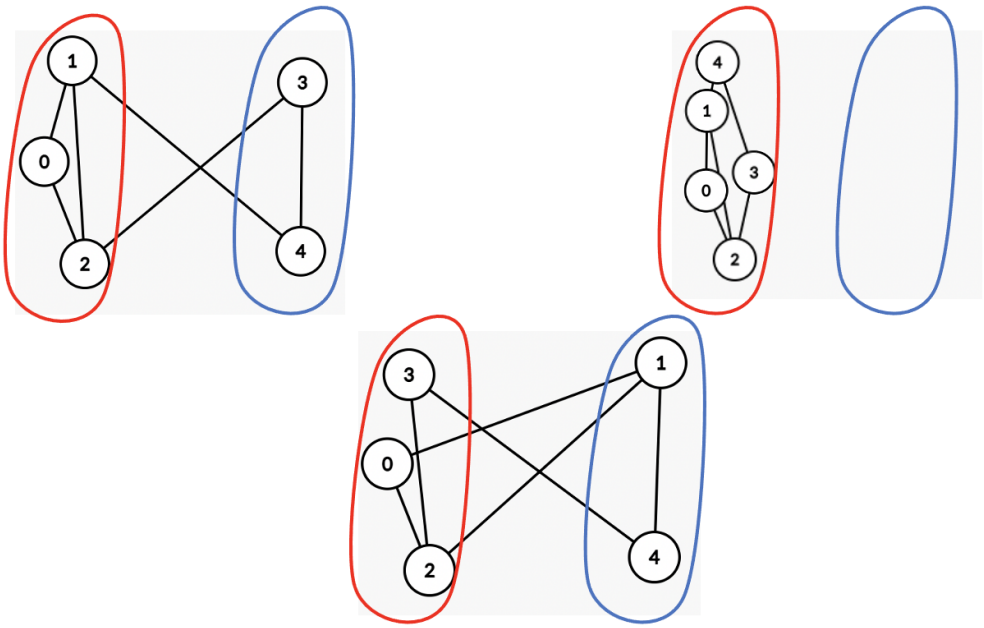

Figure 1.5: Three $(S, \bar{S})$ cut-set configurations sampled from the 2^5 total configurations. The nodes in subset S, and thus having a value of 0, are circled in red while the nodes having a value of 1 in subset $\bar{S}$ are circled in blue.

Cut-set Configuration 1 (Top Left).

1. *Subset configuration:*

$$S = \{v_0, v_1, v_2\}, \quad \bar{S} = \{v_3, v_4\}, \tag{1.19}$$

2. $\boldsymbol{z}$ *value:*

$$\boldsymbol{z} = 00011. \tag{1.20}$$

3. *Cut value using (1.18):*

$$C(00011) = \frac{1}{2}\{[1-(-1)(-1)] + [1-(-1)(-1)] + [1-(-1)(-1)] + [1-(-1)(1)] + [1-(1)(1)] + [1-(1)(-1)]\} \tag{1.21}$$

$$\Phi_{00011} = C(00011) = 2, \tag{1.22}$$

which agrees with the number of crossing edges in cut-set configuration 1.

Cut-set Configuration (Top Right).

1. *Subset configuration:*

$$S = \{v_0, v_1, v_2, v_3, v_4\}, \quad \bar{S} = \{\}, \tag{1.23}$$

2. $\boldsymbol{z}$ *value:*

$$\boldsymbol{z} = 00000. \tag{1.24}$$

This is a more trivial example as $\bar{S}$ is an empty set which indicates that there are no crossing edges. It can then be deduced that this corresponds to $\Phi_{00000} = 0$ without explicitly evaluating the cost function calculation. This example was highlighted to demonstrate that this is still a valid subset configuration and will always be a subset configuration for any $n \geq 1$ graph instance.

Cut-set Configuration 3 (Bottom). The subset configuration is,

1. *Subset configuration:*

$$S = \{v_0, v_2, v_3\}, \quad \bar{S} = \{v_1, v_4\}, \tag{1.25}$$

2. $\boldsymbol{z}$ *value:*

$$\boldsymbol{z} = 01001. \tag{1.26}$$

3. *Cut value using (1.18):*

$$C(01001) = \frac{1}{2}\{[1-(-1)(1)] + [1-(-1)(-1)] + [1-(1)(-1)] + [1-(-1)(-1)] + [1-(-1)(1)] + [1-(1)(1)]\}, \tag{1.27}$$

$$\Phi_{01001} = C(01001) = 3, \tag{1.28}$$

which also agrees with the number of crossing edges in subset 3. None of these subsets represent the optimal cut value.

Fig 1.6 illustrates the optimal cut and bit string $(\Phi_{opt}, \boldsymbol{z_{opt}})$ which we can easily verify. Taking this 5-bit string as an input to the cost function we get,

$$C(01010) = \frac{1}{2}\{[1-(-1)(1)] + [1-(-1)(-1)] + [1-(1)(-1)] + [1-(-1)(1)] + [1-(-1)(-1)] + [1-(1)(-1)]\}, \tag{1.29}$$

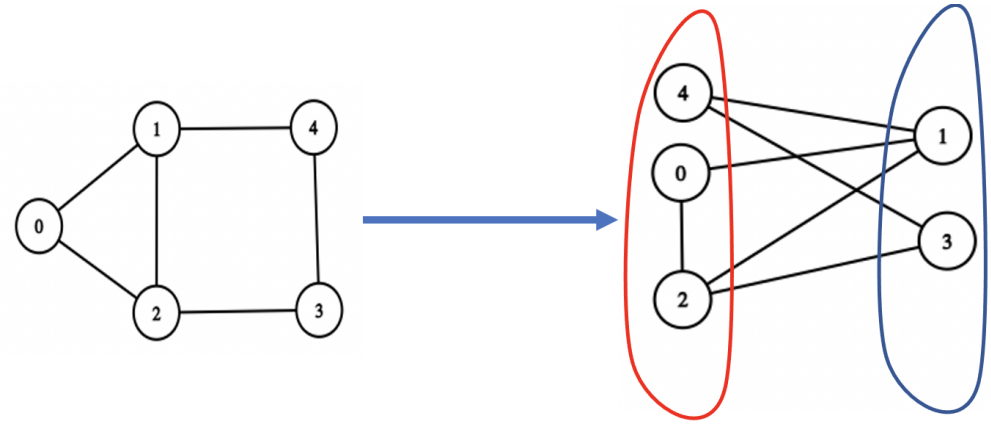

Figure 1.6: Depiction of the optimal 5-bit string, $\boldsymbol{z}^* = \boldsymbol{z}_{\boldsymbol{opt}} = 01010$. The corresponding cut to 01010 are the number of crossing edges, $\Phi^* = \Phi_{opt} = 5$. Note that this solution is doubly degenerate as swapping all nodes to the opposite subset, resulting in $\boldsymbol{z} = 10101$, is also an optimal 5-bit string.

$$\Phi_{opt} = C(01010) = 5. \tag{1.30}$$

This detailed example will provide an essential foundation for the translation of the MaxCut problem into a NISQ era and adiabatic quantum computing framework in Chapters 3 and 4.

1.4 Thesis Outline

An overview of the fundamentals of quantum computing is highlighted in the Chapter 2, specifically the basics of the qubit, the mechanics of how a quantum state evolves through the application of unitary operations, and key mathematical operations used as the base of all quantum algorithms. Chapter 3 deeply describes all the components of QAOA, how the quantum processor and classical optimizer are connected, and the performance of QAOA on a Maximium Cut problem instance using novel machine learning techniques. In Chapter 4, adiabatic quantum computing is introduced along with the methods employed to solve problems by encoding solutions in a problem Hamiltonian. Lastly, the Conclusion highlights potential extensions to this work, from new optimizing techniques that can be employed in QAOA to other Hamiltonians that may make an alternative quantum algorithm more adaptive to more optimization problems.

Chapter 2

Background

This chapter highlights the fundamental components and mathematical concepts needed to construct any NISQ or non-NISQ quantum algorithm. Quantum circuit diagrams are utilized to provide a succinct visualization of gate-model quantum algorithms. Qubits are represented as horizontal wires with each of their input states depicted on the far left of each diagram. Gates lying on those wires represent unitary operators acting on a particular qubit at a particular time step.

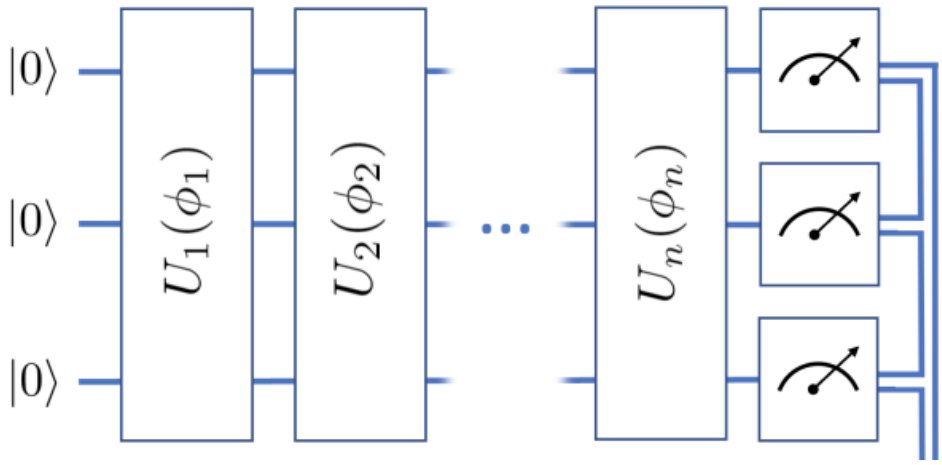

Figure 2.1: General Parametrized Quantum Circuit example for a 3-qubit system. The circuit is read left to right, where the initial state $|000\rangle$ is acted on by n blocks of unitary operators. The measurement operators are represented by the farthest right boxes[13].

The circuit representation of the input 3-qubit state is represented in 2.1, and the resulting output state is written as,

$$|\psi\rangle = \hat{U}_n(\phi_n)\hat{U}_{n-1}(\phi_{n-1})...\hat{U}_2(\phi_2)\hat{U}_1(\phi_1)\,|000\rangle\,. \tag{2.1}$$

Qubits, gates, circuit depth, and all other quantum algorithm components will be described in detail throughout Chapter 2.

2.1 Fundamentals of Quantum Computation

The most basic chunk of quantum information is the qubit, a two-level quantum system that can be represented as several different physical materials (superconducting circuits, internal states of trapped ions, spin states of trapped electrons, etc.)[4]. Unlike classical bits, qubits hold the fundamental property of potentially being a superposition state of its two levels, implying a probabilistic nature for the qubit. A qubit fully in its up or down state is denoted as:

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \qquad |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}. \tag{2.2}$$

More generally, a qubit can be written as a superposition of its two levels,

$$|z\rangle = c_0 \begin{bmatrix} 1 \\ 0 \end{bmatrix}, + c_1 \begin{bmatrix} 0 \\ 1 \end{bmatrix} = c_0 |0\rangle + c_1 |1\rangle \quad c_1, c_2 \in \mathbb{C}. \tag{2.3}$$

A general two-level quantum system is commonly represented as some vector on the Bloch Sphere, which is a unit sphere oriented in such a way that the $+\hat{z}$ direction corresponds to a qubit state of $|0\rangle$ and $-\hat{z}$ to a $|1\rangle$ state[7]. Using the visualization from Fig. 2.2, the general 1-qubit state can be represented as,

$$|\psi\rangle = \cos\frac{\theta}{2}|0\rangle + e^{i\phi}\sin\frac{\theta}{2}|1\rangle. \tag{2.4}$$

To solve many of the interesting, real-world problems currently out of reach for today's classical computers requires lots of computational power. This can be achieved with the inclusion of more qubits that can interact with each other in our quantum system. Since each qubit is a two-level system, an n-qubit quantum system can represent up to $N = 2^n$ basis states, each with a specific complex amplitude.

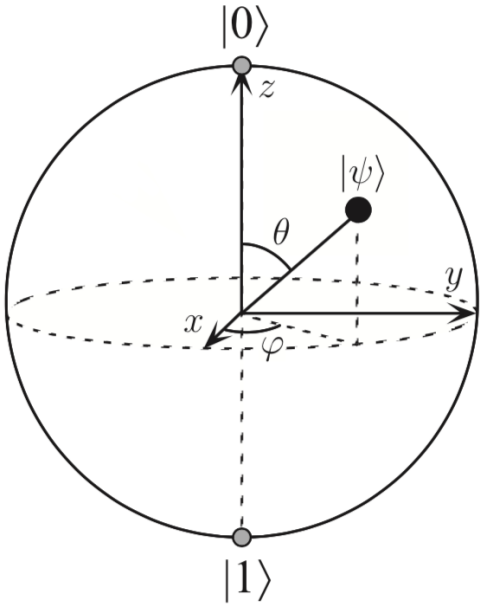

Figure 2.2: Bloch Sphere representation of a General Qubit state $|\psi\rangle$ [14].

2.1.1 Tensor Products and Fundamental Quantum Gates

The tensor/Kronecker product is a central mathematical operation in constructing unitary operators and quantum states on any qubit system comprised of more than 1 qubit. This operation will continually be used throughout this thesis.

Tensor Product. *Suppose A is an $m \times n$ matrix and B a $p \times q$ matrix. The tensor product of A and B then results in a $nq \times mp$ matrix* [7],

$$F = A \otimes B = \begin{bmatrix} A_{11}B & A_{12}B & \dots & A_{1n}B \\ A_{21}B & \ddots & \ddots & A_{2n}B \\ \vdots & \ddots & \ddots & \vdots \\ A_{m1}B & \dots & \dots & A_{mn}B \end{bmatrix} = \begin{bmatrix} A_{11}B_{11} & A_{12}B_{12} & \dots & A_{1n}B_{1q} \\ A_{11}B_{21} & \ddots & \ddots & A_{1n}B_{2q} \\ \vdots & \ddots & \ddots & \vdots \\ A_{m1}B_{p1} & \dots & \dots & A_{mn}B_{pq} \end{bmatrix}, \quad (2.5)$$

$$F \in \mathbb{C}^{nq \times mp}. \quad (2.6)$$

The notation for a tensor product of qubits used throughout this thesis is,

$$|\psi_a\rangle \otimes |\psi_b\rangle = |\psi_a \psi_b\rangle \,. \quad (2.7)$$

Pauli Matrices. It is also important to define the Pauli Matrices/gates (along with the 2×2 identity matrix), which are unitary operators that perform transformations on an input

1-qubit state ($|\psi\rangle$) on the Bloch Sphere. These are crucial as they represent some of the most used 1-qubit quantum gates. These will be defined throughout this thesis as [15],

$$\sigma^X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \quad \sigma^Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}, \quad \sigma^Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}, \quad I = \sigma^I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}. \tag{2.8}$$

Eq.'s (2.5) and (2.8) are crucial to describe the behavior of a quantum system as any Hamiltonian can be written as a combination of tensor products of Pauli matrices. Formally, for an n-qubit system, the Hamiltonian, the operator which governs the behavior through time of our quantum state, can be written as [16],

$$\hat{H} = \sum_{j=0}^{k} h_j M_{j,1} \otimes M_{j,2} \otimes ... \otimes M_{j,n}, \quad h_j \in \mathbb{R}, \quad M_{j,l} \in \{I, \sigma^X, \sigma^Y, \sigma^Z\} \ \ \forall \ \ j, l, \tag{2.9}$$

where k is proportional to the depth of the circuit (the number of U_i blocks in Fig. 2.1).

In the Bloch Sphere representation (Eq. (2.4)), these Pauli Matrices all correspond to a $\theta = \pi$ phase rotation around their respective axes. As an example, given a 1-qubit input state $|0\rangle$,

$$\sigma^X |0\rangle = |1\rangle, \quad \sigma^Y |0\rangle = i |1\rangle, \quad \sigma^Z |0\rangle = |0\rangle. \tag{2.10}$$

1-Qubit Rotation Gates. For an arbitrary θ rotation, each matrix in Eq. (2.8) can be parametrized by θ, resulting in the rotation operators [7],

$$R_X(\theta) = \exp\left\{-\frac{i\theta\sigma^X}{2}\right\} = \cos\frac{\theta}{2}\sigma^I - i\sin\frac{\theta}{2}\sigma^X = \begin{bmatrix} \cos\frac{\theta}{2} & -i\sin\frac{\theta}{2} \\ -i\sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{bmatrix} \tag{2.11}$$

$$R_Y(\theta) = \exp\left\{-\frac{i\theta\sigma^Y}{2}\right\} = \cos\frac{\theta}{2}\sigma^I - i\sin\frac{\theta}{2}\sigma^Y = \begin{bmatrix} \cos\frac{\theta}{2} & -\sin\frac{\theta}{2} \\ -\sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{bmatrix} \tag{2.12}$$

$$R_Z(\theta) = \exp\left\{-\frac{i\theta\sigma^Z}{2}\right\} = \cos\frac{\theta}{2}\sigma^I - i\sin\frac{\theta}{2}\sigma^Z = \begin{bmatrix} e^{-\frac{i\theta}{2}} & 0 \\ 0 & e^{\frac{i\theta}{2}} \end{bmatrix}. \tag{2.13}$$

CNOT Gate.

The Controlled-Not (CNOT) gate is a 2-qubit quantum that involves the state of a "target" qubit depdendent on what the state of some "control" qubit is. This gate flips the bit of some target qubit if the "control" qubit is in the state $|1\rangle$. Thus, the space of possibilities for a CNOT operation is,

$$CNOT\,|00\rangle = |00\rangle \quad CNOT\,|01\rangle = |01\rangle \quad CNOT\,|10\rangle = |11\rangle \quad CNOT\,|11\rangle = |10\rangle, \tag{2.14}$$

assuming the input state is $|z_{control} z_{target}\rangle$. For completeness, the matrix representation of the CNOT gate is[7],

$$CNOT = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}. \tag{2.15}$$

The CNOT and rotation gates will will provide clear intuition for the alternate construction of the cost Hamiltonian, $\hat{H}_C$, in Fig. 3.3.

2.1.2 Multi-qubit Quantum Systems and Evolution

An n-qubit system combines each individual qubit using tensor products and can be represented as a superposition of all $N = 2^n$ basis states,

$$|\psi_{tot}\rangle = \sum_{i=0}^{N} c_i |\mathbf{z}_i\rangle, \quad \mathbf{z}_i \in \{0,1\}^n, \quad |\psi_{tot}\rangle \in \mathbb{C}^N. \tag{2.16}$$

Example: 3-qubit System. Using Eq. (2.16), a 3-qubit system can generally be described as,

$$|\psi_{tot}\rangle = c_0|000\rangle + c_1|001\rangle + c_2|010\rangle + c_3|011\rangle + c_4|100\rangle + c_5|101\rangle + c_6|110\rangle + c_7|111\rangle, \tag{2.17}$$

$$|\psi_{tot}\rangle = c_0 \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} + c_1 \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} + c_2 \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} + c_3 \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} + c_4 \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} + c_5 \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} + c_6 \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} + c_7 \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \\ c_6 \\ c_7 \end{bmatrix}. \tag{2.18}$$

In gate-model quantum computation, quantum gates perform operations on the input quantum state (some n-qubit system, or register), manipulating the system's amplitudes in such a way that leaves the output quantum state encoding the desired answer to the problem with high probability. All quantum gates are reversible and thus can be represented as a unitary operator on an input state $|\psi_0\rangle$. This unitary operator represents the time evolution of $|\psi_0\rangle$ governed by some Hamiltonian, $\hat{H}$, and is defined as[15],

$$\hat{U} = \exp\left\{-\frac{i\hat{H}t}{\hbar}\right\}. \tag{2.19}$$

For completeness, the matrix representation of the Hamiltonian is defined as,

$$\hat{H} = \begin{bmatrix} \langle 1|\hat{H}|1\rangle & \langle 1|\hat{H}|2\rangle & \dots & \langle 1|\hat{H}|N\rangle \\ \langle 2|\hat{H}|1\rangle & \ddots & \ddots & \langle 2|\hat{H}|N\rangle \\ \vdots & \ddots & \ddots & \vdots \\ \langle N|\hat{H}|1\rangle & \dots & \dots & \langle N|\hat{H}|N\rangle \end{bmatrix} = \begin{bmatrix} \hat{H}_{11} & \hat{H}_{12} & \dots & \hat{H}_{1j} \\ \hat{H}_{21} & \ddots & \ddots & \hat{H}_{2j} \\ \vdots & \ddots & \ddots & \vdots \\ \hat{H}_{N1} & \dots & \dots & \hat{H}_{NN} \end{bmatrix}, \tag{2.20}$$

$$\hat{H}_{ij} = \langle i|\hat{H}|j\rangle, \tag{2.21}$$

where the basis states, $|\boldsymbol{z}_{int}\rangle$, are,

$$|\boldsymbol{z}_{int}\rangle = |z_1 z_2 ... z_n\rangle = |z_1\rangle \otimes |z_2\rangle \otimes ... \otimes |z_n\rangle. \tag{2.22}$$

z_{int} specified here is just the base-10 (integer) representation each n-bit string representing a basis state in Eq. (2.16),

$$z_{int} = \sum_{j=0}^{n} z_{n-j} 2^{n-j}, \quad z_{n-j} \in \{0, 1\}, \tag{2.23}$$

and was introduced to simplify notation.

Base-2 to Base-10 Example.

$$\left|1(2^3) + 1(2^2) + 0(2^1) + 1(2^0)\right\rangle = |13\rangle. \tag{2.24}$$

The unitary operator in Eq. (2.19) can also be parametrized by some phase ϕ, resulting in its more general representation,

$$\hat{U}(\hat{H}, \phi) = \exp\left\{-\frac{i\phi\hat{H}}{\hbar}\right\}. \tag{2.25}$$

We will revisit the importance of this phase parameter to NISQ algorithms, specifically its relevance to the Quantum Approximate Optimization Algorithm, in Chapter 4.

Once the unitary operator is applied to the quantum state, a measurement operator is then applied which allows us to determine whether we have the desired answer to our encoded problem. In quantum computation, The measurement is almost always in the computational, or Z, basis and we will continue that throughout this thesis. Although the measurement problem is a fascinating and constantly researched area [4], it is not the focus of this thesis and will not be discussed in great detail.

2.2 NISQ Era Quantum Computation

2.2.1 Hardware Challenges

As mentioned in Chapter 1, NISQ era quantum devices today must be engineered to be robust to errors throughout the evolution of the quantum system. Decoherence is the process of the quantum system becoming more and more entangled with its outside environment which causes the quantum state to lose key information encoded within it as it loses its pure state characteristics[4]. This, coupled with errors inherent in each individual quantum gate, amplifies the probability of error in our NISQ devices. Due to this, NISQ era quantum algorithms must contain these crucial properties to succeed on NISQ devices[4]:

1. Low circuit depth (p)
2. Low number of qubits
3. Probability of error per qubit for each measurement is below a certain threshold

The first two criteria will be explored in detail when exploring the QAA and QAOA similarities in Chapter 4.

2.3 Main Research Questions and Motivation

When exploring these NISQ era algorithms, we are motivated by the the main question:

What computational frameworks are best suited for the NISQ era?

To address this, we must first examine the high level computational workflow for any NISQ era algorithm. These are all hybrid quantum-classical, or variational, algorithms which implies that there must be some interaction of information between a QPU and a CPU. The CPU passes some initial parameters into the QPU, parametrizing the input quantum state, $|\psi_0\rangle$. The quantum circuit then gets executed and the measurement information of some observable is then passed back into the CPU. This is repeated and thus forms a closed QPU-CPU iterative loop described in detail in Fig. **??**.

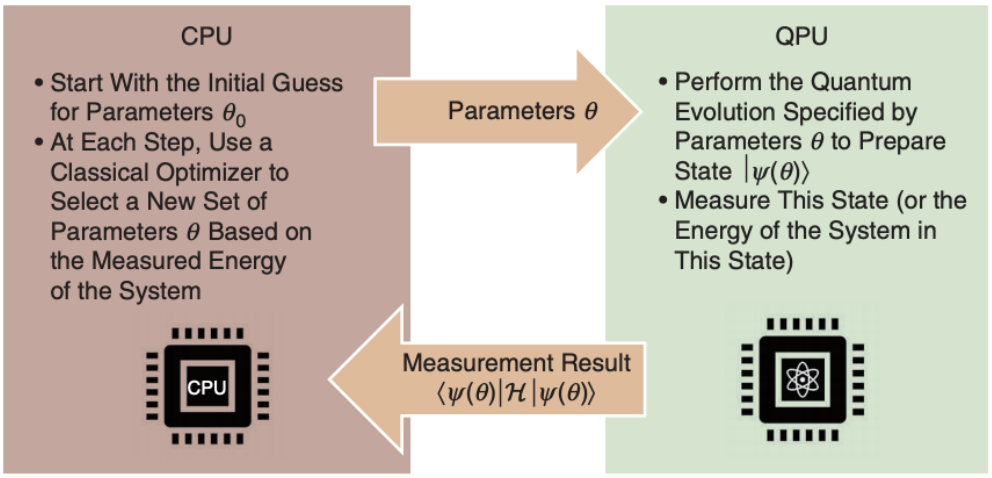

Figure 2.3: Computational Workflow for NISQ era Quantum Computing[17].

With this framework in mind, the development of NISQ era quantum algorithms has resulted in the Variational Quantum Eigensolver (VQE) and the Quantum Approximate Optimization Algorithm (QAOA) mentioned in Chapter 1.1, the two most promising for a quantum advantage on NISQ devices.

2.3.1 Variational Quantum Eigensolver

The Variational Quantum Eigensolver (VQE)[9] is a NISQ era algorithm used to solve quantum many-body systems. These problems cannot be accurately solved with classical algorithms and thus there is no classical bench-mark to compare the results of this algorithm to. This is an optimization algorithm that parametrizes a quantum circuit based on some **prepared input quantum state** and varies this parameter until a minimum is found for the ground state energy, E_0, of a quantum system. This stems from the Variational Principle, which states,

If any normalizable function $|\psi\rangle$ is chosen, an upper bound on the ground state, E_0, can be found [15],

$$E_0 \leq \langle\psi|\,\hat{H}\,|\psi\rangle\,. \tag{2.26}$$

It is a hybrid algorithm because there is an outer loop classical optimizer changing the variational parameter for each new iteration accodring to some objective, or cost, function. We will see that his this outer optimization loop on a CPU is a feature shared between VQE and QAOA and is an open area of research [18]. Although there has been in depth simulations of VQE on various problems[9], this is not the quantum algorithm of interest for this thesis and will not be explored.

2.3.2 Quantum Approximate Optimization Algorithm (QAOA)

When VQE is applied to combinatorial optimization problems, the tunable parameters (optimized by the outer loop CPU) are optimized **over an evolution with discrete time steps**, resulting in QAOA. These time steps are represented by p circuit layers, each layer containing parametrized unitary operators. QAOA is the discretized version of the Quantum Adiabatic Algorithm (QAA), that uses a non-parametrized optimization schedule function to find the solution to some quantum system's evolution, and is discussed in Chapter 4.

QAOA can be seen as a special case of VQE, one that solves combinatorial optimization problems versus quantum-many body problems. But for MaxCut, QAOA is the better NISQ algorithm to use due to its connections with QAA. Due to the variational nature of NISQ algorithms, QAOA can also be seen as optimizing a parametrized schedule function, adding a layer of complexity relative to QAA. The details of these parameters in QAOA and how they propagate throuhgout the QPU-CPU framework is detailed in Chapter 3.4.

Chapter 3

Analytical Analysis on QAOA

3.1 Intro to QAOA

The Quantum Approximate Optimization Algorithm was first created by Farhi et. al [1] and was designed to approximate optimal solutions to existing discrete, combinatorial optimization problems. It is mainly applied to classical NP-hard computational problems (MaxCut, 3-SAT, etc.) and thus can be benchmarked with the best classical algorithms on these same problems.

QAOA involves a cost function, so if we recall Eq. (1.8), we can write this even more generally as the sum of all the constraints, or clauses, for some input graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$. These clauses will take on some higher value, if satisfied, and some lower value if not satisfied,

$$C_m(\boldsymbol{z}) = \begin{cases} q_1 & \text{if } C_m \text{ is satisfied by } \boldsymbol{z} \\ q_2 & \text{otherwise} \end{cases}, \quad q_1 \geq q_2.$$

With this formulation, QAOA on MaxCut can be viewed as a maximization problem of the total number of satisfied clauses.

3.2 Entire QAOA CPU-QPU Process

At a high level, the most general QAOA process can be summarized as follows:

1. (CPU) Initialize parameters $(\boldsymbol{\beta}, \boldsymbol{\gamma})$ randomly or with specific values through other techniques

2. (QPU) Start with a uniform superposition of the n-qubit register:
$$|\psi_0\rangle = H^{\otimes n} |0\rangle^{\otimes n}.$$

3. (QPU) Construct a parametrized quantum circuit (PQC) using the most recent $2p$ $(\boldsymbol{\beta}, \boldsymbol{\gamma})$ parameters, resulting in:
$$|\boldsymbol{\beta}, \boldsymbol{\gamma}\rangle = \hat{U}(\hat{H}_D, \beta_p)\hat{U}(\hat{H}_C, \gamma_p)...\hat{U}(\hat{H}_D, \beta_0)\hat{U}(\hat{H}_C, \gamma_0)H^{\otimes n} |0\rangle^{\otimes n}.$$

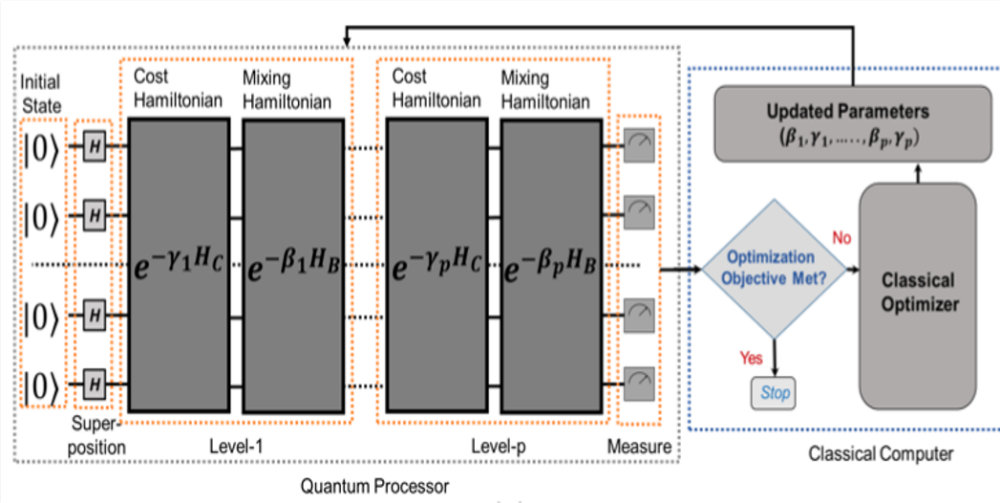

Figure 3.1: End-to-end QAOA process [19]

4. (QPU) Sample from output state, calculating the expectation value of cost Hamiltonian:
$$\langle \boldsymbol{\beta}, \boldsymbol{\gamma} | \hat{H}_C | \boldsymbol{\beta}, \boldsymbol{\gamma} \rangle .$$

5. (CPU) Choose the next set of $(\boldsymbol{\beta}, \boldsymbol{\gamma})$ parameters through the desired classical optimization technique.

6. Repeat (1)-(4) until the optimal $(\boldsymbol{\beta}, \boldsymbol{\gamma})$ parameters are found that successfully approximate the optimal MaxCut cut value (Φ_{opt}).

Fig. 3.1 encapsulates all of these steps and each will be explained in detail with their own sections in Chapter 3.

3.3 QAOA on MaxCut

Building off of Chapter 1.3, we will constrain our subspace of input $\mathcal{G}(\mathcal{V}, \mathcal{E})$ graphs to unweighted n-node, d-regular graphs. The "d" parameter is the **degree** of a node, or in an unweighted graph is the amount of edges coming out of it. A regular graph is one where each node in some $\mathcal{G}(\mathcal{V}, \mathcal{E})$ has the same degree.

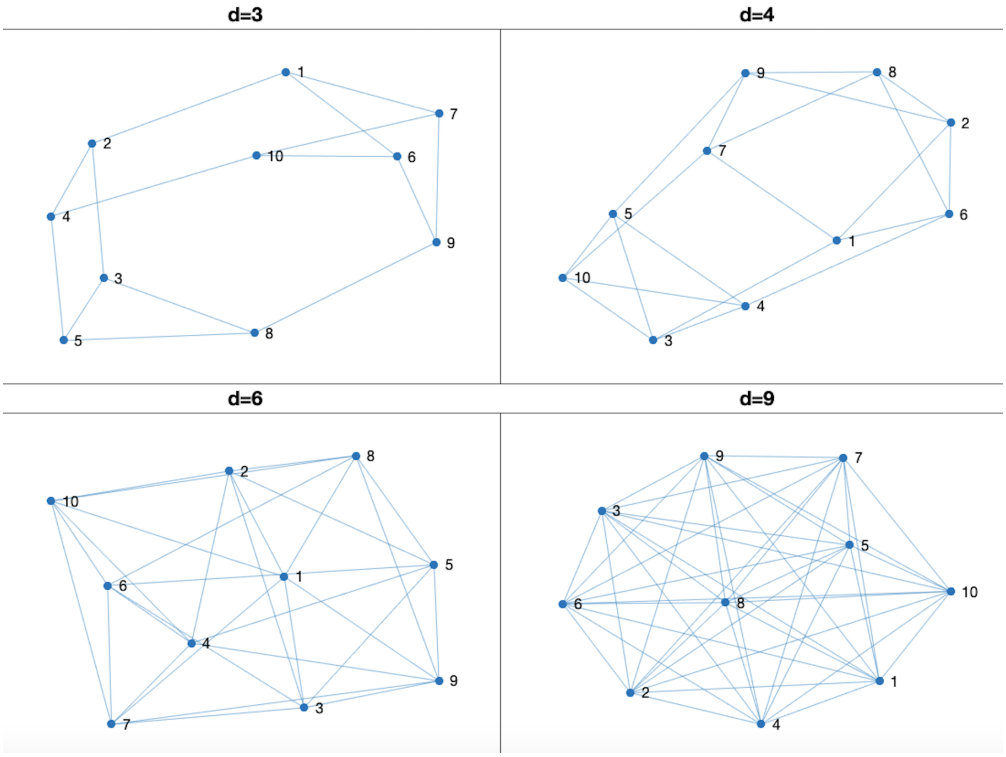

Figure 3.2: A 10-node $\mathcal{G}(\mathcal{V}, \mathcal{E})$ instance with different d-regular connectivity. The $d = 9$ case is an example of a fully connected simple graph as each node has the maximum amount of $n - 1$ edges.

3.3.1 Initial state, $|\psi_0\rangle$

Each bit in the input n-bit string, $\boldsymbol{z}$, of the specific optimization problem is mapped to a qubit. In other words, each node in our input graph is represented by a qubit. This results in an n-qubit register as the input state of QAOA,

$$\text{Given: } \boldsymbol{z} = z_1 ... z_n$$

$$\boldsymbol{z} \rightarrow |\boldsymbol{z}\rangle \in \mathbb{C}^{2^n}.$$

This input n-qubit state is usually initialized initialized as $|0\rangle^{\otimes n}$, where,

$$|0\rangle^{\otimes n} = |0\rangle \otimes ... \otimes |0\rangle \quad (n \text{ times}),$$

in the form of Eq. (2.16). This is then operated on by the Hadamard transform, a unitary operator that manipulates the state so it is in a uniform superposition of all basis states[7]. In other words, if measured in this state, finding the optimal bit string, $\boldsymbol{z_{opt}}$, would be equivalent to enumerating all states as each has equal probability. The Hadamard transform is defined as,

$$|\psi_0\rangle = |+\rangle^{\otimes n} = H^{\otimes n}\,|0\rangle^{\otimes n}\,, \text{ where } \quad H = \frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}.$$

This operator can alternatively be written in terms of the basis states of $|\psi_0\rangle$,

$$|\psi_0\rangle = \frac{1}{\sqrt{2^n}}\sum_{\boldsymbol{z}}|\boldsymbol{z}\rangle = \frac{1}{\sqrt{2^n}}(|00...0\rangle + |00...1\rangle + ... + |11...1\rangle).$$

3.3.2 Driving Hamiltonian Formulation, $\hat{H}_D$

This encoding of the MaxCut problem can then be transformed into a quantum system by two main expressions: The driving Hamiltonian ($\hat{H}_D$), which is an easily solvable system, and the cost Hamiltonian ($\hat{H}_C$). The driving Hamiltonian is a parametrized σ^X gate that is applied to all qubits,

$$\hat{H}_D = \sum_{i=0}^{n}\sigma_i^X, \tag{3.1}$$

where,

$$\hat{H}_D = \sigma_0^x \otimes ... \otimes \sigma^I + \sigma^I \otimes \sigma_1^x \otimes ... \otimes \sigma^I + ... + \sigma^I \otimes ... \otimes \sigma_n^x. \tag{3.2}$$

This agrees with our general definition of a Hamiltonian in Eq. (2.9) as this is expanded into a summation of tensor products of Pauli matrices.

The structure of $\hat{H}_D$ is independent of any MaxCut problem instance and scales with the total number of nodes and p circuit blocks (described in Chaper 4). It is chosen as it's very easy to implement on NISQ devices and does not commute with the cost Hamiltonian[1], $\hat{H}_C$, which we will construct in the next section. $\hat{H}_D$ is then parametrized by a rotation angle β_0. Using Eq. (2.19),

$$\hat{U}(\hat{H}_D, \beta_0) = \exp\Big\{-i\beta_0\hat{H}_D\Big\} = \prod_{i=0}^{n}\exp\{-i\beta_0\sigma_i^X\}.$$

3.3.3 Cost Hamiltonian Formulation, $\hat{H}_C$

The formulation of the cost Hamiltonian is similar to Eq. (1.8) and is dependent on the characteristics of the input $\mathcal{G}(\mathcal{V}, \mathcal{E})$ graph. As stated in Chapter 1.3, if two nodes share an edge, one being in S and the other in $\bar{S}$, then the edge weight value, w_{jk} should be added to the cost function (here we are still assuming $w_{jk} = 1$). With this, Eq.(1.8) transforms into,

$$\hat{H}_C = \sum_{(z_j, z_k)\in\mathcal{E}_{bin}}\hat{C}_{(j,k)\in\mathcal{E}_{bin}}, \;\; \hat{C}_{(j,k)\in\mathcal{E}_{bin}} = \frac{1}{2}(\sigma^{I^{\otimes n}} - \sigma_j^Z \otimes \sigma_k^Z), \tag{3.3}$$

where $\sigma^{I^{\otimes n}}$ is the $2^n \times 2^n$ identity matrix and will be denoted as $\mathbb{I}$.

This Hamiltonian involves the sum of all clauses, where each edge in $\mathcal{E}_{bin}$ in the input $\mathcal{G}(\mathcal{V}, \mathcal{E})$ is a clause. The quantum analog to the classical cut value, $\Phi_{\boldsymbol{z}}$, is the expectation value of the cost Hamiltonian,

$$\begin{aligned}\langle \hat{H}_C \rangle &= \langle \boldsymbol{z} | \hat{H}_C | \boldsymbol{z} \rangle \\ &= \langle z_a z_b | \hat{C}_{a,b} | z_a z_b \rangle + ... + \langle z_f z_g | \hat{C}_{f,g} | z_f z_g \rangle \quad \forall z_j z_k \in \mathcal{E}_{bin} \\ &= \langle z_a z_b | \frac{1}{2}(\mathbb{I} - \sigma_a^Z \otimes \sigma_b^Z) | z_a z_b \rangle + ... + \langle z_f z_g | \frac{1}{2}(\mathbb{I} - \sigma_a^Z \otimes \sigma_b^Z) | z_f z_g \rangle \quad \forall z_j z_k \in \mathcal{E}_{bin}.\end{aligned} \tag{3.4}$$

Since z_j, z_k are binary values, there are only four possible $|z_j z_k\rangle$ states each clause can operate on. For simplicity, I've "truncated" the basis states to 2-qubit states instead of n-qubit states. This was done because for all nodes besides the two sharing an edge, the trivial σ^I gate is applied to them. For the two nodes evolving non-trivially, the total possible expectation value clause outcomes are (Eq. (A.4)-(A.6)),

$$\langle 00 | \hat{C} | 00 \rangle = 0 \quad \langle 10 | \hat{C} | 10 \rangle = 1 \quad \langle 01 | \hat{C} | 01 \rangle = 1 \quad \langle 11 | \hat{C} | 11 \rangle = 0, \tag{3.5}$$

or more concisely,

$$\langle \hat{C}_{(j,k)\in\mathcal{E}_{bin}} \rangle = \begin{cases} 0, & |z_j\rangle = |z_k\rangle \\ 1, & \text{otherwise} \end{cases}.$$

This is then parametrized by an angle rotation, γ_0, resulting in a unitary operator definition of,

$$\begin{aligned}\hat{U}(\hat{H}_C, \gamma_0) &= \exp\Big\{-i\gamma_0 \hat{H}_C\Big\} \\ &= \prod_{\langle j,k\rangle \in E} \exp\Big\{-i\gamma_0 \hat{C}_{(j,k)\in\mathcal{E}_{bin}}\Big\}.\end{aligned}$$

3.4 General Quantum State, $|\boldsymbol{\beta}, \boldsymbol{\gamma}\rangle$, for arbitrary p

$\hat{H}_C$ is applied before the $\hat{H}_D$, resulting in a final quantum state that can be described as,

$$|\Psi\rangle = \hat{U}(\hat{H}_D, \beta_0)\hat{U}(\hat{H}_C, \gamma_0) H^{\otimes n} |0\rangle^{\otimes n}.$$

In Fig. 3.3, the binary edge set, $\mathcal{E}_{bin}$, is,

$$\mathcal{E} = \{(z_0, z_1), (z_0, z_2), (z_1, z_3), (z_2, z_3)\}.$$

The cost Hamiltonian for this particular $\mathcal{G}(\mathcal{V}, \mathcal{E})$ instance is then,

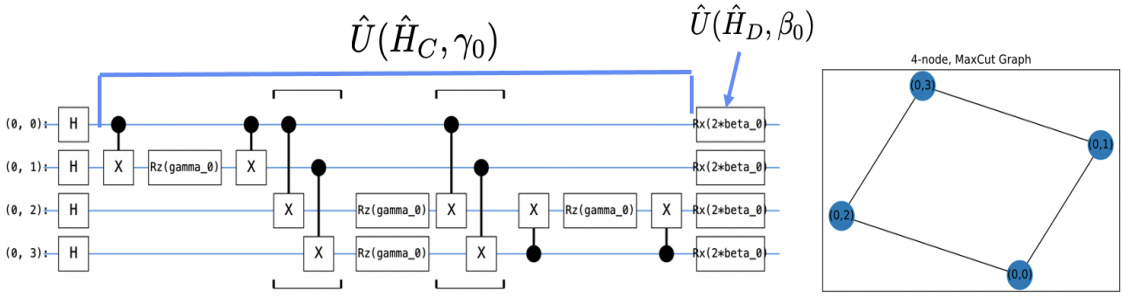

Figure 3.3: QAOA on MaxCut circuit diagram for a 4-node, 2-regular graph, $p = 1$. Note here that the clauses, $\hat{C}_{(j,k)\in\mathcal{E}_{bin}}$, are alternatively represented as a CNOT gate, followed by an $R_z(\gamma_0)$ rotation gate, followed by the same CNOT gate. This construction is **equivalent** to what's defined in Eq. (3.3) and is popularly used in QAOA simulations [3]. This simply means that the cost Hamiltonian can alternatively be represented with these quantum gates.

$$\hat{H}_C = \frac{1}{2}[(\mathbb{I} - \sigma_0^Z\sigma_1^Z) + (\mathbb{I} - \sigma_0^Z\sigma_2^Z) + (\mathbb{I} - \sigma_1^Z\sigma_3^Z) + (\mathbb{I} - \sigma_2^Z\sigma_3^Z)]$$

$$\hat{H}_C = \frac{1}{2}[(\mathbb{I} - \sigma^Z \otimes \sigma^Z \otimes \sigma^I \otimes \sigma^I) + (\mathbb{I} - \sigma^Z \otimes \sigma^I \otimes \sigma^Z \otimes \sigma^I) + (\mathbb{I} - \sigma^I \otimes \sigma^Z \otimes \sigma^I \otimes \sigma^Z) + (\mathbb{I} - \sigma^I \otimes \sigma^I \otimes \sigma^Z \otimes \sigma^Z)], \quad (3.6)$$

where $\mathbb{I}$ denotes the 16×16 identity matrix in this case. As a matrix representation, $\hat{H}_C$ becomes,

$$\hat{H}_C = \frac{1}{2}\begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 6 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 6 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 6 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 6 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 6 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 6 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 4 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 4 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 4 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}$$

To output an even more accurate approximation, these alternating cost and driving Hamiltonians are applied successively for p blocks, each block containing a specific set of rotation angles (β_h, γ_h). This results in the input state, $|\psi_0\rangle$, being parametrized by $2p$ rotation angles $(\boldsymbol{\beta}, \boldsymbol{\gamma} \in \mathbb{R}^p)$. The general QAOA unitary operator that encapsulates these p blocks is then,

$$\hat{U}(\boldsymbol{\beta}, \boldsymbol{\gamma}) = \prod_{h=0}^{p} \exp\Big\{-i\beta_h \hat{H}_D\Big\} \exp\Big\{-i\gamma_h \hat{H}_C\Big\}, \quad \boldsymbol{\beta}, \boldsymbol{\gamma} \in \mathbb{R}^p.$$

This allows us to construct the entire end-to-end QAOA parametrized quantum circuit as,

$$|\boldsymbol{\beta}, \boldsymbol{\gamma}\rangle = \hat{U}(\hat{H}_D, \beta_p)\hat{U}(\hat{H}_C, \gamma_p)...\hat{U}(\hat{H}_D, \beta_0)\hat{U}(\hat{H}_C, \gamma_0)H^{\otimes n} |0\rangle^{\otimes n}.$$

Once this parametrized unitary operator is applied, the output state is then measured in the Z basis, resulting in the expectation value of the specific objective function for the problem instance at hand. Since this objective function has been mapped to the cost Hamiltonian, the expectation value is then,

$$\langle \hat{H}_C \rangle = \langle \boldsymbol{\beta}, \boldsymbol{\gamma}| \hat{H}_C |\boldsymbol{\beta}, \boldsymbol{\gamma}\rangle.$$

Once this expectation value is measured, a CPU is used with classical optimization techniques to choose the next set of $(\boldsymbol{\beta}, \boldsymbol{\gamma})$ parameters. The algorithm is then repeated l times until a $(\boldsymbol{\beta}, \boldsymbol{\gamma})$ set is found that provides an n-bit string, $\boldsymbol{z^*}$, that results in an expectation value, Φ^*, satisfying a minimum approximation ratio (Eq. (1.8)),

$$r = \frac{\Phi^*}{\Phi_{opt}}.$$

The most straightforward optimization technique is a brute force parameter search, or sweeping the $(\boldsymbol{\beta}, \boldsymbol{\gamma})$ parameters over some interval and finding the combination of these $2p$ parameters that provides the best n-bit string, $\boldsymbol{z}$. As the number of parameters increase (proportional with the increase in the p parameter), this optimization method becomes too computationally costly[20] and other classical optimization methods must be explored.

As an example, the cost Hamiltonian expectation value can be fed into a loss function that's a component of a hybrid quantum-classical machine learning model,

$$loss(\boldsymbol{\beta}, \boldsymbol{\gamma}) = loss(\langle \boldsymbol{\beta}, \boldsymbol{\gamma}| \hat{H}_C |\boldsymbol{\beta}, \boldsymbol{\gamma}\rangle).$$

This loss function mapping can then be minimized through the training of the model over several iterations (or epochs) to ideally result in the optimal parameter set $(\boldsymbol{\beta^*}, \boldsymbol{\gamma^*})$ that maximizes $\Phi_{\boldsymbol{z}}$.

3.5 MaxCut Simulation for $p = 1$ layers

In this section we describe an example of MaxCut for a 6-node, 3-regular graph and use the TensorFlow Quantum API to create a hybrid quantum-classical machine learning model with an outer loop loss function for this instance.

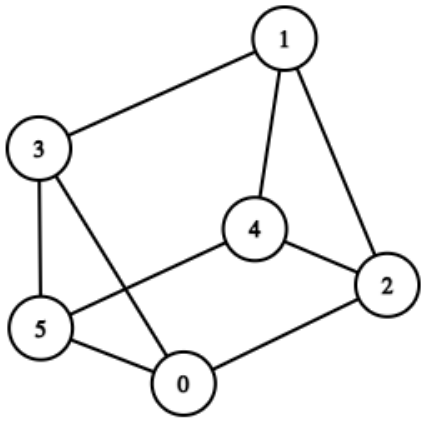

Figure 3.4: The 6-node, 3-regular graph used for the TensorFlow Quantum Model

Following the description of QAOA in earlier sections, we can start by constructing the driving Hamiltonian, which just applies a σ^X gate to all qubits,

$$\begin{aligned}\hat{H}_D = \sigma^x \otimes \sigma^I \otimes \sigma^I \otimes \sigma^I \otimes \sigma^I \otimes \sigma^I + \sigma^I \otimes \sigma^x \otimes \sigma^I \otimes \sigma^I \otimes \sigma^I \otimes \sigma^I \\ + \sigma^I \otimes \sigma^I \otimes \sigma^x \otimes \sigma^I \otimes \sigma^I \otimes \sigma^I + \sigma^I \otimes \sigma^I \otimes \sigma^I \otimes \sigma^x \otimes \sigma^I \otimes \sigma^I \\ + \sigma^I \otimes \sigma^I \otimes \sigma^I \otimes \sigma^I \otimes \sigma^x \otimes \sigma^I + \sigma^I \otimes \sigma^I \otimes \sigma^I \otimes \sigma^I \otimes \sigma^I \otimes \sigma^x. \quad (3.7)\end{aligned}$$

Using notation from Eq. (3.2),

$$\hat{H}_D = \sigma_0^X + \sigma_1^X + \sigma_2^X + \sigma_3^X + \sigma_4^X + \sigma_5^X. \quad (3.8)$$

The cost Hamiltonian for this instance is then,

$$\begin{aligned}\hat{H}_C = \frac{1}{2}[(\mathbb{I} - \sigma_0^Z\sigma_5^Z) + (\mathbb{I} - \sigma_0^Z\sigma_3^Z) + (\mathbb{I} - \sigma_0^Z\sigma_2^Z) + (\mathbb{I} - \sigma_5^Z\sigma_3^Z) + (\mathbb{I} - \sigma_5^Z\sigma_4^Z) + (\mathbb{I} - \sigma_2^Z\sigma_4^Z) \\ + (\mathbb{I} - \sigma_2^Z\sigma_1^Z) + (\mathbb{I} - \sigma_3^Z\sigma_1^Z) + (\mathbb{I} - \sigma_1^Z\sigma_4^Z)]. \quad (3.9)\end{aligned}$$

Since this QAOA circuit has $p = 1$ layers, $\boldsymbol{\beta}, \boldsymbol{\gamma} \in \mathbb{R}$. These parameters were randomly initialized and were updated via a gradient descent method after each iteration of the $\hat{H}_C$ expectation value, $\langle \hat{H}_C \rangle$, was passed into the outer loop optimizer.

3.5.1 QAOA on MaxCut for $p = 1$ layers Python Implementation in TensorFlow Quantum

For the outer loop classical optimizer, the gradient descent algorithm chosen was the Adaptive Moment Estimation (Adam) optimizer. This is an adaptive learning rate method that is commonly accepted as the best overall optimization technique using gradient descent for any type of input data[21]. Fig 3.5 is a detailed description of each component that was contained in the hybrid quantum-classical ML model for QAOA using TensorFlow Quantum.

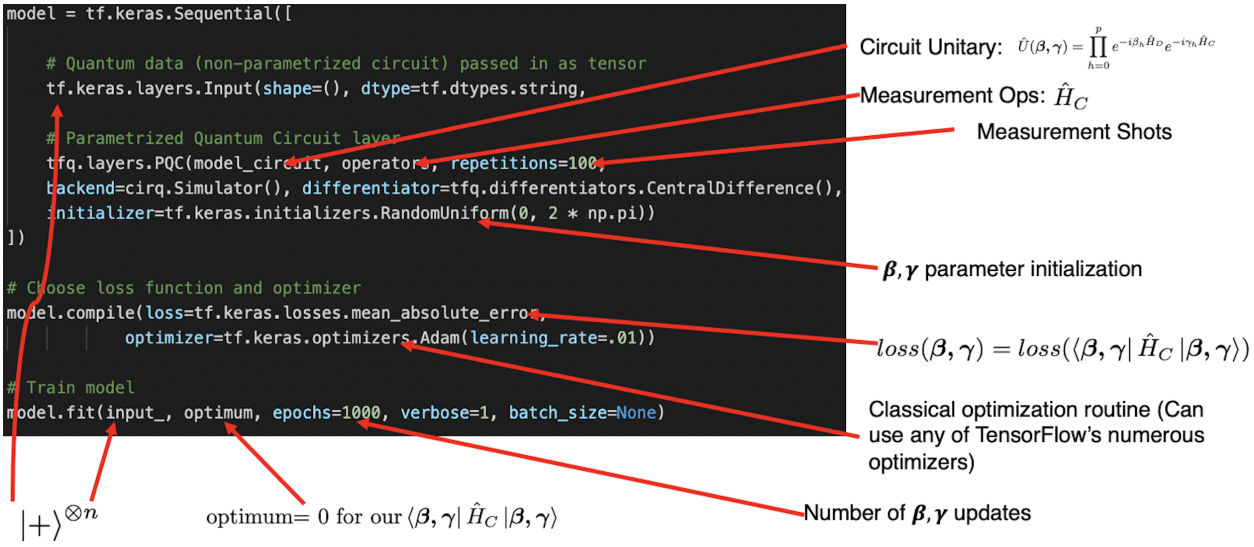

Figure 3.5: TensorFlow Quantum QAOA on MaxCut ML Model

After training, the model was bench-marked by the approximation ratio and the probability of each basis state in our quantum state, each corresponding to a possible cut/partition of nodes. Fig 3.6 depicts the results of the simulation. When sampling 100 measurement outcomes, the optimal bit-strings occurred with sufficient frequency to yield an average cut of 5.98 and an approximation ratio of .8551. Note that averaging over the **whole possible outputs** yielded the expectation value of 3.061.

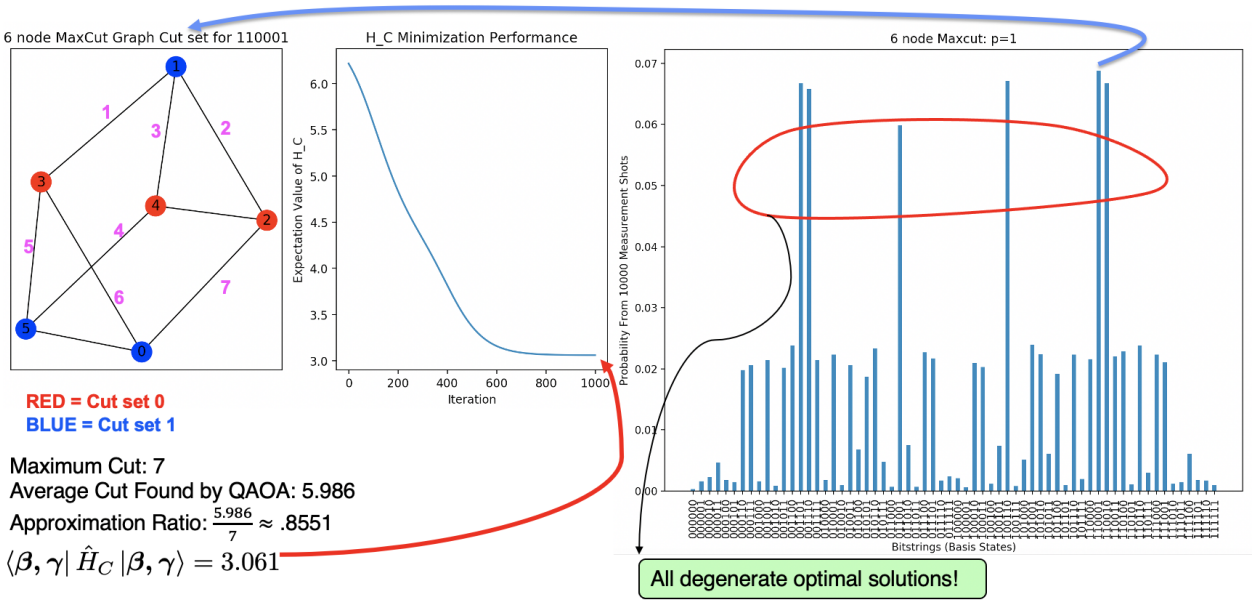

Figure 3.6: Results using TFQ

From Fig 3.6, there are 6 optimal 6-bit strings, resulting in a 6-fold degenerate cut value of,

$$\langle \hat{H}_C \rangle = 7. \tag{3.10}$$

All of these $\boldsymbol{z}_{opt}$ strings are the most probable out of our space of 64 possible $(S, \bar{S})$ configurations. It's important to note that the average cut is the mean of **all** resulting 6-bit string values found over 1000 measurements. In other words, this value reflects the mean of all the counts in the probability histogram in Fig. 3.6.

Using Eq. (1.11), the maximum cut value for this smaller graph is the optimal value, resulting in an approximation ratio of,

$$r = \frac{\langle \hat{H}_C \rangle}{\Phi_{opt}} = 1. \tag{3.11}$$

Extensions to this model for future research are explained in the Conclusion.

Chapter 4

Quantum Adiabatic Algorithm (QAA)

4.1 Intro to Adiabatic Quantum Computing

Adiabatic quantum computing differs from gate-model quantum computers in that it aims to start in the ground state of a solvable Hamiltonian and slowly evolve into a "problem Hamiltonian" whose ground state encodes the solution to some problem at hand. As mentioned in Chapter 2, this is extremely similar to QAOA except it is a continuous evolution of our input state $|\psi_0\rangle$. Alternatively, adiabatic quantum computation is roughly the construction of QAOA as $p \to \infty$. Formally,

Starting from $|\psi_0\rangle$, an eigenstate of $\hat{H}_D$, we evolve by the time-dependent Hamiltonian going from from $\hat{H}_D \to \hat{H}_C$. If the evolution is sufficiently slow (governed by the parameter T), the final state will be an eigenstate of $\hat{H}_C$.

The quantum state time evolution is governed by the time-dependent Schrödinger equation (TDSE),

$$i\hbar \frac{d\,|\psi(t)\rangle}{dt} = \hat{H}(t)\,|\psi(t)\rangle\,. \tag{4.1}$$

To evolve from a solvable Hamiltonian, $\hat{H}_D$, to the problem Hamiltonian of interest, $\hat{H}_C$, the general Hamiltonian applied to some input quantum state must be,

$$\hat{H}(t) = (1 - s(t))\hat{H}_D + s(t)\hat{H}_C, \quad s(t) \in [0, T],$$

where $s(t)$ is the schedule function with constraints,

$$s(0) = 0, s(T) = 1.$$

The schedule function determines the rate at which the solvable Hamiltonian evolves into the problem Hamiltonian. The usual choice of the schedule function is simply a linear evolution,

$$s(t) = \frac{t}{T}$$

resulting in a general Hamiltonian of,

$$\hat{H}(t) = (1 - \frac{t}{T})\hat{H}_D + \frac{t}{T}\hat{H}_C.$$

The total time T for the adiabatic evolution to occur is a crucial parameter in QAA as it determines the duration of our algorithm, which can be seen as a cost metric for QAA. Finding the optimal time, or lower bound, for T for a system allows us to find an optimal run time for QAA and this is explicitly formulated by the adiabatic theorem.

4.1.1 The Adiabatic Theorem

This theorem puts a bound on the adiabatic evolution time, T, and proves that a time T exists that allows the system to stay in the ground state (E_0) and not cross over to the first excited state (E_1) during the time evolution interval, $[0, T]$.

The Adiabatic Theorem *A quantum system stays in its ground state, $|e_{0,t}\rangle$, given a perturbation if this perturbation acts slowly enough such that there is a minimum gap between the ground state and the first excited state, $|e_{1,t}\rangle$ [22]. Formally,*

$$|\psi_0\rangle = \sum_{n=0}^{N} c_{n,0} |e_{n,0}\rangle \implies |\psi_T\rangle = \sum_{n=0}^{N} c_{n,T} \exp\{-\frac{i}{\hbar}\int_0^{t'} E_{n,t'}dt'\} |e_{n,T}\rangle ,$$

$$\text{iff} \quad \sum_{m\neq n} \frac{2\hbar |\langle e_{m,t}| \frac{d\hat{H}_t}{dt} |e_{n,t}\rangle|}{(E_{n,t} - E_{m,t})^2} \ll 1. \tag{4.2}$$

One important result of this theorem is that given some linear schedule $s(t)$,

$$\lim_{T\to\infty} s(t) = \lim_{T\to\infty} (\frac{t}{T}) = 0. \tag{4.3}$$

Intuitively, this condition states that as the adiabatic time approaches infinity, the total error in the problem of interest, or the probability of resulting in an eigenstate not encoding the problem solution, goes to 0.

4.2 QAA on MaxCut

4.2.1 Relation to $\hat{U}(\hat{H}_D, \boldsymbol{\beta}), \hat{U}(\hat{H}_C, \boldsymbol{\gamma})$ QAOA Operators

The construction of the QAA Hamiltonian for MaxCut is very similar to the QAOA formulation seen in Chapter 3. In fact, we have the same cost and driving Hamiltonians

and we simply do not parametrize our unitary operator with $(\boldsymbol{\beta}, \boldsymbol{\gamma})$ values. This results in a Hamiltonian of,

$$\hat{H}(t) = (1 - s(t))\hat{H}_D + s(t)\hat{H}_C, \tag{4.4}$$

where,

$$\hat{H}_D = \sum_{i=0}^{n} \sigma_i^x,$$

$$\hat{H}_C = \sum_{\langle j,k \rangle \in \mathbb{E}} (\mathbb{I} - \sigma_j^z \sigma_k^z). \tag{4.5}$$

The output quantum state measured after a time T should encode the optimal cut value, Φ_{opt}, for our graph instance.

4.2.2 QAA on MaxCut Algorithm Construction

With the construction of our driving and problem Hamiltonians, Eq. (4.4) can be used to implement QAA on MAXCUT for some n-node, d-regular graph. To simulate the adiabatic process, a small time step dt must used and continually updated as the state continually evolves over the interval $[0, T]$. This parameter is tuned by N_1, so as this gets larger we approximate a smooth, analog adiabatic path with greater accuracy. The algorithm is as follows,

Algorithm 2 QAA on MaxCut

Require: $\hat{H}_D, \hat{H}_C, dt$ (time step), $\mathcal{G}(\mathcal{V}, \mathcal{E}), N_1$ (time step parameter)
  $|\psi\rangle \leftarrow H^{\otimes n} |0\rangle^{\otimes n}$
  $dt \leftarrow \frac{T}{N_1}$
  $t \leftarrow 0$
  for $k \in [0 : T]$ **do**
    $t \leftarrow t + \frac{dt}{2}$
    $\hat{H} \leftarrow (1 - s(t))\hat{H}_D + s(t)\hat{H}_C$
    $t \leftarrow t + \frac{dt}{2}$
    $|\psi\rangle \leftarrow e^{-i\hat{H}dt} |\psi\rangle$
    $E_{avg} \leftarrow \langle \hat{H} \rangle$
  end for
  return $|\psi\rangle$

After the state $|\psi\rangle$ is returned, we are able to find the probabilities of each of the possible n-bit strings ($\boldsymbol{z}$), which are contained in the set of all possible cuts. Consistent with the analysis of the n-bit strings for QAOA on MaxCut, the probability of each cut is given by,

$$\text{Prob}(\boldsymbol{z}) = |c_{\boldsymbol{z}}|^2,$$

$$\text{where } \; c_{\boldsymbol{z}} = \langle \boldsymbol{z}|\psi\rangle \, .$$

This is the standard use of Fourier's trick to calculate the complex coefficients each basis state contributes to $|\psi\rangle$. This would allow us to see if the degenerate optimal n-bit strings for smaller graphs are the most probable states or if there is a different probability distribution relative to what was found with QAOA using ML methods as the outer loop optimizer.

Fig. 4.1 corresponds to the QAA algorithm on MaxCut numerical results for a 6-node, 3-regular graph instance. This numerical simulation resulted in six $\boldsymbol{z}$ values that were most probable, and each result with a cut value of,

$$\Phi_{\boldsymbol{z}} = 7. \tag{4.6}$$

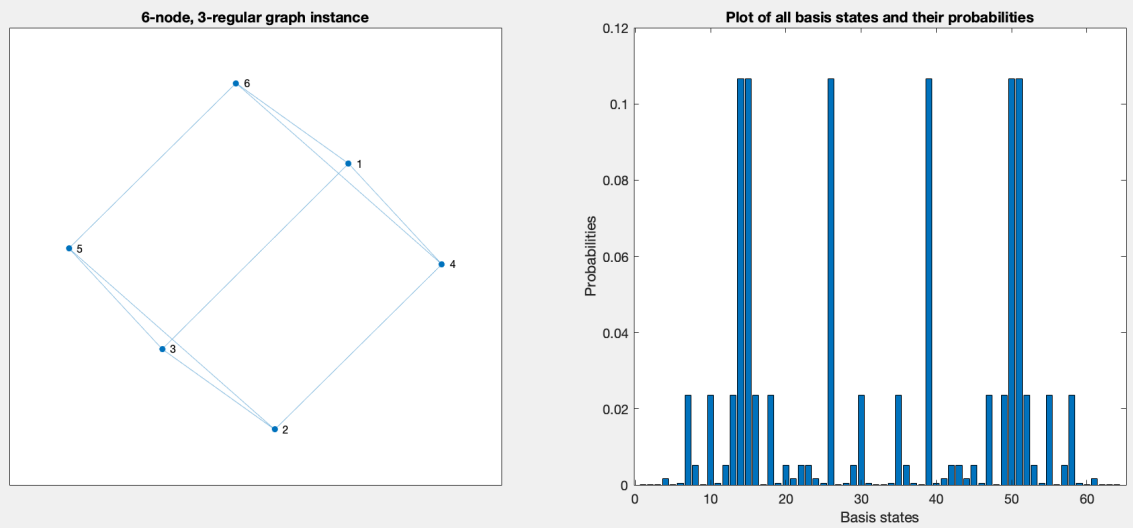

Figure 4.1: Results of QAA on MaxCut for a 6-node, 3-regular graph instance. The most probable 6-bit strings are the 6-fold degenerate $\boldsymbol{z}_{opt}$'s.

In the next section, we explore to concrete relationship between QAA and QAOA, specifically how the $2p$ parameters in QAOA may allow this NISQ era algorithm to have superior performance on MaxCut.

4.3 QAOA and QAA Relation

There are some clear similarities between QAOA and QAA, but one of the main distinctions is the time steps in which each algorithm is run. For QAOA, the "time steps" are represented by the parameter p, which physically is the number of alternating $\hat{U}(\hat{H}_D, \beta_h)\hat{U}(\hat{H}_C, \gamma_h)$ circuit blocks in our QAOA construction. In an ideal adiabatic setting, the input quantum state would evolve in continuous time ($dt \to 0$), but this comes at the trade-off of having an arbitrarily long (and even infinite) adiabatic time, T. If we had this condition, this would allow us to find the optimal n-bit string, $\boldsymbol{z_{opt}}$ satisfying any MaxCut graph instance, but this is computationally infeasible. In the frame of QAOA, this condition is [16],

$$\lim_{p\to\infty} \langle \boldsymbol{\beta}, \boldsymbol{\gamma}| \, \hat{H}_C \, |\boldsymbol{\beta}, \boldsymbol{\gamma}\rangle = \boldsymbol{\Phi_{opt}}.$$

The relationship between the solution path QAOA and QAA take is then fully dependent on the modulation of p layers. When simulating QAA on a classical computer, the dt time step cannot actually become zero and must broken up in to time steps just large enough that the computation is feasible dependent on the CPU used. The nuances between the analog adiabatic algorithm, the simulated adiabatic algorithm, and QAOA are highlighted in Fig. 4.2[5].

Although this suggests that one should just have $p \gg 1$, it is important to highlight two crucial reasons why this is not feasible.

First, recall that QAOA is being run on NISQ devices and one of the fundamental characteristics of a NISQ algorithm is a low total gate count because of the decoherence time constraint. Allowing $p \gg 1$ theoretically provides a strictly better optimal bitstring, $\boldsymbol{z_{opt}}$, for our MaxCut instance, but this is not feasible on NISQ devices. If each step of the evolution is proportional to a 2-qubit gate, the total number of gates scale as nmp, where,

1. n = Total number of qubits
2. m = Total number of clauses in the objective function (Representing the total number of edges for MaxCut)
3. p = Total number of QAOA layers

This gate scaling with p clearly poses a barrier on what is experimentally possible on NISQ devices.

Second, as $p \to \infty$, the space of total $(\boldsymbol{\beta}, \boldsymbol{\gamma})$ parameters also increases linearly (Recall that $(\boldsymbol{\beta}, \boldsymbol{\gamma}) \in \mathbb{R}^p$). This will increase the computational resources for the CPU needed to find the optimal bitstring as there is now a larger and larger solution space for our tunable parameters. Specifically, this not only causes any brute force method to become obsolete but also could result in gradient descent optimizers (Adam, SGD, Adadelta, etc.) getting stuck in a local minima, returning an inferior bit string and thus a worse $\Phi_{\boldsymbol{z}}$ value. Novel gradient descent optimizers and other classical optimizers to combat this problem of QAOA at larger p are continually being explored in the space [19].

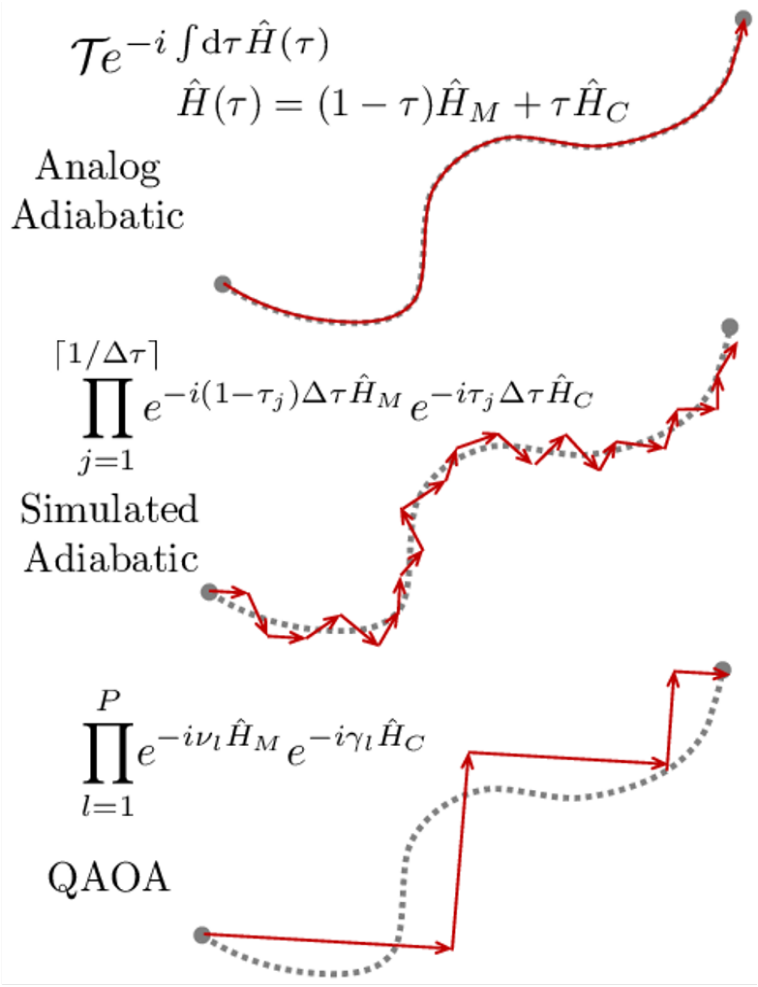

Figure 4.2: QAOA and QAA $\lim_{p\to\infty}$ relation, where τ is our dt [5]

Chapter 5

Conclusion

5.1 Summary of Results

In Chapter 3, the QAOA on MaxCut algorithm was simulated using TensorFlow Quantum, which incorporated Adam as the outer loop classical optimizer. This resulted in all the most probable 6-bit strings being a valid $\boldsymbol{z}_{opt}$ as all had resulting cut values of,

$$\Phi_{\boldsymbol{z}} = 7, \tag{5.1}$$

which demonstrates that using machine learning techniques does in fact yield the correct MaxCut results. This then could be a great alternate to a brute force exhaustive search as the outer loop optimizer, consisting of searching through all possible $(\boldsymbol{\beta}, \boldsymbol{\gamma})$ values.

In Chapter 4, QAA, the continuous analog to QAOA, was introduced and a step-by-step algorithm to implement this was explained. Following the algorithm, QAA was then implemented in MATLAB for another 6-node, 3-regular input $\mathcal{G}(\mathcal{V}, \mathcal{E})$. The results are depicted in Fig. 4.1 which depicts the distribution of all resulting $\boldsymbol{z}$'s during the evolution. Again, this algorithm resulted in the the most probable 6-bitstrings being all of the $\boldsymbol{z}_{opt}$ bit strings, with a corresponding cut value of,

$$\Phi_{\boldsymbol{z}} = 7. \tag{5.2}$$

Intuitively, this demonstrates a successful, efficient search for the MaxCut solution on this smaller, 6-node graph, given the correct adiabatic evolution time, T. The limitations of the continuous framework of QAA was then highlighted in the end of Chapter 4 along with the direct connection between QAA and QAOA.

5.2 AQAOA Hybrid Algorithm

The close similarities between QAOA and QAA spark another area of future research regarding a hybrid QAOA "continuous" algorithm. Gate-model quantum computation constructs quantum algorithms in discrete steps with a discrete set of quantum gates. This

allows the Hamiltonian to be written according to Eq. (2.9). If the adiabatic evolution components of QAA described in Chapter 4 are cleverly combined with the discrete parametrized gate framework for QAOA in Chapter 3, a new "AQAOA" algorithm could be bench-marked on MaxCut.

Specifically for future work, one can construct the new cost and driving Hamiltonians (which may or may not explicitly be feasible on NISQ era QPUs) for this new algorithm and bench-mark it on some input graph. These results can then be compared to the QAOA and QAA implementations described in Chapters 3 and 4 to determine if there are any advantages to this AQAOA algorithm.

5.3 Extensions to QAOA on MaxCut

5.3.1 Use of Different Optimizers

An analysis of a set of different optimizers could be experimentally simulated for QAOA using TensorFlow Quantum[3]. The set of optimizers could include[23],

1. Nelder-Mead
2. BFGS
3. Stochastic Gradient Descent (SGD, as a baseline performance metric)

One can simulate the same input $\mathcal{G}(\mathcal{V}, \mathcal{E})$ graph in Chapter 3.4 and find approximation ratios, average cuts, etc. With these results, one can bench-mark with respect to the results from the Adam optimizer in Fig. 3.6 and determine the most efficient optimizer for this input graph. Since Adam, Adagrad, Adadelta, and RMSProp are very similar optimization algorithms to Adam[23], it may not be totally useful to bench-mark each of their performances on QAOA. TensorFlow Quantum contains numerous other options for varying the outer loop CPU optimizer for QAOA simulation, allowing a user to determine which optimizer truly yields the best $\boldsymbol{z}^*$ while fixing the input graph.

5.3.2 Variations on the input $\mathcal{G}(\mathcal{V}, \mathcal{E})$

Alternatively, one can fix all the parameters (outer loop optimizer, number of measurement shots, etc.) in the TensorFlow Quantum model and vary the input $\mathcal{G}(\mathcal{V}, \mathcal{E})$ graph. In this thesis, the space of all possible graphs was restricted to unweighted, n-node, d-regular graphs, but this can be loosened. Regarding some of the specific graph components, one can vary,

1. Total number of nodes
2. Number of edges

3. Degree of each node (doesn't have to be a regular graph where all nodes have the same degree)

4. Allowed values for edge weights

An analysis of how the scaling each of these components affect the difficulty of finding the optimal n-bit string would then be an interesting simulation to execute for future research.

5.4 QAOA on other Combinatorial Optimization Problems

MaxCut is undoubtedly the most researched computational problem for this QAOA, but there are some implementations of QAOA on other problems as well, such as the Travelling Salesman problem [24]. There have also been alternate NISQ algorithms proposed and constructed to solve these other problems.

As further research using TensorFlow Quantum, one can use this API to map the Travelling Salesman problem (or other problems) onto a QAOA framework. There are numerous parameters of this model that can be tuned, including simulations using different optimizers mentioned in 5.3.1, that can be bench-marked with current results using alternate NISQ algorithms.

Appendix A

Appendix

A.1 Extended Calculations

From Chapter 3.2, the explicit calculations for each expectation value in Eq. (3.5) is below. Note that this is only demonstrating a 2-qubit state but in reality this is an n-qubit state. This was done to explicitly highlight how the clauses work.

$$\langle 00|\,\hat{C}\,|00\rangle = 0 \quad \langle 10|\,\hat{C}\,|10\rangle = 1 \quad \langle 01|\,\hat{C}\,|01\rangle = 1 \quad \langle 11|\,\hat{C}\,|11\rangle = 0, \tag{A.1}$$

The (truncated) matrix representation of the clause function is,

$$\hat{C} = \frac{1}{2}(\mathbb{I} - \sigma_j^Z \sigma_k^Z) = \begin{bmatrix} 0 & -1 & 0 & 0 \\ -1 & 2 & 0 & 0 \\ 0 & 0 & 2 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}, \tag{A.2}$$

resulting in clause expectation values of,

$$\langle 00|\,\hat{C}\,|00\rangle = \frac{1}{2}\begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & -1 & 0 & 0 \\ -1 & 2 & 0 & 0 \\ 0 & 0 & 2 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \frac{1}{2}(0) = 0 \tag{A.3}$$

$$\langle 10|\,\hat{C}\,|10\rangle = \frac{1}{2}\begin{bmatrix} 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & -1 & 0 & 0 \\ -1 & 2 & 0 & 0 \\ 0 & 0 & 2 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} = \frac{1}{2}(2) = 1 \tag{A.4}$$

$$\langle 01|\,\hat{C}\,|01\rangle = \frac{1}{2}\begin{bmatrix} 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 0 & -1 & 0 & 0 \\ -1 & 2 & 0 & 0 \\ 0 & 0 & 2 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} = \frac{1}{2}(2) = 1 \tag{A.5}$$

$$\langle 11|\,\hat{C}\,|11\rangle = \frac{1}{2}\begin{bmatrix}0 & 0 & 0 & 1\end{bmatrix}\begin{bmatrix}0 & -1 & 0 & 0\\ -1 & 2 & 0 & 0\\ 0 & 0 & 2 & 1\\ 0 & 0 & 1 & 0\end{bmatrix}\begin{bmatrix}0\\0\\0\\1\end{bmatrix} = \frac{1}{2}(0) = 0 \tag{A.6}$$

A.2 Adiabatic Theorem Proof

Proof.

1. Consider a Hamiltonian that evolves from a driving (known) Hamiltonian ($\hat{H}_D$) to a problem Hamiltonian ($\hat{H}_C$) and commutes with itself for all times,

$$\hat{H}(t) = (1 - s(t))\hat{H}_D + s(t)\hat{H}_P, \quad [\hat{H}(t), \hat{H}(t')] = 0 \quad \forall t, t' \in [0, T],$$

2. Assume our energy eigenstates are constant in time allowing us to write the state at $t = 0$ as a superposition of them,

$$|\psi_0\rangle = \sum_{n=0}^{N} c_n\,|e_n\rangle\,, \quad N = 2^n, c_n \in \mathbb{C}$$

3. Define $\{\hat{H}(t), E_n(t), |e_n(t)\rangle\}$ for each specific time step t,

$$\hat{H}_t\,|e_{n,t}\rangle = E_{n,t}\,|e_{n,t}\rangle\,, \quad \text{where: } \{\hat{H}_t, E_{n,t}, |e_{n,t}\rangle\} = \{\hat{H}(t), E_n(t), |e_{n,t}\rangle\}.$$

4. The time evolution of our state is given by,

$$|\psi_t\rangle = \sum_{n=0}^{N} \exp\{-\frac{i}{\hbar}\int_0^{t'} E_{n,t'}dt'\}c_{n,t}\,|e_{n,t}\rangle$$

   (a) Solving the TDSE this specific time t yields,

$$i\hbar\frac{d\,|\psi(t)\rangle}{dt} = \hat{H}(t)\,|\psi(t)\rangle \tag{A.7}$$

,

$$i\hbar\frac{d\left|\psi_t\right\rangle}{dt} = i\hbar\sum_{n=0}^{N}\frac{dc_{n,t}}{dt}\exp\{-\frac{i}{\hbar}\int_0^t E_{n,t'}dt'\}\left|e_{n,t}\right\rangle + i\hbar\sum_{n=0}^{N}c_{n,t}\frac{-iE_{n,t}}{\hbar}\exp\{-\frac{i}{\hbar}\int_0^t E_{n,t'}dt'\}\left|e_{n,t}\right\rangle + i\hbar\sum_{n=0}^{N}c_{n,t}\exp\{-\frac{i}{\hbar}\int_0^t E_{n,t'}dt'\}\frac{d\left|e_{n,t}\right\rangle}{dt} \quad \text{(LHS of Eq. (4.3))}, \quad \text{(A.8)}$$

$$\hat{H}_t\left|\psi_t\right\rangle = \sum_{n=0}^{N}\exp\{-\frac{i}{\hbar}\int_0^t E_{n,t'}dt'\}E_{n,t}c_{n,t}\left|e_{n,t}\right\rangle \quad \text{(RHS of Eq. (4.3))}, \quad \text{(A.9)}$$

(b) Equating Eqs. (4.4), (4.4) and simplifying results in,

$$i\hbar\sum_{n=0}^{N}\frac{dc_{n,t}}{dt}\exp\{-\frac{i}{\hbar}\int_0^t E_{n,t'}dt'\}\left|e_{n,t}\right\rangle + i\hbar\sum_{n=0}^{N}c_{n,t}\exp\{-\frac{i}{\hbar}\int_0^t E_{n,t'}dt'\}\frac{d\left|e_{n,t}\right\rangle}{dt} = 0 \quad \text{(A.10)}$$

5. Taking the matrix element of $\langle e_{m,t}|$ on both sides and solving for $\frac{dc_{m,t}}{dt}$,

$$\langle e_{m,t}|\, i\hbar\sum_{n=0}^{N}\frac{dc_{n,t}}{dt}\exp\{-\frac{i}{\hbar}\int_0^t E_{n,t'}dt'\}\left|e_{n,t}\right\rangle + \langle e_{m,t}|\, i\hbar\sum_{n=0}^{N}c_{n,t}\exp\{-\frac{i}{\hbar}\int_0^t E_{n,t'}dt'\}\frac{d\left|e_{n,t}\right\rangle}{dt} = 0 \quad \text{(A.11)}$$

$$i\hbar\sum_{n=0}^{N}\frac{dc_{n,t}}{dt}\exp\{-\frac{i}{\hbar}\int_0^t E_{n,t'}dt'\}\delta_{n,m} + i\hbar\sum_{n=0}^{N}c_{n,t}\exp\{-\frac{i}{\hbar}\int_0^t E_{n,t'}dt'\}\left\langle e_{m,t}\right|\frac{d}{dt}\left|e_{n,t}\right\rangle = 0$$

$$i\hbar\frac{dc_{m,t}}{dt}\exp\{-\frac{i}{\hbar}\int_0^t E_{n,t'}dt'\} = -i\hbar\sum_{n=0}^{N}c_{n,t}\exp\{-\frac{i}{\hbar}\int_0^t E_{n,t'}dt'\}\left\langle e_{m,t}\right|\frac{d}{dt}\left|e_{n,t}\right\rangle$$

$$\frac{dc_{m,t}}{dt} = -\sum_{n=0}^{N}c_{n,t}\exp\{-\frac{i}{\hbar}\int_0^t (E_{n,t'}-E_{m,t'})dt'\}\left\langle e_{m,t}\right|\frac{d}{dt}\left|e_{n,t}\right\rangle \quad \text{(A.12)}$$

6. Rewriting the matrix element $\langle e_{m,t}| \frac{d}{dt} |e_{n,t}\rangle$ for $n = m$,

$$\langle e_{n,t}|e_{n,t}\rangle = 1 \quad \therefore \quad \frac{d}{dt}\langle e_{n,t}|e_{n,t}\rangle = 0,$$

(a) Let $\eta_n = \langle e_{n,t}| \frac{d}{dt} |e_{n,t}\rangle$,

$$\begin{aligned} 0 &= \langle e_{n,t}| \frac{d}{dt} |e_{n,t}\rangle \\ &= \langle e_{n,t}| \frac{d|e_{n,t}\rangle}{dt} + \frac{d\langle e_{n,t}|}{dt} |e_{n,t}\rangle \\ &= \eta_n^* + \eta_n \Longrightarrow \eta_n^* = -\eta_n \end{aligned}$$

(b) Using a gauge transformation, we define and can simplify:

$$|\tilde{e}_{n,t}\rangle = \exp\{-\int_0^t \eta_n dt'\} |e_{n,t}\rangle ,$$

$$\frac{d}{dt}\langle \tilde{e}_{n,t}|\tilde{e}_{n,t}\rangle = \langle \tilde{e}_{n,t}| \frac{d|\tilde{e}_{n,t}\rangle}{dt} + \frac{d\langle \tilde{e}_{n,t}|}{dt} |\tilde{e}_{n,t}\rangle$$

$$\frac{d}{dt}\langle \tilde{e}_{n,t}|\tilde{e}_{n,t}\rangle = \langle \tilde{e}_{n,t}| \frac{d}{dt}(\exp\{-\int_0^t \eta_n dt'\} |e_{n,t}\rangle) + \frac{d}{dt}(\langle e_{n,t}| \exp\{-\int_0^t \eta_n dt'\}) |\tilde{e}_{n,t}\rangle$$

$$\begin{aligned} \frac{d}{dt}\langle \tilde{e}_{n,t}|\tilde{e}_{n,t}\rangle = \langle \tilde{e}_{n,t}| (-\eta_n) \exp\{-\int_0^t \eta_n dt'\} |e_{n,t}\rangle + \langle e_{n,t}| e^{-\int_0^t \eta_n^* dt'} e^{-\int_0^t \eta_n dt'} \frac{d|e_{n,t}\rangle}{dt} \\ + \langle e_n(t)| (-\eta_n^*) e^{-\int_0^t \eta_n^* dt'} |\tilde{e_n}(t)\rangle + \frac{d\langle e_{n,t}|}{dt} e^{-\int_0^t \eta_n^* dt'} e^{-\int_0^t \eta_n dt'} |e_{n,t}\rangle \end{aligned}$$

$$\frac{d}{dt}\langle \tilde{e}_{n,t}|\tilde{e}_{n,t}\rangle = -\eta_n \langle \tilde{e}_{n,t}|\tilde{e_{n,t}}\rangle + \langle e_{n,t}| \frac{d|e_{n,t}\rangle}{dt} + -\eta_n^* \langle \tilde{e_n}(t)|\tilde{e_n}(t)\rangle + \frac{d\langle \tilde{e}_n(t)|}{dt} |e_n(t)\rangle$$

$$\frac{d}{dt}\langle \tilde{e}_{n,t}|\tilde{e}_{n,t}\rangle = -\eta_n^* + \eta_n^* - \eta_n + \eta_n$$

$$\frac{d}{dt}\langle \tilde{e}_{n,t}|\tilde{e}_{n,t}\rangle = 0$$

(c) **Result**: There exists a choice of phases that causes all of the $n = m$ matrix elements to be zero.

7. Expanding all $n \neq m$ cases,

   (a) Take derivative of the eigenvalue equation:

$$\hat{H}_t \left|e_{n,t}\right\rangle = E_{n,t} \left|e_{n,t}\right\rangle$$

$$\frac{d\hat{H}_t}{dt} \left|e_{n,t}\right\rangle + \hat{H}_t \frac{d\left|e_{n,t}\right\rangle}{dt} = \frac{dE_{n,t}}{dt} \left|e_{n,t}\right\rangle + E_{n,t} \frac{d\left|e_{n,t}\right\rangle}{dt}$$

   (b) Take matrix element with $\langle e_{m,t}|$:

$$\left\langle e_m(t)\right| \frac{d\hat{H}(t)}{dt} \left|e_n(t)\right\rangle + \left\langle e_m(t)\right| \hat{H}(t) \frac{d\left|e_n(t)\right\rangle}{dt} = \frac{dE_n(t)}{dt} \left\langle e_m(t)|e_n(t)\right\rangle + E_n \left\langle e_m(t)\right| \frac{d\left|e_n(t)\right\rangle}{dt},$$

$$\left\langle e_m(t)\right| \frac{d\hat{H}(t)}{dt} \left|e_n(t)\right\rangle + \left\langle e_m(t)\right| E_m(t) \frac{d\left|e_n(t)\right\rangle}{dt} = E_n(t) \left\langle e_m(t)\right| \frac{d\left|e_n(t)\right\rangle}{dt}$$

$$\left\langle e_m(t)\right| \frac{d\hat{H}(t)}{dt} \left|e_n(t)\right\rangle = \left(E_n(t) - E_n(t)\right) \left\langle e_m(t)\right| \frac{d}{dt} \left|e_n(t)\right\rangle \tag{A.13}$$

   (c) Recall Eq.(4.8) and plug in:

$$\frac{dc_{m,t}}{dt} = -\sum_{n=0}^{N} c_{n,t} \exp\{-\frac{i}{\hbar} \int_0^t (E_{n,t'} - E_{m,t'})dt'\} \left\langle e_{m,t}\right| \frac{d}{dt} \left|e_{n,t}\right\rangle \tag{A.14}$$

$$\frac{dc_{m,t}}{dt} = -\sum_{n=0}^{N} c_{n,t} \exp\{-\frac{i}{\hbar} \int_0^t (E_{n,t'} - E_{m,t'})dt'\} \frac{\left\langle e_{m,t}\right| \frac{d\hat{H}_t}{dt} \left|e_{n,t}\right\rangle}{E_{n,t} - E_{m,t}} \tag{A.15}$$

   (d) Assume that at $t = 0$, just $c_{1,t}$ is nonzero. This can be configured by the choice of the schedule function, resulting in:

$$\frac{dc_{m,t}}{dt} = -c_{1,t} \exp\{-\frac{i}{\hbar} \int_0^t (E_{1,t'} - E_{m,t'})dt'\} \frac{\left\langle e_{m,t}\right| \frac{d\hat{H}_t}{dt} \left|e_{1,t}\right\rangle}{E_{1,t} - E_{m,t}} \quad \forall\, m \tag{A.16}$$

   (e) For $t \ll 1$, we can approximate and integrate our exponential term assuming that the $\frac{\langle e_m(t)|\frac{d\hat{H}(t)}{dt}|e_1(t)\rangle}{E_1(t) - E_m(t)}$ term is roughly constant in t:

$$\Delta c_{m,t} = \frac{\hbar}{(E_{1,t} - E_{m,t})^2} \left\langle e_{m,t}\right| \frac{d\hat{H}_t}{dt} \left|e_{1,t}\right\rangle \left(e^{-i\Delta t} - 1\right) \quad \forall\, m$$

(f) Taking the absolute value gives us:

$$|\Delta c_{m,t}| = \frac{\hbar}{(E_{1,t} - E_{m,t})^2} \langle e_{m,t}| \frac{d\hat{H}_t}{dt} |e_{1,t}\rangle \, |e^{-i\Delta t} - 1| \quad \forall \, m$$

$$|\Delta c_{m,t}| \leq \frac{2\hbar}{(E_{1,t} - E_{m,t})^2} \langle e_{m,t}| \frac{d\hat{H}_t}{dt} |e_{1,t}\rangle \quad \forall \, m,$$

Giving us the correct adiabaticity condition/constraint for the time evolution of our quantum state. One important result of this theorem is that given some schedule $s(t)$, as $T \to \infty$ the error in the problem of interest goes to 0.

A.3 Python Documentation

VARIABLES

1. **model_circuit**- The total unitary operator for the driving Hamiltonian
2. **operators**- The operators that the user wants to be measured after the entire circuit. For our simulation, this was our cost Hamiltonian, $\hat{H}_C$.
3. **input_**- The input quantum state to our QAOA circuit, $H^{\otimes n} |0\rangle^{\otimes n}$
4. **optimum**- The optimal value the outer loop optimizer can attain. This will vary depending on which optimizer is chosen, but for our choice of Adam this parameter was 0.

MODEL FUNCTIONS

1. **repetitions**- The number of times each circuit with **the same** $(\boldsymbol{\beta}, \boldsymbol{\gamma})$ values were measured. This parameter was set to 100 for our simulation.

Bibliography

[1] E. Farhi, J. Goldstone, and S. Gutmann, arXiv preprint arXiv:1411.4028 (2014).

[2] M. X. Goemans and D. P. Williamson, Journal of the ACM (JACM) **42**, 1115 (1995).

[3] M. Broughton, G. Verdon, T. McCourt, A. J. Martinez, J. H. Yoo, S. V. Isakov, P. Massey, M. Y. Niu, R. Halavati, E. Peters, et al., arXiv preprint arXiv:2003.02989 (2020).

[4] J. Preskill, Quantum **2**, 79 (2018).

[5] G. Verdon, M. Broughton, and J. Biamonte, arXiv preprint arXiv:1712.05304 (2017).

[6] S. Kumar, arXiv preprint arXiv:1511.05956 (2015).

[7] M. A. Nielsen and I. L. Chuang, Phys. Today **54**, 60 (2001).

[8] J. M. Gambetta, J. M. Chow, and M. Steffen, npj Quantum Information **3**, 1 (2017).

[9] J. R. McClean, J. Romero, R. Babbush, and A. Aspuru-Guzik, New Journal of Physics **18**, 023023 (2016).

[10] I. Drori, A. Kharkar, W. R. Sickinger, B. Kates, Q. Ma, S. Ge, E. Dolev, B. Dietrich, D. P. Williamson, and M. Udell, arXiv preprint arXiv:2006.03750 (2020).

[11] N. Vesselinova, R. Steinert, D. F. Perez-Ramirez, and M. Boman, IEEE Access **8**, 120388 (2020).

[12] A. Jazayeri and H. Sayama, arXiv preprint arXiv:1608.01716 (2016).

[13] M. Wilson (2019), URL `arXiv:1908.03185`.

[14] D. R. Franzosi and D. Cocchiarella (2018).

[15] D. J. Griffiths, *Introduction to Quantum Mechanics (2nd Edition)* (Pearson Prentice Hall, Upper Saddle River, NJ, 2004).

[16] G. Nannicini, Physical Review E **99**, 013304 (2019).

[17] R. Shaydulin, H. Ushijima-Mwesigwa, C. F. Negre, I. Safro, S. M. Mniszewski, and Y. Alexeev, Computer **52**, 18 (2019).

[18] W. Lavrijsen, A. Tudor, J. Müller, C. Iancu, and W. de Jong, in *2020 IEEE International Conference on Quantum Computing and Engineering (QCE)* (IEEE, 2020), pp. 267–277.

[19] M. Alam, A. Ash-Saki, and S. Ghosh, in *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)* (IEEE, 2020), pp. 686–689.

[20] D. J. Hidary, Unpublished (2000).

[21] S. Ruder, arXiv preprint arXiv:1609.04747 (2016).

[22] J. Roland and N. J. Cerf, Physical Review A **65**, 042308 (2002).

[23] F. McAndrew, Ph.D. thesis, The University of Melbourne (2020).

[24] K. Srinivasan, S. Satyajit, B. K. Behera, and P. K. Panigrahi, arXiv preprint arXiv:1805.10928 (2018).