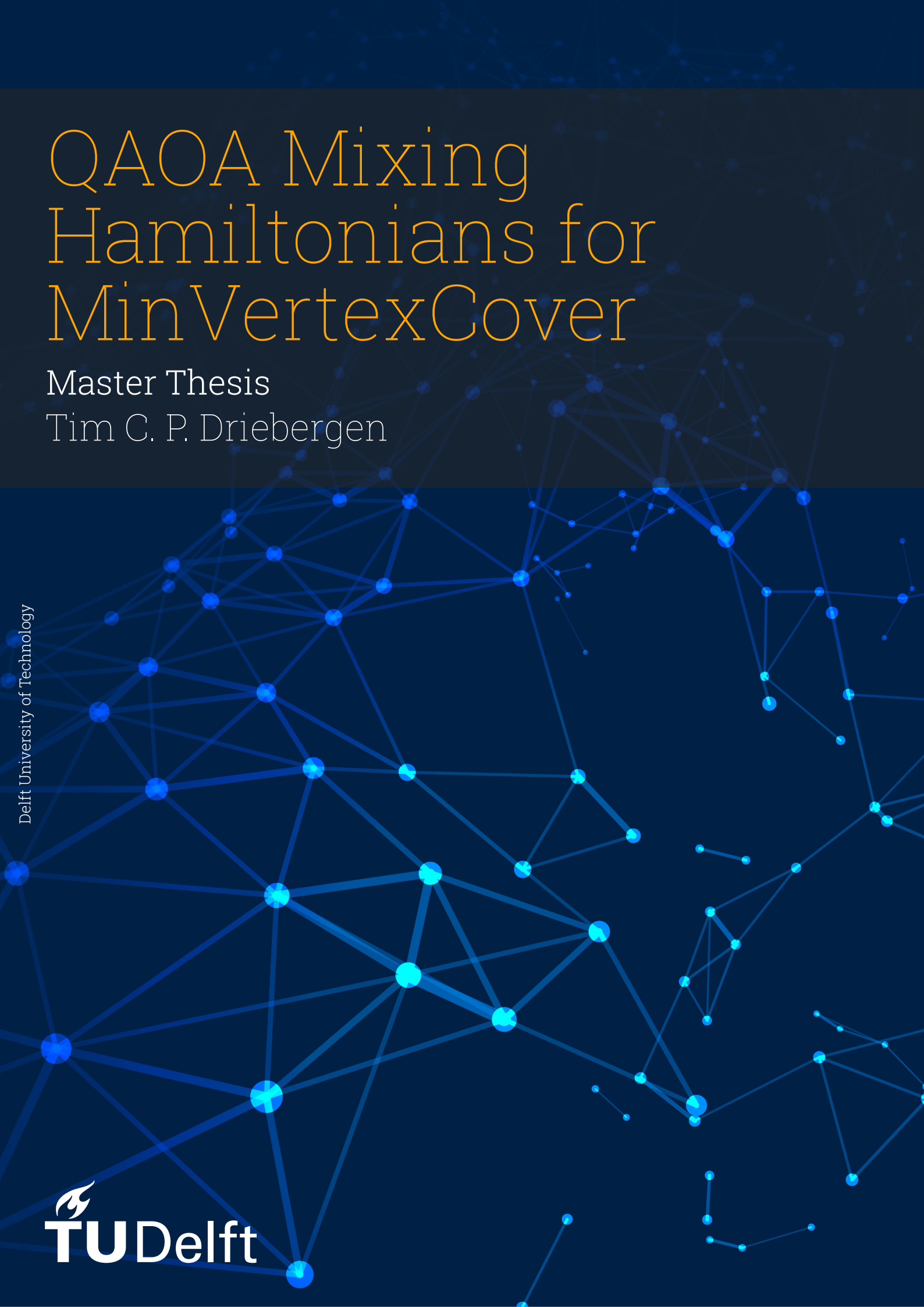
QAOA Mixing Hamiltonians for MinVertexCover

Master Thesis

Tim C. P. Driebergen



Delft University of Technology

TU Delft

QAOA Mixing Hamiltonians for MinVertexCover

by

Tim C. P. Driebergen

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Tuesday January 24, 2023 at 14:00.

Student number:	4565320
Supervisors:	Dr. M. Möller
	Dr. D. de Laat
Other committee members:	Dr. ir. L. J. J. van Iersel
Institution:	Delft University of Technology
	Faculty of Electrical Engineering,
	Mathematics & Computer Science
Master programme:	Applied Mathematics
Specialisation:	Optimization

Cover: Image by starline on Freepik (Modified)

An electronic version of this thesis is available at `http://repository.tudelft.nl/`.

TU Delft

Abstract

The minimum vertex cover problem (MinVertexCover) is an important optimization problem in graph theory, with applications in numerous fields outside of mathematics. As MinVertexCover is an NP-hard problem, there currently exists no efficient algorithm to find an optimal solution on arbitrary graphs. We consider quantum optimization algorithms, such as the Quantum Alternating Operator Ansatz (QAOA+), to find good minimum vertex covers.

This thesis presents new 'second degree' mixing Hamiltonians for MinVertexCover in the QAOA+ framework, which allow mixing between solutions Hamming distance 2 apart. The performance of these new Hamiltonians is evaluated on a small graph. Methods to extend this idea by constructing Hamiltonians which allow mixing between solutions at Hamming distance n are also presented, along with a generalization of second degree mixing Hamiltonians to other optimization problems.

Preface

It is with great pleasure that I present my Master's thesis, which marks the completion of my studies at Delft University of Technology. For the past five years, I have been engaged in research on quantum computing, and it has been an incredibly rewarding experience. The combination of quantum computing and optimization has been particularly intriguing to me, and I have greatly enjoyed delving deeper into this field.

I would like to extend my deepest gratitude to my supervisors, Matthias Möller and David de Laat, for their invaluable guidance and expertise throughout the completion of this thesis. Their advice and support have been instrumental in helping me achieve this milestone. Additionally, I would like to thank my family and friends for their ongoing support and motivation. Special thanks to my dad, who provided invaluable mentorship and encouragement throughout the process.

I hope you enjoy your reading.

Tim C. P. Driebergen
Delft, January 2023

Contents

1 Introduction

The minimum vertex cover problem (MinVertexCover) is a fundamental optimization problem in graph theory. A vertex cover is a subset of vertices in an undirected graph such that every edge of the graph is incident to at least one of the vertices in the subset. The minimum vertex cover problem is the problem of finding such a vertex cover of smallest size. MinVertexCover has applications in many other fields outside of mathematics and computer science, such as biotechnology, network design, and transportation. In biotechnology, for example, MinVertexCover can be used to optimize engineered genetic systems by eliminating repetitive genetic parts [1]. Vertex cover optimization is used to model a wide number of real-world problems, and finding minimum vertex covers efficiently is thus an important aspect to solving these problems.

MinVertexCover can be solved by iterating over all possible vertex covers to find the smallest one. However, this approach is not very efficient, as the number of possible covers (2^n) scales exponentially with n. Many different algorithms for finding and approximating minimum vertex covers have been developed, but none are able to solve the problem in polynomial time. This is due to the fact that MinVertexCover is an NP-hard problem, and thus it is unlikely that there exists an efficient algorithm to find a minimum vertex cover in an arbitrary graph. Approximations of MinVertexCover have not improved past an approximation factor of 2. Should the stronger unique games conjecture be true, then this approximation factor 2 is also the smallest possible approximation constant [2]. Thus, we look at techniques that are not bounded by the rules of classical computing to achieve better approximations.

Quantum algorithms are computational methods that use quantum mechanical phenomena to parallelize arithmetic and to solve problems more efficiently. Examples of famous quantum algorithms are Shor's algorithm to factor large integers [3], Grover's algorithm to search databases [4], and the HHL algorithm to solve systems of linear equations [5]. Along with these practical applications, the study of quantum algorithms also helps us to better understand quantum mechanics itself. In general, the field of quantum computing and quantum algorithms is a rapidly developing area of research with the potential to solve a variety of problems that classical computers are currently unable to handle.

A quantum algorithm designed to approximate unconstrained combinatorial optimization algorithms is the Quantum Approximate Optimization Algorithm (QAOA) [6]. QAOA operates by representing the possible solutions of the problem as quantum states. Two unitaries are then iteratively applied to a starting state in order to transform the probability distribution of the state to favour solutions with better objective value. Finally, the state is measured to find a solution to the original optimization problem. QAOA is not able to handle constrained optimization problems, and therefore an extension to QAOA known as the Quantum Alternating Operator Ansatz (QAOA+) was introduced [7]. QAOA+ focuses on general families of unitaries and can handle optimization problems with a desired feasible subspace defined by the constraints of the problem.

The main contribution of this thesis is two new mixing Hamiltonians for MinVertexCover in the QAOA+ setting. These new mixing Hamiltonians are able to swap between multiple vertices at the same time, while still preserving the feasible subspace. The performance of these Hamiltonians is compared to the original mixing Hamiltonian suggested in [7], using different classical optimization methods to find the optimal parameters for QAOA+. Extensions to these Hamiltonians are also proposed along with alternatives to the implementation used in this thesis.

My thesis is structured as follows. First, we start with an overview of MinVertexCover in Chapter 2, covering the definition of MinVertexCover as well as different ways of solving the problem exactly and approximating its solutions. Chapter 3 then discusses quantum systems and Hamiltonian evolution, topics important to understand how QAOA operates. Next, Chapter 4 covers the algorithms QAOA and QAOA+ in detail, along with a more comprehensive discussion on mixing Hamiltonians. In Chapter 5 two new mixing Hamiltonians are presented for MinVertexCover. This chapter discusses both their explicit formulas as well as how they operate to preserve and reach the feasible subspace. The implementation of the new Hamiltonians and the type of classical optimization methods used for QAOA+ are then discussed in Chapter 6. The new mixing Hamiltonians are subsequently compared to their original counterpart to produce the results presented in Chapter 7. Finally, conclusions are presented in Chapter 8, along with extensions and alternatives to the new Hamiltonians and their implementation in Chapter 9.

2

Minimum Vertex Cover Problem

In this chapter, we will delve into MinVertexCover and various classical approaches for solving it. We will begin by defining the problem and examining its complexity. Next, we will examine exact solutions for specific types of graphs, including tree graphs, complete graphs, cycle graphs, and bipartite graphs. Finally, we will discuss the approximability of MinVertexCover, focusing on algorithms that can find solutions that are close to optimal, but may not be the optimal solution itself.

2.1. Problem definition and complexity

A vertex cover C of an undirected graph $G = (V, E)$ is a subset of V such that for every edge $uv \in E$ we have $u \in C$ or $v \in C$, so every edge in G has one (or both) of its endpoints in C. A minimum vertex cover is a vertex cover of smallest size. A minimum vertex cover is different from a minimal vertex cover, which is a vertex cover C where none of the vertices in C can be removed from C while preserving the property that C is a vertex cover. Note that a minimum vertex cover is by definition minimal, but a minimal vertex cover is not necessarily a minimum vertex cover, as can be seen in Figure 2.1.

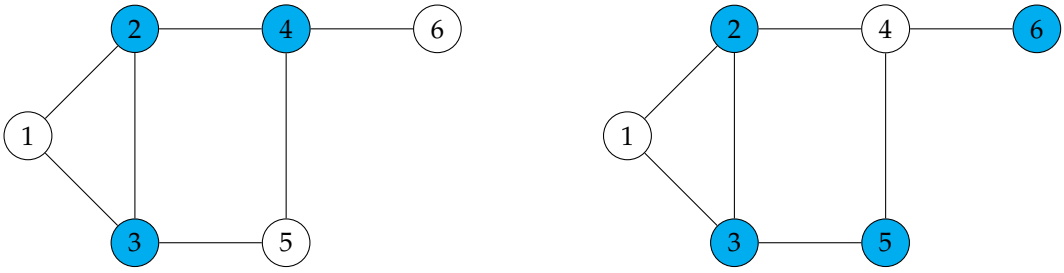


Figure 2.1: Examples of vertex covers with the vertices in the vertex cover shown in cyan. Note that both of these covers are minimal vertex covers, but only the left cover is a minimum vertex cover.

The minimum vertex cover problem (MinVertexCover) is the optimization problem of finding a smallest size vertex cover in a graph. Note that there is not necessarily exactly one smallest size vertex cover, as for many graphs multiple smallest size vertex covers are possible. Finding any (not all) of these smallest size covers solves the problem. The corresponding decision problem is called the vertex cover problem, and answers the question whether, given a positive integer k, there exists a vertex cover of size k. The vertex cover number $\tau(G)$ of a graph G is defined as the size of a smallest vertex cover. MinVertexCover can be formulated as the integer linear program (ILP) given by:

$$\min \quad \sum_{v \in V} x_v \tag{2.1}$$

$$\text{s.t.} \quad x_u + x_v \geq 1 \qquad \text{for all } \{u, v\} \in E, \tag{2.2}$$

$$x_v \in \{0, 1\} \qquad \text{for all } v \in V, \tag{2.3}$$

where a solution to the ILP x corresponds to the vertex cover C through the relations $x_v = 1 \iff v \in C$ and $x_v = 0 \iff v \notin C$. The objective function (2.1) is exactly the size of the vertex cover C, and constraint (2.2) implements the constraint that at least one of the endpoints of every edge $uv \in E$ has to be in C. Lastly, constraint (2.3) formulates that every vertex is either in C or not in C.

A decision problem is said to be in NP if a 'yes' instance of the problem can be verified in polynomial time. In the case of the vertex cover problem this translates to, given a certain vertex cover C, verify that it is indeed a vertex cover. This is done by checking all the possible constraints (2.2), of which there are at most $\frac{|V|(|V|-1)}{2}$, as a graph with $|V|$ vertices has at most $\frac{|V|(|V|-1)}{2}$ edges, which is verifiable in polynomial time. A problem is said to be NP-hard if any other problem in NP is polynomial-time reducible to said problem. The decision variant of MinVertexCover is an NP-complete problem as was shown by Karp as part of Karps 21 NP-complete problems [8], meaning it is both in NP as well as NP-hard. Karp proved the NP-completeness of MinVertexCover by reducing another NP-complete problem to MinVertexCover, namely by first reducing the Boolean satisfiability problem to the Clique problem, and then reducing the Clique problem to MinVertexCover. The Boolean satisfiability problem is NP-complete as stated by the Cook-Levin theorem [9], making the Clique problem NP-complete as well, and thus MinVertexCover is also an NP-complete problem.

As the decision variant of MinVertexCover is NP-complete, it is unlikely that there exists an efficient algorithm to find a minimum vertex cover for arbitrary graphs. In fact, the decision variant of MinVertexCover remains NP-complete even in graphs where all vertices have degree 3 (so-called cubic graphs) [10]. Even if all vertices have degree 3 and the graph is planar, meaning it can be drawn on a plane without its edges crossing at points other than the vertices, the problem still remains NP-complete [11]. However, for some specific types of graphs solutions can be found in polynomial time, as will be discussed in Section 2.2.

2.2. Exact solutions to MinVertexCover for different types of graphs

While the general version of MinVertexCover is NP-complete, some types of graphs are structured in such a way that for these graphs MinVertexCover can be solved in polynomial time. Here some of these graph types are discussed, namely tree graphs, complete graphs, cycle graphs, and bipartite graphs. For these graphs we look at the method of finding a minimum vertex cover and show the size $\tau(G)$ of this minimum cover (if this number is possible to formulate as a function of $|V|$) [12].

2.2.1. Tree graphs

A tree graph $T = (V, E)$ is a connected graph without a cycle, an example of which can be seen in Figure 2.2.

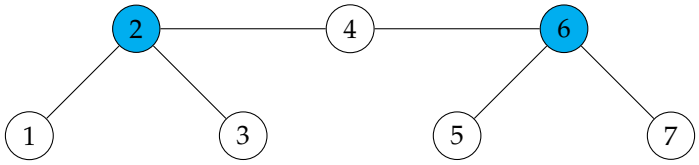


Figure 2.2: Example of a tree graph with a minimum vertex cover shown in cyan.

To find a minimum vertex cover in a tree graph, first note that any tree graph always has a leaf (a vertex of degree 1). Starting at this leaf, add the vertex adjacent to this leaf (of which there is by definition only 1) to C and remove both the leaf and the vertex adjacent to it from the graph, which should create another leaf. Repeating this process until no edges are left results in C being a minimum vertex cover. This process is a variant of the algorithm to find a maximum independent set on trees discussed in [13].

2.2.2. Complete graphs

A complete graph $K_n = (V, E)$ is a graph of n vertices where every pair of vertices is connected by an edge, an example of which can be seen in Figure 2.3.

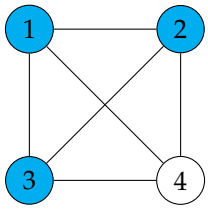


Figure 2.3: Example of a complete graph with a minimum vertex cover shown in cyan.

To find a minimum vertex cover in a complete graph, we first show that such a vertex cover can never be of smaller size than $|V| - 2$. Assume a cover C with a size smaller than $|V| - 2$ exists. This would mean that there exist two vertices u and v which are not in C. As the graph is complete, the edge (u, v) exists, which is not covered by C as both u and v are not in C. Thus, C is not a vertex cover and any vertex cover has to be of size at least $|V| - 1$. It is clear that any set of vertices of this size is indeed a vertex cover, as for any edge not to be covered there need to be at least 2 vertices not in the cover. Therefore, any set of vertices of size $|V| - 1$ is a minimum vertex cover for a complete graph, and $\tau(K_n) = |V| - 1$.

2.2.3. Cycle graphs

A cycle graph $C_n = (V, E)$ is a graph consisting of a single cycle of n vertices, two examples of which can be seen in Figure 2.4.

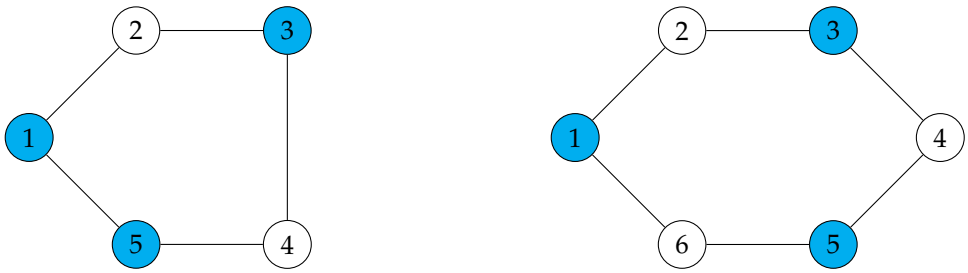


Figure 2.4: Cycle graphs C_5 and C_6 with possible minimum vertex covers shown in cyan.

To find a minimum vertex cover of a cycle graph, first note that for a cycle graph we have $|E| = |V|$. All vertices have degree 2 and can therefore cover at most two edges, thus the size of any vertex cover has to be at least $\lceil \frac{|V|}{2} \rceil$. If we pick a starting vertex to add to a set C and go around the cycle adding every other vertex to C, then C will be a vertex cover, as no edge is left uncovered. This vertex cover C is of size $\frac{|V|}{2}$ in the case of an even cycle and $\frac{|V|+1}{2}$ in the case of an odd cycle. In both cases this size is equal to the minimum size of $\lceil \frac{|V|}{2} \rceil$, which means C is indeed a minimum vertex cover, and thus $\tau(C_n) = \lceil \frac{|V|}{2} \rceil$.

2.2.4. Bipartite graphs

A bipartite graph $G = (V, E)$ is a graph whose vertices can be divided into two disjoint sets A and B such that no edge of the graph connects two vertices in A or two vertices in B, that is every edge connects a vertex in A to a vertex in B. An example can be seen in Figure 2.5.

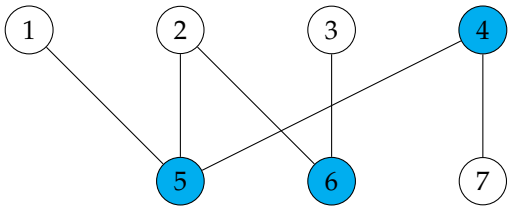


Figure 2.5: Example of a bipartite graph with a minimum vertex cover shown in cyan.

Note that any tree graph is also bipartite, and so is any cycle graph with an even number of nodes. In fact, the bipartite graphs are exactly all graphs that do not contain a cycle of odd length [14]. Both A and B are vertex covers of G, but not necessarily minimum vertex covers. To find a minimum vertex cover of a bipartite graph, we first define the concept of a matching. A matching is a set of edges M such that each vertex is connected to at most one edge in M. In other words, no edges in M have common endpoints. A maximum matching is a matching of largest size. Note that there can be multiple maximum matchings for a given graph. An example of a matching can be seen in Figure 2.6.

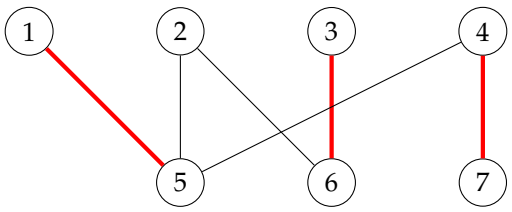


Figure 2.6: Example of a bipartite graph with a matching shown in red.

König linked the size of maximum matchings and minimum vertex covers in the following theorem.

Theorem 2.1 (König [15])**.** *In any bipartite graph, the number of edges in a maximum matching equals the number of vertices in a minimum vertex cover.*

The Hopcroft-Karp algorithm can be used to find a maximum matching in a bipartite graph in polynomial time [16]. Combining this algorithm with the theorem by König shows that, if there is an algorithm to produce a minimum vertex cover given a maximum matching, then the problem is solvable in polynomial time. One such algorithm can be found in [17], showing that in any bipartite graph a minimum vertex cover can be found in polynomial time.

A special case of a bipartite graph is a complete bipartite graph. In a complete bipartite graph with disjoint vertex sets A and B, every vertex in A is connected to every vertex in B. Without loss of generality, let $|A| \leq |B|$. In this case, there are at least $|A|$ different edges with all different endpoints as the graph is complete. Therefore, any vertex cover needs to have at least size $|A|$. Thus in general, we have $\tau(G) = \min(|A|, |B|)$ and the smaller of the two sets A and B is a minimum vertex cover.

2.3. Approximations of MinVertexCover

Currently there is no known algorithm to solve MinVertexCover on arbitrary graphs in polynomial time. However, we do know that if MinVertexCover (or any other NP-complete problem for that matter) is solvable in polynomial time, then all other problems in NP are also solvable in polynomial time [18]. As such an algorithm has not been found for any NP-complete problem so far, we look for other algorithms to obtain not necessarily optimal, but approximate solutions to MinVertexCover.

2.3.1. Factor 2-approximation

To find a factor 2-approximation (or simply 2-approximation) we use the algorithm by Gavril and Yannakakis [19], which first greedily constructs a maximal matching M. Next a set C is constructed based on this matching M by taking C as set of all the endpoints of the edges in M. Note that this set C is a vertex cover. Suppose an edge e is not covered by C, then neither of its endpoints are in C. This means none of the edges in M are adjacent to e and therefore $M + \{e\}$ would also be a matching, which is a contradiction as M was assumed to be maximal.

To show this vertex cover C is a 2-approximation, first note that $|C| = 2|M|$. See that for any edge $e \in M$, at least one of its endpoints has to be in any vertex cover. This means the size of the optimal cover has to be at least the size of our maximal matching, as every edge in M corresponds to at least one point in C. In other words, $|M| \leq |C_{opt}|$. Combining this with the fact that $|C| = 2|M|$ we see that $|C| = 2|M| \leq 2|C_{opt}|$, in other words, our constructed vertex cover C is at most twice the size of any other cover, including the optimal one.

Another way to find a 2-approximation is by using the ILP formulation of MinVertexCover given by (2.1), (2.2), and (2.3), where we follow the structure of [20]. To find this 2-approximation we look at the linear programming (LP) relaxation of the ILP, given by:

$$\min \quad \sum_{v \in V} x_v \tag{2.4}$$

$$\text{s.t.} \quad x_u + x_v \geq 1 \qquad \text{for all } \{u, v\} \in E, \tag{2.5}$$

$$x_v \geq 0, x_v \leq 1 \qquad \text{for all } v \in V. \tag{2.6}$$

Let us solve this LP relaxation in polynomial time and call the optimal solution x. If this solution is integral, then the problem is solved exactly, so assume it is not integral and thus fractional. Take this optimal fractional solution x and define our approximation x^* as follows. If $x_v < \frac{1}{2}$, let $x_v^* = 0$ and if $x_v \geq \frac{1}{2}$ let $x_v^* = 1$. Next define the set $C := \{v \mid x_v^* = 1\}$. The following two statements now hold for C:

1. C is a vertex cover. Suppose C is not a vertex cover, then $x_u^* + x_v^* = 0$ for some edge $\{u, v\}$. Therefore, $x_u^* = x_v^* = 0$ and thus $x_u < \frac{1}{2}$ and $x_v < \frac{1}{2}$. It follows that $x_u + x_v < \frac{1}{2} + \frac{1}{2} = 1$. But this contradicts the fact that for the LP relaxation we have $x_u + x_v \geq 1$ for all $\{u, v\} \in E$. Thus, C has to be a vertex cover.
2. The size of C is at most twice the size of a minimum vertex cover C_{opt}, as $\sum_{v \in V} x_v^* \leq 2 \cdot \sum_{v \in V} x_v = 2 \cdot \text{opt}(LP) \leq 2 \cdot |C_{\text{opt}}|$, where $\text{opt}(LP)$ is the minimum value of the LP.

C is thus a vertex cover with a size at most two times the size of a minimum vertex cover, meaning this algorithm gives another factor 2-approximation for vertex cover.

2.3.2. Other approximations

The algorithm with the best approximation-factor found so far was found by Karakostas in 2009 [21], with a factor of $2 - \Theta(1/\sqrt{\log|V|})$, but no further improvements have been made. In 2005 it was already shown by Dinur and Safra that MinVertexCover cannot be approximated for any sufficiently large vertex degree within a factor of 1.3606 [22] under the assumption that $P \neq NP$ (as if $P = NP$ then MinVertexCover can be solved exactly in polynomial time). This makes MinVertexCover part of the APX class, the class of NP optimization problems with polynomial-time algorithms with an approximation ratio bounded by a constant (again assuming $P \neq NP$). Similar to how MinVertexCover is NP-complete, MinVertexCover is also APX-complete, meaning it is in APX and every other problem in APX can be PTAS-reduced to MinVertexCover [22]. A PTAS-reduction is a reduction that preserves the property of the optimization problem having a polynomial time approximation scheme (PTAS). In other words, any problem that has a polynomial-time approximation algorithm with the approximation ratio bound by a constant can be transformed into MinVertexCover. In 2017 the approximation bound was improved to $\sqrt{2} - \epsilon$ for all $\epsilon > 0$ in [23] using the 2-to-2 games conjecture [24] (assuming $P \neq NP$). Furthermore, if the stronger unique games conjecture [25] is true, this bound would be even higher at $2 - \epsilon$ for all $\epsilon > 0$ [2]. This would mean the 2-approximation algorithms discussed before would be the best possible approximation algorithms for MinVertexCover.

All relevant knowledge necessary to understand MinVertexCover and its solutions and approximations has now been discussed. Next Chapter 3 will cover quantum systems and Hamiltonian evolution, which will build the way to understanding how QAOA and its successors work.

3

Quantum Systems and Time evolution

Before introducing the quantum algorithm used in this thesis to approximate solutions to MinVertexCover (which will be covered in Chapter 4), first the setting and background of quantum systems will be discussed to better understand what the Quantum Approximate Optimization Algorithm is based on and how it operates. Here we describe the setup of quantum systems as based on the postulates of quantum mechanics [26].

3.1. Schrödinger Equation

The state of a quantum system is defined as a vector $|\psi\rangle$ which belongs to a Hilbert space $\mathcal{H}$. State vectors are normalized vectors under the inner product of $\mathcal{H}$, $\langle |\psi\rangle , |\psi\rangle \rangle = 1$. The total energy of a quantum system is given by a Hamiltonian H, which is a Hermitian linear operator acting on the Hilbert space. This Hamiltonian H has eigenvalues λ_i and eigenvectors v_i (as quantum mechanical calculations are performed on finite-dimensional Hilbert spaces), which correspond to the potential energy levels of a system and the states which attain this energy level when the system is measured. Any quantum state will be a linear combination of the eigenstates of the system, known as a superposition. When the energy of a system, which is then in a certain state $|\psi\rangle$, is measured, this state will collapse to an eigenstate v with probability $|\langle v, |\psi\rangle\rangle|^2$. Note that when measured the states $|\psi\rangle$ and $e^{-i\gamma} |\psi\rangle$ can not be distinguished. This means in a physical sense quantum states are well defined up to a global phase, a property which will be important later when discussing QAOA.

The evolution of a quantum state through time is described by the Schrödinger equation [27], given by

$$i\frac{d}{dt} |\psi\rangle (t) = H(t) |\psi\rangle (t), \tag{3.1}$$

where the reduced Planck constant $\hbar$ is included in the Hamiltonian. If the Hamiltonian H is time-independent, then the solution of this differential equation is given by

$$|\psi(t)\rangle = e^{-iHt} |\psi(0)\rangle = U(t) |\psi(0)\rangle , \tag{3.2}$$

where the exponential of the Hamiltonian is defined through its power series. Thus, $U(t) = e^{-iHt}$ gives the quantum state of the system at any time t when applied to the initial state of the system. This operator U has to be unitary, as it describes time evolution of a closed quantum system. This means no energy is added to the system or can leave the system, and as U is indeed unitary as it preserves the inner product of $\mathcal{H}$.

3.2. Adiabatic Evolution

Now that we understand how a quantum state evolves over time, we can start using this to our advantage. The Hamiltonians belonging to systems we describe from this chapter forward will not be independent of time, and thus we can not use the solution of the Schrödinger equation as described by (3.2). Although an exact general solution to the Schrödinger equation is not known, we can make make statements about eigenstates of the Hamiltonian under certain conditions. Here we look at the adiabatic theorem, which was first stated by Born and Fock as:

Theorem 3.1 (Born, Fock [28])**.** *A physical system remains in its instantaneous eigenstate if a given perturbation is acting on it slowly enough and if there is a gap between the eigenvalue and the rest of the Hamiltonian's spectrum.*

In other words, if the systems Hamiltonian changes slowly enough, then it will not leave the eigenstate it was in. Remember that this only holds if there is a non-zero gap between the eigenvalues of the Hamiltonian at all times. We can use this property to our advantage by considering an eigenstate of a constant original Hamiltonian and defining a new Hamiltonian that evolves the system from this original Hamiltonian to a new constant target Hamiltonian. By the adiabatic theorem, if the system evolves slow enough, we will end up in the same eigenstate of the target Hamiltonian. This is exactly what the Quantum Adiabatic Algorithm does.

The Quantum Adiabatic Algorithm (QAA, also known as Quantum Annealing) [29] considers an initial Hamiltonian H_M and a target Hamiltonian H_P. It then uses adiabatic evolution to evolve from the known ground state (lowest energy eigenstate) of H_M to the unknown ground state of H_P by evolving under a new Hamiltonian defined by

$$H(t) = (1 - t/T)H_M + (t/T)H_P, \tag{3.3}$$

where T controls the speed which the system evolves with. Note that $H(0) = H_M$ and $H(T) = H_P$. There are also newer methods that change the path from the H_M to H_P by including a term known as a catalyst Hamiltonian [30, 31]. An example of Hamiltonian evolution using a catalyst would be

$$H(t) = (1 - t/T)H_M + (t/T)(1 - t/T)H_C + (t/T)H_P, \tag{3.4}$$

where H_C is the catalyst Hamiltonian. Again note that still $H(0) = H_M$ and $H(T) = H_P$.

To ensure the gap between eigenvalues it is adequate for most problems to take

$$T \gg \frac{1}{g_{\min}^2}, \tag{3.5}$$

as suggested in [29]. Here the minimum gap $g_{\min}$ is defined by

$$g_{\min} = \min_{0 \le t \le T} \left(\lambda_1(t) - \lambda_0(t)\right), \tag{3.6}$$

where λ_0 and λ_1 are the first two eigenvalues of H. By the adiabatic theorem as T tends to ∞ the chance that we end up in the ground state of H_P tends to 1, as by increasing T the evolution speed of the system is slowed down. The general goal of catalyst Hamiltonians is to increase the size of this minimum gap $g_{\min}$.

The idea behind QAA is to pick a Hamiltonian H_M whose eigenstates are easy to construct, and to pick another Hamiltonian H_P whose eigenstates are solutions to an optimization problem. These constructions will be discussed in Section 4.1 when discussing QAOA. The system is then evolved according to the Schrödinger equation from (3.1) for time T as in (3.5), and the final state $|\psi(T)\rangle$ will be arbitrarily close to the ground state of H_P. This state is then measured to find the optimal solution of the optimization problem H_P was based on.

One of the weaknesses of the Quantum Adiabatic Algorithm is that as T increases, the probability of finding an optimal solution does not necessarily increase. As can be seen in Figure 3.1, there are examples where the probability of success drops suddenly, even though for very large T it will eventually go to 1. Therefore, it can be hard to determine a running time required to exactly solve the problem [32].

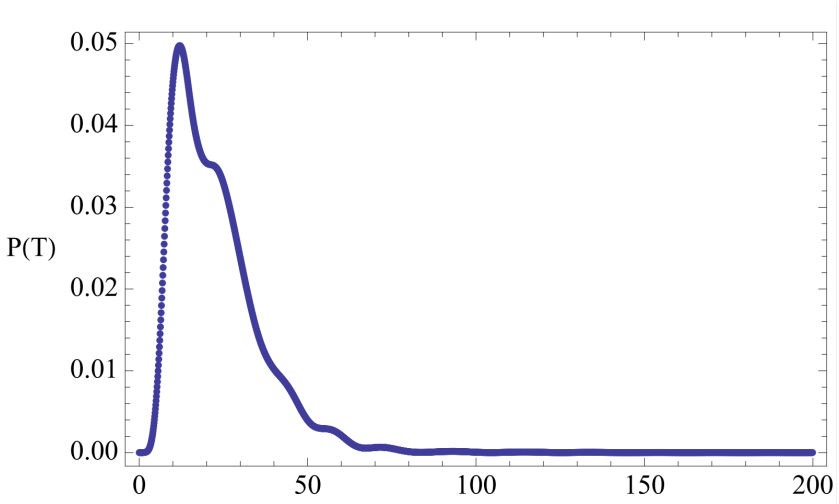


Figure 3.1: The success probability of QAA as a function of total evolution time T for an instance of MAX 2-SAT with n = 20 from [32].

While the Quantum Adiabatic Algorithm is designed to find the optimal solution of an optimization problem if the runtime T is long enough, we are focused on finding an approximate solution to instances of MinVertexCover. Thus, we want to find an approximation to the adiabatic evolution as described in this section, which will be obtained through the process of Trotterization.

3.3. Trotterization

Let us look again at the Schrödinger evolution (3.1) of the system under the Hamiltonian of (3.3). To find an approximation to the solution we can split the time interval $(0, T)$ into N intervals of length ΔT. $H(T)$ will be approximately constant over the interval $(t, t + \Delta t)$ if $\Delta T \ll T$. This way, we can approximate the time evolution in this system by N total timesteps of size ΔT, each having the solution given by (3.2) for their interval. Following the steps in [33], this gives the approximation

$$U(t) \approx \prod_{k=1}^{N} e^{-i\Delta t H(k\Delta t)}, \tag{3.7}$$

where $H(t)$ is still given by (3.3). Substituting this into (3.7) results in

$$U(t) \approx \prod_{k=1}^{N} e^{-i\Delta t((1-k\Delta t)H_M + k\Delta t H_P)}. \tag{3.8}$$

This Hamiltonian is a sum of two other Hamiltonians, so we can use the Suzuki-Trotter expansion [34] given by

$$e^{A+B} = \lim_{n\to\infty} (e^{A/n} e^{B/n})^n, \tag{3.9}$$

or alternatively

$$e^{\delta(A+B)} = e^{\delta A} e^{\delta B} + O(\delta^2). \tag{3.10}$$

Note that

$$\begin{aligned} e^{A+B} &= \sum_{k=0}^{\infty} \frac{(A+B)^k}{k!} \\ &= \sum_{k=0}^{\infty} \sum_{l=0}^{k} \binom{k}{l} \frac{A^l B^{k-l}}{k!} \\ &= \sum_{k=0}^{\infty} \sum_{l=0}^{k} \frac{A^l B^{k-l}}{l!(k-l)!}, \end{aligned} \tag{3.11}$$

using the binomial theorem and the fact that $\binom{k}{l} = \frac{k!}{l!(k-l)!}$. If A and B commute we can reorder all of the A and B terms and use the Cauchy product formula [35] to find

$$\begin{aligned} e^{A+B} &= \sum_{k=0}^{\infty} \sum_{l=0}^{k} \frac{A^l B^{k-l}}{l!(k-l)!} \\ &= \sum_{l=0}^{\infty} \frac{A^l}{l} \sum_{m=0}^{\infty} \frac{B^m}{m!} \\ &= e^A e^B. \end{aligned} \tag{3.12}$$

Note that this is generally not true in the case that A and B do not commute. Using this Suzuki-Trotter expansion from (3.10), we find our approximation from (3.8) to be

$$U(t) \approx \prod_{k=1}^{N} e^{-i\Delta t(1-k\Delta t)H_M} e^{-i\Delta t k \Delta t H_P}, \tag{3.13}$$

where the $O(\Delta t)$ term vanishes as we assume $\Delta t \ll T$ with this approximation.

This process is known as Trotterization. As the two terms of the Hamiltonian given by (3.3) may not commute, we use this Trotterization to evolve the Hamiltonian by repeatedly switching between the two Hamiltonian terms and evolving each of them for a short amount of time. This way, each of the Hamiltonians can be implemented individually, instead of their sum.

Now that the basics of quantum systems are covered, we can use equation (3.13) to introduce a full approximation algorithm based on the Quantum Adiabatic Algorithm. This approximation algorithm will be discussed next in Chapter 4.

4

Quantum Approximate Optimization Algorithms

In this chapter, we will bring together the topics of MinVertexCover and quantum systems to examine how quantum computing can be used to solve optimization problems. We will focus on the Quantum Approximate Optimization Algorithm (QAOA) and its extension, the Quantum Alternating Operator Ansatz (QAOA+), and how they can be used to approximate solutions to the MinVertexCover problem. We will also delve into the concept of mixing Hamiltonians and how they can be utilized in this context.

4.1. The Quantum Approximate Optimization Algorithm

The Quantum Approximate Optimization Algorithm as introduced in [6] attempts to optimize a function f on the n-bit strings. The unconstrained problem is given by

$$\max \quad f(x) \tag{4.1}$$
$$\text{s.t.} \quad x \in \{0,1\}^n. \tag{4.2}$$

A problem Hamiltonian H_P which operates on the n-qubit space is then defined through the equation

$$H_P |x\rangle = f(x) |x\rangle , \tag{4.3}$$

which scales each of the n-qubit strings by their objective value $f(x)$. Note that throughout this thesis $f(x)$ and $f(|x\rangle)$ are used interchangeably. The matrix representation of the operator H_p onto the standard computational basis is a diagonal matrix of size $2^n \times 2^n$ with the diagonal elements being the objective value of each of the n-bit strings in binary order. This matrix representation is given by

$$H_P |x\rangle = \begin{pmatrix} f(|0\rangle^{\otimes n}) & 0 & \cdots & 0 & 0 \\ 0 & f(|0\rangle^{\otimes n-1}|1\rangle) & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & f(|1\rangle^{\otimes n-1}|0\rangle) & 0 \\ 0 & 0 & \cdots & 0 & f(|1\rangle^{\otimes n}) \end{pmatrix}. \tag{4.4}$$

Note that by definition the largest eigenvalue of H_P is $f(x_{\text{opt}})$ with corresponding eigenstate $|x_{\text{opt}}\rangle$, with x_{opt} being the optimal solution to the unconstrained optimization problem. Similarly if the problem was a minimization problem the smallest eigenvalue of H_P would be $f(x_{\text{opt}})$. Highlighting these eigenvalues is important, as the optimization problem is solved by finding one of the corresponding eigenstates.

Next a mixing Hamiltonian H_M is defined by

$$H_M = \sum_{j=1}^{n} X_j, \tag{4.5}$$

where X_j is the Pauli X operator acting on the jth qubit. This Hamiltonian (also known as the transverse field Hamiltonian) is not dependent on the optimization problem, and thus its eigenstates with largest eigenvalues are the same for every instance of the problem.

Lemma 4.1. *The eigenstate with largest eigenvalue of the transverse field Hamiltonian is given by*

$$|s\rangle = \frac{1}{\sqrt{2^n}} \sum_x |x\rangle \,, \tag{4.6}$$

where the sum is over all computational basis states, which correspond exactly to all 2^n *possible* n*-bit strings.*

Proof. First define H_M^n as the mixing Hamiltonian acting on the n-qubit space. Note that the mixing Hamiltonian in the single qubit space is just the Pauli-X gate given by

$$H_M^1 = X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \tag{4.7}$$

and the transverse field Hamiltonian in a $(n+1)$-qubit space can be written as

$$H_M^{n+1} = \begin{pmatrix} H_M^n & I_n \\ I_n & H_M^n \end{pmatrix}. \tag{4.8}$$

Let v be an eigenvector of a square matrix A with eigenvalue λ_v and define $A' = \begin{pmatrix} A & I \\ I & A \end{pmatrix}$ and $w = \begin{pmatrix} v \\ v \end{pmatrix}$, then

$$\begin{aligned} A'w &= \begin{pmatrix} A & I \\ I & A \end{pmatrix} \begin{pmatrix} v \\ v \end{pmatrix} \\ &= \begin{pmatrix} Av + Iv \\ Iv + Av \end{pmatrix} \\ &= \begin{pmatrix} (\lambda+1)v \\ (1+\lambda)v \end{pmatrix} \\ &= (\lambda+1)w, \end{aligned} \tag{4.9}$$

which shows that $w = \begin{pmatrix} v \\ v \end{pmatrix}$ is an eigenstate of A'. Similarly we see that $\begin{pmatrix} v \\ -v \end{pmatrix}$ is an eigenstate of A' with eigenvalue $\lambda - 1$. The two eigenstates of H_M^1 are $\frac{1}{\sqrt{2}}\begin{pmatrix} 1 \\ 1 \end{pmatrix}$ and $\frac{1}{\sqrt{2}}\begin{pmatrix} 1 \\ -1 \end{pmatrix}$ with eigenvalues 1 and -1 respectively. Thus, $|s\rangle$ is the eigenstate of H_M with largest eigenvalue (equal to n). □

The state $|s\rangle$ as defined in (4.6) is known as the initial state or starting state. While the problem Hamiltonian H_P is dependent on the problem structure as it depends on the to be optimized function f, it is still constructed the same for each type of optimization problem. The contrary is true for the mixing Hamiltonian H_M, which is independent of the type of optimization problem, although many different mixing Hamiltonians are possible for the same problem. The motivation behind mixing Hamiltonians will be covered more in depth in Section 4.4.1.

Next we define the unitaries $U(H_P, \gamma)$ and $U(H_M, \beta)$ as

$$U(H_P, \gamma) = e^{-i\gamma H_P}, \tag{4.10}$$

$$U(H_M, \beta) = e^{-i\beta H_M}, \tag{4.11}$$

where $\gamma \in [0, 2\pi)$ and $\beta \in [0, \pi)$ are a pair of real-valued angles. As seen earlier in Section 3.1, these are the solutions to the time-dependent Schrödinger equation with a constant Hamiltonian. Next we use a total of $2p$ of these angle parameters to create the parameterized quantum state $|\boldsymbol{\gamma}, \boldsymbol{\beta}\rangle$ by alternately applying the unitaries of both H_P and H_M p times to the initial state, resulting in

$$\begin{aligned} |\boldsymbol{\gamma}, \boldsymbol{\beta}\rangle &= U(H_M, \beta_p)U(H_P, \gamma_p) \cdots U(H_M, \beta_1)U(H_P, \gamma_1) |s\rangle \\ &= e^{-i\beta_p H_M} e^{-i\gamma_p H_P} \cdots e^{-i\beta_1 H_M} e^{-i\gamma_1 H_P} |s\rangle , \end{aligned} \tag{4.12}$$

with $|s\rangle$ again defined as in (4.6).

This state is referred to as $|\boldsymbol{\gamma}, \boldsymbol{\beta}\rangle$, as the state only depends on the choice of these sets of angles. Remember that this state is still a superposition of all possible n-qubit strings, and we can calculate the expectation of H_P in this state by measuring in the computational basis and evaluating

$$F_p(\boldsymbol{\gamma}, \boldsymbol{\beta}) = \langle \boldsymbol{\gamma}, \boldsymbol{\beta}| H_P |\boldsymbol{\gamma}, \boldsymbol{\beta}\rangle . \tag{4.13}$$

This expectation value is different for each set of angles $\boldsymbol{\gamma}, \boldsymbol{\beta}$. We are looking for the state that produces the highest possible expectation value, which in turn means the goal is to find the set of angles that produces this state. We define this maximum expectation value dependent on p as

$$M_p = \max_{\boldsymbol{\gamma}, \boldsymbol{\beta}} F_p(\boldsymbol{\gamma}, \boldsymbol{\beta}). \tag{4.14}$$

The algorithm uses a classical optimization routine to find the optimal set of angles $\boldsymbol{\gamma}, \boldsymbol{\beta}$ and then runs the algorithm with these angles to find the solution with the best expectation of H_P. As H_P encodes the objective value of the optimization problem, the quantum state that maximizes the expectation of H_P will be the desired approximation.

It turns out if this process is repeated indefinitely (in terms of p), we do indeed end up with the optimal solution, the eigenstate of H_P with highest eigenvalue (which is then also the optimal value of the optimization problem). This is because H_M only has non-negative off diagonal elements. Therefore, by the Perron-Frobenius theorem its largest eigenvalue is unique, and the same will be true for the Hamiltonian used in the evolution of (3.3). Thus, the adiabatic theorem applies and the resulting state is the eigenstate corresponding to the largest eigenvalue of H_P. Furthermore, in contrast to the Quantum Adiabatic algorithm, the probability of success monotonically increases with p. This is because the optimization process at $p - 1$ is also a solution at p through $(\gamma_{p-1}, 0, \beta_{p-1}, 0)$.

Finding these angles $\boldsymbol{\gamma}, \boldsymbol{\beta}$ is not trivial at all, and many classical algorithms can be used to try to find these angles, which will be discussed in 6.2. This concludes the original Quantum Approximate Optimization Algorithm, but this algorithm was quickly expanded on in [7], as will be discussed in Section 4.2.

4.2. Quantum Alternating Operator Ansatz

Although we discussed the setup of the original Quantum Approximate Optimization Algorithm, we can not apply this algorithm to MinVertexCover. MinVertexCover is a constrained optimization problem, while QAOA is designed to handle unconstrained optimization problems. This is why we look to the extension of QAOA, known as the Quantum Alternating Operator Ansatz as introduced in [7]. This section is largely based on this work. The setup of QAOA+ is similar to that of QAOA. We again consider an optimization problem to approximate a function f, however, we now also consider a domain of feasible points F, which is not necessarily all n-bit strings. Let $\mathcal{F}$ be the Hilbert space of dimension n, then QAOA+ considers two families of operators on $\mathcal{F}$:

- A family of problem operators $U_P(\gamma)$ analogous to the original problem operator as defined in (4.10)
- A family of mixing operators $U_M(\beta)$ analogous to the original mixing operator as defined in (4.11)

where β and γ are again real-valued parameters. Just as in Section 4.1, we then produce the parameterized quantum state $|\boldsymbol{\gamma}, \boldsymbol{\beta}\rangle$ through the application of these operators to a starting state $|s\rangle$, resulting in

$$|\boldsymbol{\gamma}, \boldsymbol{\beta}\rangle = U_M(\beta_p)U_P(\gamma_p)\cdots U_M(\beta_1)U_P(\gamma_1)\,|s\rangle\,. \tag{4.15}$$

So far QAOA+ is similar to QAOA, except with the addition of a feasible subspace F which is not necessarily equal to the entire space of n-bit strings. This feasible subspace makes QAOA+ applicable to more different kinds of optimization problems than QAOA, as QAOA only considers unconstrained problems. That is to say, QAOA only considers problems with the property that any n-bit string is a feasible solution, which is not the case for many optimization problems. The unitary problem operator is constructed the same as before through a combination of (4.3) and (4.10). However, with this possibility of considering constrained problems some restrictions have to be applied to the mixing unitaries.

Firstly, the mixing unitary must preserve the feasible subspace. For any value β the resulting family of mixing unitaries $U_M(\beta)$ must map feasible states to other feasible states. This achieves that the resulting state is a superposition of only feasible states, i.e. states $|x\rangle$ where $x \in F$. This requirement is easy to fulfill by looking at the matrix representation of a mixing Hamiltonian when designing such an operator and making sure no mixing is happening between feasible and infeasible states.

Secondly, the family of mixing operators must be able to reach the entire feasible subspace. For any combination of feasible basis states $x, y \in F$ there is some angle β' and some positive integer r such that

$$|\langle x|\, U_M^r(\beta')\,|y\rangle| > 0. \tag{4.16}$$

In other words, all states in the feasible subspace are connected by the mixer, even if it takes multiple applications ($r > 1$) of the mixer to reach said state. The combination of these two requirements ensures that we check all feasible solutions to the optimization problem without infeasible solutions being part of the state. It should be said that with these restrictions to the unitary, there are still many different possibilities for mixing Hamiltonians and unitaries.

Lastly, we consider the initial state $|s\rangle$, the state which the operators $U_p(\gamma)$ and $U_M(\beta)$ are applied to produce the state $|\boldsymbol{\gamma}, \boldsymbol{\beta}\rangle$. For QAOA the initial state was the state $|s\rangle$ as defined in (4.6), however, it is now possible that this state is not completely feasible (this would be the case if a single basis state of the superposition is not feasible). In fact, if the problem has any non-trivial constraints at all this initial state is no longer feasible. By definition $|s\rangle$ is the superposition of all possible n-bit strings, and if any constraints are placed on the problem it means one of these strings is no longer feasible, thus neither is the initial state. Therefore, for QAOA+ a specific suitable initial state is chosen for each problem, depending on the domain and the structure of the problem. Usually this is a state that is a feasible solution for every instance of the optimization problem. For example, in the independent set problem (the problem of finding a maximum size set of non-adjacent vertices), one would use the initial state

$$|s\rangle = |0\rangle^{\otimes n}\,, \tag{4.17}$$

as the empty set is always a feasible solution to the independent set problem.

Similar to the original QAOA, the algorithm then uses a classical optimizing routine to find the optimal set of angles $\boldsymbol{\gamma}, \boldsymbol{\beta}$ that find the highest expectation of H_P in the resulting state. As before, the quantum state that maximizes this expectation will be the desired approximation. In the next section the Quantum Alternating Operator Ansatz will be applied to the minimum vertex cover problem as it was discussed in Chapter 2.

4.3. Quantum Alternating Operator Ansatz Applied To MinVertexCover

We now apply the Quantum Alternating Operator Ansatz as described in Section 4.2 to MinVertexCover as described in Chapter 2. Remember that given $G = (V, E)$ with $|V| = n$, the goal is to minimize the size of a subset $C \subseteq V$ that covers V (so for every $(u, v) \in E$, $u \in C$ or $v \in C$). In this section we look at the problem Hamiltonian, mixing Hamiltonian and starting state.

4.3.1. Problem Hamiltonian

In QAOA+ the problem Hamiltonian is defined by (4.3). The objective value of a cover C is exactly the Hamming weight of the n-bit string representation of C. In other words, the problem Hamiltonian needs to scale a state by how many vertices have value 1 in the corresponding solution. This is achieved by defining the problem Hamiltonian as

$$H_P = \sum_{u \in V} W_u, \tag{4.18}$$

where we define $W_u = |1\rangle\langle 1|_u = I^{\otimes u-1} \otimes |1\rangle\langle 1| \otimes I^{\otimes n-u}$. To understand why this is the case we apply $|1\rangle\langle 1|$ to the two-dimensional computational basis vectors to get

$$W|0\rangle = |1\rangle\langle 1|0\rangle = \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}\begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix} = 0\begin{pmatrix} 1 \\ 0 \end{pmatrix} = 0|0\rangle, \tag{4.19}$$

$$W|1\rangle = |1\rangle\langle 1|1\rangle = \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}\begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix} = 1\begin{pmatrix} 0 \\ 1 \end{pmatrix} = 1|1\rangle. \tag{4.20}$$

Here we see that applying W to a vertex will scale that vertex's computational basis vector by 0 if the vertex is not in the cover and by 1 if it is in the cover. Applying this to all the vertices of the graph and adding the results therefore counts exactly how many vertices are in the cover. Thus, the expectation value of H_P is exactly $\text{Ham}(x)$, where $\text{Ham}(x)$ is the Hamming weight of x, defined as the amount of 1's in the bitstring x. This is shown in the following Lemma.

Lemma 4.2. *The expectation value of H_P as given by* (4.18) *in the quantum state $|x\rangle$, where x is a bitstring, is the Hamming weight of x.*

Proof. The expectation value of an observable $\mathcal{A}$ in the state ψ is given by

$$\langle A\rangle_\psi = \langle\psi|A|\psi\rangle. \tag{4.21}$$

Thus, the expectation of H_P as given by (4.18) in the state $|x\rangle$ will be

$$\begin{aligned} \langle H_P\rangle_x &= \langle x|H_P|x\rangle \\ &= \langle x|\sum_{u\in V} W_u|x\rangle \\ &= \sum_{u\in V}\langle x|W_u|x\rangle. \end{aligned} \tag{4.22}$$

For every u we get that if bit u is set to 0 in bitstring x, then $W_u|x\rangle = 0|x\rangle = 0$ by (4.19), and if bit u is set to 1 in bitstring x, then $W_u|x\rangle = 1|x\rangle = |x\rangle$ by (4.20). Therefore, we find

$$\begin{aligned} \langle H_P\rangle_x &= \sum_{u\in V}\langle x|W_u|x\rangle \\ &= \sum_{u\in V \text{ s.t. } x_u=1}\langle x|x\rangle \\ &= \text{Ham}(x), \end{aligned} \tag{4.23}$$

as the inner product $\langle x|x\rangle$ is always 1 for every bitstring x and the number of ones in a bitstring is exactly the Hamming weight of x. Thus, we see the expectation value of H_P is the Hamming weight of x. □

We want the problem Hamiltonian to encode the objective value $f(x)$ given a solution x through (4.3). In the case of MinVertexCover the objective value is the number of vertices in the cover, so the Hamming weight of a solution x, so the problem Hamiltonian H_P as given by (4.18) encodes this well.

4.3.2. Mixing Hamiltonian

For the mixing Hamiltonian we first note that mixing between states corresponds to adding or removing vertices from a cover in between solutions. For example, mixing between the states $|0101\rangle$ and $|0100\rangle$ is the same as removing the fourth vertex from the cover. Remember that the mixing rules of QAOA+ state that a mixing Hamiltonian must preserve the feasible subspace. This means we need to make sure a solution is still feasible when removing a vertex from the cover (obviously it will also still be feasible if a vertex is added to the cover because it was originally feasible as well). Therefore, a vertex u can only be swapped in or out of the current vertex cover C if all edges incident to u will still be covered if u leaves C, in which case we end up with another possible solution. In other words, we can only swap u in or out of the cover if all vertices adjacent to u are currently in the cover (and thus set to 1), this way all edges incident to u still have an endpoint in the cover. To see this idea more clearly we refer to Figure 4.1.

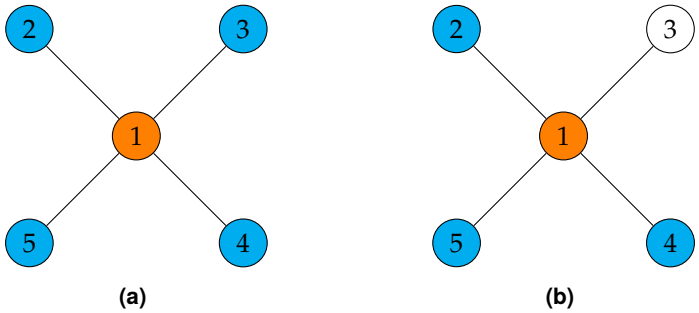


Figure 4.1: Example graph with the to be swapped vertex in orange and the rest of the current cover in cyan. In 4.1a all neighbours of the orange vertex are in the cover, while this is not the case in 4.1b.

In Figure 4.1a we see that the orange vertex can be removed from the cover as all its neighbours are in the cover as well. In Figure 4.1b the orange vertex can not be removed from the cover as vertex 3 is not in the cover, so by removing the orange vertex the edge $\{1, 3\}$ will not be covered anymore and we will have left the feasible subspace. Formulating these requirements in the general case as a mixing Hamiltonian gives

$$H_M = \sum_{u \in V} \left(X_u \prod_{v \in V \text{ s.t. } \{u,v\} \in E} W_v \right), \tag{4.24}$$

where again $W_u = I^{\otimes u-1} \otimes |1\rangle\langle 1| \otimes I^{\otimes n-u}$ and similarly we now define $X_u = I^{\otimes u-1} \otimes X \otimes I^{\otimes n-u}$. Here the product of W_v checks if all the neighbouring vertices of u are in the cover, and if this is the case, the X_u gate is applied to vertex u. We again sum over all vertices to be able to reach the entire subspace by swapping each of the vertices in or out of the cover where the vertex cover property allows it.

4.3.3. Starting State

For QAOA one would always use the initial state to be the equal superposition state of all possible solutions, given by (4.6). However, now not all n-bit strings are possible solutions, so instead an initial state is used that is a feasible state for every instance of MinVertexCover as discussed in Section 4.2. For MinVertexCover one solution that is feasible for any instance of the problem is

$$|s\rangle = |1\rangle^{\otimes n}, \tag{4.25}$$

which is the trivial vertex cover $C = V$. The only other starting states feasible for any instance of MinVertexCover are the n states of the form

$$|s^i\rangle = |1\rangle^{\otimes i-1} |0\rangle |1\rangle^{\otimes n-i}. \tag{4.26}$$

Note that these $n+1$ different initial states are indeed feasible for every MinVertexCover instance, as for any edge not to be covered both its endpoints need to not be in a cover. For the starting state defined in (4.25) all vertices are in the cover, and for the starting states defined in (4.26) exactly one vertex is not in the cover. To show that no other starting state is feasible for every MinVertexCover instance, consider any other initial solution s'. As s' is not any of the states given by (4.25) and (4.26), there is are at least 2 indices $i, j \in \{1, ..., n\}$ such that $s'_i = 0$. Now construct any graph $G = (V, E)$ with $\{v_i, v_j\} \in E$, and see that s' is not a feasible solution to MinVertexCover for this instance, as s' will not cover the edge $\{v_i, v_j\}$.

Any of the starting states from (4.25) and (4.26) is indeed a valid starting state, but usually only the starting state from (4.25) is used, as applying the mixing Hamiltonian as little as possible is preferred for computational purposes. The initial state from (4.25) will be used to produce the results in thesis. As the optimal solution is unknown and depends on the problem instance, it is not possible to know which of the starting states $|s^i\rangle$ is closest to the optimal solution, and picking one at random might result in picking a solution that is farther from the optimal than starting state $|s\rangle$ from (4.25). It is also more reasonable to use the starting state from (4.25) from a testing perspective, as using a different starting state could be an advantage in some instances, but a disadvantage in others.

Now that the explicit forms of the problem Hamiltonian, mixing Hamiltonian and initial state for MinVertexCover have all been discussed, we move on to an example to show what the matrix representations of the problem Hamiltonian en the mixing Hamiltonian look like in a specific problem instance.

Example
Let us construct the problem Hamiltonian from (4.18) and the mixing Hamiltonian from (4.24) for the triangle graph seen in Figure 4.2.

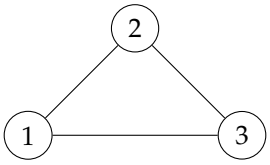


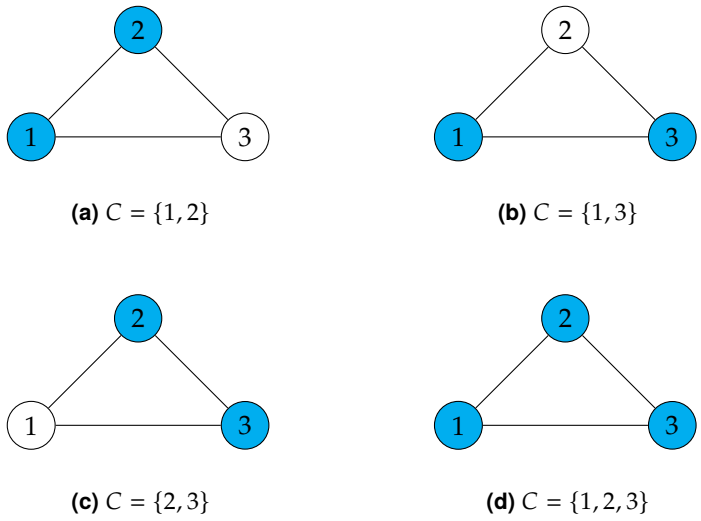
Figure 4.2: Triangle graph

Calculating the problem Hamiltonian H_P from (4.18) gives the following matrix representation of H_P:

$$H_P = \sum_{u \in V} W_u = |1\rangle\langle 1| \otimes I \otimes I + I \otimes |1\rangle\langle 1| \otimes I + I \otimes I \otimes |1\rangle\langle 1|$$
$$= \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 3 \end{pmatrix}, \tag{4.27}$$

as expected. The value of each diagonal element is equal to the number of ones in the binary number that belongs to that row (starting with 0 at row 1). For example, the fifth diagonal element is 1 as the fifth row corresponds to the state $|100\rangle$, which has exactly one 1 in it.

For the mixing Hamiltonian we note that the only feasible covers are $C = \{1,2\}, C = \{1,3\}, C = \{2,3\}$, and $C = \{1,2,3\}$ as can be seen in Figure 4.3.



(a) $C = \{1,2\}$ **(b)** $C = \{1,3\}$

(c) $C = \{2,3\}$ **(d)** $C = \{1,2,3\}$

Figure 4.3: All feasible covers of the triangle graph

This means we want the mixing Hamiltonian to map the covers to each other as follows:

$$H_M |110\rangle = |111\rangle , \tag{4.28}$$
$$H_M |101\rangle = |111\rangle , \tag{4.29}$$
$$H_M |011\rangle = |111\rangle , \tag{4.30}$$
$$H_M |111\rangle = |110\rangle + |101\rangle + |011\rangle , \tag{4.31}$$

as that would show that the feasible subspace is preserved and that every feasible solution can be reached from the initial state. To find the matrix representation of the mixing Hamiltonian we use (4.24) to find

$$H_M = \sum_{u \in V} \left(X_u \prod_{v \in V \ s.t. \ \{u,v\} \in E} W_v \right)$$

$$= X_1 W_2 W_3 + X_2 W_1 W_3 + X_3 W_1 W_2$$

$$= \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \end{pmatrix} . \tag{4.32}$$

Finally, for the initial state $|s\rangle$ we use (4.25) to get

$$|s\rangle = |111\rangle . \tag{4.33}$$

This concludes the example. Next the workings of mixing Hamiltonians will be explored more, so that in Chapter 5 new mixing Hamiltonians can be introduced.

4.4. Mixing Hamiltonians

In the previous sections the setup of both the Quantum Approximate Optimization algorithm and the more recent Quantum Alternating Operator Ansatz were introduced and applied to MinVertexCover, but the reasoning behind the construction of parts of the algorithms has not been discussed. In this section, we look at the motivation behind mixing Hamiltonians, as well as how they can be intuitively interpreted.

4.4.1. Motivation

The need for a mixing Hamiltonian in QAOA and QAOA+ is not inherently clear. One might wonder if it possible to apply the problem unitary $U(H_P, \gamma)$ from (4.10) directly to the starting state and then measure in the computational basis. As discussed before, the mixing Hamiltonian in the original QAOA framework is not problem dependent, thus one might question why it is required. Let us apply a problem unitary to the starting state (4.6) of the original QAOA framework, resulting in

$$\begin{aligned} U(H_P, \gamma) |s\rangle &= e^{-i\gamma H_P} |s\rangle \\ &= e^{-i\gamma H_P} \frac{1}{\sqrt{2^n}} \sum_x |x\rangle \\ &= \sum_x \frac{e^{-i\gamma H_P}}{\sqrt{2^n}} |x\rangle , \end{aligned} \tag{4.34}$$

where we use (4.6) in the first step. Note that H_P is a diagonal matrix with $f(x)$ on the diagonal, therefore the application of e^{H_P} to the quantum state $|x\rangle$ results in $e^{f(x)} |x\rangle$. Continuing with the above equation we get

$$\begin{aligned} U(H_P, \gamma) |s\rangle &= \sum_x \frac{e^{-i\gamma H_P}}{\sqrt{2^n}} |x\rangle \\ &= \sum_x \frac{e^{-i\gamma f(x)}}{\sqrt{2^n}} |x\rangle . \end{aligned} \tag{4.35}$$

Let us measure the resulting state in the computational basis, then the probability for any computational basis state $|y\rangle$ (which correspond exactly to the 2^n n-bit strings) to be the result of the measurement is given by

$$\begin{aligned} P(y) &= \| \langle y | U(H_P, \gamma) |s\rangle \|^2 \\ &= \left\| \langle y | \sum_x \frac{e^{-i\gamma f(x)}}{\sqrt{2^n}} |x\rangle \right\|^2 \\ &= \left\| \sum_x \frac{e^{-i\gamma f(x)}}{\sqrt{2^n}} \langle y|x\rangle \right\|^2 . \end{aligned} \tag{4.36}$$

Remember that x and y are both computational basis states. This means that $\langle y|x\rangle$ results in 1 if and only if $y = x$ and results in 0 otherwise. Continuing with the above formulate we thus get

$$\begin{aligned} P(y) &= \left\| \sum_x \frac{e^{-i\gamma f(x)}}{\sqrt{2^n}} \langle y|x\rangle \right\|^2 \\ &= \left\| \frac{e^{-i\gamma f(y)}}{\sqrt{2^n}} \right\|^2 \\ &= \frac{\left\| e^{-i\gamma f(y)} \right\|^2}{2^n} \\ &= \frac{1}{2^n}. \end{aligned} \tag{4.37}$$

This result does not depend on which computational basis state $|y\rangle$ is chosen, and thus we see that even after applying $U(H_P, \gamma)$, we still get a uniform distribution across all computational basis states when performing measurements. While applying the problem unitary, only a relative phase is added, which does not affect measuring results. To fix this problem, the mixing Hamiltonian is introduced, which transforms the quantum state so that a measurement no longer results in a uniform distribution.

4.4.2. Intuitive Interpretation of Mixing Hamiltonians

Now that we discussed why mixing Hamiltonians are needed, we will discuss what a Hamiltonian does when applied to a certain state. Understanding this process will give a good basis to construct more complex mixing Hamiltonians. Remember that in the original QAOA framework the domain F was set to all n-bit strings, so all these strings are feasible solutions. The mixing Hamiltonian applied to this problem was the transverse field Hamiltonian, given by (4.5). Let us apply this mixing Hamiltonian directly to a quantum state $|z\rangle$ of the form $|z_1\rangle |z_2\rangle |z_3\rangle ... |z_n\rangle$ with $z \in \{0,1\}^n$, corresponding to the n-bit string z. This results in

$$\begin{aligned} H_M |z_1\rangle |z_2\rangle |z_3\rangle ... |z_n\rangle = \; & |\overline{z_1}\rangle |z_2\rangle |z_3\rangle ... |z_n\rangle \\ & + |z_1\rangle |\overline{z_2}\rangle |z_3\rangle ... |z_n\rangle \\ & + |z_1\rangle |z_2\rangle |\overline{z_3}\rangle ... |z_n\rangle \\ & \; \; \; \dots \\ & + |z_1\rangle |z_2\rangle |z_3\rangle ... |\overline{z_n}\rangle \,, \end{aligned} \tag{4.38}$$

where we define $|\overline{x}\rangle = |1 - x\rangle$. This mixing Hamiltonian maps the state $|x\rangle$ of an n-string x to all other states that differ in exactly one bit from x. In other words, the Hamiltonian maps $|x\rangle$ to the superposition of all n possible outcomes when flipping exactly one bit of x. We can summarize this property as

$$\langle x| H_M |y\rangle = \begin{cases} 1, & x, y \in F \text{ and } \mathrm{Ham}(x,y) = 1, \\ 0, & \text{otherwise,} \end{cases} \tag{4.39}$$

where $\mathrm{Ham}(x, y)$ is the Hamming distance between strings x and y, defined as the number of positions the strings x and y are different. The Hamming distance between a string x and the string of zeroes (so the number of ones in string x) is again the Hamming weight of x (Ham(x)). Starting from the state of any n-bit string, we could "reach" any other n-bit string by applying the Hamiltonian enough times (although only a maximum of n applications is needed, as this is the largest Hamming distance between any two strings of length n).

This is shown by considering an arbitrary quantum state of the form

$$|\phi\rangle = \sum_{x^k \in K} c_k |x^k\rangle \,, \tag{4.40}$$

which is a non-empty superposition of n-bit strings, each of the form

$$|x^k\rangle = |x_1^k\rangle |x_2^k\rangle |x_3^k\rangle ... |x_n^k\rangle \tag{4.41}$$

with $x^k \in \{0,1\}^n$. Here K is the set of n-bit strings which are part of the superposition, and have thus been reached already. Consider a target string w of the form $|w_1\rangle |w_2\rangle |w_3\rangle ... |w_n\rangle$, again with $w_i \in \{0,1\}$, which we are trying to reach. If $w \in K$, then we are done. If not, there is some $z \in K$ with smallest Hamming distance to w. Let this Hamming distance be $m \neq 0$ (otherwise $w = z \in K$). As this distance is non-zero, there must be an $i \in \{1, .., n\}$ such that $z_i \neq w_i$. Applying the mixing Hamiltonian to the superposition ϕ results in a superposition of many states, one of which is the state

$$z^* = |z_1\rangle ... |z_{i-1}\rangle |w_i\rangle |z_{i+1}\rangle ... |z_n\rangle \,. \tag{4.42}$$

This means after applying H_M we have $z^* \in K$, and we have constructed a state in K with $\mathrm{Ham}(z^*, w) = m - 1$. Repeating this process m times results in $w \in K$, meaning the target state is part of our superposition ϕ. Therefore, by applying the mixing Hamiltonian enough times to any state, it is possible to create a superposition containing any other state.

An example of a Hamiltonian that does not reach the entire subspace is the Hamiltonian defined by

$$\begin{aligned} H_M |z_1\rangle |z_2\rangle ... |z_{n-1}\rangle |z_n\rangle = \; & |\overline{z_1}\rangle |z_2\rangle ... |z_{n-1}\rangle |z_n\rangle \\ & + \; |z_1\rangle |\overline{z_2}\rangle ... |z_{n-1}\rangle |z_n\rangle \\ & \quad \dots \\ & + \; |z_1\rangle |z_2\rangle ... |\overline{z_{n-1}}\rangle |z_n\rangle \,, \end{aligned} \tag{4.43}$$

as this Hamiltonian has no effect on the final qubit. Therefore, any state of the form $|z\rangle = |z_1\rangle ... |z_{n-1}\rangle |0\rangle$ could never reach a state of the form $w = |w_1\rangle ... |w_{n-1}\rangle |1\rangle$, and thus $|\langle z| U_M^r(\beta') |w\rangle| = 0$ for all β' and r, violating (4.16).

Now that the idea behind mixing Hamiltonians is more apparent, we can start to better understand the choice of the initial state from the original QAOA framework in the context of mixing Hamiltonians. As can be seen in (4.38), the mixing Hamiltonian maps a computational basis state to a not necessarily normalized superposition of possibly many other computational basis states. When applying the unitary $U(H_M, \beta)$ from (4.11) to a normalized quantum state, the resulting state will also be normalized. Let us apply $U(H_M, \beta)$ to quantum state ϕ to obtain state ϕ', then the probability amplitude of a basis state in ϕ is likely lower than the probability amplitude of that same basis state in ϕ'. Theoretically, some basis states could have a higher amplitude in the newer state ϕ' due to the addition of the amplitudes of multiple different states, but the total amplitude of combined states that were already in the decomposition of the previous quantum state ϕ will decrease as long as the entire feasible subspace is not reached and H_M is well defined through (4.16). See the following example.

Example
Let a quantum state ϕ be

$$|\phi\rangle = \frac{\sqrt{3}}{2} |00\rangle + \frac{\sqrt{2}}{4} |01\rangle + \frac{\sqrt{2}}{4} |10\rangle \,. \tag{4.44}$$

Note that this expression of ϕ is normalized as $(\frac{\sqrt{3}}{2})^2 + (\frac{\sqrt{2}}{4})^2 + (\frac{\sqrt{2}}{4})^2 = \frac{3}{4} + \frac{1}{8} + \frac{1}{8} = 1$. Applying the transverse field Hamiltonian as defined in (4.5) results in

$$\begin{aligned} H_M |\phi\rangle &= H_M(\frac{\sqrt{3}}{2} |00\rangle + \frac{\sqrt{2}}{4} |01\rangle + \frac{\sqrt{2}}{4} |10\rangle) \\ &= \frac{\sqrt{3}}{2} H_M |00\rangle + \frac{\sqrt{2}}{4} H_M |01\rangle + \frac{\sqrt{2}}{4} H_M |10\rangle \\ &= \frac{\sqrt{3}}{2}(|01\rangle + |10\rangle) + \frac{\sqrt{2}}{4}(|00\rangle + |11\rangle) + \frac{\sqrt{2}}{4}(|00\rangle + |11\rangle) \\ &= \frac{\sqrt{2}}{2} |00\rangle + \frac{\sqrt{3}}{2} |01\rangle + \frac{\sqrt{3}}{2} |10\rangle + \frac{\sqrt{2}}{2} |11\rangle \,. \end{aligned} \tag{4.45}$$

This expression of the state is not normalized anymore as the squares of the probability amplitudes add to $(\frac{\sqrt{2}}{2})^2 + (\frac{\sqrt{3}}{2})^2 + (\frac{\sqrt{3}}{2})^2 + (\frac{\sqrt{2}}{2})^2 = \frac{1}{2} + \frac{3}{4} + \frac{3}{4} + \frac{1}{2} = 2\frac{1}{2}$. Normalizing this expression by dividing the amplitudes by $\sqrt{2\frac{1}{2}}$ and comparing it to the original state results in

$$|\phi\rangle = \frac{\sqrt{3}}{2} |00\rangle + \frac{\sqrt{2}}{4} |01\rangle + \frac{\sqrt{2}}{4} |10\rangle \,, \tag{4.46}$$

$$\begin{aligned} H_M |\phi\rangle &= \frac{\sqrt{2}}{\sqrt{(10)}} |00\rangle + \frac{\sqrt{3}}{\sqrt{(10)}} |01\rangle + \frac{\sqrt{3}}{\sqrt{(10)}} |10\rangle + \frac{\sqrt{2}}{\sqrt{(10)}} |11\rangle \\ &= \frac{\sqrt{5}}{5} |00\rangle + \frac{\sqrt{30}}{10} |01\rangle + \frac{\sqrt{30}}{10} |10\rangle + \frac{\sqrt{5}}{5} |11\rangle \,. \end{aligned} \tag{4.47}$$

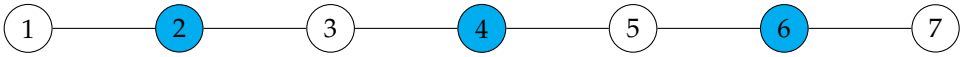
Comparing each of the states from (4.46) and (4.47), we can see that the probability amplitudes of both the $|01\rangle$ state and the $|10\rangle$ state have increased, as $\frac{\sqrt{30}}{10} > \frac{\sqrt{2}}{4}$. However, the total probability amplitude of the original states ($|00\rangle, |01\rangle$ and $|10\rangle$) has decreased from 1 to $1 - (\frac{\sqrt{5}}{5})^2 = \frac{4}{5}$. This shows that, although some of the states could have a higher amplitude after applying the Hamiltonian, the combined amplitudes of all the states that were in the superposition before applying the Hamiltonian decreases unless the original state is a superposition of the entire domain. This concludes this example.

Because the probabilities will keep decreasing for most basis states as we keep applying the unitary (until we reach the entire feasible subspace), we want to apply the mixing unitary as little as possible. This intuitively makes sense as well, as it means we want to choose our initial state as close to the desired solution state as possible. Of course the problem unitary is also applied to the state as well in between mixing unitaries to up the probability of measuring this solution state, but having to apply these operators as little as possible is always good for both runtime and noise purposes. One could argue that choosing an initial state with a good objective value is superior. This way the algorithm has apply the unitaries $U(H_P, \gamma)$ and $U(H_M, \beta)$ less frequently to reach the optimal solution, but good solutions do not necessarily have small Hamming distance to each other. This can be seen in the following example.

Example
Consider the minimum vertex cover problem on the graph of Figure 4.4a. A solution x with a good objective value can be seen in Figure 4.4b, as it has objective value 4. The optimal solution x^* can be seen in Figure 4.4c, with objective value 3.

(a) Example graph

(b) Initial state solution x ($|1010101\rangle$)



(c) Optimal state solution x^*($|0101010\rangle$)

Figure 4.4: An example graph (a) with a possible good initial state solution (b) and the optimal solution (c).

The quantum state corresponding to the solution x in Figure 4.4b is $|1010101\rangle$, while the quantum state corresponding to the optimal solution x^* in Figure 4.4c is $|0101010\rangle$. Assume the optimal solution is not known, but it is known that the optimal vertex cover has size 3. Solution x could then be considered a good initial solution, since it has an objective value very close to the value of the optimal solution. However, the Hamming distance between x and x^* is $\text{Ham}(1010101, 0101010) = 7$. This means the mixing Hamiltonian has to be applied at least 7 times before x^* has any amplitude in the superposition. This shows that the the mixer could have to be applied many times to mix between solutions that are very close in objective value. This concludes this example.

Because good solutions do not always have small Hamming distance to each other, it makes sense to pick the initial state that equally distributes the probability of measurement among all states. Through this reasoning we again arrive at the original state from (4.6). As discussed earlier, we can not use this initial state for problems with a restricted domain like MinVertexCover, so this idea is not applicable to those optimization problems. However, it is still valuable to understand mixing Hamiltonians and their effects on different initial states when designing other mixing Hamiltonians for more complex optimization problems.

In conclusion, the application of a mixing Hamiltonian is needed to shift the probability amplitudes away from a uniform distribution across all computational basis states. Furthermore, it has been discussed how a mixing Hamiltonian operates on a state and what the limitations of possible starting states are. Now that the Quantum Alternating Operator Ansatz has been applied to MinVertexCover and the ideas behind mixing Hamiltonians have been covered, we can start looking at different setups for the same problem. It has been shown earlier this section that different starting states are possible than the starting state from (4.25), however, using them does not seem to lead to a significant advantage, be it a shorter quantum circuit or a faster algorithm. Although the problem Hamiltonian is mostly fixed, many different mixing Hamiltonians are possible as long as they both preserve and reach the entire feasible subspace. This thesis will now introduce new mixing Hamiltonians in Chapter 5.

5

Second Degree Mixing Hamiltonians for MinVertexCover

Previously, in Section 4.3 we saw how the Quantum Alternating Operator Ansatz would apply to MinVertexCover. The mixing Hamiltonian introduced in Section 4.3 (given by (4.24)) will from this point forward be referred to as $H_{M_{V1}}$, as in this chapter we will construct new mixing Hamiltonians. When using $H_{M_{V1}}$, the only mixing between solutions occurs if the target and original solutions are exactly Hamming distance 1 away, while meeting the requirement that the resulting solution is a vertex cover. This mixer therefore does not mix between all pairs of solutions, and we can extend this mixer by including more terms which mix between solutions at Hamming distance 2.

5.1. Mixing Hamiltonian V2

We try to reach solutions that are either Hamming distance 1 or 2 away while still meeting the requirement that the resulting solution is in the feasible subspace. To do this we have to include more terms in the mixing Hamiltonian. Each of the terms in the Hamiltonian from (4.24) swaps a different vertex in or out of the cover if all the neighbours of a vertex are set to 1. Next, we want to add more terms to swap two vertices in or out of the cover at the same time. This is done by applying two X gates simultaneously to two different vertices. We look at applying the X gates to vertices u and v and only consider the case where $\{u, v\} \notin E$:

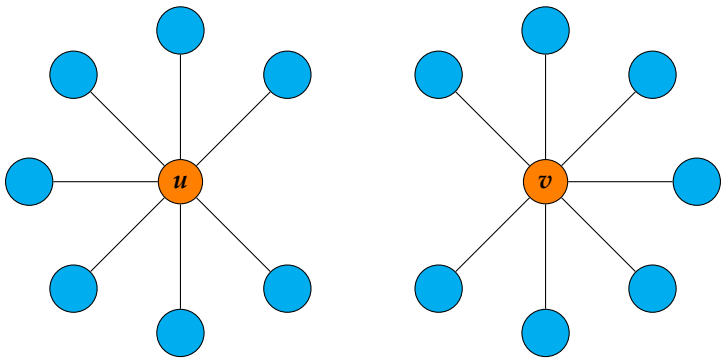


Figure 5.1: Example graph where u and v are not connected. Every vertex adjacent to u or v is in the cover.

If $\{u, v\} \notin E$, we have the same requirement as in the original mixing Hamiltonian $H_{M_{V1}}$. Vertices u and v are not connected, we only need to check for each of the vertices that all of their neighbours are set to 1, as there is no other way the resulting cover is not feasible, see Figure 5.1. This is similar to Figure 4.1 in the sense that we can consider each of the vertices separately as they are not connected, but now we swap both vertices in or out of the cover with one application of the mixing Hamiltonian. This means

that we only have to make sure all of the vertices of both u and v are already in the cover, which is reflected in the Hamiltonian by multiplying by the product of W's. If this is the case, we can apply both X_u and X_v at the same time. This results in the term given by

$$X_u X_v \prod_{\substack{w \in V \text{ s.t. } \{u,w\} \in E \\ \text{or } \{v,w\} \in E}} W_w. \tag{5.1}$$

Finishing this case, we sum over all possible combinations of vertices which are non-adjacent to find

$$\sum_{u \in V, v \in V \text{ s.t.} \{u,v\} \notin E} \left(X_u X_v \prod_{\substack{w \in V \text{ s.t. } \{u,w\} \in E \\ \text{or } \{v,w\} \in E}} W_w \right) \tag{5.2}$$

as the term of the mixer swapping two non-adjacent vertices at the same time. Combining this with the terms of the original Hamiltonian of (4.24), we get the new Hamiltonian $H_{M_{V2}}$ given by

$$\begin{aligned} H_{M_{V2}} = &\sum_{u \in V} \left(X_u \prod_{v \in V \ s.t. \ \{u,v\} \in E} W_v \right) \\ &+ \sum_{u \in V, v \in V \text{ s.t.} \{u,v\} \notin E} \left(X_u X_v \prod_{\substack{w \in V \text{ s.t. } \{u,w\} \in E \\ \text{or } \{v,w\} \in E}} W_w \right). \end{aligned} \tag{5.3}$$

This concludes the mixing Hamiltonian $H_{M_{V2}}$.

5.2. Mixing Hamiltonian V3

The case where $\{u, v\} \in E$ is more complicated, as we have to account for more scenarios where swapping vertices out of the solution results in an infeasible solution. Like before, all of the neighbours of both vertices have to be in our current solution to be able to swap either vertex out of the solution. However, this time we could face the problem that by swapping both vertices out of the solution at the same time we arrive outside the domain by not covering edge $\{u, v\}$, see Figure 5.2.

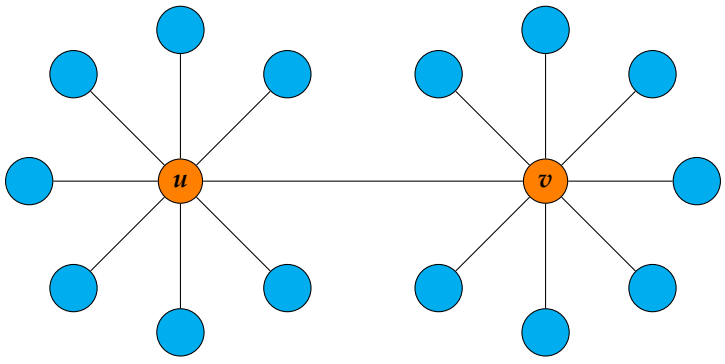


Figure 5.2: Example graph where u and v are connected. Every vertex adjacent to u or v is in the cover.

As said before, if we try to swap both vertices u and v out of the cover at the same time, the edge $\{u, v\}$ will not be covered, however, it is still possible to swap u and v at the same time without resulting in an infeasible solution. If exactly one of the two vertices is already in the cover and the other is not, we can still cover edge $\{u, v\}$. For example, assume u is in the cover but v is not, then $\{u, v\}$ will be covered by u before the swap. After u leaves the cover and v enters the cover, $\{u, v\}$ will still be covered, now by v instead of u. This idea gives the following terms:

$$X_u X_v W_u \overline{W_v} \prod_{\substack{w \in V \text{ s.t. } w \neq u,\ w \neq v \\ \{u,w\} \in E \text{ or } \{v,w\} \in E}} W_w \ + \ X_u X_v \overline{W_u} W_v \prod_{\substack{w \in V \text{ s.t. } w \neq u,\ w \neq v \\ \{u,w\} \in E \text{ or } \{v,w\} \in E}} W_w, \tag{5.4}$$

where still $W_u = |1\rangle \langle 1|_u$ and similarly we now define $\overline{W_v} = |0\rangle \langle 0|_v$. Note that we do not include $w = u$ and $w = v$ in the product, as these cases of W_w are already accounted for outside of the product. X_u and X_v again handle the swapping of u and v into and out of the cover. The first of the two terms corresponds to u already being in the cover and v not being in the cover, as $W_u \overline{W_v}$ checks that u has value 1 and v has value 0. Similarly the second term corresponds to u not being in the cover and v already being in the cover.

Originally, this term was constructed to not iterate over $w \in W$ such that $\{v, w\} \in E$ in the product of the first term. This was thought to be possible as it is known that the initial state before applying the Hamiltonian is already a vertex cover, otherwise we would not be in the feasible subspace. Outside of the product, the factor $\overline{W_v}$ is applied already, so the Hamiltonian only affects solutions where v is not in the cover. In these solutions all edges incident to v have to be covered by the neighbours of v, as v itself is not in the cover. Therefore, it was assumed there was no need to check if all of v's neighbours are in the cover, as we know they have to be, otherwise the initial state would not be a feasible solution. In contrast, we still need to check all of u's neighbours are in the cover, as they do not necessarily have to be before applying the Hamiltonian (u itself could cover these edges). The same applies in reverse to the second term, where we would not iterate over $w \in W$ such that $\{u, w\} \in E$.

Although this reasoning is correct if the Hamiltonian would only be applied to states which are indeed vertex covers, constructing the Hamiltonian this way results in a vital problem. If the Hamiltonian would not iterate over $w \in W$ such that $\{v, w\} \in E$ as well, then it would be possible to mix a solution which is not a vertex cover to a solution that is a vertex cover. Since the Hamiltonian does not mix solutions which are vertex covers to solutions which are not vertex covers by design, this makes the designed Hamiltonian not Hermitian, which is a requirement for a Hamiltonian to represent the energy of a quantum system as described in Section 3.1. Thus, we still iterate over all $w \in W$ such that $\{u, w\} \in E$. We again sum over all possible combinations of vertices which are adjacent to find

$$\sum_{u \in V, v \in V \text{ s.t.} \{u,v\} \notin E} \Bigg(X_u X_v W_u \overline{W_v} \prod_{\substack{w \in V \text{ s.t. } w \neq u,\ w \neq v \\ \{u,w\} \in E \text{ or } \{v,w\} \in E}} W_w + X_u X_v \overline{W_u} W_v \prod_{\substack{w \in V \text{ s.t. } w \neq u,\ w \neq v \\ \{u,w\} \in E \text{ or } \{v,w\} \in E}} W_w \Bigg) \tag{5.5}$$

as the term of the mixer swapping two adjacent vertices at the same time. For reading purposes it will be split into two different sums and written as

$$\sum_{u \in V, v \in V \text{ s.t.} \{u,v\} \notin E} \left(X_u X_v W_u \overline{W_v} \prod_{\substack{w \in V \text{ s.t. } w \neq u,\ w \neq v \\ \{u,w\} \in E \text{ or } \{v,w\} \in E}} W_w \right) + \sum_{u \in V, v \in V \text{ s.t.} \{u,v\} \notin E} \left(X_u X_v \overline{W_u} W_v \prod_{\substack{w \in V \text{ s.t. } w \neq u,\ w \neq v \\ \{u,w\} \in E \text{ or } \{v,w\} \in E}} W_w \right). \tag{5.6}$$

Combining these cases and adding the original mixer to swap only one vertex in or out at a time, we find the mixing Hamiltonian $H_{M_{V3}}$ which swaps either one or two vertices in/out at the same time to be

$$\begin{aligned} H_{M_{V3}} = &\sum_{u\in V}\left(X_u \prod_{v\in V \text{ s.t. } \{u,v\}\in E} W_v\right) \\ &+ \sum_{u\in V, v\in V \text{ s.t.}\{u,v\}\notin E}\left(X_u X_v \prod_{\substack{w\in V \text{ s.t. } \{u,w\}\in E \\ \text{or } \{v,w\}\in E}} W_w\right) \\ &+ \sum_{u\in V, v\in V \text{ s.t.}\{u,v\}\in E}\left(X_u X_v W_u \overline{W_v} \prod_{\substack{w\in V \text{ s.t. } w\neq u,\ w\neq v \\ \{u,w\}\in E \text{ or } \{v,w\}\in E}} W_w\right) \\ &+ \sum_{u\in V, v\in V \text{ s.t.}\{u,v\}\in E}\left(X_u X_v \overline{W_u} W_v \prod_{\substack{w\in V \text{ s.t. } w\neq u,\ w\neq v \\ \{u,w\}\in E \text{ or } \{v,w\}\in E}} W_w\right). \end{aligned} \tag{5.7}$$

We will refer to $H_{M_{V2}}$ and $H_{M_{V3}}$ as 'second degree' mixing Hamiltonians due to their ability to switch 2 vertices in or out of a cover at the same time. Similarly, the specific terms of the Hamiltonians which swap 2 specific vertices in or out at the same time will be called second degree terms. Continuing this terminology, $H_{M_{V1}}$ is thus a first degree mixing Hamiltonian. The construction of third and higher degree mixing Hamiltonians will be discussed in Section 9.1.

One could wonder how the new second degree terms of mixing Hamiltonian $H_{M_{V3}}$ compare to the square of the first degree mixing Hamiltonian $H_{M_{V1}}$. The mixing Hamiltonian $H_{M_{V3}}$ encodes all possible swaps between two solutions when swapping two vertices at the same time, while $(H_{M_{V1}})^2$ encodes all possible combinations of swapping one vertex in or out of the solution, and then another afterwards. This means $(H_{M_{V1}})^2$ takes the possibility into account that a vertex is swapped out and then swapped back in again (or vice versa), which the second degree terms of $H_{M_{V3}}$ do not do. This results in elements on the diagonal of $(H_{M_{V1}})^2$ where after swapping twice one returns to the original solution. Furthermore, in $(H_{M_{V1}})^2$ scales some elements as these solutions can be reached via multiple ways. We show these differences with an example.

Example

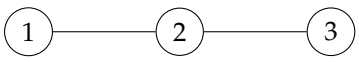


Figure 5.3: Example graph

Consider the graph given in Figure 5.3. The second degree terms of $H_{M_{V3}}$ (i.e. $H_{M_{V3}} - H_{M_{V1}}$) in matrix form as given by (5.2) and (5.6) are

$$H_{M_{V3}} - H_{M_{V1}} = (5.2) + (5.6) = \begin{pmatrix} 0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&1 \\ 0&0&0&0&0&1&1&0 \\ 0&0&0&0&0&0&0&0 \\ 0&0&0&1&0&0&1&0 \\ 0&0&0&1&0&1&0&0 \\ 0&0&1&0&0&0&0&0 \end{pmatrix}. \tag{5.8}$$

These are all possible ways to mix between solutions of the graph of Figure 5.3 which are Hamming distance 2 apart. For example, the value of the element at the third row and the final column is 1, as it is possible to swap between solutions $|010\rangle$ and $|111\rangle$ by swapping the first and third vertex at the same time. $(H_{M_{V1}})^2$ in its matrix representation is given by

$$(H_{M_{V1}})^2 = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 & 0 & 2 \\ 0 & 0 & 0 & 2 & 0 & 1 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 2 & 0 & 1 & 2 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 & 0 & 3 \end{pmatrix}. \tag{5.9}$$

As discussed earlier, there are now terms on the diagonal, since possible solutions can mix to themselves by swapping the same vertex first in and then out, or first out and then in. For example, the value of the element at the third row and third column is 2, as it both possible to mix $|010\rangle$ back to itself in two ways (mixing to $|110\rangle$ and back or mixing to $|011\rangle$ and back).

Furthermore, some of the terms that are non-zero in both Hamiltonians are scaled. The value of the element at the third row and final column in (5.9) is 2 (compared to 1 in (5.8)), as one can swap between $|010\rangle$ and $|111\rangle$ by either first swapping in the third vertex and then the first, or swapping in the first vertex first and then the third. Thus, the second degree terms of $H_{M_{V3}}$ can be viewed as $(H_{M_{V1}})^2$ with all diagonal elements set to 0 and all other nonzero elements set to 1.

Now that we constructed the new mixing Hamiltonians $H_{M_{V2}}$ and $H_{M_{V3}}$ we will next try to implement them to see how they compare against the original mixing Hamiltonian $H_{M_{V1}}$. The implementation of these Hamiltonians along with the choices of graph type and classical optimization methods will be discussed next in Chapter 6.

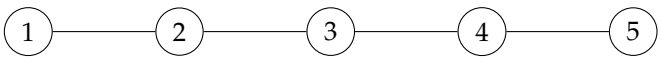
6 Implementation

In this chapter, we will discuss the graph used for benchmarking the new Hamiltonians $H_{M_{V2}}$ and $H_{M_{V3}}$, along with how both Hamiltonians were implemented. We also briefly discuss each of the classical optimization methods used to optimize the parameters used in the unitaries of QAOA+. Optimizing these parameters is crucial for finding good approximate solutions to MinVertexCover.

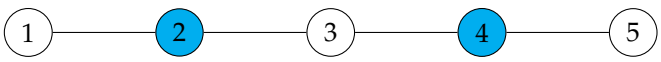
6.1. Graph choice and Hamiltonian construction

To test the performance of new mixing Hamiltonians $H_{M_{V2}}$ and $H_{M_{V3}}$, we run QAOA multiple times on a single test graph. Because the quantum circuit will become deeper (i.e. have more layers of gates) due to the difficulty of implementing the Hamiltonians through (5.3) and (5.7), we want to use a small graph where these new mixing Hamiltonians still have a possible advantage. As $H_{M_{V2}}$ and $H_{M_{V3}}$ swap two vertices in or out of the cover at the same time, we need our graph to still have two different solutions after removing two vertices. This way $H_{M_{V2}}$ and $H_{M_{V3}}$ can swap between these solutions. For example, for the triangle graph as seen in Figure 4.2 there is no difference between using $H_{M_{V1}}$ or $H_{M_{V2}}$ as mixing Hamiltonian, as it is not possible to remove two vertices from the cover without leaving the feasible subspace.

One of the smallest non-trivial graphs where it is possible to remove two vertices from the initial cover without leaving the feasible subspace would be four nodes in a connected line, known as the path graph P_4. However, since it is more clear to use a graph with a unique optimal solution for benchmarking purposes, we will instead use the path graph P_5.



(a) The path graph P_5



(b) Optimal solution of MinVertexCover on P_5

Figure 6.1: The path graph P_5 (a) and its optimal solution for MinVertexCover (b).

This graph was implemented in Python using the networkx package [36]. Quantum circuits were implemented in Python using qiskit [37] on the Aer simulator, while the implementation of the QAOA algorithm itself is largely based on code found at [38]. To implement the problem unitary of (4.18) we

note that $W = \frac{(I-Z)}{2}$. Summing over all these operators we see that we can take our unitary as

$$U_P = e^{-i\gamma \sum_{u \in V} Z_u}, \tag{6.1}$$

as the constant term caused by the I-terms will only affect the global phase as seen in [7]. Thus, we can implement this unitary using an $R_z(2\gamma)$-gate for each vertex of the graph.

The mixing Hamiltonians $H_{M_{V1}}$, $H_{M_{V2}}$, and $H_{M_{V3}}$ were implemented differently. Although it is likely possible to implement these Hamiltonians in a similar way to the problem Hamiltonian, these Hamiltonians were all implemented directly through their matrix form. All matrices were computed directly from equations (4.24), (5.3), and (5.7), after which the matrices were multiplied by their respective $-i\beta$-factor. The exponential of the resulting matrices was then computed, after which these exponentials were converted to an operator to be appended to the quantum circuit.

6.2. Classical Optimization Algorithms

As seen earlier in Chapter 4, both QAOA and QAOA+ use a classical optimization method to find parameters that produce a high expectation of H_P. In this section the optimization methods used in our implementation of QAOA will be discussed, although the use of other optimization methods is possible as well. Each of the sections is largely based on the reference directly after the name of each of the methods.

6.2.1. Powell

Powell's method [39] is an optimization algorithm used for finding the minimum of an objective function. The algorithm uses a bi-directional search along search vectors to find this minimum. Powell's method does not use derivatives, and thus the to be optimized function does not need to be differentiable. An initial point x_0 is given to the method, as well as N initial search vectors h_1 through h_N. A single iteration of the algorithm works as follows.

The algorithm first minimizes the function along each search vector (using for example Golden-section search [40]) in order (so the optimum along this first search vector is the starting point for the second search vector). The resulting point x_1 can be expressed as a linear combination of each of the search vectors in the form

$$x_1 = x_0 + \sum_{i=1}^{N} a_i h_i = x_0 + h_{n+1}, \tag{6.2}$$

where a_i is the scalar that optimizes the search in direction h_i. The vector h_{n+1} becomes a new search vector, and the search vector with the largest term in the linear combination (the term for which $|a_i|\|h_i\|$ is largest) is removed from the search vectors. This process is then repeated until the difference between x_n and x_{n+1} is small enough. An example of Powell's method with $N = 2$ can be seen in Figure 6.2.

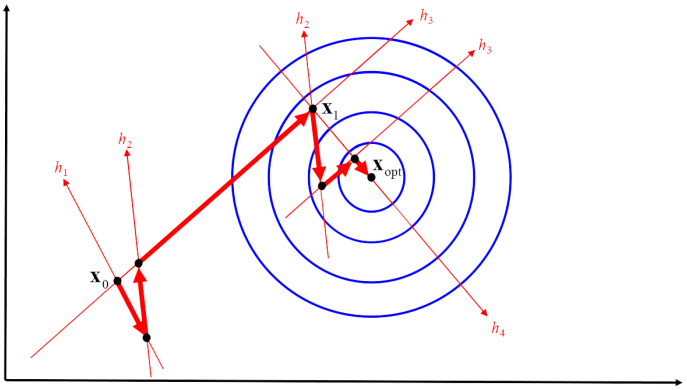


Figure 6.2: Powell's method in two dimensions. Figure adjusted from [41].

It can handle many different optimization problems and is relatively easy to implement, but turned out to be quite slow in practice.

6.2.2. Nelder-Mead

The Nelder-Mead method [42] is an optimization algorithm to find the minimum of an objective function. Similar to Powell's method it does use the derivatives of the to be optimized function, making it a direct search method. The algorithm works by creating a simplex, and then moving the vertices of the simplex to find the minimum of the function. At each step, the algorithm compares the function values at each vertex of the current simplex and uses this information to determine which of the vertices should be moved and in which direction. The algorithm terminates when certain criteria are met, which could be convergence, runtime, a maximum number of iterations, or a maximum number of function evaluations. We look at a single iteration of a variant of the algorithm in a two-dimensional space.

We minimize a function $f(x)$ with $x \in \mathbb{R}^2$. This means our simplex is a triangle in $\mathbb{R}^2$ with vertices $x_1, x_2, x_3 \in \mathbb{R}^2$. Assume without loss of generality that $f(x_1) \leq f(x_2) \leq f(x_3)$. After checking if the algorithm should terminate, we calculate the center of x_1 and x_2 given by $x_0 = \frac{1}{2}(x_1 + x_2)$, and move to step 1.

Step 1: Reflection
We reflect the worst point x_3 through the edge (x_1, x_2) of the simplex to find a reflected point $x_r = x_0 + (x_0 - x_3)$. If x_r has a higher objective value than x_1 but a lower objective value than x_2 (i.e. $f(x_1) \leq f(x_r) \leq f(x_3)$), then this direction is not very good, but at least its better than the worst vertex x_3, so we replace x_3 with x_r to make a new simplex and end this iteration. If not, move to step 2.

Step 2: Expansion
If x_r is the best point so far (i.e. $f(x_r) < f(x_1)$), then we are searching in the right direction. Similar to Powell's method we expand the reflected point to find the expanded point $x_e = x_r + (x_r - x_0)$. If this expanded point is better than the reflected point (i.e. $f(x_e) < f(x_r)$), then we replace x_3 with x_e to make a new simplex and end this iteration. If this expanded point is not better than the reflected point (i.e. $f(x_e) \geq f(x_r)$), then we replace x_3 with x_r to make a new simplex and end this iteration. If x_r is not the best point so far, move to step 3.

Step 3: Contraction
If x_r is worse than x_1 and x_2, then it is not useful, as it would only become the new worst point if we replace x_3 with it. Instead, we contract the worst point x_3 to a new point. We consider two cases:

- If $f(x_r) < f(x_3)$, we compute a new contracted point $x_c = x_0 + \frac{1}{2}(x_r - x_0)$. If this point is better than x_r, we replace x_3 with x_c to make a new simplex and end this iteration. If x_c is not better than x_r, move to step 4.
- If $f(x_r) \geq f(x_3)$, we compute a new contracted point $x_c = x_0 + \frac{1}{2}(x_3 - x_0)$. If this point is better than x_3, we replace x_3 with x_c to make a new simplex and end this iteration. If x_c is not better than x_r, move to step 4.

Step 4: Shrinkage
If we end up at step 4, then none of the computed points x_r, x_e and x_c are currently better than the worst vertex x_3. Therefore, we will shrink the simplex towards the best point x_1 according to the formulas $x_2 = x_1 + \frac{1}{2}(x_2 - x_1)$ and $x_3 = x_1 + \frac{1}{2}(x_3 - x_1)$. This finishes the iteration.

In this variation the standard coefficients for reflecting ($\alpha = 1$), expanding ($\gamma = 2$), contracting ($\rho = \frac{1}{2}$), and shrinking ($\sigma = \frac{1}{2}$) were used, although different coefficients could improve the performance of the algorithm [43]. A visual example of each of the possible new vertices of Nelder-Mead is shown in Figure 6.3.

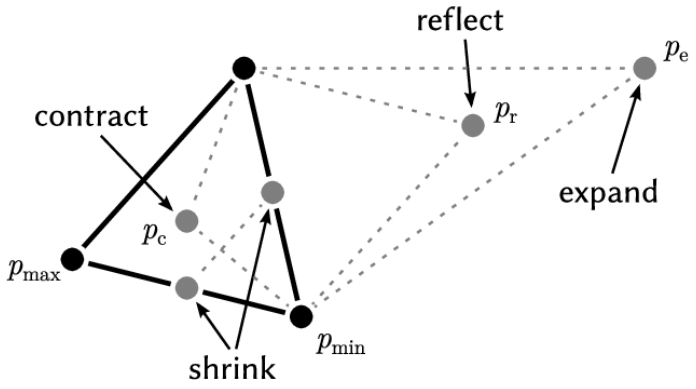


Figure 6.3: An iteration of the Nelder-Mead method in two dimensions where $p_{\max}$ is the worst point, $p_{\min}$ is the best point, p_r is the reflected point, p_e is the expanded point, and p_c is the contracted point. If none of the constructed points are better than $p_{\max}$, then the simplex shrinks towards $p_{\min}$. Figure adjusted from [44].

The Nelder-Mead algorithm is easy to implement, and very resistant to bad initial guesses, unlike Powell's method. Among the algorithms discussed in this thesis, it is one of the faster algorithms, with only COBYLA being faster.

6.2.3. COBYLA

Constrained Optimization BY Linear Approximation (COBYLA) [45] is another optimization algorithm to find the minimum of a function. COBYLA is also a derivative-free method, but unlike Powell and Nelder-Mead, COBYLA can also handle constrained optimization problems. COBYLA uses an initial guess x_0 to make linear approximations to the objective function and the constraints. This linear programming problem is then solved, resulting in another point x_1. This point x_1 is then evaluated in the original objective function to determine the direction in which the objective function is decreasing the most. Using this information a new linear approximation is made, and the process repeats until the difference between candidate solutions is small enough. COBYLA is an easy to implement optimization algorithm and is very efficient in most cases, although it may not perform well when the problem contains complex constraints.

6.2.4. BFGS

The Broyden-Fletcher-Goldfarb-Shanno (BFGS) method [46, 47, 48, 49] is an algorithm used for finding the minimum of an objective function. BFGS is a gradient-based method unlike Powell, Nelder-Mead, and COBYLA, meaning it uses gradient evaluations to look for a search direction at each iteration. It is a quasi-Newton method, meaning it is an alternative to Newton's method [50]. Quasi-Newton methods use an approximation to the Hessian matrix B instead of the exact Hessian matrix. We again start with an initial point x_0 and an initial approximation of the Hessian matrix B_0. We then iteratively find points x_k used to construct new approximations B_k. The sequence $\{x_k\}$ should then converge to the minimum of f. An iteration of Newton's method starts by considering the second-order Taylor expansion of f around x_k given by

$$f(x_k + \Delta x) \approx f(x_k) + \nabla f(x_k)^\top \Delta x + \frac{1}{2}\Delta x^\top B \Delta x, \tag{6.3}$$

where ∇f is the gradient of f and B is the exact Hessian matrix. We want $f(x_k + \Delta x)$ to be a local minimum, and thus we look at the gradient of this approximation with respect to Δx to find

$$\nabla f(x_k + \Delta x) \approx \nabla f(x_k) + B\Delta x. \tag{6.4}$$

We want to be at the point where the gradient is equal to zero, as then we will be at the minimum when constructing our next point. Thus, we want to choose our direction Δx to be the solution of

$$B\Delta x = -\nabla f(x_k). \tag{6.5}$$

Solving for Δx gives a search direction at step k. Similar to Powell's method, a line search is then performed to find a minimum along this line. In practice, an inexact line search is usually performed to find an α_k that satisfies the Wolfe conditions [51, 52]. These conditions make sure both the function f and the gradient ∇f decrease sufficiently. A new point x_{k+1} is then constructed through

$$x_{k+1} = x_k + \Delta x = x_k - \alpha_k B \nabla f(x_k). \tag{6.6}$$

This process is repeated until certain termination conditions are met. Quasi-Newton methods do not use the exact Hessian B, but an approximation of the Hessian B_k that is updated each iteration. Starting with an initial approximation B_0 (usually of the form $B_0 = \beta I$), first a new point x_{k+1} is constructed as above with B_k instead of B, and then B_k is updated to B_{k+1} using an update formula. Different quasi-Newton methods use different update formulas. For BFGS this update formula is of the form

$$B_{k+1} = B_k + \frac{y_k y_k^\top}{y_k^\top \Delta x_k} - \frac{B_k \Delta x_k (B_k \Delta x_k)^\top}{\Delta x_k^\top B_k \Delta x_k}, \tag{6.7}$$

where $y_k = \nabla f(x_{k+1} - \nabla f(x_k)$ is the difference in gradients between x_{k+1} and x_k. Similarly the inverse of B^{-1} can be updated directly using

$$B_{k+1}^{-1} = \left(I - \frac{\Delta x_k y_k^\top}{y_k^\top \Delta x_k}\right) B_k^{-1} \left(I - \frac{y_k \Delta x_k^\top}{y_k^\top \Delta x_k}\right) + \frac{\Delta x_k \Delta x_k^\top}{y_k^\top \Delta x_k}. \tag{6.8}$$

One of the main advantages of quasi-Newton methods like BFGS is that the approximate Hessian matrix does not need to be inverted and can be updated directly. When using Newton's method, the Hessian matrix needs to be inverted. Computing the Hessian itself can also be computationally expensive, especially for larger optimization problems. BFGS is easy to implement and known to be a very fast method in comparison to some other methods like Powell. However, BFGS can be very sensitive to the initial guess, which is why BFGS is often run with multiple initial guesses.

6.2.5. CG

The conjugate gradient method (CG) [53] is an optimization algorithm to find the minimum of an objective function. Like BFGS it is a gradient-based method, meaning it uses the gradient of the objective function to find an optimum. The CG method was originally designed to solve a system of linear equations $Ax = b$, where A is a symmetric positive definite matrix. This is equivalent to minimizing the function

$$f(x) = \frac{1}{2} x^\top A x - b^\top x. \tag{6.9}$$

This method was later extended to non-linear functions in [54]. CG is an iterative method and starts with an initial guess x_0. To start the algorithm we calculate the gradient at x_0 and define $r_0 = -\nabla f(x_0)$. This r_0 will also be our initial search direction p_0. From here we perform a line search along p_0 to find a new point x_1. A single iteration of the algorithm then works as follows.

At the start of an iteration we are at the point x_k. We evaluate the gradient at x_k, which will be $r_k = -\nabla f(x_k)$. At this point the gradient descent method would perform a line search in direction r_k, as this is the direction of steepest descent. Instead, we want to search in a direction conjugate to all previous directions. CG has the special property that a new direction conjugate to all previous directions can be constructed only using the information of the last used direction. We therefore adjust our search direction to obtain the new search direction

$$p_k = r_k + \beta_k p_{k-1}, \tag{6.10}$$

where the factor β_k will guarantee that p_k and p_{k-1} are conjugate with respect to the Hessian. Finally a line search is performed in direction p_k to find an α_k such that $f(x_k + \alpha_k p_k)$ is minimal, thus the next point x_{k+1} is given by

$$x_{k+1} = x_k + \alpha_k p_k. \tag{6.11}$$

This ends the iteration.

Different β_k result in different conjugate gradient methods in the case that f is non-linear, in this thesis the Polak-Ribière formula [55] for β_k is used, given by

$$\beta_k = \frac{r_k^\top (r_k - r_{k-1})}{r_k^\top r_k}. \tag{6.12}$$

An example of the conjugate gradient method compared to gradient descent in two dimensions can be seen in Figure 6.4.

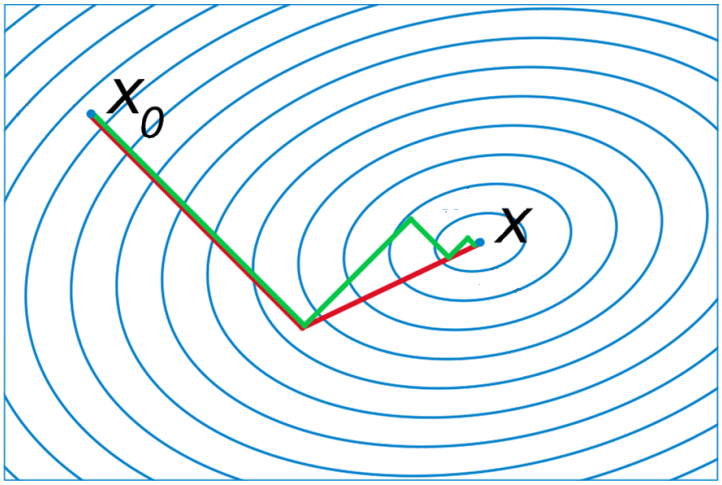


Figure 6.4: A comparison of gradient descent (in green) and conjugate gradient (in red) with initial point x_0 and optimal point x. Note that gradient descent follows a zigzag pattern, which conjugate gradient avoids by choosing a direction conjugate to the previous direction instead of simply following the gradient. Figure adapted from [56].

CG is known as one of the more computationally efficient algorithms as it does not use the Hessian in its iterations, although it may require a good initial guess to find an optimum.

All of these optimization methods were implemented to find the angles $\boldsymbol{\gamma}, \boldsymbol{\beta} \in \mathbb{R}^n$ which have the best expectation of H_P as described in sections 4.1 and 4.2. As we will see next in Section 7.1, the expectation landscapes suggest the optimal solution might not be found for an arbitrary initial guess for most algorithms, therefore all of the algorithms were run with 10^3 initial points (similarly to [57], but with 10^3 initial points instead of 10^4). The gradient of f needed for BFGS and CG is numerically approximated using finite differences, but parameter shift rules could also be used and are further discussed in Section 9.2.1.

7

Results

7.1. Expectation Landscape

To better understand the function we are optimizing over, we will look at some landscapes of $F_p(\boldsymbol{\gamma}, \boldsymbol{\beta})$ as given in (4.13). For all these landscapes and results we will use the problem Hamiltonian H_P as given by (4.18), where G is the path graph P_5 described in Section 6.1. Note that when we apply unitary $U(H_P, \gamma)$ from (4.10) to the state $|s\rangle$ from (4.25) we get

$$U(H_P, \gamma)\,|s\rangle = e^{-i\gamma H_P}\,|1\rangle^{\otimes n} = e^{-i\gamma (H_P)_{nn}}\,|1\rangle^{\otimes n}\,, \tag{7.1}$$

where $(H_P)_{nn}$ is the element of H_P in the nth column and nth row. As $\left\|e^{-i\gamma (H_P)_{nn}}\right\| = 1$, we see that the probability of measuring the state $|1\rangle^{\otimes n}$ is still 1 just as before. Therefore, applying the first problem unitary only affects the global phase of the state, so the expectation value $F_1(\gamma_1, \beta_1)$ is only dependent on β_1. The expectation value $F_1(\beta_1)$ as a function of β_1 in the interval $(0, 2\pi)$ is shown in Figure 7.1 for each of the Hamiltonians $H_{M_{V1}}$, $H_{M_{V2}}$, and $H_{M_{V3}}$.

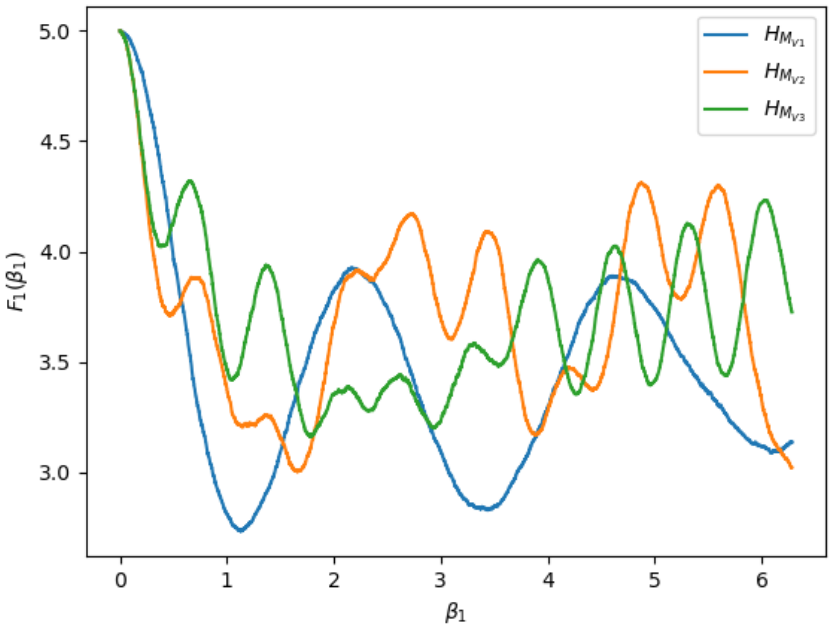


Figure 7.1: The expectation value $F_1(\beta_1)$ as a function of β_1 for each of the mixing Hamiltonians $H_{M_{V1}}$, $H_{M_{V2}}$, and $H_{M_{V3}}$.

We see that $H_{M_{V2}}$ and $H_{M_{V3}}$ have a much more fluctuating behaviour than $H_{M_{V1}}$. In the case that $p = 1$ we also see that $H_{M_{V1}}$ has by far the lowest expectation at the local optimum of β_1, followed by $H_{M_{V2}}$ and only then $H_{M_{V3}}$. For $p = 1$ this suggests that using the first degree mixing Hamiltonian $H_{M_{V1}}$ is much better, however, note that the minimum of $F_1(\beta_1)$ when using $H_{M_{V1}}$ is only slightly lower than 3. When running QAOA using the optimal β, we want to find an expectation much lower than 3 to find the optimal solution with objective value 2 much more often. Running QAOA with $p = 1$ using the β_1 in the first local minimum of $H_{M_{V1}}$ in Figure 7.1 gives the distribution seen in Figure 7.2.

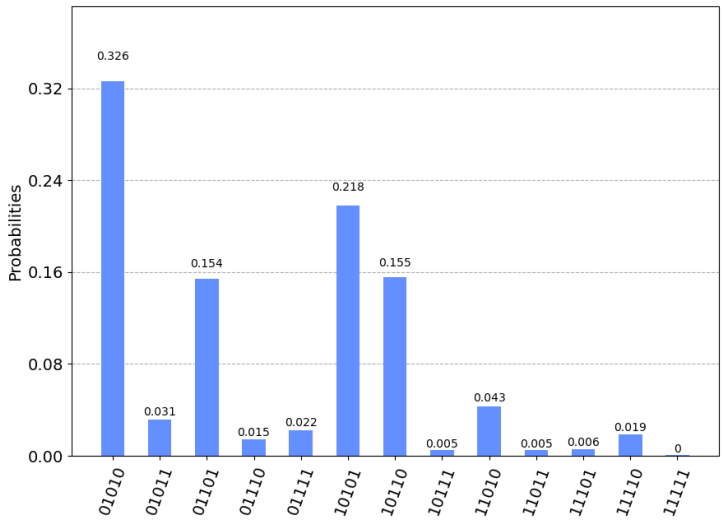
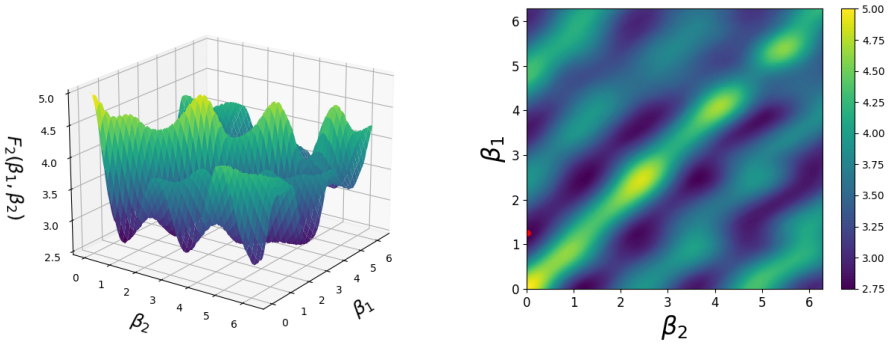


Figure 7.2: Sample distribution of the solutions when running QAOA using the β_1 of the local minimum for $H_{M_{V_1}}$.

We see that the optimal solution $|01010\rangle$ with objective value 2 is only measured about one third of the time, with other solutions like $|01101\rangle$, $|10101\rangle$, $|10110\rangle$ all still having a probability of measurement of over 0.1. Therefore, we will need to look at higher values of p to find lower expectation values $F_p(\boldsymbol{\gamma}, \boldsymbol{\beta})$.

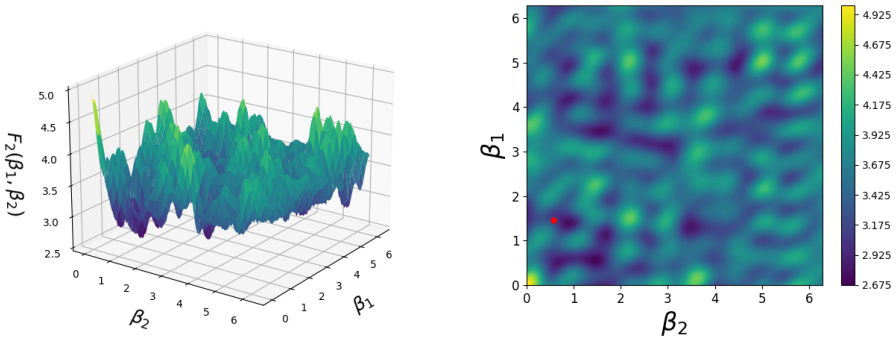
Next, we will try to visualize the landscape of $F_p(\boldsymbol{\gamma}, \boldsymbol{\beta})$ in the case that $p = 2$. Note that as seen in the start of this section F_p still does not depend on γ_1. We try to visualize the landscape by choosing a predetermined value for γ_2, and then plotting F_p only as a function of β_1 and β_2 to better compare between Hamiltonian versions. As the unitary $U(H_P, \gamma)$ has a periodicity of π due to its implementation using R_z-gates as described in Section 6, we will choose a random $\gamma_2 \in (0, \pi)$ to visualize the landscape of F_2, which can be seen in Figure 7.3.



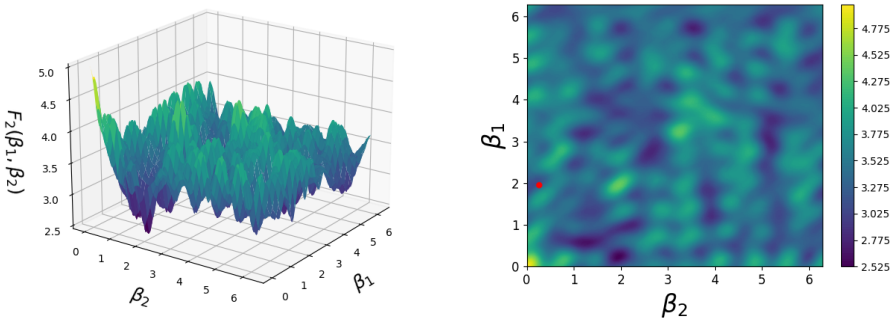
(a) 3D Plot of F_2 for $H_{M_{V1}}$

(b) Colorplot of F_2 for $H_{M_{V1}}$

Figure 7.3: Landscapes of $F_2 = (\beta_2, \beta_1)$ for each of the Hamiltonians $H_{M_{V1}}$, $H_{M_{V2}}$, and $H_{M_{V3}}$ with γ_2 set to $\frac{37}{57}\pi$. β_1 and β_2 range from 0 to 2π in all plots and are divided into 200 grid points. F_2 is determined using 512 samples from the resulting state, with the minimum indicated by a red dot in the colorplots.



(c) 3D Plot of F_2 for $H_{M_{V2}}$ **(d)** Colorplot of F_2 for $H_{M_{V2}}$



(e) 3D Plot of F_2 for $H_{M_{V3}}$ **(f)** Colorplot of F_2 for $H_{M_{V3}}$

Figure 7.3: Landscapes of $F_2 = (\beta_2, \beta_1)$ for each of the Hamiltonians $H_{M_{V1}}$, $H_{M_{V2}}$, and $H_{M_{V3}}$ with γ_2 set to $\frac{37}{57}\pi$. β_1 and β_2 range from 0 to 2π in all plots and are divided into 200 grid points. F_2 is determined using 512 samples from the resulting state, with the minimum indicated by a red dot in the colorplots.

As can be observed in the 3D surface plots of Figures 7.3a, 7.3c, and 7.3e, we again see that the landscapes of $H_{M_{V2}}$ and $H_{M_{V3}}$ fluctuate much more than the landscape of $H_{M_{V1}}$. The expectation values of $H_{M_{V2}}$ and $H_{M_{V3}}$ also seem to have a much smaller range than those of $H_{M_{V1}}$. Most of the expectation values of $H_{M_{V2}}$ and $H_{M_{V3}}$ are within the range $(3.0, 4.0)$, while most the expectation values of $H_{M_{V1}}$ are within the much broader range of $(2.75, 4.75)$. However, in the heat maps of Figures 7.3b, 7.3d, and 7.3f we see low local minima are still obtained for $H_{M_{V2}}$ and $H_{M_{V3}}$. In fact, later we will see that the minima of 7.3d and 7.3f are lower than the minimum found in 7.3b. Next, in Section 7.2.1 we will look at the exact values of these minima and how they compare between Hamiltonians for different p.

One might wonder why figures 7.1 and 7.3 only use values of β_1 and β_2 between 0 and 2π. As the decomposition of the mixing Hamiltonians for MinVertexCover has not been explored, we do not know if there is any periodicity of the unitary $U(H_M, \beta)$ in the parameter β. As mentioned before, we can find a periodicity of π in the γ parameter for the unitary $U(H_P, \gamma)$, as the unitary is implemented using R_z-gates. As $U(H_M, \beta)$ is directly implemented through its matrix form, we can not be sure that the optimal β is indeed in the interval $(0, 2\pi)$. Nevertheless, there is no uniform distribution over $\mathbb{R}^p$ with integral 1, so for the results of this thesis we will restrict the region of initial points to $(0, 2\pi)^p$, although it is possible that other minima outside this region that result in a better expectation exist.

7.2. Performance Comparison of Hamiltonians

7.2.1. Comparison for set p

As mentioned in the previous sections, we evaluate each of the mixing Hamiltonians $H_{M_{V1}}$, $H_{M_{V2}}$ and $H_{M_{V3}}$ by the expectation $F_p(\boldsymbol{\gamma}, \boldsymbol{\beta})$ given in (4.13). We now show the resulting expectations when running QAOA using each of the Hamiltonians $H_{M_{V1}}$, $H_{M_{V2}}$ and $H_{M_{V3}}$ and each of the classical optimization methods described in Section 6.2. We show results for $p = 1$ through $p = 4$. Each time the algorithm is performed 10 times on the path graph P_5 with 10^3 initial points.

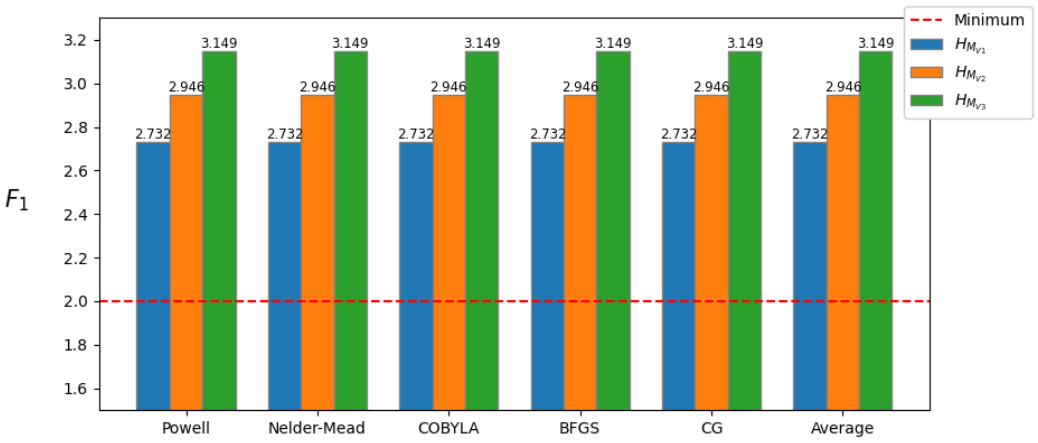


Figure 7.4: Expectation F_1 reached by each of the Hamiltonians $H_{M_{V1}}$, $H_{M_{V2}}$, and $H_{M_{V3}}$ for different classical optimization methods. Expectations given are averages of 10 runs on P_5 with 10^3 initial points each.

For $p = 1$ we see in Figure 7.4 that each of the optimization methods was able to find the minimum of $F_1(\beta_1)$ that was observed earlier in Figure 7.1. The only p for which all the optimization methods coincide is $p = 1$, as we will see when discussing $p = 2$ and beyond. $H_{M_{V1}}$ performs best with an expectation of 2.732 for all optimization methods, followed by $H_{M_{V2}}$ with an expectation of 2.946 and $H_{M_{V3}}$ with an expectation of 3.149. As discussed earlier, even though $H_{M_{V1}}$ outperforms both $H_{M_{V2}}$ and $H_{M_{V3}}$, the expectation of 2.732 of $H_{M_{V1}}$ is still not low enough for practical use.

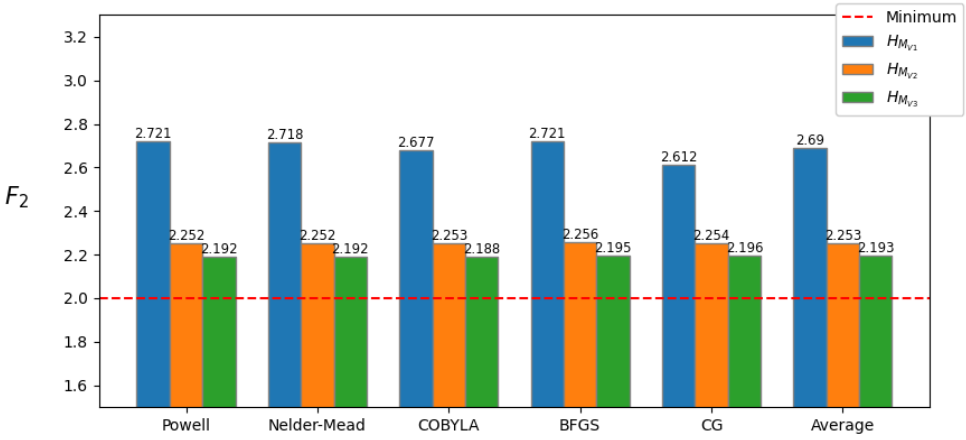


Figure 7.5: Expectation F_2 reached by each of the Hamiltonians $H_{M_{V1}}$, $H_{M_{V2}}$, and $H_{M_{V3}}$ for different classical optimization methods. Expectations given are averages of 10 runs on P_5 with 10^3 initial points each.

We see in Figure 7.5 that $p = 2$ is the lowest value of p where each optimization method finds a different expectation value for all three Hamiltonians. $H_{M_{V1}}$ performs by far the worst of the three Hamiltonians, not improving much compared to $p = 1$ with an average expectation across all optimization methods of 2.69. $H_{M_{V2}}$ and $H_{M_{V3}}$ improve much more with average expectation values of 2.253 and 2.193 for $H_{M_{V2}}$ and $H_{M_{V3}}$ respectively. $H_{M_{V3}}$ performs best, achieving the lowest expectation value yet of 2.188 when ran using the COBYLA method. Interesting to note is that the optimization methods are slightly more consistent in finding a similar minimum value for $H_{M_{V2}}$ and $H_{M_{V3}}$ than for they are for $H_{M_{V1}}$.

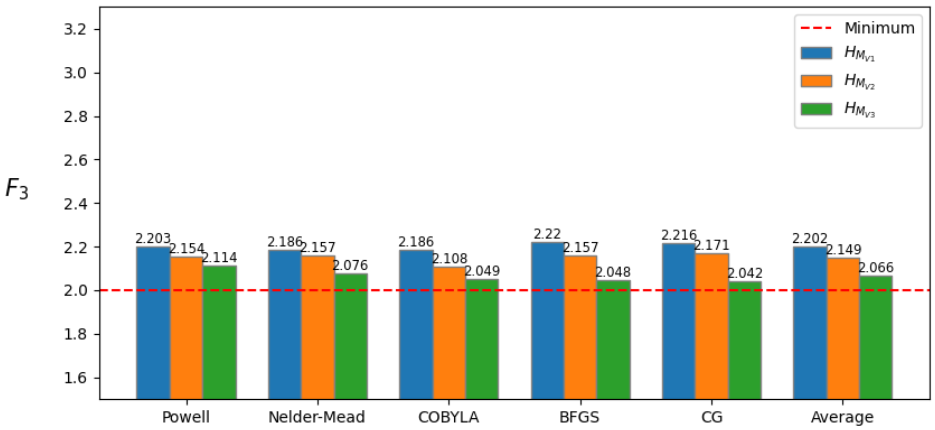


Figure 7.6: Expectation F_3 reached by each of the Hamiltonians $H_{M_{V1}}$, $H_{M_{V2}}$, and $H_{M_{V3}}$ for different classical optimization methods. Expectations given are averages of 10 runs on P_5 with 10^3 initial points each.

For $p = 3$ we see in Figure 7.6 that the expectation values of $H_{M_{V1}}$ have improved with expectation values ranging between 2.186 and 2.220. Nevertheless, $H_{M_{V2}}$ and $H_{M_{V3}}$ still both outclass $H_{M_{V1}}$, both still reaching a lower expectation value regardless of what optimization method was used. $H_{M_{V3}}$ again slightly outperforms $H_{M_{V2}}$ on average, with the gap between the two largest when using BFGS. COBYLA again performs best on average for the optimization methods, although the lowest expectation is found by CG when using $H_{M_{V3}}$.

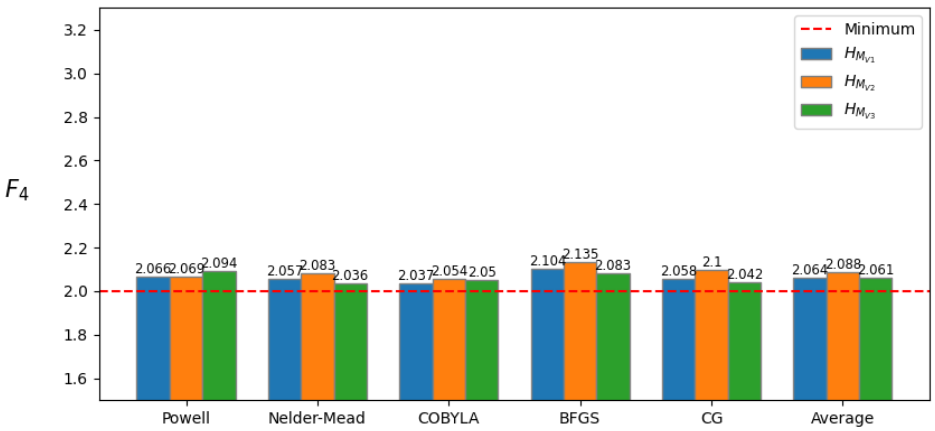


Figure 7.7: Expectation F_4 reached by each of the Hamiltonians $H_{M_{V1}}$, $H_{M_{V2}}$, and $H_{M_{V3}}$ for different classical optimization methods. Expectations given are averages of 10 runs on P_5 with 10^3 initial points each.

At $p = 4$ we finally see $H_{M_{V1}}$ catch up with $H_{M_{V2}}$ and $H_{M_{V3}}$. All three Hamiltonians reach expectation values below 2.1 on average, with the best expectation value being reached by $H_{M_{V3}}$ when using the Nelder-Mead method. $H_{M_{V2}}$ now performs the worst for almost all of the optimization methods, with the exception of Powell's method.

7.2.2. Comparison for set expectation

In the previous section the expectations of mixing Hamiltonians $H_{M_{V1}}$, $H_{M_{V2}}$, and $H_{M_{V3}}$ were compared for set p, where $p \in \{1, 2, 3, 4\}$. In this section we will instead compare the depth p needed to reach a set expectation F for each of the Hamiltonians. We do not want the expectation F_p to be too high, otherwise we will not measure the optimal solution often enough. When introducing noise into the algorithm, this chance of finding the optimum will be reduced, so we want our expectation F_p to be low. We choose the set expectation to be reached to be $F = 2.2$. In Figure 7.8 we plot the depth p needed to reach an expectation better than $F = 2.2$ against each of the Hamiltonian versions for all of the different optimization methods.

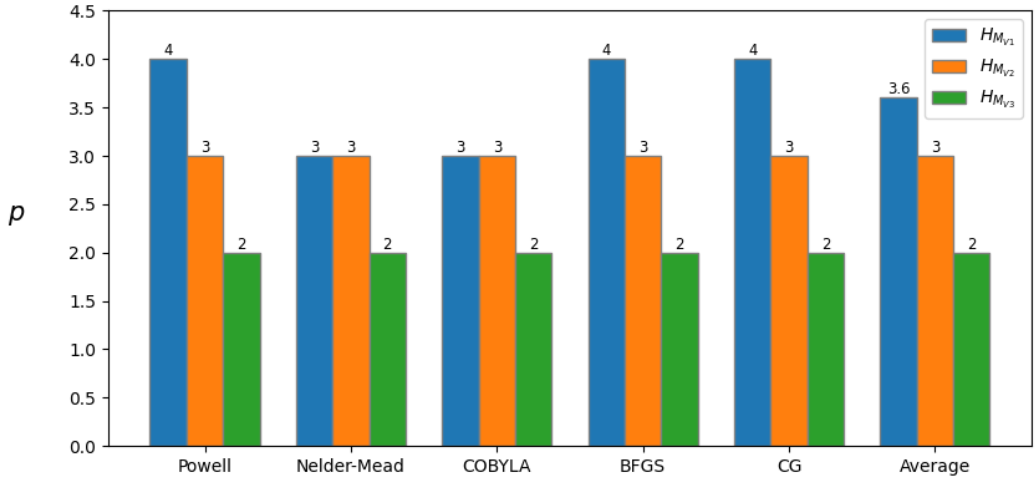


Figure 7.8: Depth p needed to reach the set expectation $F = 2.2$ for each of the Hamiltonians $H_{M_{V1}}$, $H_{M_{V2}}$, and $H_{M_{V3}}$ for different classical optimization methods. Depths p given are averages of 10 runs on P_5 with 10^3 initial points each.

Here we see that the original mixing Hamiltonian $H_{M_{V1}}$ most often needs to use a depth of $p = 4$ (when using Powell, BFGS, or CG) and sometimes a depth of $p = 3$ (when using Nelder-Mead or COBYLA) to reach an expectation of less than 2.2. $H_{M_{V2}}$ improves on this by only needing a depth of $p = 3$ for all optimization methods used. $H_{M_{V3}}$ again achieves better results than both, only needing a depth of $p = 2$ to reach an expectation less than 2.2 for all optimization methods.

Although $H_{M_{V3}}$ needs a lower depth p to reach good expectation values, one could wonder if using $H_{M_{V3}}$ instead of $H_{M_{V1}}$ is accompanied by a longer runtime as these Hamiltonians are of higher degree. Using higher degree Hamiltonians with a lower depth p is not necessarily beneficial if the runtime of the algorithm becomes much longer in doing so. We thus compare the runtime of the algorithm for each of the Hamiltonians when the depth p from Figure 7.8 is used to reach an expectation of less than 2.2, the results of which are shown in Figure 7.9.

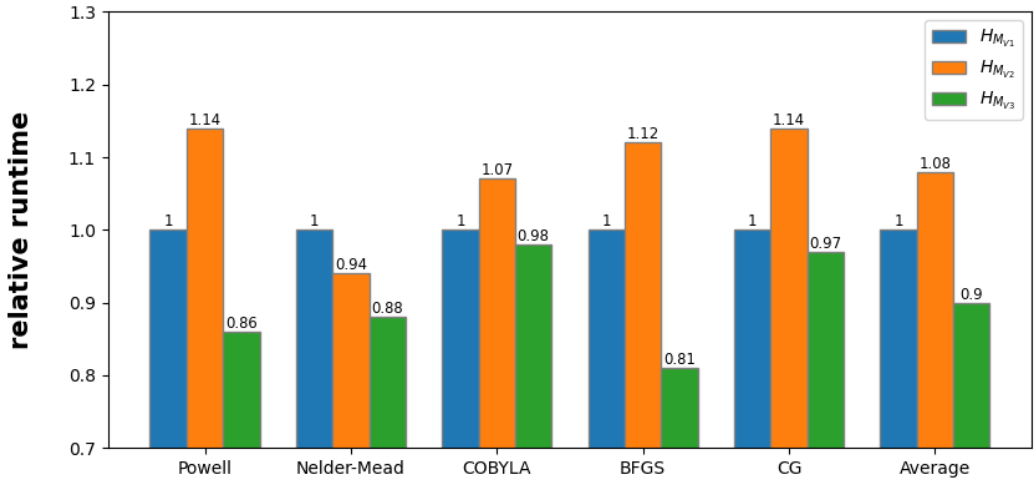


Figure 7.9: Relative runtime needed to reach the set expectation $F = 2.2$ for each of the Hamiltonians $H_{M_{V1}}$, $H_{M_{V2}}$, and $H_{M_{V3}}$ for different classical optimization methods. Relative runtimes given are averages of 10 runs on P_5 with 10^3 initial points each. The baseline is set at 1 for each of the optimization methods, and optimization methods themselves are not relative to each other.

We see that the improvement for $H_{M_{V2}}$ to only need depth $p = 3$ is not enough to keep up with $H_{M_{V1}}$ in terms of runtime. $H_{M_{V2}}$ was only faster when using the Nelder-Mead method, and in comparison to $H_{M_{V1}}$ was 8% slower on average. $H_{M_{V3}}$ outperforms both $H_{M_{V1}}$ and $H_{M_{V2}}$ for all optimization methods used, decreasing runtime by 10% on average compared to $H_{M_{V1}}$.

8

Conclusions and Discussion

In this thesis we explored the use of the Quantum Approximate Optimization Algorithm (QAOA) and the Quantum Alternating Operator Ansatz (QAOA+) for solving the minimum vertex cover problem (MinVertexCover), which is a classic combinatorial optimization problem with applications in many fields, such as biotechnology, network design and transportation. We discussed how MinVertexCover is solved and approximated on classical computers, as well as the background of both the QAOA and QAOA+ algorithms. Next, the application of QAOA+ to MinVertexCover was examined, which uses a first degree mixing Hamiltonian $H_{M_{V1}}$. This mixing Hamiltonian mixes between solution states of MinVertexCover which are exactly Hamming distance 1 apart.

This thesis introduced two new second degree mixing Hamiltonians $H_{M_{V2}}$ and $H_{M_{V3}}$. These Hamiltonians are based on the premise to be able to mix between solutions of MinVertexCover which are Hamming distance 1 or 2 apart. $H_{M_{V2}}$ and $H_{M_{V3}}$ were constructed by looking at all possibilities to switch 2 vertices at the same time and then adding these terms to the original mixing Hamiltonian. The new mixing Hamiltonians were subsequently implemented and benchmarked against the original first degree mixing Hamiltonian for MinVertexCover using multiple different classical optimization methods, testing for both depth p needed to reach a set expectation as well as runtime.

While the second degree mixing Hamiltonians $H_{M_{V2}}$ and $H_{M_{V3}}$ do not perform well at the low depth $p = 1$, for $p = 2$ and $p = 3$ they achieve a better expectation F_p than $H_{M_{V1}}$ for all optimization methods used. $H_{M_{V3}}$ performs by far the best of all mixing Hamiltonians. It achieves a significantly better average expectation for $p = 2$, keeps a slight advantage at $p = 3$, and still attains equal results to $H_{M_{V1}}$ at $p = 4$. $H_{M_{V2}}$ and $H_{M_{V3}}$ also reach the desired expectation $F = 2.2$ at an earlier p on average than the first degree mixing Hamiltonian $H_{M_{V1}}$. $H_{M_{V3}}$ only needs a depth of $p = 2$ to reach this expectation F in comparison to the depth $p = 4$ (or sometimes $p = 3$) for $H_{M_{V1}}$. Even though $H_{M_{V2}}$ and $H_{M_{V3}}$ are more computationally expensive to compute, the benefit of having a smaller depth p compensates for this fact in terms of runtime. Although on average $H_{M_{V2}}$ takes longer to reach $F = 2.2$ than $H_{M_{V1}}$, $H_{M_{V3}}$ outperforms $H_{M_{V1}}$ regarding runtime for all optimization methods used with a relative runtime improvement of 10% on average. $H_{M_{V3}}$ even reaches runtime improvements of as much as 20% when using the very commonly utilized BFGS method.

When running the algorithm, the runtime of the optimization methods imposed significant limitations on the amount of total runs. Even COBYLA, the fastest optimization method across the board, had a runtime of at least 15 minutes when run with 10^3 initial points on a graph with only 5 vertices. Presumably, this is due to hardware limitations of the computer used. It would be interesting for future work to explore how the Hamiltonians $H_{M_{V2}}$ and $H_{M_{V3}}$ perform on larger graphs. As these Hamiltonians are currently implemented using their matrix, one could first look into their decomposition into elementary gates to calculate the gradient more easily to speed up the optimization process. Using this improvement, one could test each of the Hamiltonians on substantially larger graphs using gradient-based methods to see if $H_{M_{V3}}$ still reaches better expectation values sooner.

Another reason to examine the decomposition of $H_{M_{V2}}$ and $H_{M_{V3}}$ would be to find periodicity in these Hamiltonians. Using this periodicity, the search space for optimal parameters would be considerably easier to explore and the optima found could possibly be determined to be global optima. Currently, the initial points are chosen within the region of $(0, 2\pi)^p$, but different choices for the region of initial points are possible. As discussed before, we can not take initial points from the full parameter space, as there exists no uniform distribution over this subspace, however, different initial regions could result in different Hamiltonian behaviour and thus different results.

Should the results for $H_{M_{V2}}$ and $H_{M_{V3}}$ recur in larger graphs, then the use of second and possibly higher degree mixing Hamiltonians will become a tradeoff of the depth of the quantum circuit and the size of the classical optimization space. Higher degree Hamiltonians will be more susceptible to noise as implementing them requires more gates. On the contrary, as they need a smaller p to reach a certain expectation they do not need to search as large of a parameter space as compared to the first degree Hamiltonian. Provided that quantum computers become robust enough in the future, these new Hamiltonians could prove to be useful in significantly speeding up the classical optimization element of QAOA and QAOA+. Keeping the depth p low even for increasingly large optimization problems could prove essential in the use of QAOA as a route to Quantum Supremacy.

9

Extensions and Alternatives

In this chapter we will look at extensions of the Hamiltonians constructed in sections 5.1 and 5.2, along with the same principles applied to mixing Hamiltonians of other optimization problems. We will also look at the idea of penalty methods and how they can be used to manipulate Hamiltonians.

9.1. Extensions

First, we will look at two ways to extend the idea of second degree mixing Hamiltonians. We will start by trying to construct third and higher degree mixing Hamiltonians for MinVertexCover, after which we look at second degree mixing Hamiltonians for other optimization problems.

9.1.1. Third and Higher Degree Mixing Hamiltonians for MinVertexCover

In Section 5.1 and Section 5.2 we saw the introduction of new mixing Hamiltonians $H_{M_{V2}}$ and $H_{M_{V3}}$. These Hamiltonians are based on the idea of also mixing vertex covers with Hamming distance 2 between them. We can continue this pattern to construct Hamiltonians which will mix vertex covers with larger Hamming distance between them. This section will cover the complete construction of the third degree Hamiltonian, along with a generalisation for higher degree Hamiltonians.

To design a Hamiltonian that mixes vertex covers with Hamming distance 3 between them, we need to perform X-gates to three different vertices while making sure the resulting solution is still a feasible cover. We look at all possible subgraphs of size 3, which can be seen in Figure 9.1.

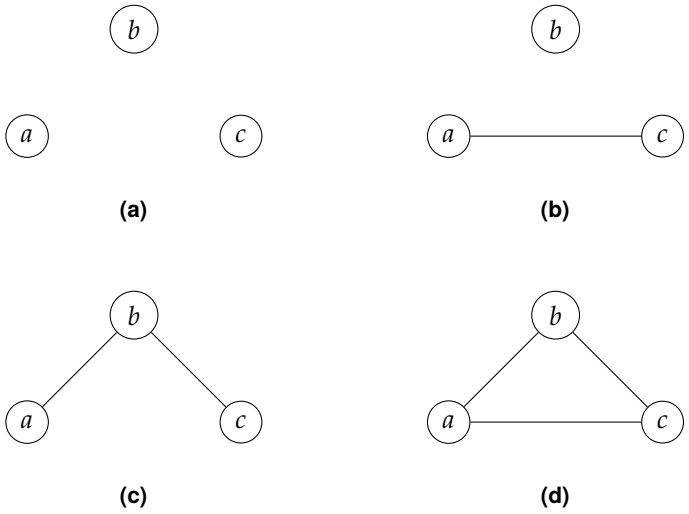


Figure 9.1: All possible subgraphs of size 3 with their case number.

We consider each of the cases (a) through (d) and construct a Hamiltonian term for each, defining the neighbourhood $N(V')$ of a set of vertices V' as the set of vertices in $G \setminus V'$ which are adjacent to at least one vertex in V'.

Case (a): None of the vertices are connected
In the case that none of the vertices are connected, we can repeat the process of Section 5.1 to find the Hamiltonian term to be

$$X_a X_b X_c \prod_{v \in N(\{a,b,c\})} W_v, \tag{9.1}$$

where again $N(\{a,b,c\})$ is the set of vertices adjacent to a, b or c which are not a, b, or c themselves. Finishing this case we sum over all possible combinations of triples of vertices which are non-adjacent to find

$$\sum_{\substack{a,b,c \in V \text{ s.t. } \{a,b\} \notin E, \\ \{a,c\} \notin E,\ \{b,c\} \notin E}} \left(X_a X_b X_c \prod_{v \in N(\{a,b,c\})} W_v \right) \tag{9.2}$$

as the term of the mixing Hamiltonian swapping three non-adjacent vertices at the same time.

Case (b): Two of the vertices are connected with a single edge
In the case that just two of the vertices are connected, we can use a combination of sections 5.1 and 5.2. Assume without loss of generality that $\{a,b\} \notin E$, $\{a,c\} \in E$, $\{b,c\} \notin E$. Here we are in the situation seen in Figure 9.2.

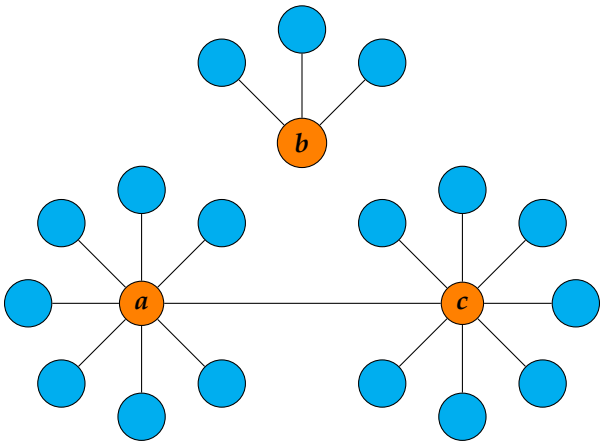


Figure 9.2: Example graph where a and c are connected.

This is again similar to the terms used in $H_{M_{V3}}$ in Section 5.2, as we see that a and c can not both be set to 0 or 1 at the same time. This would result in either not a vertex cover if both were set to 0 (as $\{a,c\}$ is then not covered) or if both were set to 1 we would mix out of the feasible subspace. All vertices adjacent to a, b and c that are not a, b or c themselves must already be included in the cover, as was previously stated. This can be confirmed by including the W_v terms. Thus, in the case that $\{a,b\} \notin E$, $\{a,c\} \in E$, $\{b,c\} \notin E$ this leads to the term

$$\begin{aligned} &X_a X_b X_c W_a \overline{W_c} \prod_{v \in N(\{a,b,c\})} W_v \\ &+ X_a X_b X_c \overline{W_a} W_c \prod_{v \in N(\{a,b,c\})} W_v. \end{aligned} \tag{9.3}$$

We sum over all possible subgraphs with a single edge between two vertices to find the Hamiltonian term if two of the vertices are connected to be

$$\begin{aligned}
&\sum_{\substack{a,b,c \in V \text{ s.t. } \{a,b\}\in E,\\ \{a,c\}\notin E,\ \{b,c\}\notin E}} \Bigg(X_a X_b X_c W_a \overline{W_b} \prod_{v\in N(\{a,b,c\})} W_v \\
&\qquad + X_a X_b X_c \overline{W_a} W_b \prod_{v\in N(\{a,b,c\})} W_v \Bigg) \\
&+ \sum_{\substack{a,b,c \in V \text{ s.t. } \{a,b\}\notin E,\\ \{a,c\}\in E,\ \{b,c\}\notin E}} \Bigg(X_a X_b X_c W_a \overline{W_c} \prod_{v\in N(\{a,b,c\})} W_v \\
&\qquad + X_a X_b X_c \overline{W_a} W_c \prod_{v\in N(\{a,b,c\})} W_v \Bigg) \\
&+ \sum_{\substack{a,b,c \in V \text{ s.t. } \{a,b\}\notin E,\\ \{a,c\}\notin E,\ \{b,c\}\in E}} \Bigg(X_a X_b X_c W_b \overline{W_c} \prod_{v\in N(\{a,b,c\})} W_v \\
&\qquad + X_a X_b X_c \overline{W_b} W_c \prod_{v\in N(\{a,b,c\})} W_v \Bigg).
\end{aligned} \tag{9.4}$$

Case (c): All of the vertices are connected through two edges
In the case that all of the vertices are connected through two edges, we need to construct an entirely new term. Assume without loss of generality that $\{a,b\} \in E$, $\{a,c\} \notin E$, $\{b,c\} \in E$. Here we are in the situation seen in Figure 9.3.

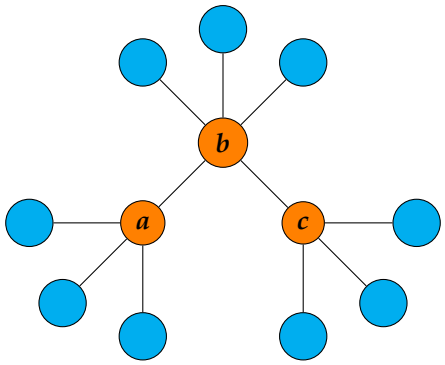


Figure 9.3: Example graph where $\{a,b\} \in E$ and $\{b,c\} \in E$.

For our Hamiltonian we need to make sure both the original state as well as the state that will be mixed to are vertex covers. If we swap a, b and c all at the same time this is only possible if $|abc\rangle = |101\rangle$ or $|abc\rangle = |010\rangle$. We again check for these situations using W_v operators, in the case of the example this leads to the term

$$\begin{aligned}
&X_a X_b X_c W_a \overline{W_b} W_c \prod_{v\in N(\{a,b,c\})} W_v \\
&+ X_a X_b X_c \overline{W_a} W_b \overline{W_c} \prod_{v\in N(\{a,b,c\})} W_v.
\end{aligned} \tag{9.5}$$

We again sum over all possible subgraphs with all vertices connected through two edges to find the term for the third degree Hamiltonian to be

$$\begin{aligned}
&\sum_{\substack{a,b,c \in V \text{ s.t. } \{a,b\}\in E,\\ \{a,c\}\notin E,\ \{b,c\}\in E}} \Bigg(X_a X_b X_c W_a \overline{W_b} W_c \prod_{v\in N(\{a,b,c\})} W_v \\
&\qquad + X_a X_b X_c \overline{W_a} W_b \overline{W_c} \prod_{v\in N(\{a,b,c\})} W_v \Bigg) \\
&+ \sum_{\substack{a,b,c \in V \text{ s.t. } \{a,b\}\in E,\\ \{a,c\}\in E,\ \{b,c\}\notin E}} \Bigg(X_a X_b X_c W_a \overline{W_b W_c} \prod_{v\in N(\{a,b,c\})} W_v \\
&\qquad + X_a X_b X_c \overline{W_a} W_b W_c \prod_{v\in N(\{a,b,c\})} W_v \Bigg) \\
&+ \sum_{\substack{a,b,c \in V \text{ s.t. } \{a,b\}\notin E,\\ \{a,c\}\in E,\ \{b,c\}\in E}} \Bigg(X_a X_b X_c W_a W_b \overline{W_c} \prod_{v\in N(\{a,b,c\})} W_v \\
&\qquad + X_a X_b X_c \overline{W_a W_b} W_c \prod_{v\in N(\{a,b,c\})} W_v \Bigg).
\end{aligned} \tag{9.6}$$

Case (d): All of the vertices are connected through three edges
In the case that all vertices are connected, we are in the situation seen in Figure 9.4.

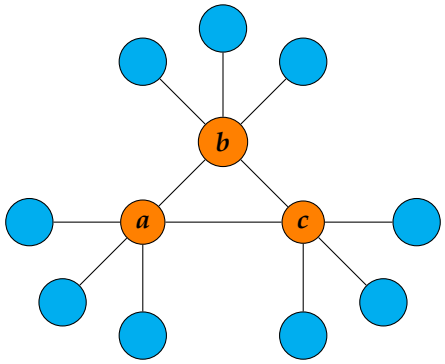


Figure 9.4: Graph where $\{a,b\} \in E$, $\{a,c\} \in E$, and $\{b,c\} \in E$.

This is a special case, as no non-zero Hamiltonian term can be constructed for this case. Even if all the neighbours of a, b and c (shown in the figure in blue) are in the vertex cover, there is still no possible vertex cover which is still feasible after swapping all of a, b and c (unlike the cover $|abc\rangle = |101\rangle$ from case (c)). This concludes this case.

Combining all cases gives the following third degree term Hamiltonian given as a sum of equation numbers:

$$H_M = (9.2) + (9.4) + (9.6). \tag{9.7}$$

Adding the terms from the first and second degree Hamiltonian, this would lead to the full third degree Hamiltonian to be

$$H_M = (4.24) + (5.2) + (5.6) + (9.2) + (9.4) + (9.6), \tag{9.8}$$

again written as a sum of equation numbers.

Through analogous ways it is possible to construct even higher degree Hamiltonians for MinVertexCover as well. In fact, it is possible to create a Hamiltonian term for each possible subgraph G' as long as the subgraph has a vertex cover C such that $H \setminus C$ is also a vertex cover. This is exactly the case when G' is bipartite as is proven in the following theorem.

Theorem 9.1. *The Hamiltonian term $H_{G'}$ corresponding to a subgraph $G' = (V', E')$ of $G = (V, E)$ is non-zero if and only if G' is bipartite.*

Proof. $\Rightarrow$:
If the Hamiltonian term $H_{G'}$ of degree k corresponding to subgraph G' is non-zero then it has at least one non-zero element $(H_{G'})_{ij}$ which mixes bitstring i to bitstring j. Remember that any Hamiltonian must preserve the feasible subspace, so bitstrings i and j correspond to vertex covers C_i and C_j. $H_{G'}$ is the Hamiltonian term of degree k corresponding to G', so we know $|V'| = k$ and the term swaps all k vertices of G' at the same time. Thus, bitstring i and j are exactly the same for vertices in $G \setminus G'$ but the opposite for vertices in G'. Let $A \in G'$ be all vertices in G' set to 1 in solution i and $B \in G'$ be all vertices in G' set to 1 in solution j. Note that A and B are disjoint and $A \cup B = V'$. Furthermore, as C_i and C_j are both vertex covers of G we must have that A and B are both vertex covers of G'. If A is a vertex cover of G' then B must be independent, because if two vertices of B were adjacent in G' then this edge would not be covered by A as A and B are disjoint. Similarly B is a vertex cover of G', thus A is independent as well. Thus, we have found two disjoint, independent sets A and B such that $A \cup B = V'$. Therefore, G' is bipartite.

$\Leftarrow$:
If G' is bipartite then we can divide its vertices in two disjoint, independent subsets A and B. As we saw in section 2.2, both A and B are vertex covers of G'. This means we can construct a Hamiltonian term $H_{G'}$ that swaps all vertices of G'. In other words, if we apply $H_{G'}$ to a cover C_1 with $A \in C_1$, then H will mix this cover C_1 to the cover with C_2 with $B \in C_2$ and $C_2 \setminus B = C_1 \setminus A$. To construct this term we again note all vertices adjacent to vertices in G' need to be set to 1. Furthermore, we need to have exactly all vertices in A or all vertices in B set to 1, thus we get the following Hamiltonian $H_{G'}$ term corresponding to G':

$$\begin{aligned} H_{G'} = &\prod_{u \in V'} X_u \prod_{a \in A} W_a \prod_{b \in B} \overline{W_b} \prod_{v \in N(V')} W_v \\ &+ \prod_{u \in V'} X_u \prod_{a \in A} \overline{W_a} \prod_{b \in B} W_b \prod_{v \in N(V')} W_v, \end{aligned} \tag{9.9}$$

where $N(V')$ is the set of vertices in $G \setminus G'$ adjacent to at least one vertex in G'. □

Note that although a disconnected graph can have more than one bipartition [58], every bipartition of the same subgraph leads to the same Hamiltonian term. This procedure for constructing mixing Hamiltonians can be used to find Hamiltonian terms of any degree $k \leq K$ where K is the size of the largest bipartite subgraph of G. We can find the nth degree mixing Hamiltonian by summing over all possible bipartite subgraphs of size $\leq n$, resulting in

$$\begin{aligned} H_M = \sum_{i=1}^{n} \sum_{\substack{G' \in G_i \text{ s.t.} \\ G' \text{is bipartite}}} \Bigg(&\prod_{u \in V'} X_u \prod_{a \in A} W_a \prod_{b \in B} \overline{W_b} \prod_{v \in N(V')} W_v \\ &+ \prod_{u \in V'} X_u \prod_{a \in A} \overline{W_a} \prod_{b \in B} W_b \prod_{v \in N(V')} W_v \Bigg), \end{aligned} \tag{9.10}$$

where G_i is the set of subgraphs of G of size i and (A, B) is a bipartition of subgraph G'.

9.1.2. Second Degree Mixing Hamiltonians for Other Optimization Problems

The mixing Hamiltonians $H_{M_{V2}}$ and $H_{M_{V3}}$ as introduced in sections 5.1 and 5.2 were designed specifically for MinVertexCover, but second degree Hamiltonians can also be constructed for other optimization problems. The simplest one is a second degree version of the transverse field Hamiltonian for unconstrained optimization problems as seen in (4.5), where the second degree Hamiltonian is given by

$$H_M = \sum_{v \in V} X_v + \sum_{v \in V, w \in V} X_v X_w. \tag{9.11}$$

Second degree Hamiltonians are always of the form $H_M = \sum_{v \in V} X_v \cdot a + \sum_{v \in V, w \in V} X_v X_w \cdot b$, where a and b are dependent on the problem structure, as these factors make sure the feasible subspace is not left while mixing. In the case where there are no restrictions to the domain (such as with the maximum cut problem), these terms are not needed and can be taken as I. As seen earlier, for MinVertexCover these terms are usually of the form $\prod_{v \in V} W_v$ to check that all neighbours of a flipped vertex are also in the cover so that no edge is left uncovered. To give an example on how a second degree mixing Hamiltonian can be constructed we look at the independent set problem.

Example

An independent set of a graph $G = (V, E)$ is a subset of vertices $S \subseteq V$ such that no pair of vertices in C is adjacent. The maximum independent set problem (MaxIndependentSet) is the problem of finding an independent set of maximum size. To construct the first degree mixing Hamiltonian for the independent set problem we have to look at the mixing rules. A vertex can only be swapped in or out of the independent set S if none of its neighbours are currently in S. Thus, we iterate over all vertices and its neighbours to find the first degree mixing Hamiltonian to be

$$H_M = \sum_{u \in V} \left(X_u \prod_{v \in V \ s.t. \ \{u,v\} \in E} \overline{W_v} \right). \tag{9.12}$$

To construct the second degree Hamiltonian for MaxIndependentSet we again consider two cases. Let the to be swapped vertices be u and v. If u and v are not adjacent then they can both be swapped as long as all of their neighbours are not in S, resulting in the term

$$\sum_{u \in V, v \in V \text{ s.t.} \{u,v\} \notin E} \left(X_u X_v \prod_{\substack{w \in V \text{ s.t. } \{u,w\} \in E \\ \text{or } \{v,w\} \in E}} \overline{W_w} \right). \tag{9.13}$$

Only including these terms leads to the MaxIndependentSet-version of the Hamiltonian from (5.3), given by

$$\begin{aligned} H_M = &\sum_{u \in V} \left(X_u \prod_{v \in V \ s.t. \ \{u,v\} \in E} \overline{W_v} \right) \\ &+ \sum_{u \in V, v \in V \text{ s.t.} \{u,v\} \notin E} \left(X_u X_v \prod_{\substack{w \in V \text{ s.t. } \{u,w\} \in E \\ \text{or } \{v,w\} \in E}} \overline{W_w} \right). \end{aligned} \tag{9.14}$$

If u and v are adjacent, then they can not be swapped in at the same time, as then the set S would not be independent anymore. As swapping in or out is a symmetric operation, they therefore can also not be swapped out at the same time. Thus u can only be swapped in if v is being swapped out at the same time and vice versa (similar to MinVertexCover). Adding these terms leads to the full second degree mixing Hamiltonian

$$
\begin{aligned}
H_M = &\sum_{u\in V}\left(X_u \prod_{v\in V \text{ s.t. } \{u,v\}\in E} \overline{W_v}\right) \\
&+ \sum_{u\in V, v\in V \text{ s.t.} \{u,v\}\notin E}\left(X_u X_v \prod_{\substack{w\in V \text{ s.t. } \{u,w\}\in E \\ \text{or } \{v,w\}\in E}} \overline{W_w}\right) \\
&+ \sum_{u\in V, v\in V \text{ s.t.} \{u,v\}\in E}\left(X_u X_v W_u \overline{W_v} \prod_{\substack{w\in V \text{ s.t. } w\neq u,\ w\neq v \\ \{u,w\}\in E \text{ or } \{v,w\}\in E}} \overline{W_w}\right) \\
&+ \sum_{u\in V, v\in V \text{ s.t.} \{u,v\}\in E}\left(X_u X_v \overline{W_u} W_v \prod_{\substack{w\in V \text{ s.t. } w\neq u,\ w\neq v \\ \{u,w\}\in E \text{ or } \{v,w\}\in E}} \overline{W_w}\right).
\end{aligned} \tag{9.15}
$$

This concludes this example.

Second degree Hamiltonians can be constructed in a similar way for other optimization problems. In the general sense one only needs to see what the restrictions of the feasible subspace are, and from there it is possible to quantify what requirements need to be met in order to swap two vertices in or out at the same time. How these second degree Hamiltonians perform for other problems could be problem dependent and is an interesting question for future research.

9.2. Alternatives

Here we will look at two ways to implement second degree mixing Hamiltonians differently within the QAOA+ framework. Firstly, we will look at how parameter-shift rules can greatly improve the runtime of gradient methods. Secondly, we will explore how penalty methods can be used to move constraints to the problem Hamiltonian. This opens up the possibility to use the general unconstrained second degree mixing Hamiltonian discussed in 9.1.2, which could be easier to implement compared to second degree mixing Hamiltonians specifically designed for each optimization problem.

9.2.1. Parameter-shift rules

Earlier in section 6.2 we saw that for both BFGS and CG the gradient of the to be optimized function f is used in the algorithm. For the results from BFGS and CG in this thesis the gradient Δf was approximated using finite differences, but other methods for approximating the gradient are also possible, like automatic differentiation [59] and parameter-shift rules [60]. The problem with using finite difference is that the high errors of near-term quantum devices might make them difficult to use. Parameter-shift rules are a different option for estimating the derivative of quantum functions, having several advantages compared to using finite differences. Parameter-shift rules can require fewer circuit evaluations and are more robust to the errors caused by quantum hardware [61].

Parameter-shift rules allow the gradient of a quantum circuit to be calculated by evaluating the quantum circuit at various shifted points. We follow the example from [62] and consider a quantum circuit where the only unitary is the R_x gate. The R_x gate is of the form

$$U(\theta) = R_x(\theta) = e^{-i\frac{\theta}{2}X}. \tag{9.16}$$

The gradient of the R_x gate with respect to θ is given by

$$\nabla_\theta U(\theta) = \nabla_\theta R_x(\theta) = -\frac{i}{2}Xe^{-i\frac{\theta}{2}X} = -\frac{i}{2}XU(\theta) = -\frac{i}{2}U(\theta)X. \tag{9.17}$$

We want to know how our parameter θ influences the expectation of an observable H in its final state. We apply the R_x gate to a starting state $|\psi\rangle$ and measuring the observable H to get the function

$$f(\psi, \theta) = \langle\psi| U^\dagger(\theta) H U(\theta) |\psi\rangle . \tag{9.18}$$

The gradient of this function $f(x, \theta)$ with respect to θ is then given by

$$\begin{aligned}\nabla_\theta f(x,\theta) &= \nabla_\theta \langle\psi| U^\dagger(\theta) H U(\theta) |\psi\rangle \\ &= \langle\psi| \nabla_\theta (U^\dagger(\theta) H U(\theta)) |\psi\rangle \\ &= \langle\psi| \nabla_\theta U^\dagger(\theta) H U(\theta) + U^\dagger(\theta) H \nabla_\theta U(\theta) |\psi\rangle ,\end{aligned} \tag{9.19}$$

where we use the product rule. Using (9.17) we can rewrite this as

$$\begin{aligned}\nabla_\theta f(\psi,\theta) &= \langle\psi| \nabla_\theta U^\dagger(\theta) H U(\theta) + U^\dagger(\theta) H \nabla_\theta U(\theta) |\psi\rangle \\ &= \langle\psi| (-\frac{i}{2} U^\dagger(\theta) X^\dagger) H U(\theta) + U^\dagger(\theta) H (-\frac{i}{2} X U(\theta)) |\psi\rangle \\ &= -\frac{i}{2} \langle\psi| U^\dagger(\theta)(XH + HX) U(\theta) |\psi\rangle ,\end{aligned} \tag{9.20}$$

using the linearity of the inner product and the fact that $X^\dagger = X$. Next we make use of a commutator identity from [60] to evaluate $XH + HX$, this leads to

$$\begin{aligned}\nabla_\theta f(\psi,\theta) &= -\frac{i}{2} \langle\psi| U^\dagger(\theta)(XH + HX) U(\theta) |\psi\rangle \\ &= -\frac{i}{2} \langle\psi| U^\dagger(\theta)(-i(U^\dagger(\tfrac{\pi}{2}) H U(\tfrac{\pi}{2}) - U^\dagger(-\tfrac{\pi}{2}) H U(-\tfrac{\pi}{2}))) U(\theta) |\psi\rangle .\end{aligned} \tag{9.21}$$

Using the fact that $U(\theta)U(\frac{\pi}{2}) = U(\frac{\pi}{2})U(\theta) = U(\theta + \frac{\pi}{2})$ and $U^\dagger(\theta)U^\dagger(\frac{\pi}{2}) = U^\dagger(\theta + \frac{\pi}{2})$ together with linearity, we can rewrite this as

$$\begin{aligned}\nabla_\theta f(\psi,\theta) &= \frac{1}{2} \langle\psi| U^\dagger(\theta) U^\dagger(\tfrac{\pi}{2}) H U(\tfrac{\pi}{2}) U(\theta) |\psi\rangle \\ &\quad - \frac{1}{2} \langle\psi| U^\dagger(\theta) U^\dagger(-\tfrac{\pi}{2}) H U(-\tfrac{\pi}{2}) U(\theta) |\psi\rangle \\ &= \frac{1}{2} \langle\psi| U^\dagger(\theta + \tfrac{\pi}{2}) H U(\theta + \tfrac{\pi}{2}) |\psi\rangle \\ &\quad - \frac{1}{2} \langle\psi| U^\dagger(\theta - \tfrac{\pi}{2}) H U(\theta - \tfrac{\pi}{2}) |\psi\rangle .\end{aligned} \tag{9.22}$$

Note that we can rewrite the gradient as a sum of function evaluations through

$$\begin{aligned}\nabla_\theta f(\psi,\theta) &= \frac{1}{2} \langle\psi| U^\dagger(\theta + \tfrac{\pi}{2}) H U(\theta + \tfrac{\pi}{2}) |\psi\rangle \\ &\quad - \frac{1}{2} \langle\psi| U^\dagger(\theta - \tfrac{\pi}{2}) H U(\theta - \tfrac{\pi}{2}) |\psi\rangle \\ &= \frac{1}{2}(f(\psi, \theta + \tfrac{\pi}{2}) - f(\psi, \theta - \tfrac{\pi}{2})).\end{aligned} \tag{9.23}$$

This idea can be generalised to find derivatives of the form $U(\theta) = e^{i\theta G)}$ where G is the Hermitian generator G which has exactly two eigenvalues [63]. We can shift these eigenvalues to be of the form $\lambda_{1,2} = \pm r$, as the global phase has no influence on the expectation of U. The gradient can then be computed using the parameter-shift rule

$$\nabla f(\psi, \theta) = \frac{r}{2\sin(rs)}(f(\psi, \theta + s) - f(\psi, \theta - s)), \tag{9.24}$$

where $s \in (0, \pi)$ is a freely chosen shift parameter. Further generalizations to these formulas were also constructed in [63].

These parameter-shift rules can be very useful in determining the gradient of the expectation give in (4.13). This gradient can be used to make the optimization more efficient in the BFGS and CG methods described in sections 6.2.4 and 6.2.5. Instead, the gradient was approximated using finite differences in this thesis. This method was used because the Hamiltonians $H_{M_{V2}}$ and $H_{M_{V3}}$ are implemented directly via their matrix form, as described in section 6.1. As an alternative the circuit could be differentiated with general parameter-rules if the spectral decomposition of $H_{M_{V2}}$ and $H_{M_{V3}}$ is examined [64]. Similar if $H_{M_{V2}}$ and $H_{M_{V3}}$ were to be decomposed into standard gates one could differentiate each of them using parameter-shift rules as in (9.24) as seen in [65]. Using these rules could prove useful in testing if second and higher degree mixing Hamiltonians hold up when applied to larger graphs, as they could greatly reduce the computational cost of gradient-based algorithms.

9.2.2. Penalty Methods

In Section 4.1 we saw that the original Quantum Approximate Optimization Algorithm considers an unconstrained optimization problem as given by (4.1) and (4.2). The setup for QAOA implemented the objective function in the problem Hamiltonian, while the transverse field Hamiltonian was the mixing Hamiltonian for every problem. The Quantum Alternating Operator Ansatz expanded on this setup by considering families of problem and mixing Hamiltonians, so that constrained optimization problems could also be approximated using quantum algorithms. The constraints are implemented into mixing Hamiltonians by making sure they preserve the feasible subspace, which is defined by the constraints. The objective function is still implemented through the problem Hamiltonian. Up to this point the objective function was associated with the problem Hamiltonian, while the constraints were associated with the mixing Hamiltonian. However, we can remove constraints and integrate them within the objective function by using penalty methods. The goal of this subsection is to show that constraints are not necessarily limited to being implemented into the mixing Hamiltonian, and can be implemented through the problem Hamiltonian as well.

Penalty methods are a class of optimization algorithms used to solve constrained problems. The idea is to not completely disallow certain solutions to the problem, but instead penalize them so that their objective value decreases. The constraints are removed and a new objective function is constructed based on both the old objective function and the constraints. This is done through the addition of a penalty function to the objective function, which is also multiplied by a penalty coefficient. The penalty function penalizes any solution outside of the feasible subspace, and adds 0 penalty to any solution within the feasible subspace. The original constrained optimization problem is then replaced by a series of unconstrained optimization problems which then converge to the solution of the original problem. To begin, the series of problems a starting penalty coefficient is chosen, and the new unconstrained optimization problem is solved. For the next iteration the penalty coefficient is increased and this new unconstrained optimization problem is solved again, using the solution of the previous iteration as a starting solution. By increasing the penalty coefficient at each iteration the solution eventually converges, as a very large penalty coefficient corresponds to solutions outside the feasible subspace never being the optimum [66].

Many possible penalty functions and series of penalty coefficients are possible, here we look at a distance-based penalty function as described in [67]. Consider the constrained optimization problem given by

$$\max \quad f(x) \tag{9.25}$$

$$\text{s.t.} \quad c_j(x) \leq 0 \qquad \text{for all } j \in \{1, .., m\}. \tag{9.26}$$

where f is the objective function and each c_j is one of m total constraints. As this is a maximization problem, we want to penalize the solutions which do not follow all constraints by subtracting a penalty function. Note that this penalty function is a function of the constraints, and not of the solution itself. Define our example penalty function as

$$p(c_j(x)) = \max(0, c_j(x)). \tag{9.27}$$

We want $c_j(x)$ to be negative as defined by (9.26), but we do not want to make our solution greater by subtracting a negative penalty function, so we take the maximum between the constraint result and 0. It should be noted that this penalty is not severe and is only used for demonstrative purposes, usually at least a quadratic loss penalty is chosen. As we want to penalize a solution for each of the constraints, it violates we sum over all constraints and multiply by a penalty coefficient σ to get

$$\sigma \sum_{j=1}^{m} \max(0, c_j(x)). \tag{9.28}$$

Subtracting this from the original objective function and using a different penalty coefficient σ_k for the problem with index k (for instance using $\sigma_{k+1} = 2\sigma_k$) gives the following new objective functions $F_k(x)$:

$$F_k(x) = f(x) - \sigma_k \sum_{j=1}^{m} \max(0, c_j(x)). \tag{9.29}$$

To solve the constrained problem one would then iteratively solve the unconstrained problems

$$\max \quad F_k(x). \tag{9.30}$$

Penalty methods can turn a constrained optimization problem into an unconstrained optimization problem (or a series of unconstrained optimization problems). Penalty methods are not the only way of accomplishing this, and a similar transformation can be achieved through the use of other methods, such as interior point methods [68] or augmented Lagrangian methods [69, 70]. In the context of Hamiltonians this means it is possible to use more simple mixing Hamiltonians by adding the penalty functions to the construction of the problem Hamiltonian [71]. To illustrate this, instead of having to implement the constraints of the example given in (9.26), one could use the new objective function from (9.29) to directly construct a problem Hamiltonian of the form

$$\begin{aligned} H_P \left|x\right\rangle &= F_k(x) \left|x\right\rangle \\ &= \left(f(x) - \sigma_k \sum_{j=1}^{m} \max(0, c_j(x))\right) \left|x\right\rangle . \end{aligned} \tag{9.31}$$

and then use the transverse field Hamiltonian as the mixing Hamiltonian. However, implementing this penalty function as a problem Hamiltonian could prove difficult, and does not guarantee an advantage over using the more complicated mixing Hamiltonian instead. If the penalty function is too strong it may make the optimization problem converge poorly and hard to solve, but if the penalty function is too weak the provided solution might not be feasible. Designing a penalty function requires a balance between satisfying the constraints and keeping the problem relatively simple. Using a penalty method together with the second degree transverse field Hamiltonian of (9.11) could be a useful alternative to using the second degree MinVertexCover mixing Hamiltonian $H_{M_{V3}}$.

Bibliography

[1] Ayaan Hossain, Eriberto Lopez, Sean M Halper, Daniel P Cetnar, Alexander C Reis, Devin Strickland, Eric Klavins, and Howard M Salis. Automated design of thousands of nonrepetitive parts for engineering stable genetic systems. *Nature biotechnology*, 38(12):1466–1475, 2020.

[2] Subhash Khot and Oded Regev. Vertex cover might be hard to approximate to within 2- ε. *Journal of Computer and System Sciences*, 74(3):335–349, 2008.

[3] Peter W Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings 35th annual symposium on foundations of computer science*, pages 124–134. Ieee, 1994.

[4] Lov K Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 212–219, 1996.

[5] Aram W. Harrow, Avinatan Hassidim, and Seth Lloyd. Quantum algorithm for linear systems of equations. *Physical Review Letters*, 103(15), oct 2009.

[6] Edward Farhi, Jeffrey Goldstone, and Sam Gutmann. A quantum approximate optimization algorithm. *arXiv preprint arXiv:1411.4028*, 2014.

[7] Stuart Hadfield, Zhihui Wang, Bryan O'gorman, Eleanor G Rieffel, Davide Venturelli, and Rupak Biswas. From the quantum approximate optimization algorithm to a quantum alternating operator ansatz. *Algorithms*, 12(2):34, 2019.

[8] Richard M. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, pages 85–103. Springer US, 1972.

[9] Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing - STOC '71*. ACM Press, 1971.

[10] M R Garey, D S Johnson, and L Stockmeyer. Some simplified NP-complete problems. In *Proceedings of the sixth annual ACM symposium on Theory of computing - STOC '74*, New York, New York, USA, 1974. ACM Press.

[11] M. R. Garey and D. S. Johnson. The rectilinear steiner tree problem is np-complete. *SIAM Journal on Applied Mathematics*, 32(4):826–834, June 1977.

[12] John Adrian Bondy, Uppaluri Siva Ramachandra Murty, et al. *Graph theory with applications*, volume 290. Macmillan London, 1976.

[13] Jon Kleinberg and Eva Tardos. *Algorithm Design*. Addison-Wesley Longman Publishing Co., Inc., USA, 2005.

[14] R. Diestel. *Graph Theory*. Graduate Texts in Mathematics. Springer, 2005.

[15] Dénes König. Gráfok és mátrixok. *Matematikai és Fizikai Lapok*, 38:116–119, 1931.

[16] John E. Hopcroft and Richard M. Karp. An $n^{5/2} $ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing*, 2(4):225–231, December 1973.

[17] James A. Storer. *An Introduction to Data Structures and Algorithms*, page 319. Progress in Computer Science and Applied Logic Series. Springer, 2001.

[18] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2022.

[19] Christos H Papadimitriou and Kenneth Steiglitz. *Combinatorial optimization: algorithms and complexity*. Courier Corporation, 1998.

[20] Vijay V Vazirani. *Approximation algorithms*, volume 1. Springer, 2001.

[21] George Karakostas. A better approximation ratio for the vertex cover problem. *ACM Transactions on Algorithms*, 5(4):1–8, October 2009.

[22] Irit Dinur and Samuel Safra. On the hardness of approximating vertex cover. *Annals of Mathematics*, 162(1):439–485, July 2005.

[23] Khot Subhash, Dor Minzer, and Muli Safra. Pseudorandom sets in grassmann graph have near-perfect expansion. In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 592–601. IEEE, 2018.

[24] Subhash Khot. On the proof of the 2-to-2 games conjecture. *Current Developments in Mathematics*, 2019(1):43–94, 2019.

[25] Subhash Khot. On the power of unique 2-prover 1-round games. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing - STOC '02*. ACM Press, 2002.

[26] Claude Cohen-Tannoudji. *Quantum Mechanics*. Wiley-VCH Verlag GmbH & Co. KGaA, 2020.

[27] R Shankar. *Principles of quantum mechanics*. Kluwer Academic/Plenum, New York, NY, 2 edition, August 1994.

[28] M. Born and V. Fock. Beweis des adiabatensatzes. *Zeitschrift för Physik*, 51(3-4):165–180, March 1928.

[29] Edward Farhi, Jeffrey Goldstone, Sam Gutmann, and Michael Sipser. Quantum computation by adiabatic evolution, 2000.

[30] Edward Farhi, Jeffrey Goldstone, and Sam Gutmann. Quantum adiabatic evolution algorithms with different paths, 2002.

[31] Natasha Feinstein, Louis Fry-Bouriaux, Sougato Bose, and P. A. Warburton. Effects of xx-catalysts on quantum annealing spectra with perturbative crossings, 2022.

[32] Elizabeth Crosson, Edward Farhi, Cedric Yen-Yu Lin, Han-Hsuan Lin, and Peter Shor. Different strategies for optimization using the quantum adiabatic algorithm, 2014.

[33] Jaimie Suzette Stephens, Ojas D. Parekh, and Ciaran Ryan-Anderson. Quantum approximate optimization algorithm (qaoa) on constrained optimization problems. `https://www.osti.gov/servlets/purl/1577041`, Aug 2018.

[34] Naomichi Hatano and Masuo Suzuki. Finding exponential product formulas of higher orders. In *Quantum Annealing and Other Optimization Methods*, pages 37–68. Springer Berlin Heidelberg, November 2005.

[35] Claudio Canuto and Anita Tabacco. *Mathematical Analysis II*, volume 85. Springer, 2015.

[36] Aric Hagberg, Pieter Swart, and Daniel S Chult. Exploring network structure, dynamics, and function using networkx. Technical report, Los Alamos National Lab.(LANL), Los Alamos, NM (United States), 2008.

[37] Sajid Anis et al. Qiskit: An open-source framework for quantum computing, 2021.

[38] Solving combinatorial optimization problems using qaoa. `https://qiskit.org/textbook/ch-applications/qaoa.html#QAOA`, Nov 2022.

[39] Michael JD Powell. An efficient method for finding the minimum of a function of several variables without calculating derivatives. *The computer journal*, 7(2):155–162, 1964.

[40] Jack Kiefer. Sequential minimax search for a maximum. *Proceedings of the American mathematical society*, 4(3):502–506, 1953.

[41] Raymond C. Rumpf. Lecture – powells method - empossible. https://empossible.net/wp-content/uploads/2020/08/Lecture-Powells-Method-1.pdf, 2020.

[42] John A Nelder and Roger Mead. A simplex method for function minimization. *The computer journal*, 7(4):308–313, 1965.

[43] Michael Baudin. Nelder mead user's manual, 2009.

[44] Jade Yu Cheng and Thomas Mailund. Ancestral population genomics using coalescence hidden markov models and heuristic optimisation algorithms. *Computational Biology and Chemistry*, 57:80–92, August 2015.

[45] Michael JD Powell. Direct search algorithms for optimization calculations. *Acta numerica*, 7:287–336, 1998.

[46] Charles George Broyden. The convergence of a class of double-rank minimization algorithms 1. general considerations. *IMA Journal of Applied Mathematics*, 6(1):76–90, 1970.

[47] Roger Fletcher. A new approach to variable metric algorithms. *The computer journal*, 13(3):317–322, 1970.

[48] Donald Goldfarb. A family of variable-metric methods derived by variational means. *Mathematics of Computation*, 24(109):23–26, 1970.

[49] David F Shanno. Conditioning of quasi-newton methods for function minimization. *Mathematics of computation*, 24(111):647–656, 1970.

[50] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge University Press, 2004.

[51] Philip Wolfe. Convergence conditions for ascent methods. *SIAM review*, 11(2):226–235, 1969.

[52] Philip Wolfe. Convergence conditions for ascent methods. ii: Some corrections. *SIAM review*, 13(2):185–188, 1971.

[53] Stephen Wright, Jorge Nocedal, et al. Numerical optimization. *Springer Science*, 35(67-68):7, 1999.

[54] Reeves Fletcher and Colin M Reeves. Function minimization by conjugate gradients. *The computer journal*, 7(2):149–154, 1964.

[55] Elijah Polak and Gerard Ribiere. Note sur la convergence de méthodes de directions conjuguées. *Revue française d'informatique et de recherche opérationnelle. Série rouge*, 3(16):35–43, 1969.

[56] Oleg Alexandrov. Conjugate gradient illustration. https://commons.wikimedia.org/wiki/File:Conjugate_gradient_illustration.svg, 2007.

[57] Leo Zhou, Sheng-Tao Wang, Soonwon Choi, Hannes Pichler, and Mikhail D Lukin. Quantum approximate optimization algorithm: Performance, mechanism, and implementation on near-term devices. *Physical Review X*, 10(2):021067, 2020.

[58] Gary Chartrand and Ping Zhang. *Chromatic graph theory*. Chapman and Hall/CRC, 2008.

[59] Atilim Gunes Baydin, Barak A Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind. Automatic differentiation in machine learning: a survey. *Journal of Marchine Learning Research*, 18:1–43, 2018.

[60] Kosuke Mitarai, Makoto Negoro, Masahiro Kitagawa, and Keisuke Fujii. Quantum circuit learning. *Physical Review A*, 98(3):032309, 2018.

[61] Maria Schuld, Ville Bergholm, Christian Gogolin, Josh Izaac, and Nathan Killoran. Evaluating analytic gradients on quantum hardware. *Physical Review A*, 99(3):032331, 2019.

[62] Parameter-shift rules - pennylane. https://pennylane.ai/qml/glossary/parameter_shift.html.

[63] David Wierichs, Josh Izaac, Cody Wang, and Cedric Yen-Yu Lin. General parameter-shift rules for quantum gradients. *Quantum*, 6:677, 2022.

[64] Oleksandr Kyriienko and Vincent E Elfving. Generalized quantum circuit differentiation rules. *Physical Review A*, 104(5):052417, 2021.

[65] Gavin E Crooks. Gradients of parameterized quantum gates using the parameter-shift rule and gate decomposition. *arXiv preprint arXiv:1905.13311*, 2019.

[66] Alice E. Smith and David W. Coit. Constraint-handling techniques c5.2 penalty functions. 1997.

[67] Jon T Richardson, Mark R Palmer, Gunar E Liepins, and Mike R Hilliard. Some guidelines for genetic algorithms with penalty functions. In *Proceedings of the 3rd international conference on genetic algorithms*, pages 191–197, 1989.

[68] Florian A Potra and Stephen J Wright. Interior-point methods. *Journal of computational and applied mathematics*, 124(1-2):281–302, 2000.

[69] Magnus R Hestenes. Multiplier and gradient methods. *Journal of optimization theory and applications*, 4(5):303–320, 1969.

[70] Michael JD Powell. A method for nonlinear constraints in minimization problems. *Optimization*, pages 283–298, 1969.

[71] Jaimie Suzette Stephens, Ciaran Ryan-Anderson, William Bolden, and Ojas D. Parekh. Quantum approximate optimization algorithm (qaoa) on constrained optimization problems. 2 2019.