Fakultät für Mathematik
Lehrstuhl für Scientific Computing

Quantum algorithms for variants of the traveling salesperson problem

Master's Thesis by Aubrey Coffey

Examiner:	Prof. Dr. Thomas Huckle
Advisor:	Prof. Dr. Christian Mendl and Lilly Palackal
Submission Date:	November 15, 2022

I hereby confirm that this is my own work, and that I used only the cited sources and materials.

München, November 15, 2022

Aubrey Coffey

Abstract

In this thesis we investigate quantum algorithms for the traveling salesperson problem (TSP) and its variants and introduce a new quantum algorithm for the TSP, which we call the graph state alternating operator ansatz (GSAOA). We also investigate optimization formulations to be used in quantum algorithms for the TSP and its variants and introduce two new optimization formulations for a variant of the TSP. For the quantum algorithms we investigated, our main focus is on hybrid quantum-classical algorithms that utilize an alternating operator, amplitude amplification, and classical tuning of the variational parameters. The GSAOA is inspired by these algorithms and incorporates these features. The GSAOA is distinct in that it encodes an instance of the TSP using graph states, instead of the typical enumeration encoding used in the other quantum algorithms investigated.

Zusammenfassung

In dieser Arbeit untersuchen wir Quantenalgorithmen für das Traveling Salesperson Problem (TSP) und seine Varianten und stellen einen neuen Quantenalgorithmus für das TSP vor, den wir den Graph State Alternating Operator Ansatz (GSAOA) nennen. Wir untersuchen auch Optimierungsformulierungen, die in Quantenalgorithmen für das TSP und seine Varianten verwendet werden können, und stellen zwei neue Optimierungsformulierungen für eine Variante des TSP vor. Bei den von uns untersuchten Quantenalgorithmen liegt unser Hauptaugenmerk auf hybriden quantenklassischen Algorithmen, die einen alternierenden Operator, Amplitudenverstärkung und klassische Optimierung der Variationsparameter verwenden. Der GSAOA ist von diesen Algorithmen inspiriert und beinhaltet diese Methoden. Der GSAOA unterscheidet sich von den anderen untersuchten Quantenalgorithmen dadurch, dass er eine Instanz des TSP mit Graphenzuständen kodiert, anstelle der typischen Enumerationskodierung, die in den anderen untersuchten Quantenalgorithmen verwendet wird.

Contents

Chapter 1

Introduction

Quantum computing is of interest because it promises improved computational speed over classical computing [Gro96]. Over the last couple of decades, it has been proven, for certain problems, that quantum algorithms outperform the best current classical algorithms [Sho94]. There is so much potential in quantum computing that companies, such as Infineon Technologies, are investing in research in quantum computing despite the fact that the physical devices for quantum computing are still in the early stages of development. Currently, quantum computing hardware can only handle a small number of qubits and thus cannot outperform current classical methods. Also, it is currently unknown if quantum computing can provide an advantage for all, or even most, problems.

A common approach to solving combinatorial optimization problems classically is using heuristic algorithms that provide an approximate solution. This means that the algorithm is not guaranteed to return to the optimal solution, but hopefully something close to it. In the case of some approximation algorithms, we can prove bounds on how close the returned solution is to the optimal solution. These classical approximation algorithms have motivated the development of quantum approximation algorithms. The hope is that these quantum approximation algorithms may eventually be able to outperform classical approximation algorithms. The success of companies is strongly dependent on their supply chains, which is where combinatorial optimization becomes important. Proper planning of manufacturing and delivery is essential to optimizing the supply chain and reducing costs. Many problems in supply chain optimization, such as vehicle routing problems, can be writtten as combinatorial optimization problems, or combinations of combinatorial optimization problems. Although there are already many classical approaches for tackling these problems, there is hope that faster and more optimal solutions can be reached using quantum computing.

At Infineon there are several use cases involving vehicle routing. Vehicle routing problems include variants of the traveling salesman problem (TSP), such as capacitated vehicle routing problem (CVRP) and multiple depot capacitated vehicle routing problem (MDCVRP). These variants can have very complicated formulations. One common approach to solving the general class of vehicle routing problems is to use a heuristic method and split the problem into subproblems, with one of the subproblems being a

TSP. This is the main motivation for our study of quantum algorithms for the TSP and its variants. We first explored simulated annealing to solve these problems. Then, quantum heuristic algorithms, such as the quantum approximate optimization algorithm (QAOA) were examined. One advantage to the QAOA is that it has been proven to outperform the quantum adiabatic algorithm, which is what quantum annealing is based on. Also, there has been significant research into enhancements of the QAOA. For these reasons, the QAOA is an appropriate starting point to find the best quantum algorithm approach for the TSP. The QAOA has convergence guarantees to find the optimum solution, but only as the number of iterations of the algorithm approaches infinity. A quantum circuit that contains the convergence guarantees of the QAOA is not physically possible. Therefore, the accuracy of the QAOA varies as we cannot obtain a convergence guarantee. It is very difficult to prove bounds on the approximation accuracy for the QAOA when the number of iterations is bounded. Regardless, for some combinatorial optimization problems, the QAOA has shown good results with circuits of a reasonable number of iterations.
One extension of the QAOA is the generalized alternating operator ansatz (AOA) [Had+19] which provides part of the motivation for our own attempt at an algorithm. In this work we decided to base our algorithm on graph states. This is because the TSP is a graph problem, and we expect that by using graph states in a quantum algorithm we will be able to take advantage of the graph structure and geometry in the algorithm. The first goal of this work is to develop our own quantum algorithm for the TSP. The second goal of this work is to research and benchmark existing quantum algorithms on the TSP and its variants. The third goal of this work is to develop new optimization formulations for vehicle routing problems and benchmark these against existing ones. Because of the limitations of current quantum hardware, the benchmarking experiments will be done with small instance examples and run on simulators.

1.0.1 Outline

In Chapter 2, we introduce the TSP and its variants and the use cases at Infineon that the work from this thesis could be applied to. We also outline some classical approaches for these problems. In Chapter 3, we introduce optimization formulations for describing the TSP and its variants so that the formulations can be solved with quantum algorithms. In Chapter 4, we provide an introduction to quantum computing and quantum information. In Chapter 5, we give an overview of current quantum algorithms for the TSP and benchmark them. Lastly, in Chapter 6, we introduce our own quantum algorithm for the TSP, the graph state approximate optimization algorithm (GSAOA).

1.0.2 Acknowledgement

First, I would like to thank my supervisors Lilly Palackal, Prof. Dr. Christian Mendl, and Hans Ehm for their consistent guidance and advice throughout this thesis and Prof. Dr. Thomas Huckle for giving me the opportunity to write my Master's thesis in his department. Furthermore, I want to thank the members of Infineon's quantum team for all the ideas and discussion that have helped me with this work. Finally, I want to thank Infineon Technologies for giving me the opportunity to write my Master's thesis with their company.

Chapter 2

Applications

In this chapter we give an introduction to the optimization problems studied in this work and their relevancy to supply chain problems at Infineon. We also discuss the possibilities of quantum computing and current classical approaches for solving these optimization problems.

2.1 Infineon and supply chain

Infineon Technologies AG is one of the largest semiconductor manufacturers worldwide. Supply chain management entails the organization and deployment of the processes that ensure successful production and delivery of a product or service. The production process transforms the raw materials into the product, in this case semiconductors. Supply chain manangement has many intricate and often inter-connected processes to ensure the proper creation and delivery of the product. Optimal planning within the supply chain is vital to reducing costs and giving companies a competitive advantage. As a company grows and becomes more international, supply chain optimization becomes even more crucial as these production and delivery processes become even more complicated. The semiconductor industry has seen continuous growth but with above-average market volatility. The price-performance ratio, the quality of the performance for a product compared to its price, in the semiconductor industry is constantly improving at a very high rate compared to other industries. This causes changes in the semiconductor market to occur very rapidly. Due to this, semiconductor companies, such as Infineon, have a high need for flexibility and optimal supply chain managment in order to adjust to the volatile market changes.
The supply chain at Infineon follows the supply chain operations reference (SCOR) model. The SCOR model consists of five processes: plan, source, make, deliver, and return. These steps handle all processes within the production and delivery of the semiconductors. Manufactoring procesess within the supply chain are facilitated using almost 50 unique facilities, as many highly specialized facilities are needed. One can see how quickly supply chain optimization becomes extraodinarily complicated, as one has to optimize delivery amounts for production and delivery routes within all these facilities. There is extensive optimization of delivery routes in supply chain

optimization, in the delivery process and also in the other SCOR procceses. This is where vehicle routing problems (VRP) become critical. VRPs encompass a wide range of problems, with varying specificity, but the general goal is to find the optimal set of routes for a fleet of vehicles to utilize to deliver to a group of customers. Throughout this work we focus on two optimization problems that are also VRPs, the traveling salesman problem (TSP) and the capacitated vehicle routing problem (CVRP).

2.2 Combinatorial optimization

VRPs are combinatorial optimization problems, as they focus on optimizing over a discrete set of options instead of a continous spectrum. Combinatorial optimization is vital to successful supply chain implementation. Many problems in supply chain optimization, e.g., vehicle routing problems, job shop scheduling, and others, can be writtten as combinatorial optimization problems, or combinations of combinatorial optimization problems.

2.2.1 Traveling salesman problem

The TSP for a given graph $G = (V, E)$ is to find a route that satisfies the following constraints:

- Every vertex must be visited exactly once.
- The route must return to its starting point.
- We want to minimize the total cost of the route given a distance matrix that specifies the travel cost, also referrred to as edge cost, between any two vertices.

An instance of the TSP is defined by the number of vertices in the graph, n, and the distance matrix, W, which defines the cost of each edge in the graph. In our treatment of the TSP and its variants, we assume an undirected and complete graph. If one is not given a complete graph, one can be created to be used in these formulations by appending the sum of all distances to the unkown edge values. Also, the first two constraints specify that the route must be a Hamiltonian cycle, and throughout this work we will often refer to Hamiltonian cycles as tours. A feasible solution is one that meets the first two constraints, but is not necessarily optimal. The feasible solutions for the TSP are the set of Hamiltonian cycles for $G = (V, E)$. VRPs can be seen as more specialized variants of the TSP.

2.2.2 Capacitated vehicle routing problem

The CVRP is a VRP in which a fixed fleet of vehicles of uniform capacity must deliver commodities to customers with known demand from a common depot. The goal is to minimize the cost of the route while never exceeding the vehicles' capacity. Sometimes, there is an additional constraint to minimize the vehicle fleet. Also, in some interpretations, servicing every customer is not a requirement for a feasible solution. However, for our treatment of the CVRP, we will assume a feasible solution must service all customer demands.
Constraints:

- Truck must start and end at the depot.
- Truck capacity must never be exceeded. There is a fixed vehicle fleet with all vehicles having the same capacity.
- Service every customer's delivery demands.
- Optimize route to minimize travel cost.

A CVRP problem is specified by three variables: the distance matrix, W, the demand for the delivery vertices, d_v, and the truck capacity, C. As an example, the following variables define a CVRP of size $n = 3$, illustrated in Figure 2.1:

$$W = \begin{pmatrix} 0 & 8 & 16 & 10 \\ 8 & 0 & 11 & 13 \\ 16 & 11 & 0 & 9 \\ 10 & 13 & 9 & 0 \end{pmatrix}, d_v = \begin{pmatrix} 12 \\ 8 \\ 10 \end{pmatrix}, C = 20.$$

The route from the feasible solution in Figure 2.1 can be specified by the set of routes within the route: $\{(3), (1), (2, 1)\}$. This means, the truck starts at the depot, travels to delivery vertex 3 and delivers 10 items, and travels back to the depot; this is the purple route. Then the truck travels to delivery vertex 1 and delivers 10 items, which means that vertex still needs 2 items delivered to it, and travels back to the depot; this is the green route. Lastly, the truck delivers 8 items to vertex 2, then travels to vertex 1 and delivers 2 items, then returns to the depot; this is the orange route. The order in which the routes within the route are completed does not affect the cost or quality of the solution. However, the ordering of a route within the route can affect the quality of the solution, but only if the size of the route is greater than 2, where the size does not include the depot vertex.
We introduce a third VRP that is discussed in this work and that is the multiple depot capacitated vehicle routing problem (MDCVRP). This is the same problem as the CVRP but now there are multiple depots, instead of one, that can be used by the trucks to pick up the commodities to be delivered. An MDCVRP problem can have

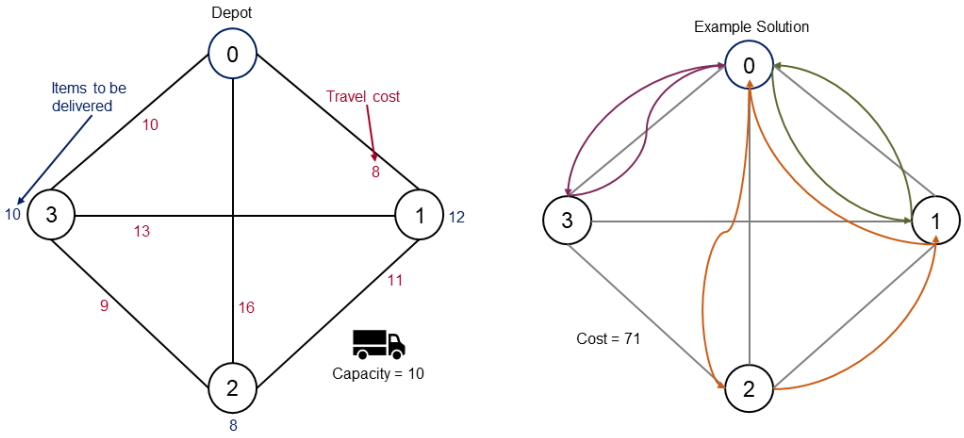


Figure 2.1: On the left is an example of a CVRP of size 3 with vehicle capacity, $C = 10$. In blue is the demand for each delivery vertex, d_v, and in red are the edge costs from the distance matrix, W_{ab}. On the right is a route for this problem, corresponding to the feasible solution $\{(3), (1), (2, 1)\}$.

the constraint that the trucks start and end at one of the depots or, alternatively, start and end at a new garage location, which is irrelevant for pickup and delivery.

2.2.3 Exact, approximate, or heuristic

There is a distinction in categorizing many combinatorial optimization algorithms as either exact, approximation, or heuristic algorithms. An exact algorithm is guaranteed to find the optimal solution. An approximation algorithm is not guaranteed to find the optimal solution, but it has a provable bound on how close the returned solution will be to the optimal solution. A heuristic algorithm is one which is not guaranteed to find the optimal solution and does not have probable guarantees on how close the returned solution will be to the optimal solution. Approximation and heuristic algorithms generally have much better runtime performance than exact algorithms, which is their advantage.
In this work we discuss exact and heuristic approaches for solving the CVRP and the MDCVRP. One heuristic approach is to split the CVRP or MDVRP problem into two subproblems to be solved sequentially. The first subproblem is a clustering, bin packing, or knapsack problem, to determine which points are visited in each route, each route being defined by a start and return to a depot. The second subproblem is a TSP for how to optimize the individual routes. In the case of the MDCVRP, there is also the option of clustering delivery vertices to depots, and then the second subproblem

is solving a CVRP for each cluster. The exact approach is to solve all the problem constraints under one problem. The subproblems in the heuristic approach are simpler to implement than the problem in the exact approach. However, the heuristic approach excludes some feasible, and possibly optimal, solutions.

2.2.4 Applications at Infineon

The supply chain department at Infineon has two applications that this work is quite relevant for. We call these applications the large container problem and the glass container problem. Both problems are vehicle routing problems related to waste management. Both problems can make use of TSP and CVRP solvers, either in a simplified form of the problem or in a subproblem of the realistic form of the problem. Suppose one has containers distributed over a town that need to be emptied. This is currently done by sending trucks through the city, collecting the waste according to a plan. Now, one observes that sometimes containers are overfilled and people deposit their trash beside the container. This is expensive as the drivers need to clean before continuing to the next container. On the other hand, some containers are still almost empty when the driver goes to empty them; these are unnecessary journeys. Having a less optimal route for a VRP leads to waste, financially and in terms of carbon emissions. Infineon has suggested the use of sensors in these containers to determine how full each container is. Then, we can further optimize the routing to containers, knowing the priority of containers, how full each container is. We will now elucidate the two VRPs that motivated this work. For both problems, some approaches require solving a TSP or a CVRP. So, the work done for quantum algorithms on the TSP can also be useful for these problems, despite them being more complicated than the TSP. The glass container problem is, essentially, a "reverse" MDCVRP. The trucks must travel around the city collecting glass waste from containers until the trucks are at capacity, and then they must take the glass waste to disposal sites. This is a "reverse" MDCVRP because instead of the trucks filling up at the depots and delivering to the delivery vertices, the trucks fill up at the trash sites and then deliver to the disposal sites. The distinction is with the MDCVRP, when the truck travels to one of the multiple depots, it generally fills to capacity. Whereas, in the glass container problem, the truck travels to multiple trash sites to fill to capacity and then empties all or most of its weight at the disposal sites. The realistic case for this problem is even more complicated as there can be limits to how much waste the desposal sites can take in. For a simplified version of this problem, one can ignore the limits at the disposal site and treat the problem as an MDCVRP, where the demand is reversed in the formulation. For the realistic case, one approach could be a heuristic approach involving clustering trash pickup sites to disposal facilities based on the limits at the facilities. In the large container problem, trucks can only carry one waste container at a time. In this problem, there are multiple types of waste containers and the disposal

facilities only accept certain types of waste containers. Since the truck can only carry one container at a time, the truck must travel from a pickup site to a disposal facility for every pickup site and then return a container of the same trash type to the pickup site within a reasonable amount of time. There are several approaches for this problem. One approach is to assign every pickup site to the closest disposal facility that accepts the trash container of that type. For each pickup site, the truck travels from the pickpup site to the disposal facility and back. Then, one can solve a TSP on the pickup sites to find the best route to minimize cost. The resulting route can be split among the available trucks. Formulations for solving TSPs, CVRPs, and MDCVRPs will be detailed in the next chapter.

2.3 Classical approaches

There are already many classical approaches for solving the TSP and its variants. Approximation algorithms are often written as an α-approximation algorithm, $\alpha \in \mathbb{R}^+$. This means the algorithm is guaranteed to return a solution that is within a factor of α from the optimal solution in terms of the cost.

2.3.1 TSP

The best exact algorithm for the TSP is the Held-Karp algorithm, which utilizes dynamic programming and solves in time $\mathcal{O}(n^2 2^n)$ [HK61]. Other exact algorithms include branch-and-bound algorithms, progressive improvement algorithms, which utilize techniques similar to linear programming, branch-and-cut algorithms, and hybrid algorithms that are combinations of these approaches.
The best approximation algorithm on the TSP is Christofides' 1.5-approximation algorithm, which solves in time $\mathcal{O}(n^3)$ [Chr22]. There is also a linear programming relaxation, a hybrid Held-Karp relaxation and Christofides method, conjectured to give a 4/3-approximation [HK71]. Heuristic algorithms for the TSP include pairwise exchange, K-opt heuristic, V-opt heuristic, randomized improvement, ant colony optimization, and constructive heuristics, such as nearest neighbor (NN), match twice and stitch (MTS), and the bitonic tour method. Heuristic algorithms can find solutions, with high probability and in a reasonable time, for large problems, millions of vertices, that are within 2 to 3 percent of the optimal tour cost [Reg+11].

2.3.2 CVRP

The current best exact algorithm for the CVRP is from [Fuk+06], which combines branch-and-cut with Lagrangean relaxation. There are other promising exact algorithms that can solve large instances, up to one hundred delivery vertices, to optimality. One from [BHM04] uses a linear programming relaxation and two-commodity network flow

approach. [LLE04] uses a branch-and-cut scheme. A hybrid algorithm from [BCM08] combines a set partitioning formulation, Lagrangean relaxation, and a pricing and cut generation technique. Most exact algorithms for the CVRP are some form of branch-and-bound [TV02], branch-and-cut, column generation/Lagrangean relaxation [ACG17], or hybrid algorithms that are combinations of these approaches.
There are some approximation algorithms for certain special cases of the CVRP. For example, there is the 2.5-approximation algorithm of [AH92], which runs in polynomial time, and the 2-approximation algorithm of [CM99] which runs in polynomial time. For the MDCVRP, there is the 5-approximation algorithm from [CKR01], which also runs in polynomial time. Heuristic algorithms are another promising CVRP approach, examples include genetic algorithms, Tabu search, simulated annealing and adaptive large neighborhood search (ALNS). Heuristic algorithms can find solutions for very large problems, hundreds of thousands of delivery vertices, that are .5 or 1 percent away from the optimal solution [Vid+14]. There is also a heuristic approach that uses a separation methodology for solving the CVRP [Ral+03], where the problem is split into two subproblems, the bin packing problem and the TSP, and solved sequentially.

2.4 Quantum computing

Some consider Richard Feynman's lecture "Simulating Physics with Computers" as the beginning of quantum computing. Feynman explained that simulating quantum mechanical systems is intractable for classical computers due to having to keep track of the probabilites corresponding to the superposition of the particles. This is when interest in a quantum computing device grew, as quantum systems can keep track of these probabilities by design. Feynman conjectured a quantum computing system may be able to solve problems we cannot classically solve. Quantum algorithms are algorithms that are built to be run on quantum devices and take advantage of the properties endowed to quantum devices that are not available with a classical computer. Quantum algorithms can be run on a classical computer using a simulator, a software that simulates the behavior of a quantum computer, but they cannot otherwise run on a classical computer. This is because the fundamental bit for a quantum device is a qubit, quantum bit, instead of a classical bit. However, this will be explained in more detail in Chapter 4. Quantum computing has already been proven to give a computational speed up for certain problems. For example, Shor's algorithm [Sho94] has been proven to solve the prime factorization problem in polynomial time on a quantum device. Grover's algorithm [Gro96] solves the database search problem in time $\mathcal{O}(\sqrt{n})$. Some hypothesize we can get closer to the global optimum for many optimization problems using quantum computing. Due to the additional mathematical properties of quantum computing that are not available in classical computing, there is a strong possibility that quantum algorithms could be the key to finding the best

possible algorithm, in terms of time and space complexity, for problems in computing and optimization.

2.4.1 Computational complexity

Complexity theory entails classifying computational problems by their time and space usage, dependent on the input parameters for the problem. Problems are usually identified with the time and space complexity of the most efficient known algorithm for solving them. For time complexity, the goal is to count how many basic operations are executed throughout the duration of an algorithm, as a function of the size of the input to the algorithm. The size of the input to the algorithm is often referred to as the problem size, and often denoted with n. These basic operations are assumed to take constant time, in that they do not depend on the problem size. However, the number of these basic operations is highly dependent on the problem size.
Many decision problems in computing are classified into three time complexity classes: $\mathcal{P}$, $\mathcal{NP}$-complete, and $\mathcal{NP}$-hard. Problems in the $\mathcal{P}$ class are ones for which there is an algorithm that can solve the problem in polynomial time. A problem is $\mathcal{NP}$-complete if there is no existing algorithm to solve the problem in polynomial time but the correctness of a possible solution can be checked in polynomial time. A $\mathcal{NP}$-hard problem is one for which no polynomial time algorithm exists for solving it and the correctness of a possible solution cannot be verified in polynomial time. It remains an open question if $\mathcal{P} = \mathcal{NP}$-complete. However, if one $\mathcal{NP}$-complete problem can be solved in polynomial time, then all problems in the class can, as all problems in the class can be transformed into each other in polynomial time. If $\mathcal{P} = \mathcal{NP}$-complete then $\mathcal{P} = \mathcal{NP}$-hard, and, therefore, $\mathcal{P} = \mathcal{NP}$. The TSP, CVRP, and MDCVRP are all $\mathcal{NP}$-hard problems.
For quantum algorithms a different classification system is used, one specifically for quantum computational complexity [Wat08]. However, it is possible to determine a bound on some quantum complexity classes using classical complexity classes. In this way, one can prove that, for certain problems, if a quantum algorithm is classified according to a certain computational complexity class we can determine its "equivalent" time complexity classically. This is how quantum advantage was proven for Shor's algorithm and Grover's algorithm. Some hypothesize that quantum algorithms could be the key to proving whether $\mathcal{P} = \mathcal{NP}$.

Chapter 3

QUBO formulations

Most of the quantum algorithms discussed in this work solve quadratic unconstrained binary optimization (QUBO) formulations. The motivation for using QUBO formulations is that they can be translated to Ising models and used in quantum algorithms. In this chapter we introduce the standard QUBO formulation for the TSP and introduce two of our own QUBO formulations for the CVRP. We also investigate other CVRP formulations and MDCVRP formulations. We then benchmark several of these formulations on a small dataset. At the end of this chapter, we explain how these formulations can be adapted for the applications at Infineon.
A QUBO problem is an $\mathcal{NP}$-hard combinatorial optimization problem.

Definition 3.1 (QUBO Problem)
Let $f : \{0,1\}^n \to \mathbb{R}$ be a quadratic polynomial over binary variables.

$$f(x) = \sum_{a=1}^{n} \sum_{b=1}^{n} c_{a,b} x_a x_b. \tag{3.1}$$

Where, $x \in \{0,1\}^n$, $x_a, x_b \in \{0,1\}, \forall a, b \in \{1,\dots,n\}$, and $c_{a,b} \in \mathbb{R}, \forall a, b \in \{1,\dots,n\}$. Here, $x_i, i \in \{1,\dots,n\}$ is the ith index of the binary vector x. Solving the QUBO problem means finding the binary vector, $x^* \in \{0,1\}^n$ that gives the minimum value for Equation (3.1) compared to all other possibilities for $x \in \{0,1\}^n$.

$$x^* = \arg\min_{x \in \{0,1\}^n} f(x). \tag{3.2}$$

QUBO problems can also be formulated as maximization problems instead of minimization problems. Equation (3.1) can also be written in matrix representation as

$$f(x) = x^\intercal Q x = x^\intercal R x + a^\intercal x. \tag{3.3}$$

Where, $Q, R \in \mathbb{R}^{n \times n}, a \in \mathbb{R}^n$. $x_i \in \{0,1\}, \forall i \in \{1,\dots,n\}$. Therefore, $x_i^2 = x_i$, which is why the second equality in Equation (3.3) holds. One can use classical solvers, such as CPLEX or Gurobi, or quantum approaches, such as quantum annealing or quantum algorithms such as the QAOA, to solve a QUBO problem. When creating

a QUBO formulation for an optimization problem, the goal is for the binary vector that is minimum among all possibilities to correspond to the optimal solution for that optimization problem. So, for a QUBO formulation, one can have as many variables as desired as long as all variables are binary and no variables are more than quadratic in their order. For more background and examples of QUBO formulations, see [GK18] and [Luc14].
The following formulations are designed for the traditional CVRP and MDVRP, but they can be augmented to work for the applications at Infineon. At the end of this chapter, we explain how this can be done.

3.1 Exact approach

We will begin with the exact QUBO formulations for the TSP. Then, we will delve into methods for extending to the CVRP and the MDCVRP.

3.1.1 TSP

The following QUBO formulation for the TSP is directly from [Luc14]. The exact approach for the TSP introduced here always includes the optimal solution as a feasible solution and should return the optimal solution given proper weighting constants and a solver than can find the minimum cost of the formulation.
Let $G = (V, E)$, and $N = |V|$. Let the vertices of the graph be labeled $0, \ldots, N-1$. N will also be the number of timesteps in our formulation, with 0 being the first timestep. Therefore, if we sum over all timesteps, we will sum from 0 to $N-1$. Let W be the distance matrix defining the travel cost between every vertex of the graph, i.e., W_{uv} gives the cost of traveling from vertex u to vertex v. The formulation uses N^2 variables, $x_{v,j}$, where v represents the vertex and j represents its order, timestep position, in a prospective cycle,$v, j \in \{0, \ldots, N-1\}$. $x_{v,j} = 1$ if vertex v is visited at timestep j and $x_{v,j} = 0$ otherwise.
First, we will address the feasibility constraints. Let A and B be weighting constants. At every timestep, j, only one vertex, v, can be visited.

$$H_{A1} = A \sum_{j=0}^{N-1} \left(1 - \sum_{v=0}^{N-1} x_{v,j}\right)^2 . \tag{3.4}$$

The next feasibility constraint is that every vertex in the graph must be visited exactly once.

$$H_{A2} = A \sum_{v=0}^{N-1} \left(1 - \sum_{j=0}^{N-1} x_{v,j}\right)^2 . \tag{3.5}$$

We combine these terms to get the feasbility constraint, H_A.

$$H_A = A \sum_{j=0}^{N-1} \left(1 - \sum_{v=0}^{N-1} x_{v,j}\right)^2 + A \sum_{v=0}^{N-1} \left(1 - \sum_{j=0}^{N-1} x_{v,j}\right)^2. \tag{3.6}$$

Equation (3.6) will only reach its minimum of zero if the selection of $x_{v,j}$ corresponds to an ordering of vertices where each vertex is included exactly once. This corresonds to a Hamiltonian cycle on the graph or a Hamiltonian path, technically, but the optimality constraint will ensure it is a cycle.
The previous constraint ensures that the solution that minimizes the QUBO is a feasible solution, a Hamiltonian cycle. The term H_B will address the optimality constraint for the TSP.

$$H_B = B \sum_{(uv)\in E} W_{uv} \sum_{j=0}^{N-1} x_{u,j} x_{v,j+1 \mod N}. \tag{3.7}$$

H_B will minimize the edge cost of the vertex ordering. When combining these two terms, $H = H_A + H_B$, the formulation will find lowest cost cycle among the Hamiltonian cycles. The final Hamiltonian is

$$H = A \sum_{j=0}^{N-1} \left(1 - \sum_{v=0}^{N-1} x_{v,j}\right)^2 + A \sum_{v=0}^{N-1} \left(1 - \sum_{j=0}^{N-1} x_{v,j}\right)^2 + B \sum_{(uv)\in E} W_{uv} \sum_{j=0}^{N-1} x_{u,j} x_{v,j+1 \mod N}. \tag{3.8}$$

Where, $A, B \in \mathbb{R}^+$ and B is small enough that it is never favorable to violate the constraints of H_A to satisfy H_B. One such constraint is $B \max(W) < A$.

3.1.2 CVRP

Later in this chapter we will outline exact CVRP and MDCVRP formulations from other sources. For now, we will outline our own two attempts at an exact QUBO CVRP formulation.

3.1.3 First CVRP formulation

We begin by describing the formulation if one uses one truck. Later, we will explain how to extend the formulation to work for multiple trucks. We start by extending the TSP formulation. Let $G = (V, E)$, and $n + 1 = |V|$. Let the distance matrix, W, be designed such that the depot is always set to be the 0th vertex, the first vertex, and the delivery locations will be vertices $1, \ldots, n$. And there are $n + 1$ total vertices and n delivery vertices. Let N be the number of timesteps in our formulation, with 0 being the first timestep. Therefore, if we sum over all timesteps, we will sum from 0 to $N - 1$. Define $x_{v,j}, \forall v \in \{0, \ldots, n\}, j \in \{0, \ldots, N-1\}$, where v represents the vertex and j

represents its order in a prospective solution. $x_{v,j} = 1$ if vertex v is visited at timestep j and $x_{v,j} = 0$ otherwise. Let C be the capacity of the truck. Let d_v be the customer demand at each delivery vertex. For d_v, for $v \in \{1, \ldots, n\}$, d_v will be negative, as the truck's weight will decrease when it visits those vertices and delivers the items. When the truck visits a delivery vertex, it delivers that vertex's full demand to that vertex and the truck's weight also decreases by that amount. For our formulation we will let the depot have a demand value, unlike the example in Figure 2.1. Let $d_0 = 1$ be the demand of the depot, which will be positive, as the truck will increase in weight whenever it is at the depot. Our total number of timesteps is

$$N = -\sum_{v=1}^{n} d_v + n + 1. \tag{3.9}$$

By setting $d_0 = 1$ and allowing for more timesteps, enough to satisfy all customer demand, the truck can fill exactly as much as needed by staying at the depot for multiple timesteps. Let A and B be weighting constants, $A, B \in \mathbb{R}^+$.
First constraint: The truck must start and end at the depot.
To enforce that the truck starts at the depot, one needs $x_{0,0} = 1$ and $x_{v,0} = 0, \forall v \neq 0$. We add the following term to our Hamiltonian:

$$H_{A1} = A((x_{0,0} - 1)^2 + \sum_{v=1}^{n} x_{v,0}^2). \tag{3.10}$$

To enforce that the truck ends at the depot, one needs $x_{0,N-1} = 1$ and $x_{v,N-1} = 0, \forall v \neq N-1$. We add the following term to our Hamiltonian:

$$H_{A2} = A((x_{0,N-1} - 1)^2 + \sum_{v=1}^{n} x_{v,N-1}^2). \tag{3.11}$$

So, our final term to describe this constraint is

$$H_A = H_{A1} + H_{A2} = A((x_{0,0} - 1)^2 + \sum_{v=1}^{n} x_{v,0}^2) + A((x_{0,N-1} - 1)^2 + \sum_{v=1}^{n} x_{v,N-1}^2). \tag{3.12}$$

Second constraint: Service every customer's delivery demands.
To satisfy this, one must ensure that the route visits every delivery vertex exactly once and that the weight of the truck never becomes negative; this will be handled with the capacity constraint. This is also assuming that every customer's individual delivery demands are less than the capacity of the truck. Later, we will explain how to extend the formulation for these cases. To ensure every delivery vertex is visited exactly once,

$$H_{B1} = A \sum_{v=1}^{n} \left(1 - \sum_{j=1}^{N-2} x_{v,j}\right)^2. \tag{3.13}$$

We also need to ensure that at every timestep a vertex is being visited,

$$H_{B2} = A \sum_{j=1}^{N-2} \left(1 - \sum_{v=0}^{n} x_{v,j}\right)^2. \tag{3.14}$$

Then, we add the following term to our Hamiltonian:

$$H_B = H_{B1} + H_{B2} = A \sum_{v=1}^{n} \left(1 - \sum_{j=1}^{N-2} x_{v,j}\right)^2 + A \sum_{j=1}^{N-2} \left(1 - \sum_{v=0}^{n} x_{v,j}\right)^2. \tag{3.15}$$

Third constraint: Truck capacity must never be exceeded.
Additionally, we need to ensure that truck weight must never be negative. If this happens, it means the truck has delivered items that it hasn't picked up from the depot. Our formulation works by treating delivery vertices as having negative demand and the depot as having positive demand. By keeping the truck weight greater than or equal to 0 and less than or equal to the capacity, C, and by visiting every delivery vertex, we will ensure that every delivery vertex has its demand met and truck capacity is not exceeded. However, because this constraint is an inequality, it is difficult to make it a QUBO problem. Let $J \in \{1, \ldots, N-1\}$, one needs to ensure that

$$\forall J, \sum_{v=0}^{n} \sum_{j=0}^{J} x_{v,j} \cdot d_v \leq C, \tag{3.16}$$

$$\forall J, \sum_{v=0}^{n} \sum_{j=0}^{J} x_{v,j} \cdot d_v \geq 0. \tag{3.17}$$

We use slack variables to implement this inequality. Define $\lambda_m^p, \forall p \in \{0, \ldots, N-1\}, \forall m \in \{0, \ldots, C\}$. For a given timestep, p, $\lambda_m^p = 1$ if the truck weight is m and $\lambda_m^p = 0$ otherwise. Since the truck weight must be equal to some value between 0 and C, inclusive, and not more than one value, we maintain that

$$\forall p \sum_{m=0}^{C} \lambda_m^p = 1. \tag{3.18}$$

This can be included in the Hamiltonian as

$$H_{C1} = A \sum_{p=0}^{N-1} (1 - \sum_{m=0}^{C} \lambda_m^p)^2. \tag{3.19}$$

Then, the truck weight, determined by vertex demand, must be set equal to the weight determined by the slack variables.

$$\forall p \sum_{m=0}^{C} \lambda_m^p m = \sum_{v=0}^{n} \sum_{j=0}^{p} x_{v,j} d_v. \tag{3.20}$$

This can be included in the Hamiltonian as

$$H_{C2} = A \sum_{p=0}^{N-1} (\sum_{m=0}^{C} \lambda_m^p m - \sum_{v=0}^{n} \sum_{j=0}^{p} x_{v,j} d_v)^2. \tag{3.21}$$

So, our final Hamiltonian term for this constraint will be

$$H_C = H_{C1} + H_{C2} = A \sum_{p=0}^{N-1} (1 - \sum_{m=0}^{C} \lambda_m^p)^2 + A \sum_{p=0}^{N-1} (\sum_{m=0}^{C} \lambda_m^p m - \sum_{v=0}^{n} \sum_{j=0}^{p} x_{v,j} d_v)^2. \tag{3.22}$$

Fourth constraint: Minimize the cost of the route.
To satisfy this constraint we add the following term to our Hamiltonian:

$$H_D = B(\sum_{a=0}^{n} \sum_{b=0}^{n} W_{ab} \sum_{j=0}^{N-2} x_{a,j} x_{b,j+1}), \tag{3.23}$$

where W_{ab} is the cost of the travel time between vertices a and b.
There is an optional constraint that can be added to the formulation. After testing with our classical QUBO solver, we found that the formulation returns the same or equivalent solutions with or without this constraint. However, with this constraint, the optimal solution is found by the solver much faster. For this reason, we choose to include this in the formulation. This constraint ensures the depot is visited the correct number of times in order to make all necessary deliveries. One needs to ensure that

$$\sum_{j=0}^{N-2} x_{0,j} = -\sum_{v=1}^{n} d_v. \tag{3.24}$$

So, the following term is added to the Hamiltonian:

$$H_E = A(\sum_{j=0}^{N-2} x_{0,j} + \sum_{v=1}^{n} d_v)^2. \tag{3.25}$$

Then the final Hamiltonian is

$$H = H_A + H_B + H_C + H_D + H_E. \tag{3.26}$$

For the weighting of the constants, H_A, H_B, H_C and H_E are all necessary for the feasibility of the solution; whereas, H_D optimises the travel cost. B should be smaller than A such that it is never favorable to violate the feasibility constraints to satisfy H_D. One such constraint is $B \max(W) < A$.

To extend to multiple trucks, one can split the routes, where a route starts at the

depot and returns to the depot, among the available trucks. This will not change the total travel time incurred by the fleet. And if the total number of routes is less than the total number of vehicles in the fleet, the optional constraint to minimize the number of vehicles used will also be satisfied. This method for extending to multiple trucks also works for other CVRP formulations and MDCVRP formulations.

There is one issue left with this formulation. It cannot handle the cases where a delivery vertex requires multiple visits because its demand is greater than the truck's capacity. To fix this, we need to edit the formulation slightly and do some data pre-processing and post-processing. Let r_v be a vector that defines for each vertex how many times the truck will have to visit the depot in order to service that vertex.

$$r_v = \left\lceil \frac{-d_v}{C} \right\rceil. \tag{3.27}$$

For delivery vertices, i, where $r_i > 1$, we will create $r_i - 1$ duplicate delivery vertices to be added to the distance matrix, W, and the demand vector, d_v. Let $r_i = a$ for a given vertex i, where $a > 1$. The $a - 1$th duplicate delivery vertex will have demand equal to $d_i \mod C$, unless that value is 0, then it will be C instead of 0. And the other duplicate delivery vertices for i and the original vertex will have demand equal to C. One also updates the distance matrix accordingly. Basically, there will be a vertices corresponding to one location. One issue with this approach is that by assigning the demand to the duplicate vertices in advance we are eliminating feasible solutions, and possibly the optimal solution, from being considered by the formulation. For example, if $C = 10$ and vertex i has a demand of 21, we will be creating 2 duplicate vertices, and have 3 vertices to represent this vertex, having demand values of $10, 10$, and 1. However, the optimal solution for this instance may deliver the demand to vertex i in the form of three trips each delivering 7 items. And, in this approach, that feasible solution will not be considered. To complete this approach we need to update our timestep calculation.

$$N = -\sum_{v=1}^{n} d_v + \sum_{v=1}^{n} r_v + 1. \tag{3.28}$$

Example 3.2
Let there be a CVRP instance of size $n = 2$, defined by the following variables:

$$W = \begin{pmatrix} 0 & 8 & 16 \\ 8 & 0 & 11 \\ 16 & 11 & 0 \end{pmatrix}, d_v = \begin{pmatrix} 1 \\ -8 \\ -12 \end{pmatrix}, C = 10.$$

The pre-processing necessary for this formulation to work on this instance will result

in the instance

$$W = \begin{pmatrix} 0 & 8 & 16 & 16 \\ 8 & 0 & 11 & 11 \\ 16 & 11 & 0 & 0 \\ 16 & 11 & 0 & 0 \end{pmatrix}, d_v = \begin{pmatrix} 1 \\ -8 \\ -10 \\ -2 \end{pmatrix}, C = 10.$$

We solve this instance using the formulation. Then, in the post-processing of the solution, we reassign the duplicate vertices to their original vertex.

While looking for solutions to the previously mentioned problem, how to still consider all feasible solutions when a vertex requires multiple deliveries, a new issue became apparent. There are instances where the optimal solution requires multiple visits to a delivery vertex even if it is undercapacity.

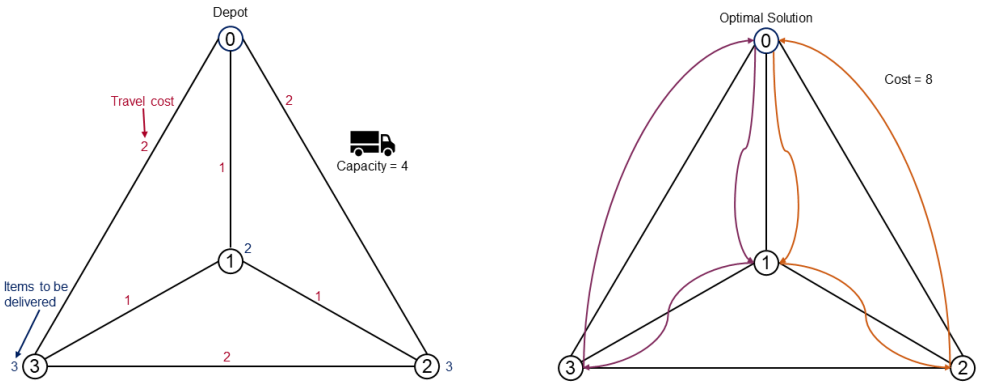


Figure 3.1: On the left is an example of a CVRP of size 3 with vehicle capacity, $C = 4$. In blue is the demand for each delivery vertex, d_v, and in red are the edge costs from the distance matrix, W_{ab}. On the right is the optimal route for this problem, corresponding to the feasible solution $\{(12), (13)\}$.

Example 3.3
Let there be a CVRP instance of size $n = 3$, defined by the following variables:

$$W = \begin{pmatrix} 0 & 1 & 2 & 2 \\ 1 & 0 & 1 & 1 \\ 2 & 1 & 0 & 2 \\ 2 & 1 & 2 & 0 \end{pmatrix}, d_v = \begin{pmatrix} 1 \\ -2 \\ -3 \\ -3 \end{pmatrix}, C = 4.$$

This example is illustrated in Figure 3.1. When the first formulation solves this instance, it will give a solution that visits each delivery vertex exactly once, since each is at or

under capacity. For this instance, $N = -(-3-3-2)+3+1 = 12$. The optimal solution found by the first formulation will be three routes $\{(1)(2)(3)\}$. And, in any order, i.e., the solutions $\{(1)(2)(3)\}, \{(1)(3)(2)\}, \{(2)(1)(3)\}, \{(2)(3)(1)\}, \{(3)(2)(1)\}, \{(3)(1)(2)\}$ will all have equivalent travel cost and all be feasible. Let $x_{v,j\ldots k} = x_{v,j}, x_{v,j+1}, \ldots, x_{v,k}$. Then the variable specification for the optimal solution according to the first formulation will be

$$x_{0,0\ldots1} = 1, x_{1,2} = 1, x_{0,3\ldots5} = 1, x_{2,6} = 1, x_{0,7\ldots9} = 1, x_{3,10} = 1, x_{0,11} = 1,$$

with the remaining $x_{v,j} = 0$. This route has cost $(2+2)+(2+2)+(1+1) = 10$. However, the true optimal solution to this problem involves visiting vertex 1 twice and making two routes instead of three. The true optimal solution is $\{(12)(13)\}$. For the first route, the truck fills to capacity, 4, travels from the depot to vertex 1 and delivers 1 item, then travels to vertex 2 and delivers 3 items, then returns to the depot. On the second route, the truck fills to capacity, 4, travels from the depot to vertex 1 and delivers 1 item, then travels to vertex 3 and delivers 3 items, then returns to the depot. The cost of this route is $(1+1+2)+(1+1+2) = 8$. Changing the order of the routes inside the route will lead to equivalent solutions, as will taking the permutations of the vertices inside the individual routes since each individual route has a size less than 3.

We conclude that the first formulation will always return a feasible solution, but not necessarily the optimal solution, even when each delivery vertex has demand less than the capacity. This is when we decided to formulate a second CVRP formulation.

3.1.4 Second CVRP formulation

In the first formulation, whenever a truck visits a delivery vertex, it will deliver the maximum load possible, which is either the capacity of the truck, the full demand of that delivery vertex, or the remainder of the full demand when divided by the capacity. The idea behind the second formulation is that, instead of delivering the maximum possible load when visiting a delivery vertex, the truck delivers one item for each timestep it is at the vertex. This is similar to how the truck fills up at the depot in the first formulation, except now this is implemented when the truck fills up and when it delivers. By allowing the truck to deliver one item for each timestep it is at a delivery vertex, we make the delivery amount variable, which allows us to handle special cases, such as the one in Example 3.3. For example, if the truck is at delivery vertex i for 3 timesteps before it moves to a new vertex, then the truck delivered 3 items to vertex i on that visit to the vertex. Most of the second formulation remains the same as the first, but there are a few updates. The Hamiltonian terms H_A, H_{B2}, H_{C1}, H_D and H_E will remain the same. This formulation does not require the pre-processing that the first formulation does when a vertex's demand is overcapacity.

First, we edit the number of timesteps in the formulation.

$$N = -2\sum_{v=1}^{n} d_v + 1. \tag{3.29}$$

Next, the term H_{B1} needs to be updated to ensure that the number of times a delivery vertex is visited is equal to the demand at that vertex.

$$H_{B1} = A\sum_{v=1}^{n}\left(-d_v - \sum_{j=1}^{N-2} x_{v,j}\right)^2. \tag{3.30}$$

Lastly, the H_{C2} term needs to be updated. Now, every timestep the truck is at a delivery vertex the truck weight will decrease by 1 instead of by d_v. So, essentially, replace d_v with 1 in H_{C2}.
We want, for $J \in \{1, \ldots, N-1\}$

$$\forall J, \sum_{v=0}^{n}\sum_{j=0}^{J} x_{v,j} \leq C, \tag{3.31}$$

$$\forall J, \sum_{v=0}^{n}\sum_{j=0}^{J} x_{v,j} \geq 0. \tag{3.32}$$

Again, slack variables are used. Most of the construction stays the same. However, the calculation of the truck weight at timestep p needs to be updated

$$\forall p \sum_{m=0}^{C} \lambda_m^p m = \sum_{v=0}^{n}\sum_{j=0}^{p} x_{v,j}. \tag{3.33}$$

This can be included in the Hamiltonian as

$$H_{C2} = A\sum_{p=0}^{N-1}(\sum_{m=0}^{C} \lambda_m^p m - \sum_{v=0}^{n}\sum_{j=0}^{p} x_{v,j})^2. \tag{3.34}$$

Then the final Hamiltonian is

$$H = H_A + H_B + H_C + H_D + H_E. \tag{3.35}$$

And, the weighting constants, A and B, can be set using the same method as in the first formulation.

Example 3.4
We will illustrate how the second formulation would solve the instance from Example 3.3, the instance illustrated in Figure 3.1. Let $x_{v,j\dots k} = x_{v,j}, x_{v,j+1}, \dots, x_{v,k}$. The number of timesteps for this instance, using the second formulation, will be $N = -2 \cdot (-2-3-3) + 1 = 17$. Then the variable specification for an optimal solution found by the second formulation will be

$$x_{0,1\dots3} = 1, x_{1,4} = 1, x_{2,5\dots7} = 1, x_{0,8\dots11} = 1, x_{1,12} = 1, x_{3,13\dots15} = 1, x_{0,16} = 1,$$

with the remaining $x_{v,j} = 0$. This route has cost $(1+1+2)+(1+1+2) = 8$. This is the route depicted on the right in Figure 3.1. There are additional variable assignments that would lead to equally optimal solutions in terms of cost.

3.1.5 Comparing our formulations

The second formulation will always return the optimal solution; whereas, the first will always return a feasible solution and possibly the optimal solution. The main issue with both formulations is that they become very costly in the number of variables and constraints. This makes it difficult to use either formulation to solve any larger instances of the CVRP.

Number of variables

For both formulations the total number of variables will be $Nn + NC$. However, the timestep calculation is much larger in the second formulation. One can see this by comparing Equations (3.28) and (3.29). The difference between the N values is $\sum_{v=1}^{n} d_v - \sum_{v=1}^{n} r_v$.

3.2 Heuristic approach

Instead of solving the CVRP or the MDCVRP with an exact formulation, we can also use a heuristic approach. Here, we split the problem into multiple subproblems and solve them sequentially. One issue with the heuristic approach is that it excludes more feasible solutions, and thus possibly the optimal solution, than the exact approach does. One advantage to the heuristic approach is that by splitting into subproblems one decreases the problem size, and the solution can generally be obtained faster and used on larger problem instances.
In both problems, CVRP and MDCVRP, certain approaches involve using a clustering, or knapsack, phase as a subproblem. The clustering phase can be done classically or quantumly, but if one wants to use a fully quantum approach, it should be done using a QUBO formulation.

3.2.1 CVRP

For heuristic CVRP, we need to cluster the delivery points among multiple routes depending on the capacity, then solve a TSP for each route. There are several approaches to doing this. First, we will attempt the fully QUBO heuristic approach outlined in [Fel+19]. The most successful method in [Fel+19] uses a hybrid-quantum approach, quantum QUBO TSP subproblem and classical clustering subproblem. There is also a quantum exact approach and a quantum heuristic approach. The exact approach outlined in [Fel+19] did not return feasible solutions but the heuristic approach did. For this heuristic approach the problem is broken into two QUBOs, a knapsack subproblem and a TSP subproblem. However, solving the knapsack phase with a QUBO was found to be impractical and instead a hybrid solution where the clustering phase is solved classically and the TSP is solved with a quantum annealer was preferred. We implement the fully quantum heuristic approach, solving both subproblems as a QUBO.
The TSP for this approach is solved with the same QUBO outlined previously. For the knapsack QUBO, the authors use the following QUBO formulation, which is a variation of the knapsack QUBO from [Luc14]. Here, N is the number of delivery vertices, $\alpha \in \{1, \dots, N\}$ is the delivery vertex, $k \in \{1, \dots, m\}$ is the route, D is the distance matrix, W is the capacity of the truck, and m is the total number of routes. $y_n = 1$ if the final weight of the knapsack is n, and $y_n = 0$ otherwise, where, $n \in \{1, \dots, W\}$. w_α is the demand of each delivery vertex, which is positive, as it is in most formulations. $x_\alpha^k = 1$ if the delivery vertex is contained in route k and $x_\alpha^k = 0$ otherwise.

$$H_A = X \sum_{k=1}^{m} (1 - \sum_{n=1}^{W} y_n^k)^2 + A \sum_{k=1}^{m} (\sum_{n=1}^{W} n y_n^k - \sum_{\alpha} w_\alpha x_\alpha^k)^2, \tag{3.36}$$

$$H_B = X \sum_{\alpha} (1 - \sum_{k=1}^{m} x_\alpha^k)^2, \tag{3.37}$$

$$H_C = C \sum_{k=1}^{m} (\sum_{u,v \in E} D_{uv} x_v^k x_u^k). \tag{3.38}$$

Where the final Hamiltonian, H, is defined as

$$H = H_A + H_B + H_C. \tag{3.39}$$

This can be seen as a multiple knapsack where distance of objects is taken into account using the H_C term. For the weighting of the constants, H_A and H_B are necessary for the feasibility of the solution; whereas, H_C optimises the travel cost. X is used as a penalty in H_A and H_B and must be set very high. X must be greater than A, and A must be greater than C. The recommended settings for the constants are $X = A^2$ and $A = \max(D) \cdot N$. The authors mention one problem in this formulation is that

the C constant has to be chosen for each dataset, making it less practical than the classical approach. In their formulation, they choose m to be the number of vehicles, and thus also the number of routes. This also assumes, however, that m is large enough to handle the demand of all delivery vertices, assuming each truck does one route, as they do. We chose to calculate m as below,

$$m = \left\lceil \frac{\sum_{v=1}^{n} d_v}{W} \right\rceil . \tag{3.40}$$

However, we realized an issue with this calculation of m. This cannot handle the special cases such as the one in Example 3.3. Using Equation (3.40), would lead to infeasible solutions for the instance mentioned, as the formulation cannot place a delivery vertex into multiple knapsacks, only one. And, given the capacity in that example, and $m = 2$, according to Equation (3.40), would not be enough routes to service all vertices. Our solution to this was to increase m by 1,

$$m = \left\lceil \frac{\sum_{v=1}^{n} d_v}{W} \right\rceil + 1. \tag{3.41}$$

This ensures feasible solutions but is certainly less than optimal. We also include the data pre-processing and post-processing from Section 3.1.3, seen in Example 3.2, for instances where a delivery vertex's demand is greater than the capacity. In our benchmarking we tested the formulation using both Equations (3.41) and (3.40) for m. Future work could be to find a knapsack/clustering QUBO formulation that does not have these disadvantages.

3.2.2 MDCVRP

Exact MDCVRP formulations are of interest, but challenging. They did not return feasible solutions, as described in Section 3.3. For heuristic MDCVRP, there are several approaches for how to split the problem into subproblems. One approach is to assign all the vertices to the depot closest to them and then solve a CVRP for each depot cluster. Another approach is to cluster the delivery points into multiple routes, respecting truck capacity, and assign them to depots, then implement TSPs for each route. There are many options for the clustering phase. Doing a hybrid-quantum method with a classical clustering phase and a QUBO TSP is one option. If one wants a fully quantum approach, one could use the same clustering technique as detailed in Subsection 3.2.1 to split the delivery vertices into different routes and then assign each route to the depot closest to the center of the route. Alternatively, one can use a QUBO more suited to clustering for MDCVRP. Future work could be to finish this QUBO formulation.

3.3 Attempted formulations

For [Fit+21] and [Har+20] we created coding implementations of the exact QUBO formulations detailed therein. In both cases, the heterogeneous vehicle routing problem (HVRP) and the MDCVRP, the formulations can also be used for CVRP problems, as they are both more complicated problems than the CVRP and can reduce to the CVRP depending on the problem input.

- HVRP [Fit+21] - the HVRP is the same as the CVRP except the trucks in the fleet can have varying capacity. This formulation did not consistently return feasible solutions.
- MDCVRP [Har+20]- We were unable to obtain feasible solutions from this formulation as well. Our classical solver was not able to solve this formulation because the QUBO appears to have cubic terms. After attempting to solve an MDCVRP instance with a version of the formulation where the part containing cubic terms was removed, we were still unable to obtain feasible solutions.

For these formulations, we were not able to obtain feasible solutions. We are not sure if it is the formulation itself, the coding implementation of it, or the weighting of the constants, which are not always clearly specified in the papers. Also, for our work, we only consider a solution feasible if it satisfies the demand of all delivery vertices.

3.4 Adapting for Infineon applications

We do not have an exact QUBO approach for the MDCVRP that returns feasible solutions. However, the problem can always be solved using the heuristic method. The heuristic approaches do not need to be edited extensively to work for the glass container problem. For the exact approach for the CVRP, we need to reverse the signs in the demand. So, $d_0 = -1$ and d_v is positive for all pickup points. Since, in the glass container problem, we are picking up trash from pickup points, increasing the truck weight, and then delivering it to trash disposal facilities, decreasing the truck weight. Although, for all approaches we need to handle the depot vertex differently. As, in the traditional CVRP, the garage and depot point are the same. For the glass container problem, the garage point is different from the disposal facilities. For the exact approach, we can add an additional garage vertex, $n+1$th vertex, and change the first constraint so that the truck starts and ends at the garage instead of the disposal facility or depot. We also need to add 1 to the number of timesteps. For the heuristic approach, it will depend on how many trucks there are, but we can try to optimize the route from the garage to the first pickup location on a given truck's route and also optimize the route from the last disposal facility to the garage.
The larger issue is how to handle disposal facilities which have limits to how much trash

they can accept. The realistic instance of the glass container problem is an MDCVRP with reversed demand, except it is a bit more complicated in that it is limited on both sides. The dropoff points may not be able to accept unlimited amounts of trash, in which case dropoff points become unusable once their threshold is reached. This makes the problem much more complicated. One solution for this could be in the clustering phase of the heuristic method. We could assign all pickup points to their closest disposal facility, and for disposal facilities that are overcapacity we reassign some of its pickup points to another disposal facility. Then we solve a CVRP for each disposal facility cluster.

3.5 Benchmarking

The classical solver we use for benchmarking is DOcplex, which is a Python implementation of CPLEX. If DOcplex returns a solution for a QUBO problem, that solution is the optimal solution for that QUBO. For this reason we chose to use DOcplex instead of quantum annealing for our benchmarking. This ensures the best comparison of the three formulations we benchmark. We chose not to benchmark any of the attempted formulations since they did not consistently return feasible solutions. We choose to benchmark four formulations, our own first and second exact CVRP formulations, and the heuristic formulation detailed in Section 3.2.1. For the heuristic formulation, we ran two versions, which we will refer to as the first heuristic approach, which uses Equation (3.40) for the calculation of m, and the second heuristic approach, which uses Equation (3.41) for the calculation of m.
To calculate the optimal CVRP route, and cost, for each instance we used a brute force classical approach. We used a small dataset for this benchmarking, eight CVRP instances. For these instances, all are size $n = 3$ and the value for the capacity and the values for the demand at each delivery vertex were randomly chosen from $\{1, 2\}$. We had to use such small ranges because when using our own second exact CVRP formulation the number of variables increases dramatically with more delivery vertices, a higher capacity, and higher demand values, such that, the problems were no longer solvable by our classical solver, at least not with the computing resources we had available. We found that for seven of the eight problems, all four formulations returned the same cost route, which was also the optimal route. For one problem, both heuristic approaches returned no solution. And, both exact approaches returned the same optimal solution. Future work on the exact CVRP formulations could be to benchmark them on larger instances and do more extensive benchmarking to have a better estimate of the effectiveness of the formulations. [1]

[1] All code implemented in this work is available in this git repository, https://github.com/aubreycoffey/TSP-Quantum-Computing

Chapter 4

Preliminaries in quantum information theory

This chapter provides a short overview of the concepts and results from quantum information theory that are relevant to this work and will become useful later. All stated definitions and theorems until Section 4.5 can be found in [NC10], unless otherwise specified. Definitions and theorems from Section 4.5 can be found in [Hei+06]. We assume that the reader has some familiarity with quantum computation. For more information, please see [Hei+06] and [NC10].

4.1 Mathematical Review

This section provides a quick overview of concepts and theorems from linear algebra and group theory that will be used throughout this work.

4.1.1 Linear algebra

Concepts from linear algebra are fundamental in quantum computing. Here we give some basic definitions and more specific theorems that will be utilized in this work.

Definition 4.1 (Unitary)
A unitary operator is a surjective, bounded operator on a Hilbert space that preserves the inner product. Let $\mathcal{H}$ be a complex Hilbert space. $U : \mathcal{H} \to \mathcal{H}$. If $\mathcal{H}$ is a finite-dimensional vector space, the unitary operators are quadratic matrices. A unitary matrix, U, is a matrix that fulfills the following condition:

$$U^\dagger U = \mathbb{I}.$$

Where, $U^\dagger$ is the adjoint of U, obtained by taking the transpose and then the complex conjugate of U.

Definition 4.2 (Hermitian Operator)
An operator, A, is Hermitian if it is equal to its own adjoint, i.e.,

$$A = A^\dagger = (A^*)^T.$$

$A : \mathcal{H} \to \mathcal{H}$. If $\mathcal{H}$ is a finite-dimensional vector space, the Hermitian operators are quadratic matrices.

Definition 4.3 (Normal Operator)
An operator, A, is normal if

$$AA^\dagger = A^\dagger A.$$

An operator is normal if and only if it is diagonalizable. $A : \mathcal{H} \to \mathcal{H}$. If $\mathcal{H}$ is a finite-dimensional vector space, the normal operators are quadratic matrices.

Definition 4.4 (Positive Operator)
A positive operator, A, is one for which $\langle\psi| A |\psi\rangle \geq 0$ for all $|\psi\rangle$. This means A must be a positive semi-definite matrix. A matrix is positive semi-definite if its eigenvalues are real and non-negative.

Definition 4.5 (Commuting Operators)
Two operators, A and B, commute if $AB = BA$. We say A commutes with B or B commutes with A. We also define the commutator as $[A, B] = AB - BA$.

Definition 4.6 (Anti-commuting Operators)
Two operators, A and B, anti-commute if $AB = -BA$. We say A anti-commutes with B or B anti-commutes with A. We also define the anti-commutator as $\{A, B\} = AB + BA$.

Definition 4.7 (Kronecker Product)
Let $A \in \mathbb{C}^{m\times n}, B \in \mathbb{C}^{p\times q}$, then the Kronecker product of A and B is given by

$$A \otimes B = \begin{bmatrix} a_{11}B & \cdots & a_{1n}B \\ \vdots & \ddots & \vdots \\ a_{m1}B & \cdots & a_{1mn}B \end{bmatrix} \tag{4.1}$$

The Kronecker product is bilinear and associative.

Lemma 4.8
Here, we enumerate some useful properties of the Kronecker product.

1. *The Kronecker product is associative:* $(A \otimes B) \otimes C = A \otimes (B \otimes C)$, $\forall A \in \mathbb{C}^{m\times n}, B \in \mathbb{C}^{p\times q}, C \in \mathbb{C}^{r\times s}$.

2. *Let* $A \in \mathbb{C}^{m\times m}$ *and* $B \in \mathbb{C}^{n\times n}$. *Let* $\lambda \in \sigma(A)$ *with corresponding eigenvector* x, *and let* $\mu \in \sigma(B)$ *with corresponding eigenvector* y. *Then* $\lambda\mu$ *is an eigenvalue of* $A \otimes B$ *with corresponding eigenvector* $x \otimes y$. *Any eigenvalue of* $A \otimes B$ *arises as such a product of eigenvalues of* A *and* B.

3. *The product of two Kronecker products yields another Kronecker product:* $(A \otimes B)(C \otimes D) = AC \otimes BD$, $\forall A \in \mathbb{C}^{p\times q}, B \in \mathbb{C}^{r\times s}, C \in \mathbb{C}^{q\times k}, D \in \mathbb{C}^{s\times l}$.

4. *The product of two Kronecker products, containing multiple Kronecker products, yields another Kronecker product:* $(A_1 \otimes A_2 \otimes \cdots \otimes A_n)(B_1 \otimes B_2 \otimes \cdots \otimes B_n) = A_1B_1 \otimes A_2B_2 \otimes \cdots \otimes A_nB_n$, $\forall A_i \in \mathbb{C}^{p\times q}, B_i \in \mathbb{C}^{q\times k}, \forall i \in \{1, \ldots, n\}$.

Lemma 4.9
If A and B are Hermitian matrices with eigenvalues $a_1 \geq a_2 \geq \ldots \geq a_n$ and $b_1 \geq b_2 \geq \ldots \geq b_n$, and the eigenvalues of $C = A+B$ are $c_1 \geq c_2 \geq \ldots \geq c_n$, then $c_{i+j-1} \leq a_i + b_j$ and $c_{n-i-j} \geq a_{n-i} + b_{n-j}$, [Ful99].

4.1.2 Group theory

We will give a quick review of group theory to aid in understanding the stabilizer formalism, which we introduce in Section 4.4.

Definition 4.10 (Group)
A group is a set G along with a binary operation $\cdot$ that meets the following properties:

- Closed: $\forall a, b \in G, c = a \cdot b \in G$.
- Associative: $\forall a, b, c \in G, (a \cdot b) \cdot c = a \cdot (b \cdot c)$.
- Identity: $\exists e \in G$ such that e is unique and $\forall a \in G, e \cdot a = a \cdot e = a$.
- Inverse: $\forall a \in G, \exists b \in G$ such that b is unique and $a \cdot b = b \cdot a = e$.

Definition 4.11 (Subgroup)
Let G be a group with binary operation, $\cdot$. Let A be a subset of G, $A \subset G$. Then A is a subgroup of G if it forms a group under the same binary operation, $\cdot$.

Definition 4.12 (Abelian Group)
Let G be a group with binary operation, $\cdot$. G is abelian if the binary operation, $\cdot$, is commutative. Meaning, the order that elements appear in the group operation does not change the result. For example, let G be an abelian group, then $\forall a, b \in G$ if $a \cdot b = c$, then $b \cdot a = c$.

Definition 4.13 (Generating set of a Group)
Let G be a group and $S \subset G$. $\langle S \rangle$ is the subgroup generated by S. $\langle S \rangle$ is the smallest subgroup of G that contains every element of S. The elements of S are called generators and S is the generating set. Essentially, every element of $\langle S \rangle$ can be found as a product of elements from S and their inverses. If $\langle S \rangle = G$, S generates G.

4.2 Notation and Conventions

A quantum system is described by a vector $|\psi\rangle$ in a complex Hilbert space, $\mathcal{H}$. The state vector $|\psi\rangle$ is referred to as a pure quantum state and in this work we only consider finite dimensional Hilbert spaces. We mostly use Hilbert spaces of the form, $\mathcal{H} = \mathbb{C}^2$, and their tensor products, as we are working with qubits, which are two dimensional quantum states, and tensor products of qubits. Throughout this work, we will use the bra-ket notation.

4.2.1 The qubit

The fundamental unit of classical computing is the bit, which can be in the state 0 or 1. The qubit is the fundamental unit of quantum computing. The qubit can be in the state 0, or 1, or in a superpositon of those states, i.e., somewhere in between. This construction demonstrates a fundamental difference between quantum and classical computing. One of the goals of quantum computing is to find ways to take advantage of the mathematical properties endowed to a qubit, due to the possibility of superposition, that are not available when working with bits.
A qubit is a quantum state with a two-dimensional state space. Let $|0\rangle, |1\rangle \in \mathbb{C}^2$ form an orthonormal basis for that state space. The states $|0\rangle$ and $|1\rangle$ are the computational basis states for this state space. Where, the matrix representation for the computational basis states is given by, $|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$ and $|1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$. The state vector $|\psi\rangle \in \mathbb{C}^2$ is the quantum state vector that defines a qubit. Any $|\psi\rangle \in \mathbb{C}^2$ can be written as

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle\,, \tag{4.2}$$

where $\alpha, \beta \in \mathbb{C}$. However, $|\psi\rangle$ must be a unit vector, $\langle\psi|\psi\rangle = 1$, so $\alpha^2 + \beta^2 = 1$. This normalization condition is necessary for all quantum states, including multi-qubit states. In this example, α and β are the amplitudes for the states. Where, α^2 is the probability that the qubit is in state $|0\rangle$ and β^2 is the probability that the qubit is in state $|1\rangle$.

4.2.2 Multiple qubit states

The composite quantum system of two quantum states $|\psi\rangle$ and $|\phi\rangle$, belonging to Hilbert spaces $\mathcal{H}_\psi$ and $\mathcal{H}_\phi$, respectively, is contained in the Kronecker product of those Hilbert spaces, i.e., $\mathcal{H}_\psi \otimes \mathcal{H}_\phi$. This is how one creates mult-qubit quantum states.
Let $|\psi\rangle \in \mathbb{C}^{2^n} = (\mathbb{C}^2)^{\otimes n} = \mathbb{C}^2 \otimes \mathbb{C}^2 \otimes \cdots \otimes \mathbb{C}^2$. Then $|\psi\rangle$ is a quantum state on n qubits. An n-qubit quantum state will have 2^n computational basis states, and also have 2^n amplitudes. The same principle follows from the single qubit quantum state, $|\psi\rangle$ can be equal to any one of the computational basis states or be equal to a

superposition of these states or a subset of them. One defines the computational basis states for an n-qubit quantum state as follows: Let $x_i \in \{0,1\}, \forall i \in \{1,\dots,n\}$, then the computational basis states are all possibilities of the form $|x_1 x_2 \dots x_n\rangle \in \mathbb{C}^{2^n}$. The computational basis states can be found by taking the n-fold Kronecker product of the computational basis states for a single qubit, i.e.,

$$|x_1 x_2 \dots x_n\rangle = |x_1\rangle \otimes \dots \otimes |x_n\rangle . \tag{4.3}$$

The matrix representation for a n-qubit computational basis state is a vector of size 2^n where all entries are 0 except for one entry which is equal to 1. The index of the vector where the entry of 1 occurs is given by $x_n 2^0 + x_{n-1} 2^1 + \dots + x_1 2^{n-1}$, where the indexing begins with the 0th index. The n-qubit quantum state, $|\psi\rangle$, can be written as

$$|\psi\rangle = a_1 |00\dots00\rangle + a_2 |00\dots01\rangle + \dots + a_{2^n} |11\dots11\rangle . \tag{4.4}$$

for $a_i \in \mathbb{C}, \forall i \in \{1,\dots,2^n\}$. The amplitudes here satisfy the completeness relation, $\sum_{i=1}^{2^n} a_i^2 = 1$, due to the bilinearity of the Kronecker product.
In a quantum system, all qubits contribute to one quantum state vector. This is another key difference between classical computing and quantum computing; this phenomenon also leads to entanglement, which will be explained later.

4.2.3 Qubit operations

Unitary operators are used to describe the evolution of a quantum state. The building blocks of quantum circuits are quantum gates. The only constraint for a quantum gate is that it must be unitary, and any unitary matrix is a valid quantum gate. Since unitary matrices satisfy $U^\dagger U = \mathbb{I}$, unitary matrices are also normal and have a spectral decomposition. Unitary matrices also preserve orthogonality; in that, if a unitary matrix is applied to each element in an orthonormal basis, the resulting basis is still orthonormal. The same holds for a set of elements that are not orthonormal; after applying a unitary matrix, the resulting set will still not be orthonormal.

Single qubit operations

One example of a single qubit operation, and also a quantum gate and a unitary matrix, is the Hadamard gate, which is defined as

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} . \tag{4.5}$$

Applying the Hadamard gate to either computational basis state, $|0\rangle$ or $|1\rangle$, results in the equal superposition state, $|+\rangle$ or $|-\rangle$, respectively.

$$\begin{aligned} H|0\rangle &= \frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}\begin{bmatrix}1\\0\end{bmatrix} = \frac{1}{\sqrt{2}}\begin{bmatrix}1\\1\end{bmatrix} = \frac{|0\rangle + |1\rangle}{\sqrt{2}} = |+\rangle\,. \\ H|1\rangle &= \frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}\begin{bmatrix}0\\1\end{bmatrix} = \frac{1}{\sqrt{2}}\begin{bmatrix}1\\-1\end{bmatrix} = \frac{|0\rangle - |1\rangle}{\sqrt{2}} = |-\rangle\,. \end{aligned} \tag{4.6}$$

Here, $|+\rangle$ and $|-\rangle$ are the equal superposition states, which also form an orthonormal basis. Any $|\psi\rangle \in \mathbb{C}^2$ can be written as

$$|\psi\rangle = \alpha|+\rangle + \beta|-\rangle\,, \tag{4.7}$$

where $\alpha, \beta \in \mathbb{C}$. In general, by the definition of a basis, any $|\psi\rangle \in \mathbb{C}^{2^n}$ can be expressed as a linear combination of states that form an orthonormal basis in $\mathbb{C}^{2^n}$. Another example of single qubit operations are the Pauli matrices:

$$X = \begin{bmatrix}0 & 1\\ 1 & 0\end{bmatrix}, Y = \begin{bmatrix}0 & -i\\ i & 0\end{bmatrix}, Z = \begin{bmatrix}1 & 0\\ 0 & -1\end{bmatrix}. \tag{4.8}$$

The Pauli matrices are Hermitian and unitary. None of the Pauli matrices commute with each other, but they do all commute with themselves. X and Y anti-commute, as do X and Z. However, Y and Z do not anti-commute. Every operation on a single qubit quantum state can be expressed as a linear combination of the Pauli matrices and the identity matrix, $\mathbb{I}$.

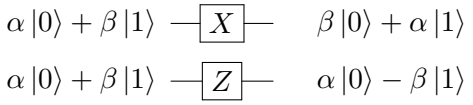


Figure 4.1: Two quantum circuits displaying the effect of quantum gates X and Z on a single qubit quantum state.

Example 4.14

The effect of applying the Pauli matrices, X or Z, to a quantum state can be seen in Figure 4.1. Where the input to both circuits is the quantum state $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle \in \mathbb{C}^2, \alpha, \beta \in \mathbb{C}$. In the circuit diagram, the quantum state to the left of the gates X and Z is the input to the gate, and the quantum state to the right of the gates is the output from the gates. As an example, the operation performed by the upper circuit can also be written as

$$X|\psi\rangle = X(\alpha|0\rangle + \beta|1\rangle) = \beta|0\rangle + \alpha|1\rangle\,. \tag{4.9}$$

The X gate is frequently referred to as the NOT gate.

Multiple qubit operations

The gates mentioned so far are all single qubit gates, i.e., they act only on one qubit, and in the examples so far we have only applied these gates to single qubit quantum states. It is also possible to have single qubit gates that act on a single qubit within a multiple qubit quantum state and multiple qubit gates that act on more than one qubit within a multiple qubit quantum state. Let $|\psi\rangle$ and $|\phi\rangle$ be two single qubit quantum states belonging to Hilbert spaces $\mathcal{H}_\psi$ and $\mathcal{H}_\phi$, respectively. Let $U_{|\psi\rangle}$ and $U_{|\phi\rangle}$ be two single qubit quantum gates acting on $|\psi\rangle$ and $|\phi\rangle$, respectively. Then, due to the properties of the Kronecker product,

$$U_{|\psi\rangle} |\psi\rangle \otimes U_{|\phi\rangle} |\phi\rangle = (U_{|\psi\rangle} \otimes U_{|\phi\rangle}) |\psi\phi\rangle . \tag{4.10}$$

This procedure holds for multi-qubit quantum states where $n > 2$ as well. Let $|\psi\rangle$ be an n-qubit quantum state. If one wants to apply unitary matrices $U_1, U_2, \ldots, U_n$ to qubits $1, 2, \ldots, n$, respectively, one applies $U_1 \otimes U_2 \otimes \cdots \otimes U_n$ to $|\psi\rangle$, i.e., $(U_1 \otimes U_2 \otimes \cdots \otimes U_n) |\psi\rangle$. The Pauli Group on n qubits is a set of unitary matrices that can perform n-qubit operations, where each gate within the full unitary operator is a single qubit gate.

Definition 4.15 (Pauli Group)
The Pauli group, PG_n, is defined on n qubits. For one qubit, the Pauli group consists of the Pauli matrices and the identity matrix, along with multiplicative factors ± 1, $\pm i$:

$$PG_1 = \{\pm\mathbb{I}, \pm i\mathbb{I}, \pm X, \pm iX, \pm Y, \pm iY, \pm Z, \pm iZ\}. \tag{4.11}$$

The general Pauli group on n qubits, PG_n, consists of all n-fold tensor products of Pauli matrices and the identity matrix, and one allows multiplicative factors of ± 1, $\pm i$.

$$PG_n = \{\pm 1, \pm i\} \cdot \{\mathbb{I}, X, Y, Z\}^{\otimes n}. \tag{4.12}$$

The Pauli Group, PG_n, forms a group with matrix multiplication as its operation.

Example 4.16
Let $|\psi\rangle = |000\rangle$ be a 3-qubit quantum state. If one wants to apply an X gate to the third qubit, one must apply $\mathbb{I} \otimes \mathbb{I} \otimes X$ to the state, i.e.,

$$(\mathbb{I} \otimes \mathbb{I} \otimes X) |000\rangle = |001\rangle .$$

Where, the unitary operator, $\mathbb{I} \otimes \mathbb{I} \otimes X$, is contained within PG_3.

So far, only single qubit gates have been introduced. The multiple qubit operations that have been introduced so far have all been separable. For U acting on a multiple qubit state, $U |\psi\phi\rangle$, where $|\psi\rangle$ and $|\phi\rangle$ can be single or multiple qubit quantum states and $|\psi\phi\rangle$ is the composite quantum state, if U can be written as $U = U_{|\psi\rangle} \otimes U_{|\phi\rangle}$ and it holds that

$$U_{|\psi\rangle} |\psi\rangle \otimes U_{|\phi\rangle} |\phi\rangle = (U_{|\psi\rangle} \otimes U_{|\phi\rangle}) |\psi\phi\rangle = U |\psi\phi\rangle , \tag{4.13}$$

then U is separable. But there are also multiple qubit gates, gates that act on multiple qubits within a multiple qubit state, and these are not seperable. One example of multiple qubit gates are the controlled unitary gates, often written controlled-U gates. This is a gate acting on two qubits; the first qubit is the control qubit and the second qubit is the target qubit. If the first qubit is in the state $|1\rangle$, then the unitary gate is applied to the second qubit, and if the first qubit is in the state $|0\rangle$, then $\mathbb{I}$ is applied to the second qubit. In either case, $\mathbb{I}$ is applied to the first qubit. Any single qubit unitary gate can be used as the U in a controlled-U gate. For example a $CNOT$ gate, controlled-X gate, is a controlled-U gate where the unitary is X and a CZ gate, controlled-Z gate, is a controlled-U gate where the unitary is Z. Any controlled-U gate can be written as

$$CU = |0\rangle\langle 0| \otimes \mathbb{I} + |1\rangle\langle 1| \otimes U. \tag{4.14}$$

Letting $U = X$ and using Equation (4.14), one can calculate the matrix representation of a CZ gate as

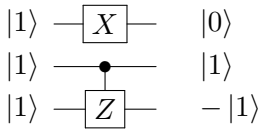
$$CZ = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \otimes \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} \otimes \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}. \tag{4.15}$$

Example 4.17
Let $|\psi\rangle = \alpha_1|00\rangle + \alpha_2|01\rangle + \alpha_3|10\rangle + \alpha_4|11\rangle \in \mathbb{C}^{2^2}, \alpha_i \in \mathbb{C}, \forall i \in \{1,2,3,4\}$. The effect of applying an X gate to any qubit is a flip, i.e., $X|0\rangle = |1\rangle$ and $X|1\rangle = |0\rangle$. Then, the result from applying a controlled-X gate to $|\psi\rangle$ is

$$CNOT|\psi\rangle = \alpha_1|00\rangle + \alpha_2|01\rangle + \alpha_3|11\rangle + \alpha_4|10\rangle.$$

If one wants to apply a multiple qubit gate to a subset of qubits within a multiple qubit state and apply single qubit gates or multiple qubit gates to the remaining qubits, the same principle as in Equation (4.10) applies.

Figure 4.2: A quantum circuit displaying the effect of a X and a CZ gate on a three qubit quantum state.



Example 4.18
Let $|\psi\rangle = |111\rangle$ be a 3-qubit quantum state. If one wants to apply an X gate to the first qubit and a CZ to the second and third qubits, one must apply $X \otimes CZ$ to the state, i.e.,

$$(X \otimes CZ)\,|111\rangle = -\,|011\rangle\,.$$

Where, the unitary operator, $CZ \otimes X$, is not contained within PG_3. The circuit for this operation is depicted in Figure 4.2.

There is an important theoretical result from classical computing that proves that any circuit on classical bits, a composition of some combination of OR, AND, XOR, $NAND$, and NOR gates, can be composed solely using $NAND$ gates. Thus, the $NAND$ gate is called a universal gate. The corresponding result from quantum computing finds that any multiple qubit logic gate can be composed solely from $CNOT$ and single qubit gates. Therefore, any quantum circuit can be composed using solely $CNOT$ and single qubit gates. The universal gate set in quantum computing is the set of all single qubit gates and the $CNOT$ gate.

4.2.4 Additional quantum properties

One advantage to quantum computation over classical computation is entanglement. Quantum hardware and quantum algorithms can take advantage of the quantum phenomenon known as entanglement, whereas classical algorithms cannot. We say that a state of a composite system having this property, that it can't be written as a product of states of its component systems, is an entangled state.

Definition 4.19 (Entanglement)
Let $|\psi\rangle$ be a pure quantum state in a composite system, $\mathcal{H}_A \otimes \mathcal{H}_B$. If $|\psi\rangle$ can be written as

$$|\psi\rangle = |\psi_A\rangle \otimes |\psi_B\rangle \tag{4.16}$$

for $|\psi_A\rangle \in \mathcal{H}_A$ and $|\psi_B\rangle \in \mathcal{H}_B$, then $|\psi\rangle$ is a product state. Quantum states that are not product states are entangled states.

A result of this definition is that, if a quantum state $|\psi\rangle$ begins as a product state, either in a computational basis state or in another basis, and one only applies single qubit gates to the state, the result will also be a product state. Introducing multiple qubit gates to a quantum circuit can generate entanglement.

Example 4.20 (Bell Basis)
A classic example of entangled states are the EPR pairs. These entangled states form

a basis in $\mathbb{C}^2 \otimes \mathbb{C}^2$ known as the Bell basis. Here, $\mathcal{H}_A = \mathcal{H}_B = \mathbb{C}^2$

$$\begin{aligned} |\psi^+\rangle &= \frac{1}{\sqrt{2}}(|01\rangle + |10\rangle). \\ |\psi^-\rangle &= \frac{1}{\sqrt{2}}(|01\rangle - |10\rangle). \\ |\phi^+\rangle &= \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle). \\ |\phi^-\rangle &= \frac{1}{\sqrt{2}}(|00\rangle - |11\rangle). \end{aligned} \tag{4.17}$$

Another difference between classical and quantum computing is the no-cloning theorem. For any classical bitstring there is a classical circuit that can copy that string, or state. However, for quantum computing, this is not always possible.

Theorem 4.21 (No-cloning theorem)
It is not possible to copy an unknown quantum state using unitary evolution. Let $|\psi\rangle \in \mathcal{H}_\psi$ and $|\delta\rangle \in \mathcal{H}_\delta$, where $|\delta\rangle$ is a pure quantum state. Then, there exists no unitary operator, U, such that

$$U(|\psi\rangle \otimes |\delta\rangle) = |\psi\rangle \otimes |\psi\rangle\,. \tag{4.18}$$

Proof. Suppose there exists a unitary U such that Equation (4.18) is satisfied. Let $|\psi\rangle, |\phi\rangle \in \mathcal{H}_\psi$. Then it follows that:

$$\begin{aligned} U(|\psi\rangle \otimes |\delta\rangle) &= |\psi\rangle \otimes |\psi\rangle \\ U(|\phi\rangle \otimes |\delta\rangle) &= |\phi\rangle \otimes |\phi\rangle\,. \end{aligned} \tag{4.19}$$

If we take the inner product of both equations, one obtains $\langle\psi|\phi\rangle = (\langle\psi|\phi\rangle)^2$. This implies $|\phi\rangle = |\psi\rangle$ or $|\phi\rangle$ and $|\psi\rangle$ are orthogonal. □

So, there exists no unitary operator that can copy any quantum state, but it is possible to copy a quantum state that is part of an orthonormal basis.

4.3 Measurement

Measuring a quantum state is done using a collection of measurement operators, $\{M_m\}$. The index m refers to the possible outcomes from the measurement. When measuring a quantum state, one can measure all qubits in the state or a subset of qubits in the state. After measurement, the qubits that have been measured collapse into one state. So, if all qubits of a quantum state are measured during the measurement, then the full state collapses into one of the possible measurement outcomes m and can no longer be in a

superposition. However, if a subset of qubits are measured, those qubits collapse into one state, which is one of the possible measurement outcomes, m, but the unmeasured qubits do not and could still be in a superposition.
Let $|\psi\rangle$ be a quantum state before being measured, then the probability the state is in outcome m is given by,

$$P(m) = \langle\psi| M_m^\dagger M_m |\psi\rangle . \tag{4.20}$$

The state of the system after measurement is given by,

$$\frac{M_m |\psi\rangle}{\sqrt{\langle\psi| M_m^\dagger M_m |\psi\rangle}}. \tag{4.21}$$

The set of measurement operators must satisfy the completeness relation,

$$\sum_m M_m^\dagger M_m = \mathbb{I}. \tag{4.22}$$

The completeness relation ensures that the sum of the probabilities of the possible outcomes is one.

4.3.1 Measuring in the computational basis

We will demonstrate the measurement of a single qubit in the computational basis for a quantum state.

Example 4.22
Let $|\psi\rangle \in \mathbb{C}^2$ and $|\psi\rangle = \alpha |0\rangle + \beta |1\rangle , \alpha, \beta \in \mathbb{C}$. Let j be the measurement outcome. Since we are measuring a single qubit, there are two outcomes, 0 and 1, with corresponding measurement operators, $M_0 = |0\rangle\langle 0|$ and $M_1 = |1\rangle\langle 1|$. If we measure $|\psi\rangle$, the probability of obtaining outcome 1 is

$$P(j=1) = \langle\psi| M_1^\dagger M_1 |\psi\rangle = \langle\psi| M_1 |\psi\rangle = \beta^2$$

For the sake of clarity, we will show the exact matrix algebra for this calculation.

$$|\psi\rangle = \alpha |0\rangle + \beta |1\rangle = \alpha \begin{bmatrix} 1 \\ 0 \end{bmatrix} + \beta \begin{bmatrix} 0 \\ 1 \end{bmatrix}.$$

$$M_1 = |1\rangle\langle 1| = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \begin{bmatrix} 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}.$$

$$\begin{aligned} P(j=1) &= \left(\alpha^* \begin{bmatrix} 1 & 0 \end{bmatrix} + \beta^* \begin{bmatrix} 0 & 1 \end{bmatrix}\right) \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} \left(\alpha \begin{bmatrix} 1 \\ 0 \end{bmatrix} + \beta \begin{bmatrix} 0 \\ 1 \end{bmatrix}\right) \\ &= \left(\alpha^* \begin{bmatrix} 1 & 0 \end{bmatrix} + \beta^* \begin{bmatrix} 0 & 1 \end{bmatrix}\right) \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} \left(\alpha \begin{bmatrix} 1 \\ 0 \end{bmatrix} + \beta \begin{bmatrix} 0 \\ 1 \end{bmatrix}\right) \\ &= \left(\alpha^* \begin{bmatrix} 0 & 0 \end{bmatrix} + \beta^* \begin{bmatrix} 0 & 1 \end{bmatrix}\right) \left(\alpha \begin{bmatrix} 1 \\ 0 \end{bmatrix} + \beta \begin{bmatrix} 0 \\ 1 \end{bmatrix}\right) \\ &= \beta^2. \end{aligned} \tag{4.23}$$

One can similarly find $P(j=0) = \alpha^2$. The state after measurement is given by

$$\begin{aligned} &\frac{M_1(\alpha |0\rangle + \beta |1\rangle)}{\sqrt{\beta^2}} \\ &= \frac{\begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} \alpha |0\rangle + \beta |1\rangle}{|\beta|} \\ &= |1\rangle \frac{\beta}{|\beta|} = |1\rangle . \end{aligned} \tag{4.24}$$

One can similarly find the state after measurement in the other case.

Next, we will demonstrate how to measure a subset of qubits in a multiple qubit state.

Example 4.23
We will demonstrate the measurement of a single qubit, within a two qubit state, in the computational basis. Let $|\psi\rangle \in \mathbb{C}^{2^2}$ and $|\psi\rangle = \alpha_1 |00\rangle + \alpha_2 |01\rangle + \alpha_3 |10\rangle + \alpha_4 |11\rangle , \alpha_i \in \mathbb{C}, \forall i \in \{1,2,3,4\}$. We choose to measure the first qubit, and let j be the measurement outcome of the first qubit. Then,

$$\begin{aligned} P(j=1) = P(|\psi\rangle = |11\rangle \vee |\psi\rangle = |10\rangle) = \alpha_3^2 + \alpha_4^2. \\ P(j=0) = P(|\psi\rangle = |00\rangle \vee |\psi\rangle = |01\rangle) = \alpha_1^2 + \alpha_2^2. \end{aligned} \tag{4.25}$$

Say we measure the first qubit and receive $j=1$. However, the second qubit may still be in a superposition and we do not know its value. Since we know $j=1$, we can elminate any options for $|\psi\rangle$ where the first qubit is $|0\rangle$, i.e.,

$$|\psi\rangle = \alpha_3 |10\rangle + \alpha_4 |11\rangle . \tag{4.26}$$

However, this expression is not normalized. After normalizing, we have the state $|\psi\rangle$ after measuring the first qubit,

$$|\psi\rangle = \frac{\alpha_3 |10\rangle + \alpha_4 |11\rangle}{\sqrt{\alpha_3^2 + \alpha_4^2}}. \tag{4.27}$$

4.3.2 Measuring in an orthonormal basis

Let $|\psi_i\rangle$ be a set of quantum states to be measured. If the states $|\psi_i\rangle$ are orthonormal, then a quantum measurement can always reliably distinguish these states. The procedure is as follows:

- Define measurement operators $M_i = |\psi_i\rangle \langle\psi_i|$ for each possible state ψ_i.
- Define a final measurement operator M_0 that is the positive square root of the operator $\mathbb{I} - \sum_{i\neq 0} |\psi_i\rangle \langle\psi_i|$. Then, the completeness relation is satisfied.
- Let the state $|\psi_i\rangle$ be measured using this set of operators, $\{M_i\}$. Then, the probability of measuring ψ_i when measuring the state $|\psi_i\rangle$ is

$$P(\psi_i) = \langle\psi_i| M_i^\dagger M_i |\psi_i\rangle = \langle\psi_i| M_i |\psi_i\rangle = \langle\psi_i|\psi_i\rangle \langle\psi_i|\psi_i\rangle = 1.$$

It is worth noting that, with the construction $M_i = |\psi_i\rangle \langle\psi_i|$, each M_i is Hermitian, and it will always hold that $M_i^\dagger M_i = M_i$ since

$$M_i^\dagger M_i = (|i\rangle \langle i|)^\dagger (|i\rangle \langle i|) = (\langle i|^* |i\rangle^*)(|i\rangle \langle i|) = |i\rangle \langle i| = M_i. \tag{4.28}$$

One can also use a quantum circuit to change the basis to be measured in. The quantum

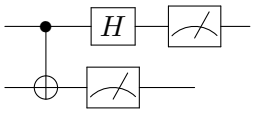


Figure 4.3: A quantum circuit displaying the necessary basis transformation to measure in the Bell basis instead of the computational basis.

software we use throughout this work is Qiskit, which measures in the computational basis. However, if we wanted to measure in the Bell basis instead of the computational basis, we can apply a basis transformation to the state and then measure in the computational basis, and this will be equivalent to measuring in the Bell basis. To measure in the Bell basis we need to add a Hadamard gate and a $CNOT$, controlled-X, gate to the quantum circuit and then measure in the computational basis. This can be seen in Figure 4.3. $CNOT(H \otimes I)$ is the measurement basis transformation from the computational basis to the Bell basis, which is the reverse ordering of the gates one would use to create the Bell basis from the computational basis, $(H \otimes I)CNOT$.

Theorem 4.24
Let $|\psi_i\rangle$ be a set of quantum states to be measured. If the states $|\psi_i\rangle$ are not orthonormal, then there is no quantum measurement that can distinguish these states with certainty.

Proof. This will be a proof by contradiction. Let $|\psi_1\rangle$ and $|\psi_2\rangle$ be two non-orthogonal quantum states. Suppose there exists a quantum measurement to distinguish these states with certainty. Then there exists measurement operators, M_1 and M_2, such that

$$P(\psi_1) = \langle\psi_1| M_1^\dagger M_1 |\psi_1\rangle = \langle\psi_1| M_1 |\psi_1\rangle = 1 \tag{4.29}$$

$$P(\psi_2) = \langle\psi_2| M_2^\dagger M_2 |\psi_2\rangle = \langle\psi_2| M_2 |\psi_2\rangle = 1. \tag{4.30}$$

Since the completeness relation must be satisfied for M_1 and M_2, we know that

$$\sum_m M_m^\dagger M_m = M_1^\dagger M_1 + M_2^\dagger M_2 = \mathbb{I}.$$

Then, we also know that

$$\sum_m \langle\psi_1| M_m^\dagger M_m |\psi_1\rangle = \langle\psi_1| M_1^\dagger M_1 |\psi_1\rangle + \langle\psi_1| M_2^\dagger M_2 |\psi_1\rangle = 1.$$

Since, $\langle\psi_1| M_1^\dagger M_1 |\psi_1\rangle = 1$, it must be true that $\langle\psi_1| M_2^\dagger M_2 |\psi_1\rangle = 0$, which implies that, $\sqrt{M_2^\dagger M_2}\,|\psi_1\rangle = 0$.
Let $|\psi_2\rangle = \alpha |\psi_1\rangle + \beta |\phi\rangle, \alpha, \beta \in \mathbb{C}$. Where, $|\phi\rangle$ is orthogonal to $|\psi_1\rangle$. Then, $\alpha^2 + \beta^2 = 1$ and $|\beta| < 1$ since $|\psi_1\rangle$ and $|\psi_2\rangle$ are not orthogonal.

$$\sqrt{M_2^\dagger M_2}\,|\psi_2\rangle = \sqrt{M_2^\dagger M_2}\alpha |\psi_1\rangle + \sqrt{M_2^\dagger M_2}\beta |\phi\rangle = \sqrt{M_2^\dagger M_2}\beta |\phi\rangle. \tag{4.31}$$

We also know that,

$$\begin{aligned}\langle\phi| M_2^\dagger M_2 |\phi\rangle &\leq \sum_m \langle\phi| M_m^\dagger M_m |\phi\rangle = \langle\phi| M_1^\dagger M_1 |\phi\rangle + \langle\phi| M_2^\dagger M_2 |\phi\rangle \\ &= \langle\phi| M_1^\dagger M_1 + M_2^\dagger M_2 |\phi\rangle = \langle\phi| \mathbb{I} |\phi\rangle = \langle\phi|\phi\rangle = 1.\end{aligned} \tag{4.32}$$

Using these statements, one proves the following:

$$\langle\psi_2| M_2^\dagger M_2 |\psi_2\rangle = |\beta^2| \langle\phi| M_2^\dagger M_2 |\phi\rangle \leq |\beta^2| < 1.$$

This implies a contradiction, given our assumption that the states can be distinguished with certainty, since we previously proved Equation (4.29). □

4.3.3 POVM

A Positive Operator-Valued Measurement (POVM) can be used to distinguish quantum states, without certainty, when they are not orthogonal. POVMs are especially useful when we are working with measuring a set of non-orthonormal quantum states and we want to know the probabilities of the measurement outcomes and are less concerned with the post-measurement state of the system.

Let a quantum system be in the state $|\psi\rangle$. Let there be measurement operators M_m for each possible measurement outcome m. One defines $E_m = M_m^\dagger M_m$. When a measurement is performed on the state, the probability of outcome m is given by,

$$P(m) = \langle\psi| M_m^\dagger M_m |\psi\rangle = \langle\psi| E_m |\psi\rangle .$$

For a POVM to be possible, the E_m operators must be Hermitian, positive, and satisfy the completeness relation.
When implementing a POVM, the circuit will require $|E_m|$ additional qubits, ancillary qubits, to perform the measurement.

4.4 Stabilizer formalism

One of the motivations for using the stabilizer formalism is that it is often easier to work with the stabilizer of a quantum state rather than the quantum state itself.

Definition 4.25 (Stabilizer Group)
Let S be an abelian subgroup of PG_n. Let V_S be a vector space on n qubits where

$$V_S = \{|\psi\rangle \in \mathbb{C}^{2^n} : s |\psi\rangle = |\psi\rangle , \forall s \in S\}.$$

S is the stabilizer group and its elements are stabilizer operators, or stabilizers. We say V_S is stabilized by S and S is the stabilizer for V_S. If $|V_S| = 1$, we call V_S a stabilizer state, otherwise it is called the stabilizer code space.

When the stabilizer group S is generated by a set, $\mathcal{G}_{V_S} = \{a_1, ..., a_m\}$, we write $S = \langle a_1, ..., a_m\rangle = \langle\mathcal{G}_{V_S}\rangle$. Throughout this work, we will write the generating set of a stabilizer group, S, which stabilizes V_S, as $\mathcal{G}_{V_S}$. We refer to the set $\mathcal{G}_{V_S}$ as the set of stabilizer generators. Generally, the set of stabilizer generators is much smaller than the stabilizer group, the set of stabilizer operators. And it is always true that $\mathcal{G}_{V_S} \subseteq S$. Let V_S be the vector space stabilized by S. Then we know for any $|\psi\rangle \in V_S$ and $s \in S$,

$$s |\psi\rangle = |\psi\rangle .$$

This also implies,

$$\langle\psi| s |\psi\rangle = \langle\psi|\psi\rangle = 1.$$

Since every stabilizer generator, $a \in \mathcal{G}_{V_S}$, is in the stabilizer group S, we also have

$$a |\psi\rangle = |\psi\rangle .$$

This is why we will often work with the generating set for the stabilizer group instead of the full group itself.

Example 4.26
Let S be a stabilizer group defined on two qubits where

$$S = \{\pm I, \pm Z\}.$$

In this case, V_S is equal to the trivial vector space. As the only solution to

$$-I\,|\psi\rangle = |\psi\rangle$$

is $|\psi\rangle = 0$.

There are two necessary and sufficient conditions for V_S to be nontrivial:

- $-I$ is not an element of S,
- the elements of S commute.

Theorem 4.27
Let V_S be a vector space stabilized by the stabilizer group, S. Let U be a unitary operation. Then, the vector space UV_S is stabilized by the group $USU^\dagger = \{UgU^\dagger | g \in S\}$.

Proof. Let $|\psi\rangle \in V_S$, then for any $g \in S$,

$$U\,|\psi\rangle = Ug\,|\psi\rangle = UgU^\dagger U\,|\psi\rangle\,. \tag{4.33}$$

Therefore, $U\,|\psi\rangle$ is stabilized by $UgU^\dagger$. Since, this holds true for any $|\psi\rangle \in V_S$, UV_S is stabilized by the group $USU^\dagger = \{UgU^\dagger | g \in S\}$. □

An important consequence of this theorem is that to determine the change in the stabilizer group one only needs to determine how the change affects the generators of the stabilizer. Since, if $S = \langle g_1, \ldots, g_l\rangle$, then $USU^\dagger = \langle Ug_1U^\dagger, \ldots, Ug_lU^\dagger\rangle$.

4.5 Graph states

Now we introduce graph states, which are fundamental to the algorithm we introduce in this work. We define a tour on a graph to be a Hamiltonian cycle, so a path where each vertex is visited exactly once and the path returns to its starting point. A subtour is a closed tour that does not visit every vertex of the graph. We will often refer to a graph state, where the corresponding graph is a tour, as a cycle graph state.

Definition 4.28 (Graph state)
Let $G(V, E)$ be an undirected graph, where V defines the vertices of the graph and E the edges. Let $n = |V|$, we will denote an edge between vertices i and j as $(ij) \in E$ and an edge is active if it exists in G and inactive if it does not. We denote an inactive edge as $\neg(ij)$ and an active edge as (ij). Every vertex is associated to one qubit and the unique graph state for this graph is defined as

$$|\psi_G\rangle = \prod_{e\in E} (CZ)_e\,|+\rangle^{\otimes n}\,. \tag{4.34}$$

Often we will write $|\psi\rangle$ in place of $|\psi_G\rangle$, when it is clear what graph is being referred to. We will also denote $E_{|\psi\rangle}$ and $V_{|\psi\rangle}$ as the edge set and vertex set, respectively, for the corresponding graph G of $|\psi\rangle$ or $|\psi_G\rangle$. One can also define a graph state using stabilizer formalism. Let us define $X_i, Z_j \in \mathbb{C}^{2^n \times 2^n}, \forall i, j \in \{1, \ldots, n\}$ as

$$X_i = \bigotimes_{a=1}^{i-1} \mathbb{I} \otimes X \otimes \bigotimes_{a=i+1}^{n} \mathbb{I}, \tag{4.35}$$

$$Z_j = \bigotimes_{a=1}^{j-1} \mathbb{I} \otimes Z \otimes \bigotimes_{a=j+1}^{n} \mathbb{I}. \tag{4.36}$$

The unique stabilizer group for a graph state, $|\psi_G\rangle$, is generated by the following set:

$$\mathcal{G}_{|\psi_G\rangle} = \left\{ X_i \prod_{a \in N(i)} Z_a | \forall i \in V \right\}. \tag{4.37}$$

Where $N(i)$ is the set of neighbors of vertex i, this is the vertex set corresponding to vertices that vertex i is attached to by an edge. Let $S_{|\psi_G\rangle}$ be the stabilizer group generated by $\mathcal{G}_{|\psi_G\rangle}$. Then, the unique graph state, $|\psi_G\rangle$, corresponding to this generating set, is the one that $\forall s \in S_{|\psi_G\rangle}$,

$$s\,|\psi_G\rangle = |\psi_G\rangle\,.$$

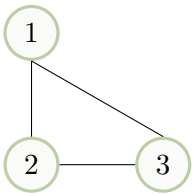


Figure 4.4: Graph of a tour on 3 vertices.

Example 4.29
Let G be the graph shown in Figure 4.4. The generating set for the graph state corresponding to this graph is

$$\mathcal{G}_{|\psi_G\rangle} = \{X_1 Z_2 Z_3, X_2 Z_1 Z_3, X_3 Z_2 Z_1\} \tag{4.38}$$

and the stabilizer group is

$$S_{|\psi_G\rangle} = \langle X_1 Z_2 Z_3, X_2 Z_1 Z_3, X_3 Z_2 Z_1 \rangle. \tag{4.39}$$

The graph state for G, $|\psi_G\rangle$, is

$$\begin{aligned}|\psi_G\rangle =&(CZ)_{12}(CZ)_{23}(CZ)_{31}\,|+\rangle^{\otimes 3}\\ =&(CZ)_{12}(CZ)_{23}(CZ)_{31}\cdot\frac{1}{\sqrt{8}}(|000\rangle+|001\rangle+|010\rangle+|100\rangle\\ &+|011\rangle+|110\rangle+|101\rangle+|111\rangle)\\ =&\frac{1}{\sqrt{8}}(|000\rangle+|001\rangle+|010\rangle+|100\rangle-|011\rangle-|110\rangle-|101\rangle-|111\rangle).\end{aligned} \tag{4.40}$$

Now, we will detail the procedure for determining a graph from its graph state when the graph state is written in the computational basis.

- Determine the sign of every computational basis state with exactly 1s. Let the 1s be located at indices i, j.
- If the sign of the computational basis state is positive, the edge between vertices i and j is inactive.
- If the sign of the computational basis state is negative, the edge between vertices i and j is active.

We introduce two lemmata that will be used often throughout this work for simplifying expressions. Often, we will omit citing the lemmata and it can be assumed they are used.

Lemma 4.30
Z_i and Z_j commute if $i = j$ or if $i \neq j$. X_i and X_j commute if $i = j$ or if $i \neq j$. X_i and Z_j commute if $i \neq j$ and anti-commute if $i = j$.

Proof. We will begin by proving the first two statements. The case when $i = j$ is clear since, $X_i \cdot X_i = X_i \cdot X_i = X_i^2$. The same is true for Z_i following the same argument. Now, we will prove the case when $i \neq j$. Without loss of generality, let $i < j$. Using

the fourth property from Lemma 4.8,

$$
\begin{aligned}
X_i \cdot X_j &= \left(\bigotimes_{a=1}^{i-1} \mathbb{I} \otimes X \otimes \bigotimes_{a=i+1}^{n} \mathbb{I}\right)\left(\bigotimes_{a=1}^{j-1} \mathbb{I} \otimes X \otimes \bigotimes_{a=j+1}^{n} \mathbb{I}\right) \\
&= \bigotimes_{a=1}^{i-1} \mathbb{II} \otimes X\mathbb{I} \otimes \bigotimes_{a=i+1}^{j-1} \mathbb{II} \otimes \mathbb{I}X \otimes \bigotimes_{a=j+1}^{n} \mathbb{II} \\
&= \bigotimes_{a=1}^{i-1} \mathbb{I} \otimes X \otimes \bigotimes_{a=i+1}^{j-1} \mathbb{I} \otimes X \otimes \bigotimes_{a=j+1}^{n} \mathbb{I}. \\
X_j \cdot X_i &= \left(\bigotimes_{a=1}^{j-1} \mathbb{I} \otimes X \otimes \bigotimes_{a=j+1}^{n} \mathbb{I}\right)\left(\bigotimes_{a=1}^{i-1} \mathbb{I} \otimes X \otimes \bigotimes_{a=i+1}^{n} \mathbb{I}\right) \\
&= \bigotimes_{a=1}^{i-1} \mathbb{II} \otimes \mathbb{I}X \otimes \bigotimes_{a=i+1}^{j-1} \mathbb{II} \otimes X\mathbb{I} \otimes \bigotimes_{a=j+1}^{n} \mathbb{II} \\
&= \bigotimes_{a=1}^{i-1} \mathbb{I} \otimes X \otimes \bigotimes_{a=i+1}^{j-1} \mathbb{I} \otimes X \otimes \bigotimes_{a=j+1}^{n} \mathbb{I}.
\end{aligned}
\tag{4.41}
$$

Therefore, $X_i \cdot X_j = X_j \cdot X_i$, and the operators commute. The same is true for Z_i following the same argument. Now, we will prove the last statement. The proof that X_i and Z_j commute if $i \neq j$ follows from the last argument if one allows Z_j to replace X_j. Let $i = j$ and we will show that X_i and Z_j anti-commute. We will use the fact that the individual Pauli operators, X and Z, anti-commute, i.e., $XZ = -ZX$. Using

the fourth property from Lemma 4.8,

$$\begin{aligned}
X_i \cdot Z_i &= \left(\bigotimes_{a=1}^{i-1} \mathbb{I} \otimes X \otimes \bigotimes_{a=i+1}^{n} \mathbb{I}\right)\left(\bigotimes_{a=1}^{i-1} \mathbb{I} \otimes Z \otimes \bigotimes_{a=i+1}^{n} \mathbb{I}\right) \\
&= \bigotimes_{a=1}^{i-1} \mathbb{II} \otimes XZ \otimes \bigotimes_{a=i+1}^{n} \mathbb{II} \\
&= \bigotimes_{a=1}^{i-1} \mathbb{I} \otimes XZ \otimes \bigotimes_{a=i+1}^{n} \mathbb{I}. \\
Z_i \cdot X_i &= \left(\bigotimes_{a=1}^{i-1} \mathbb{I} \otimes Z \otimes \bigotimes_{a=i+1}^{n} \mathbb{I}\right)\left(\bigotimes_{a=1}^{i-1} \mathbb{I} \otimes X \otimes \bigotimes_{a=i+1}^{n} \mathbb{I}\right) \\
&= \bigotimes_{a=1}^{i-1} \mathbb{II} \otimes ZX \otimes \bigotimes_{a=i+1}^{n} \mathbb{II} \\
&= \bigotimes_{a=1}^{i-1} \mathbb{I} \otimes ZX \otimes \bigotimes_{a=i+1}^{n} \mathbb{I} \\
&= -\bigotimes_{a=1}^{i-1} \mathbb{I} \otimes XZ \otimes \bigotimes_{a=i+1}^{n} \mathbb{I}.
\end{aligned} \tag{4.42}$$

Therefore, $X_i \cdot Z_i = -Z_i \cdot X_i$ and the operators anti-commute. □

Lemma 4.31
X_i and Z_j are Hermitian and unitary for all $i, j \in \{1, \ldots, n\}$.

Proof. The proof makes use of the following properties. Let A and B be square matrices.

- $(A \otimes B)^\dagger = A^\dagger \otimes B^\dagger$.
- $(A \otimes B)^{-1} = A^{-1} \otimes B^{-1}$.
- The Pauli matrices and $\mathbb{I}$ are Hermitian and unitary.
- If a matrix is Hermitian and unitary, then $A = A^\dagger = A^{-1}$.
- If one revereses the order of a product term over commuting operators, that term simplifies to its original ordering.

We begin by proving that X_i and Z_i are Hermitian. One takes the conjugate transpose of X_i and simplifies,

$$\begin{aligned} X_i^\dagger &= \bigotimes_{a=1}^{i-1} \mathbb{I}^\dagger \otimes X^\dagger \otimes \bigotimes_{a=i+1}^{n} \mathbb{I}^\dagger \\ X_i^\dagger &= \bigotimes_{a=1}^{i-1} \mathbb{I} \otimes X \otimes \bigotimes_{a=i+1}^{n} \mathbb{I} \\ X_i^\dagger &= X_i. \end{aligned} \tag{4.43}$$

Applying the same procedure to Z_i, one finds $Z_i^\dagger = Z_i$. Now, we will prove X_i and Z_i are unitary. One takes the inverse of both sides of X_i and simplifies,

$$\begin{aligned} X_i^{-1} &= \bigotimes_{a=1}^{i-1} \mathbb{I}^{-1} \otimes X^{-1} \otimes \bigotimes_{a=i+1}^{n} \mathbb{I}^{-1} \\ X_i^{-1} &= \bigotimes_{a=1}^{i-1} \mathbb{I} \otimes X \otimes \bigotimes_{a=i+1}^{n} \mathbb{I} \\ X_i^{-1} &= X_i. \end{aligned} \tag{4.44}$$

Applying the same procedure to Z_i, one finds $Z_i^{-1} = Z_i$. □

We introduce two more examples to help clarify how graph states relate to their stabilizer group and its generating set.

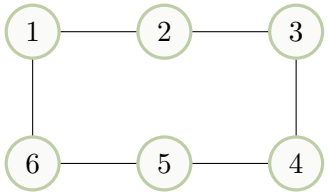


Figure 4.5: Graph of a tour on six vetices.

Example 4.32
Let G be the graph shown in Figure 4.5. The generating set for the graph state corresponding to this graph is

$$\mathcal{G}_{|\psi_G\rangle} = \{X_1Z_6Z_2, X_2Z_1Z_3, X_3Z_2Z_4, X_4Z_3Z_5, X_5Z_4Z_6, X_6Z_5Z_1\} \tag{4.45}$$

and the stabilizer group is

$$S_{|\psi_G\rangle} = \langle X_1Z_6Z_2, X_2Z_1Z_3, X_3Z_2Z_4, X_4Z_3Z_5, X_5Z_4Z_6, X_6Z_5Z_1\rangle. \tag{4.46}$$

$X_1X_2X_3X_4X_5X_6 \in S_{|\psi_G\rangle}$. One finds $X_1X_2X_3X_4X_5X_6$ as a stabilizer operator by taking the product of all the elements of the generating set. We simplify the terms using Lemmata (4.31) and (4.30),

$$\begin{aligned} &X_1Z_6Z_2 \cdot X_2Z_1Z_3 \cdot X_3Z_2Z_4 \cdot X_4Z_3Z_5 \cdot X_5Z_4Z_6 \cdot X_6Z_5Z_1 \\ &X_1Z_1Z_1 \cdot Z_2X_2Z_2 \cdot Z_3X_3Z_3 \cdot Z_4X_4Z_4 \cdot Z_5X_5Z_5 \cdot Z_6Z_6X_6 \\ &X_1 \cdot Z_2X_2Z_2 \cdot Z_3X_3Z_3 \cdot Z_4X_4Z_4 \cdot Z_5X_5Z_5 \cdot Z_6 \\ &(-1)^4X_1 \cdot Z_2Z_2X_2 \cdot Z_3Z_3X_3 \cdot Z_4Z_4X_4 \cdot Z_5Z_5X_5 \cdot Z_6 \\ &X_1 \cdot X_2 \cdot X_3 \cdot X_4 \cdot X_5 \cdot Z_6. \end{aligned} \tag{4.47}$$

However, $X_1X_2X_3, X_4X_5X_6 \notin S_{|\psi_G\rangle}$.

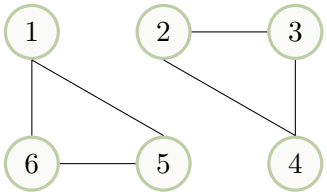


Figure 4.6: Graph on six vertices with two subtours.

Example 4.33
Let G be the graph shown in Figure 4.6. The generating set for the graph state corresponding to this graph is

$$\mathcal{G}_{|\psi_G\rangle} = \{X_1Z_6Z_5, X_5Z_1Z_6, X_6Z_5Z_1, X_2Z_4Z_3, X_3Z_2Z_4, X_4Z_3Z_2\} \tag{4.48}$$

and the stabilizer group is

$$S_{|\psi_G\rangle} = \langle X_1Z_6Z_5, X_5Z_1Z_6, X_6Z_5Z_1, X_2Z_4Z_3, X_3Z_2Z_4, X_4Z_3Z_2\rangle. \tag{4.49}$$

Where $X_4X_5X_6, X_1X_2X_3, X_1X_2X_3X_4X_5X_6 \in S_{|\psi_G\rangle}$.

4.5.1 Additional properties

Here we prove two properties about graph states and their stabilizers that will be utilized later in this work. The proofs make use of Lemma (4.31) and the properties listed in the proof there, along with the fact that, for two square matrices, A and B, $(AB)^\dagger = B^\dagger A^\dagger$.

Lemma 4.34
Let $|\psi\rangle \in \mathbb{C}^{2^n}$ be a graph state corresponding to a graph, $G = (V, E)$. Let $A \in \mathcal{G}_{|\psi\rangle}$, $A = X_i \prod_{j\in J} Z_j \in \mathbb{C}^{2^n \times 2^n}$. Where J is the set of vertices that vertex i has an edge to, and $0 \leq |J| \leq n-1$ and $i \notin J$. Then A is Hermitian and unitary.

Proof. First, we will prove that A is Hermitian. We simplify using Lemma (4.31),

$$\begin{aligned} A^\dagger &= \left(X_i \prod_{j \in J} Z_j \right)^\dagger \\ &= \prod_{j \in J} Z_j^\dagger \cdot X_i^\dagger \\ &= \prod_{j \in J} Z_j X_i \\ &= X_i \prod_{j \in J} Z_j \\ &= A. \end{aligned} \tag{4.50}$$

Next, we will demonstrate that A is also unitary,

$$\begin{aligned} A^{-1} &= \left(X_i \prod_{j \in J} Z_j \right)^{-1} \\ &= \prod_{j \in J} Z_j^{-1} \cdot X_i^{-1} \\ &= \prod_{j \in J} Z_j X_i \\ &= X_i \prod_{j \in J} Z_j \\ &= A. \end{aligned} \tag{4.51}$$

□

Lemma 4.35
Let $|\psi\rangle \in \mathbb{C}^{2^n}$ be a graph state corresponding to a graph, $G = (V, E)$. Let $U \in S_{|\psi\rangle}$. Then, U is Hermitian and unitary.

Proof. Let $g_i \in \mathcal{G}_{|\psi\rangle}$. We know from the Lemma 4.34 that g_i is Hermitian and unitary, $g_i^{-1} = g_i^\dagger = g_i$. Then, since $S_{|\psi\rangle}$ is an abelian group, U can be written as,

$$\begin{aligned} U &= \prod_{j \in J} g_j \prod_{i \in I} g_i^{-1} \\ U &= \prod_{j \in J} g_j \prod_{i \in I} g_i \\ U &= \prod_{j \in J \cup I} g_j. \end{aligned} \tag{4.52}$$

Where J is the index set of generators used in U and I is the index set of generator inverses used in U. Since $g_i \in S_{|\psi\rangle}$ and $S_{|\psi\rangle}$ is an abelian group, all of the terms in the product term in U commute. The conjugate transpose of U can be found as,

$$\begin{aligned} U^\dagger &= \left(\prod_{j \in J \cup I} g_j \right)^\dagger \\ U^\dagger &= \prod_{j \in J \cup I} g_j^\dagger \\ U^\dagger &= \prod_{j \in J \cup I} g_j \\ U^\dagger &= U. \end{aligned} \tag{4.53}$$

The inverse of U can be found as,

$$\begin{aligned} U^{-1} &= \left(\prod_{j \in J \cup I} g_j \right)^{-1} \\ U^{-1} &= \prod_{j \in J \cup I} g_j^{-1} \\ U^{-1} &= \prod_{j \in J \cup I} g_j \\ U^{-1} &= U. \end{aligned} \tag{4.54}$$

Since $U^{-1} = U^\dagger = U$, U is Hermitian and unitary. □

Chapter 5

Quantum algorithms

This chapter is dedicated to explaining quantum algorithms that have been developed and can be used on the TSP. We provide a special focus on algorithms that utilize an alternating operator method, as they relate to the algorithm we introduce in Chapter 6. All of the algorithms in this chapter can be applied to the TSP, and most can be applied to any QUBO problem. Most of these algorithms are considered hybrid quantum-classical approaches as they combine quantum amplitude amplification and classical tuning of the variational parameters. At the end of this chapter, we benchmark several of these algorithms on TSP instances.

5.1 Problem encoding

Throughout this work we refer to the QUBO for the TSP, detailed in Section 3.1.1, as the enumeration encoding. What this means is that, for an instance of the TSP, the binary variables in the QUBO formulation each correspond to a bit in a bitstring. The objective function for that QUBO is what is used as the objective function in the quantum algorithms that follow that solve QUBO problems. For a TSP instance of size n, the possible bitstrings are all possibilities, x, where $x \in \{0,1\}^{n^2}$. These possible bitstrings are what the algorithms optimize over. Every possible bitstring for a problem instance can be returned by the algorithms. However, only certain bitstrings will correspond to feasible, and also optimal, solutions. The number of bits in the bitstring, in the case of the TSP this is n^2, will be the number of qubits the algorithms need to use to solve that problem instance.

Example 5.1
Let there be a TSP instance of size $n = 4$. Then there will be $n^2 = 16$ variables, $x_{v,j}$, where v represents the vertex and j represents its order, timestep position, in a prospective cycle. For this example, $v, j \in \{0, \ldots, 3\}$. The bitstrings corresponding to this problem will be of the form

$$[x_{0,0}x_{0,1}x_{0,2}x_{0,3}x_{1,0}x_{1,1}x_{1,2}x_{1,3}x_{2,0}x_{2,1}x_{2,2}x_{2,3}x_{3,0}x_{3,1}x_{3,2}x_{3,3}].$$

The feasible solution $x_{0,0}, x_{1,1}, x_{2,2}, x_{3,3} = 1$ and the remaining $x_{v,j} = 0$ corresponds to

the bitstring

$$[1000010000100001].$$

The infeasible solution $x_{0,0}, x_{0,1}, x_{2,2}, x_{3,3} = 1$ and the remaining $x_{v,j} = 0$ corresponds to the bitstring

$$[1100000000100001].$$

The computational basis states of the quantum state correspond to the possible bitstrings. There are other encodings for the TSP besides the one we use in this work. For more information, see [GKZ20].

5.2 QAOA

The quantum approximate optimization algorithm (QAOA), introduced in [FGG14], is a quantum algorithm that returns approximate solutions for combinatorial optimization problems. QAOA depends on the number of iterations, p, of the unitary operators involved with the algorithm. The quality of the approximate solution produced by the QAOA improves as p increases.

5.2.1 Procedure

Let there be an instance of an optimization problem on bitstrings of length n which we want to find an approximate solution for. For a given instance of the optimization problem, the algorithm is specified by two Hamiltonians, H_C and H_B, and $2p$ real parameters, $\gamma_1, ..., \gamma_p, \beta_1, \ldots, \beta_p$, where $\gamma \in [0, 2\pi]$ and $\beta \in [0, \pi]$. The parameters, γ_i and β_i, indicate the time for which the chosen Hamiltonian is applied.

- The phase Hamiltonian, H_C, represents the cost of the optimization problem. Let $f(x)$ be the objective function that returns the cost of a given solution to our optimization problem. When applied to an n-qubit computational basis state, the phase Hamiltonian should return the cost of the bitstring instance encoded by the quantum state, such that,

  $$H_C |x\rangle = f(x) |x\rangle .$$

  From this Hamiltonian, one creates the following unitary:

  $$U(H_C, \gamma) = e^{-i\gamma H_C}.$$

- The mixing Hamiltonian, H_B, is used to move among the different bitstrings.

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

where X_i is the Pauli operator X acting on the ith qubit. Since the X operator performs a bitflip, this Hamiltonian allows the algorithm to change the quantum state and explore other possible bitstrings. From this Hamiltonian, one creates the following unitary:

$$U(H_B, \beta) = e^{-i\beta H_B}.$$

- The initial state is chosen to be the uniform superposition of all possible computational basis states, which represent all possible solutions that could be returned by the algorithm.

  $$|s\rangle = |+\rangle^{\otimes n}.$$

  Note that, $|s\rangle$ is the ground state, the lowest energy eigenstate, of H_B.

- One creates a parameterized quantum state by applying the two Hamiltonian unitary operators in alternating order to the initial state. The number of applications of the two unitaries is determined by p.

  $$|\gamma, \beta\rangle = U(H_B, \beta_p)U(H_C, \gamma_p)...U(H_B, \beta_1)U(H_C, \gamma_1)\,|s\rangle \tag{5.1}$$

- The state is measured in the computational basis and returns a possible solution, $|x\rangle$, with probability $|\langle x|\beta, \gamma\rangle|^2$. One repeats the state preparation and measurement to yield the expected cost of the objective function over the returned solutions as

  $$\langle f\rangle = \langle\beta, \gamma|\, H_C\, |\beta, \gamma\rangle.$$

  This expectation can be estimated using the returned solutions.

- To perform QAOA, one repeats the last two steps as part of a classical optimization loop. In this case, one optimizes the parameters, also referred to as angles, γ and β, with respect to the objective, $\langle f\rangle$.

- The sample with the lowest cost is returned as the solution.

The goal of the algorithm is to find optimal parameters γ and β, such that the quantum state $|\gamma, \beta\rangle$, seen in Equation (5.1), encodes an optimal, or near optimal, solution to the problem. An essential component of the QAOA is the choice of initial parameters or the initial parameter setting strategy; this remains an active area of research [GS17].

5.2.2 Convergence

It has been shown that for $p \to \infty$, the QAOA will always return the optimal solution of the QUBO problem it is solving. For an intuition of the proof, we should first make an observation about adiabatic quantum computing (AQC), for more information see [Far+00]. AQC relies on the adiabatic theorem to evolve a quantum state with a

time-dependent Hamiltonian. The idea in AQC is to initialize a quantum system in a ground state of the initial Hamiltonian, that is easy to prepare, and use adiabatic evolution to slowly evolve that state to the ground state of the problem Hamiltonian. The ground state of the problem Hamiltonian encodes the optimal solution of the problem. If the system is evolved slowly enough, the system will remain in the ground state and be in the ground state of the problem Hamiltonian at the conclusion. AQC is guaranteed to converge to the optimal solution if the evolution happens slowly enough. One could see the QAOA as evolving the state with the Hamiltonians H_B and H_C and approximating the adiabatic evolution that occurs in AQC. Assuming the optimal optimization of the parameters, in the limit of $p \to \infty$ this approximation becomes perfect.
However, in AQC the goal is to obtain the ground state of the problem Hamiltonian. In the QAOA the goal is to obtain a state that is similar enough to the ground state, such that it produces a good approximate solution. The procedure is also much different. AQC involves slowly transitioning between two Hamiltonians; in QAOA we repeatedly alternate the two Hamiltonians.

5.3 Warm-starting QAOA

The basic idea behind warm-starting an optimization problem is to use the optimal solution of a simplified or relaxed version of the problem to provide initial values for the problem you want to optimize. In [EMW21], the concept of warm-starting is applied to the QAOA. We will refer to warm-starting the QAOA as WS-QAOA. For WS-QAOA we relax the QUBO problem to a continuous version and use the solution of the continuous version as the initial state for QAOA, instead of the equal superposition state. For warm-starting QAOA, the mixer must also be replaced. However, the rest of the algorithm follows the same procedure as for the QAOA.

5.3.1 Relaxed QUBO

Recall Equation (3.3), if Q is positive semidefinite, the trivial continuous relaxation of the QUBO problem is

$$x^* = \arg\min_{x \in [0,1]^n} x^\intercal Q x. \tag{5.2}$$

This is a convex quadratic program (QP) and the optimal solution, x^*, can be easily found using a classical solver. However, if Q is not positive semidefinite, a different procedure must be used. We apply the procedure from [PRW95] to obtain another continuous relaxation, known as a semidefinite programming (SDP) problem. Let $\mathcal{S}^{n \times n}$ be the set of $n \times n$ symmetric matrices, e be a vector of 1s of length n, and $Y \succcurlyeq 0$ means that Y must be positive semidefinite. The SDP continuous relaxation of

a QUBO problem is given by

$$\begin{aligned} Y^* &= \arg\max_{Y \in \mathcal{S}^{n\times n}} \operatorname{tr}(QY), \\ \operatorname{diag}(Y) &= e, \\ Y &\succeq 0. \end{aligned} \tag{5.3}$$

The optimal solution, Y^*, to the SDP problem can be found using a classical solver. Given the optimal solution to the SDP problem, Y^*, there are several approaches to generating the solutions to the corresponding QUBO problem, x^*. For more information see [GW95], [BE07], and [BV97]. Once one has the optimal solution, x^*, to the continuous version of the QUBO problem, the QP or the SDP version, one is ready to build the circuit for the WS-QAOA.

5.3.2 Initial state

Let x_i^* denote the ith index of the optimal solution, x^*, to the continuous relaxation of the QUBO problem. The initial state to be used in the WS-QAOA is

$$|s\rangle = \bigotimes_{i=1}^{n} R_Y(\theta_i)\,|0\rangle\,, \tag{5.4}$$

where $R_Y(\theta_i)$ is a rotation around the Y-axis of qubit i with angle $\theta_i = 2\arcsin\sqrt{x_i^*}$.

5.3.3 Mixer Hamiltonian

We also need to replace the mixer Hamiltonian from the QAOA, $H_M = \sum_{i=1}^{n} X_i$, with $H_M = \sum_{i=1}^{n} H_{M,i}$, where

$$H_{M,i} = \begin{bmatrix} 2x_i^* - 1 & -2\sqrt{x_i^*(1-x_i^*)} \\ -2\sqrt{x_i^*(1-x_i^*)} & 1 - 2x_i^* \end{bmatrix} \tag{5.5}$$

Note that $|s\rangle$ is the ground state, i.e., the lowest energy eigenstate of H_M.

5.3.4 Convergence

The same convergence guarantees for p from the QAOA only hold for the WS-QAOA if a regularization technique is implemented, or if all $x_i^* \in (0, 1)$. This is due to a reachability issue. If $x_i^* = 0$ or $x_i^* = 1$, then qubit i would be initialized in state $|0\rangle$ or $|1\rangle$, respectively. If the phase Hamiltonian contains only Z_iZ_j and $\mathbb{I}$ operators, then the qubit will remain in its initial state throughout the QAOA optimization. If the continuous and discrete optimal solutions do not coincide for that qubit, the optimal

solution will never be reached. To solve this issue, the authors of [EMW21] introduce a regularization technique that allows for the WS-QAOA to have the same convergence guarantees of the QAOA, regardless of the values for x_i^*. We did not use the version of the WS-QAOA with the regularization technique in our coding implementation of WS-QAOA. However, we will still have the convergence guarantees of traditional QAOA because all the instances we used in the benchmarking have $x_i^* \in (0, 1)$.

5.4 AOA

In [Had+19], an extension of the QAOA algorithm is introduced, called the quantum alternating operator ansatz. However, to avoid confusion, we will refer to this algorithm as the alternating operator ansatz (AOA). The AOA is a generalization of QAOA that does not restrict to unitary operators corresponding to time evolution under a Hamiltonian. However, the operators still utilize one parameter, as in the QAOA. For certain problems, the mixer from QAOA does not only mix within the subspace of feasible solutions. This happens when the problem has hard constraints and the feasible solution subspace is smaller than the domain of all bitstrings. This is one advantage to the AOA. By allowing for a greater selection of unitary operators to be used in the alternating operators, it is possible to construct a mixer operator that mixes solely within the subspace of feasible solutions. This ensures that AOA always returns a feasible solution.

5.4.1 Procedure

Let there be an instance of an optimization problem on bitstrings of length n. Here, the optimization problem is defined by the domain, F, which is the set of feasible bitstrings, and the objective function to be optimized, $f : F \to \mathbb{R}$. For a given instance of the optimization problem, the algorithm is specified by two families of unitary operators, the phase-separation operators, U_C, and the mixing operators, U_B, and $2p$ real parameters, $\gamma_1, ..., \gamma_p, \beta_1, ..., \beta_p$, where $\gamma \in [0, 2\pi]$ and $\beta \in [0, \pi]$.

- The phase-separation operator, U_C, represents the cost of the optimization problem. Let $f(x)$ be the objective function that returns the cost of a given solution to our optimization problem. When applied to an n-qubit computational basis state, the phase Hamiltonian, H_C, should return the cost of the bitstring instance encoded by the quantum state, such that,

$$H_C |x\rangle = f(x) |x\rangle .$$

  From this Hamiltonian, one creates the following unitary:

$$U_C(\gamma) = e^{-i\gamma H_C}.$$

The family of phase-separation operators must be diagonal in the computational basis.

- The mixing operator, U_B, is used to move among the bitstrings corresponding to feasible solutions for the instance. U_B depends on the domain of the problem it is being used for. It is problem-specific and there is not a general form. U_B is dependent on the hard constraints of the optimization problem, as these are what determine the feasibility of a solution. There are two design criteria for the mixing operator for a given problem.
  - Stay in the feasible subspace: For all values of the parameter β, U_B should transform a feasible state to another feasible state. U_B should never transform a feasible state to an infeasible state.
  - Mixes among all possible feasible states: U_B should be able to reach all possible feasible states, given enough applications. This means for any two feasible states, there should be a parameter value, β, and a number of iterations of U_B, m, such that the corresponding mixer, $U_B^m(\beta)$, can transition the first feasible state to the second.
- The main requirement for the initial state in AOA is that it can be built by a constant depth circuit from the $|0, \ldots, 0\rangle$ state. Often, the initial state is chosen to be a single instance of a feasible solution for the problem, as opposed to a superposition of feasible solutions.
- One creates a parameterized quantum state by applying the two unitary operators in alternating order to the intial state. The number of applications of the two unitaries is determined by p.

$$|\gamma, \beta\rangle = U_B(\beta_p)U_C(\gamma_p)...U_B(\beta_1)U_C(\gamma_1)\,|s\rangle$$

- The state is measured in the computational basis and returns a possible solution, $|x\rangle$, with probability $|\langle x|\beta, \gamma\rangle|^2$. One repeats the state preparation and measurement to yield the expected cost of the objective function over the returned solutions as

$$\langle f\rangle = \langle\beta, \gamma|\, H_C\,|\beta, \gamma\rangle\,.$$

  This expectation can be estimated using the returned solutions.
- To perform the AOA, one repeats the last two steps as part of a classical optimization loop. In this case, one optimizes the parameters, γ and β, with respect to the objective, $\langle f\rangle$.
- The sample with the lowest cost is returned as the solution.

5.4.2 Convergence

In the AOA, the family of mixing operators used does not, in general, correspond to time evolution under a fixed Hamiltonian, as in the QAOA. For this reason, the AOA does not necessarily have the convergence guarantees of the QAOA. It instead depends on the mixer operator, which depends on the problem the AOA is being applied to.

5.5 Additional quantum algorithms for the TSP

There are many other quantum algorithms that can be applied to the TSP, besides the three detailed above. Below we separate the sections into algorithms that follow an alternating operator ansatz and algorithms that do not.

5.5.1 Additional alternating operator algorithms

In [Ben+21] the quantum walk-based optimization algorithm (QWOA) is applied to the CVRP problem. The authors found the algorithm produced high-quality solutions, which were more optimal than classical random sampling of equivalent computational effort. In QWOA the mixing operator performs a quantum walk, quantum walks are the quantum version of classical random walks, among the feasible solutions to the problem. QWOA could be seen as a variation of the AOA, in that the mixing operator restricts the ansatz to only search among feasible solutions. Similarly to the AOA, it may have the convergence guarantee of the QAOA, but it depends on the mixing operator, which depends on the problem it is being applied to. The QWOA could also be applied to the TSP. In [Sla+21], they found that the QWOA produces more effective convergence to high-quality solutions than the QAOA using fewer iterations. There are many variations of the QAOA that are quite promising. In [Pat+21], the authors propose a variational quantum algorithm that uses classical Markov chain Monte Carlo (MCMC) techniques to provably converge to global minima. In their work, they apply the MCMC method to the variational quantum eigensolver (VQE), but it could also be applied to the QAOA. In [Her+21], the authors propose a multi-angle ansatz for QAOA, the multi-angle QAOA (ma-QAOA), that reduces circuit depth and improves the approximation ratio by increasing the number of classical parameters. In [GKZ20], the authors propose a space-efficient variant of the QAOA that greatly reduces the number of qubits needed to run the QAOA at the cost of requiring deeper variational circuits. They apply their method to the TSP, although they claim the approach can be generalized for other problems where the standard enumeration encoding is inefficient. In this context, inefficient means there is duplicity in the bitstrings for the solutions they correspond to in the optimization problem, which is true for the enumeration encoding for the TSP. The recursive quantum approximate optimization algorithm (RQAOA), [Bra+22], has also shown promising results, but has

yet to be modified to work for the TSP problem. In [Rua+20], the authors introduce a variation of the QAOA for TSP where the problem constraints are encoded into the mixing Hamiltonian rather than the phase Hamiltonian. They also decrease the search space using a mapping of edges to qubits. They claim their approach can obtain a higher probability for the optimal solution for the TSP, using half the number of qubits, compared to the traditional QAOA for the TSP approach. In [BE20], the Grover mixer QAOA (GM-QAOA) is introduced. GM-QAOA uses Grover-like selective phase shift mixing operators. GM-QAOA can be applied to any QUBO problem for which it is possible to efficiently prepare an equal superposition of all feasible solutions. It is designed to particularly benefit constraint optimization problems, problems where not all possible bitstring solutions correspond to feasible solutions. They claim, for certain applications, that GM-QAOA outperforms existing QAOA approaches. Future work can be to implement and benchmark these additional QAOA variations on the TSP.

5.5.2 Further additional algorithms

There are additional quantum algorithms for solving the TSP, which are not alternating operator algorithms. Quantum annealing is a metaheuristic algorithm [HS15], which is based on AQC and utilizes quantum fluctuations that can be used to solve QUBO problems. However, the sucess of quantum annealing depends significantly on the mapping of the problem to the annealer itself [Hei+17]. The variational quantum eigensolver (VQE) [MSY20] and its enhancements [WHB19] are quite promising. The VQE is similar to the QAOA, but there is much more freedom in choosing the ansatz for the parameterized circuit used in the VQE than for that used in the QAOA. VQE is not limited to the alternating operator ansatz as in the QAOA. The QAOA can be seen as a specialized form of VQE. In [Sri+18], they introduce a quantum algorithm that utilizes phase estimation to solve the TSP. [Ban+12] proposes a quantum heuristic algorithm to solve the TSP by generalizing Grover's search. This quantum heuristic algorithm gives a quadratic speedup over the corresponding classical heuristic algorithm. In [MLM17], they introduce a hybrid quantum-classical algorithm that provides a quadratic speedup when applied to bounded-degree graphs.

5.6 Benchmarking the TSP

Now, we will benchmark QAOA, AOA, and WS-QAOA, on TSP instances. The implementation of these algorithms was done using the Python package Qiskit and the algorithms were run on a simulator of a quantum computer. For our benchmarking we chose six instances of the TSP, all of size $n = 4$. Where, each possible shape for the TSP, of which there are three, corresponds to the optimal solution in two of the instances. We ran each algorithm using the initial points specified by the individual algorithms. And we ran each algorithm using the same 20 unique sets of randomly

generated points for the intial points for the parameters in the algorithms. This was done for $p \in \{1, \ldots, 20\}$.

5.6.1 Peformance measures

We use three performance measures to evaluate the quantum algorithms in this work. Let X_{opt} be the bitstring corresponding to an optimal solution for an instance of the TSP, let X be the bitstring returned by the algorithm, and let f be the objective function for TSP, from the QUBO detailed in Section 3.1.1, that returns the objective value for any bitstring. Since the TSP is a minimization problem, $f(X_{opt}) \leq f(X)$ for all possible bitstrings X. The first performance measure is the approximation ratio, defined as

$$R = \frac{f(X_{opt})}{f(X)}. \tag{5.6}$$

If X is an optimal solution to the instance, then $R = 1$, otherwise $R \in (0, 1)$. The closer R is to 1 the better the performance of the algorithm is. One advantage of using the QUBO objective value instead of tour cost in the approximation ratio is that it can evaluate the quality of infeasible solutions; whereas, using the tour cost we cannot evaluate infeasible solutions.
The second performance measure is the feasibility ratio. We call the feasibility ratio F, and

$$F = \frac{\text{Number of feasible solutions}}{\text{Total number of solutions}}. \tag{5.7}$$

The third performance measure is the approximation ratio when only considering feasible solutions, which we call A. Here, the calculation is the same as in Equation (5.6), except we use the cost of the tour as the objective function, f.

5.6.2 Results

Figure 5.1 displays the plot of Equation (5.6), R, against p for the three algorithms. Figure 5.2 displays the plot of Equation (5.7), F, against p for the three algorithms. Figure 5.3 displays the plot of the approximation ratio of feasible solutions, A, against p for the three algorithms. One can seen from Figures 5.1, 5.2, and 5.3 that the AOA has the greatest success among the algorithms. However, this is most likely due to the fact that it is guaranteed to return a feasible solution, whereas, the QAOA and the WS-QAOA are not. The AOA does exhibit very nice convergence to the optimal solution, seen in Figures 5.1 and 5.3, which was not necessarily expected. Keep in mind, that for the AOA, the values of R and A will coincide as the AOA has a constant feasibility ratio, i.e., $F = 1$ for all p.
Both the WS-QAOA and the QAOA seem to converge to $R \approx .6$, $F \approx .3$, $A \approx .9$. However, the convergence of A is less clear. We had expected WS-QAOA to outperform

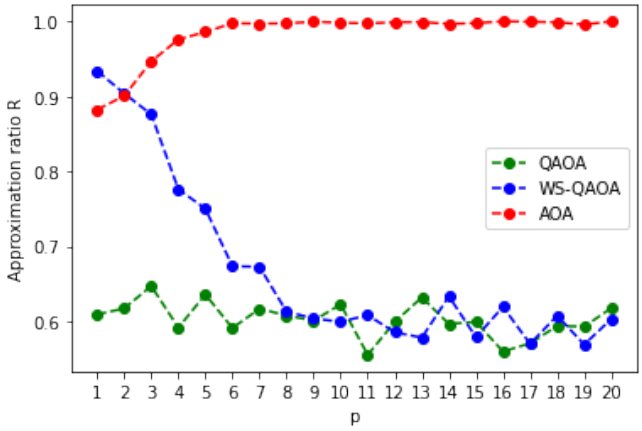


Figure 5.1: The approximation ratio of all solutions, R, plotted against p for the QAOA, the WS-QAOA, and the AOA.

QAOA. So these results were surprising, especially considering that the WS-QAOA starts at much higher values for R, F, and A than the QAOA, and then decreases, with increasing p, to maintain around the same values for R, F, and A. This could just be the behavior of QAOA and WS-QAOA on TSP instances of this size. It might be possible that very high values of p are required to get better solutions from the QAOA and the WS-QAOA. Future work could be to continue this benchmarking with higher p to further explore the performance of these algorithms on the TSP.

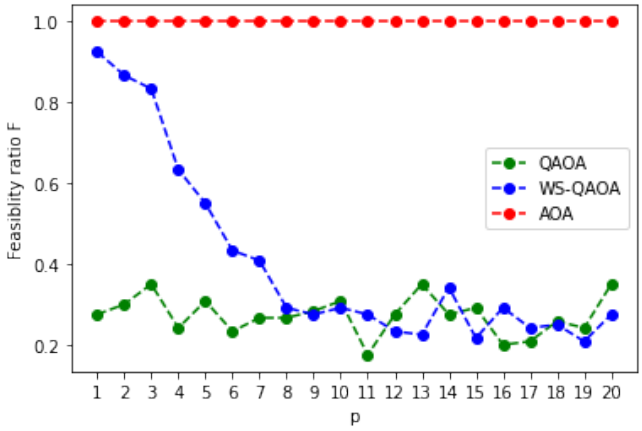


Figure 5.2: The feasibility ratio, F, of all solutions plotted against p for the QAOA, the WS-QAOA, and the AOA.

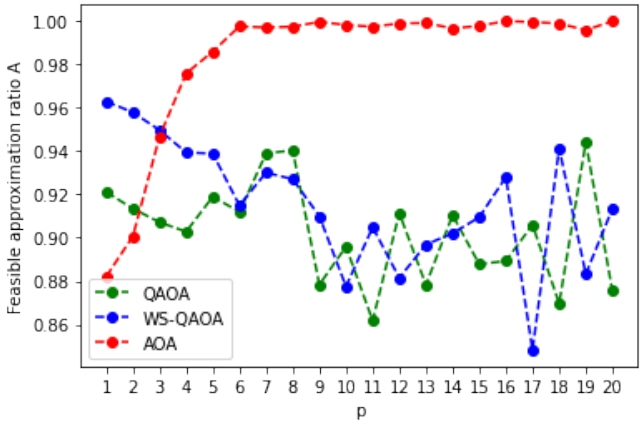


Figure 5.3: The approximation ratio of feasible solutions, A, plotted against p for the QAOA, the WS-QAOA, and the AOA.

Chapter 6

Graph state alternating operator ansatz

In this chapter, we introduce our own quantum algorithm for TSP. We call this the Graph State Alternating Operator Ansatz, or GSAOA.
Most quantum algorithms for TSP use the typical enumeration encoding, where the tour is specified by the list of vertices in the order they are visited. We believe there may be advantages to using a graph state representation instead of the enumeration representation, seen in Section 5.1, since TSP is a graph problem. The goal of this chapter is to explore ways graph states could be used in a quantum algorithm for TSP and to develop that algorithm. Our hope is that there are advantages to using graph states that can be exploited by a quantum algorithm. For example, maybe there are ways graph states can take advantage of the geometry of the graph, in a way that the enumeration encoding cannot.
The main idea behind the algorithm is to use the AOA from [Had+19] but with graph states instead of the typical enumeration encoding. The algorithm would consist of a phase Hamiltonian, which encodes the cost function, and a mixing Hamiltonian, which moves among feasible solutions for the problem. The original plan was for the mixing Hamiltonian to be of the same type as in the AOA, where the mixer constrains the ansatz to only move among feasible solutions. In this case, that would mean a mixer that only allowed cycle graph states to be considered instead of all graph states. So far, we have been unable to create a mixer Hamiltonian for graph states that only moves among cycle graph states. Instead, the mixer method for the algorithm works as a penalty. Instead of only moving among cycle graph states with a mixer Hamiltonian, we "mix" by penalizing graph states which are not cycles. For the algorithm, our phase Hamiltonian will have two parts: one that encodes the objective function and one that penalizes graph states which are not cycles. Then, the mixer Hamiltonian is a general one for graph states that moves among all graph states by toggling edges with controlled-Z gates. Enforcing that the considered graph states and the final graph state returned by the algorithm are cycle graph states is done using a penalty system in the phase Hamiltonian instead of in the mixer Hamiltonian.
This is different than the AOA, where they enforce the feasibility constraints for the problem in the mixer Hamiltonian and only move among feasible solutions. Future work on this algorithm could include looking for methods to include the cycle constraints in

a mixer Hamiltonian, separate from the phase Hamiltonian, similar to the AOA.

6.1 Procedure

Let there be an TSP on n vertices. Because graph states are being used instead of enumeration encoding, the graph state will be defined on n qubits. This is opposed to the n^2 qubits required to encode TSP for a quantum algorithm when using enumeration encoding. Here, the optimization problem is defined by the domain, F, which is the set of all possible graph states, and the objective function to be optimized, $f : F \to \mathbb{R}$, which is the cost of the TSP tour. One solves the TSP instance by minimizing the objective function. We also hope that the graph state returned by the algorithm is a cycle graph state, this is the hard feasibility constraint for TSP. For a given instance of TSP, the algorithm is specified by two Hamiltonians, H_P and H_B, and $2p$ parameters, $\gamma_1, \ldots, \gamma_p, \beta_1, \ldots, \beta_p$, where $\gamma \in [0, 2\pi]$ and $\beta \in [0, \pi]$.

- The phase Hamiltonian, H_P, represents the cost of the optimization problem and penalizes graph states which are not cycle graph states. H_C, the cost Hamiltonian, will be the Hamiltonian that represents the objective function that returns the cost of a given solution to the TSP. When applied to the current graph state, the cost Hamiltonian will return the cost of all active edges, when the active edges are connected to a vertex of degree two. In the case that the current graph state is a cycle graph state, the cost Hamiltonian will return the cost of the tour.

  $$H_C |s\rangle = f(s) |s\rangle .$$

  More details on the cost Hamiltonian can be found in Section 6.3. H_D will be the penalty Hamiltonian that penalizes graph states which are not cycle graph states. This ensures that the algorithm considers, and returns, cycle graph states instead of non-cycle graph states. More details on the penalty Hamiltonian can be found in Section 6.4. Let $H_P = H_C + H_D$. From this Hamiltonian, one creates the following unitary operator,

  $$U(H_P, \gamma) = e^{-i\gamma H_P}. \tag{6.1}$$

- The mixing Hamiltonian, H_B, is used to move among all possible graph states. To change from one graph state to another one must change which edges are active. To do this, we must apply controlled-Z gates to the graph state. The mixer Hamiltonian applies controlled-Z gates to every possible edge. More details on the mixing Hamiltonian can be found in Section 6.2.

  $$H_B = \sum_{i=1}^{n} \sum_{j=i+1}^{n} CZ_{i,j} \tag{6.2}$$

From the Hamiltonian, one creates the following unitary operator,

$$U(H_B, \beta) = e^{-i\beta H_B}. \tag{6.3}$$

- The main requirement for the initial state in the AOA is that it can be built by a constant depth circuit from the $|0, \ldots, 0\rangle$ state. There are multiple options for the intial state. It could be an empty graph state, a graph state with every possible edge active, or a single instance of a feasible solution to the problem. Since we are following the method in the AOA, we will start with the third option.

- One creates a parameterized quantum state by applying the two Hamiltonian unitary operators in alternating order to the initial state. The number of applications of the two unitary operators is determined by p.

$$|\gamma, \beta\rangle = U(H_B, \beta_p)U(H_P, \gamma_p)\ldots U(H_B, \beta_1)U(H_P, \gamma_1)\,|s\rangle \tag{6.4}$$

- The state is measured and returns a possible solution, $|x\rangle$, with probability $|\langle x|\beta, \gamma\rangle|^2$. One repeats the state preparation and measurement to yield the expected cost of the objective function over the returned solutions as

$$\langle f\rangle = \langle\beta, \gamma|\, H_P\,|\beta, \gamma\rangle\,. \tag{6.5}$$

  This expectation can be estimated using the returned solutions.

- To perform the AOA, one repeats the last two steps as part of a classical optimization loop. In this case, one optimizes the parameters, γ and β, with respect to the objective, f.

- The sample with the lowest cost is returned as the solution.

6.2 Mixing Hamiltonian

There are other options for the mixer besides Equation (6.2). For example, there could be two mixer Hamiltonians that are alternated, one which toggles half of the edges and the other which toggles the other half. Other variations of this would be feasible as well. Which approach is best still needs to be experimentally tested.
Also, in some cases, H_B could be applied directly to the graph state instead of using $U(H_B, \beta)$. However, H_B is not necessarily unitary while $U(H_B, \beta)$ will always be unitary. One controlled-Z gate is unitary, but the sum of multiple controlled-Z gates is not necessarily unitary anymore. Whereas, for H_B, we have that e^{-iH_B} is unitary.

Using $U(H_B, \beta)$ will have the same effect of toggling edges on the graph state it is applied to as using H_B would. Using the matrix exponential equation, one finds

$$\begin{aligned} U(H_B, \beta) |\psi\rangle &= e^{-i\beta H_B} |\psi\rangle \\ &= \sum_{k=0}^{\infty} \frac{(-i\beta)^k H_B^k}{k!} |\psi\rangle \\ &= |\psi\rangle - i\beta H_B |\psi\rangle - \frac{\beta^2 H_B^2}{2} |\psi\rangle + \frac{i\beta^3 H_B^3}{6} |\psi\rangle \dots \end{aligned} \tag{6.6}$$

With this, one can argue that applying $e^{-i\beta H_B}$ to the state results in a superposition. Here one of the summands is just H_B applied to $|\psi\rangle$. The summands with a higher power of H_B can be neglected, since their coefficients become arbitrarily small, $\frac{1}{k!}$.

6.3 Cost Hamiltonian

The cost Hamiltonian needs to return the cost of the tour defined in a graph state when applied to the graph state.

$$H_C |s\rangle = f(s) |s\rangle . \tag{6.7}$$

We form the following cost Hamiltonian, where w_{ab} is the weight assigned to the edge between vertices a and b, and p_1 is a weighting constant.

$$H_C = p_1 \sum_{i \in V} \sum_{j,k \in N(i)} X_i Z_j Z_k \frac{1}{2}(w_{ik} + w_{ij}) \tag{6.8}$$

This is used for the objective encoding part of the phase Hamiltonian, H_P. When H_C is applied to a graph state it will evaluate to the sum of the cost of all edges, $(ij), (ik)$, where $X_i Z_j Z_k$ is a generator for the stabilizer group for that graph state. This means, when H_C is applied to the graph state, it will evaluate to sum of the cost of all active edges, whenever the active edges are connected to a vertex of degree two. If the graph state is a cycle graph state, H_C will evaluate to the cost of the tour. By including the cost of the tour in the phase Hamiltonian, the algorithm will minimize the tour cost while searching for the optimum among feasible solutions. It is important to note that H_C is used to satisfy an optimality constraint and not a feasibility constraint. So, we need to ensure it is considered less important than the feasibility constraints using weighting constants, i.e., p_1 is weighted significantly less than the weighting constants for H_D. Let G be a fully connected graph. Let $C(G)$ be the set of all graph states corresponding to Hamiltonian cycles for the graph. Let $|C\rangle$ be the graph state corresponding to cycle $C \in C(G)$. Let $S_{|C\rangle} = \langle \mathcal{G}_{|C\rangle} \rangle$, where $\mathcal{G}_{|C\rangle}$ is the set of generators

for the stabillier group, $S_{|C\rangle}$. Equation (6.8) can also be written as below, in terms of the generators.

$$H_C = \sum_{C \in C(G)} \sum_{A \in \mathcal{G}_{|C\rangle}} A \cdot w_A. \tag{6.9}$$

6.3.1 Construction

The structure of the cost Hamiltonian is motivated by Theorem 4.27. A consequence of this theorem is that to compute the change in the stabilizer one only needs to compute how it affects the generators of the stabilizer.
Let $|\psi\rangle \in \mathbb{C}^{2^n}$ be a graph state corresponding to a graph, $G = (V, E)$. When H_C is applied to $|\psi\rangle$,

$$H_C |\psi\rangle = p_1 \left(\sum_{i \in V} \sum_{j,k \in N(i)} \frac{1}{2}(w_{ik} + w_{ij}) \langle\psi| X_i Z_j Z_k |\psi\rangle \right). \tag{6.10}$$

Let $A = X_i Z_j Z_k \in \mathbb{C}^{2^n \times 2^n}$, where $i \neq j \neq k \neq i$. For H_C to work as needed, one needs $\langle\psi| A |\psi\rangle = 0$ if $A \notin \mathcal{G}_{|\psi\rangle}$ and $\langle\psi| A |\psi\rangle = 1$ if $A \in \mathcal{G}_{|\psi\rangle}$. For this, note that if $A \in \mathcal{G}_{|\psi\rangle} \subseteq S_{|\psi\rangle}$, then

$$\langle\psi| X_i Z_j Z_k |\psi\rangle = \langle\psi|\psi\rangle = 1.$$

One needs $|\psi\rangle$ and $A|\psi\rangle$ to be orthogonal, when $A \notin \mathcal{G}_{|\psi\rangle}$. Then, when A, a generator of a cycle state, but not a generator for this graph state, $|\psi\rangle$, is applied to $|\psi\rangle$, $\langle\psi| A |\psi\rangle = 0$. The terms in the cost Hamiltonian that do not correspond to edges in the graph state, $|\psi\rangle$, will be eliminated and not contribute to the cost. Two quantum states are orthogonal when their inner product is zero. States $|\psi\rangle$ and $A|\psi\rangle$ will be orthogonal when,

$$\langle\psi| A |\psi\rangle = 0 \iff \langle\psi| A |\psi\rangle = -\langle\psi| A |\psi\rangle .$$

Lemma 6.1
Let $|\psi\rangle \in \mathbb{C}^{2^n}$ be a graph state corresponding to a graph, $G = (V, E)$. Let $A = X_i Z_j Z_k \in \mathbb{C}^{2^n \times 2^n}$, where $i \neq j \neq k \neq i$. If there exists $U \in S_{|\psi\rangle}$ and $V \in S_{A|\psi\rangle} = A S_{|\psi\rangle} A^\dagger$ such that $U = -V$, then $\langle\psi| A |\psi\rangle = 0$.

Proof. We will use the following properties in our proof.

- Let $|\phi\rangle, |\psi\rangle \in \mathbb{C}^{2^n}, A \in \mathbb{C}^{2^n \times 2^n}$, then $\langle\psi|A\phi\rangle = \langle A^\dagger\psi|\phi\rangle$.
- Let $A \in S_{|\psi\rangle}$, then $A|\psi\rangle = |\psi\rangle$.

Then,

$$\langle\psi| A |\psi\rangle = \langle\psi| VA |\psi\rangle = \langle\psi| -UA |\psi\rangle = \langle -U^\dagger\psi| A |\psi\rangle = -\langle U\psi| A |\psi\rangle = -\langle\psi| A |\psi\rangle .$$

Where the second to last equality is due to the fact that U is Hermitian, which we know from Lemma 4.35. □

It remains to determine if there exists $U \in S_{|\psi\rangle}, V \in AS_{|\psi\rangle}A^\dagger$ such that, $U = -V$
The following examples should help with understanding the construction. Let, $U \in S_{|\psi\rangle}$, $A = X_iZ_jZ_k \notin \mathcal{G}_{|\psi\rangle}$, and $V \in AS_{|\psi\rangle}A^\dagger$. A is a generator of a cycle graph state, but not a generator for $S_{|\psi\rangle}$.

Example 6.2
Let $U = X_1Z_2Z_3 \cdot X_2Z_1Z_4 = X_1Z_1Z_2X_2Z_3Z_4$, $A = X_1Z_5Z_6$. Then

$$\begin{aligned} V &= AUA^\dagger \\ &= X_1Z_5Z_6 \cdot X_1Z_1Z_2X_2Z_3Z_4 \cdot Z_6Z_5X_1 \\ &= X_1X_1Z_1X_1 \cdot Z_2X_2Z_3Z_4 \\ &= Z_1X_1Z_2X_2Z_3Z_4 \\ &= -U. \end{aligned} \tag{6.11}$$

Therefore, $\langle\psi| A |\psi\rangle = 0$.

Example 6.3
Let $U = X_1Z_2Z_3 \cdot X_2Z_1Z_4 = X_1Z_1Z_2X_2Z_3Z_4$, $A = X_1Z_2Z_6$. Then,

$$\begin{aligned} V &= AUA^\dagger \\ &= X_1Z_2Z_6 \cdot X_1Z_1Z_2X_2Z_3Z_4 \cdot Z_6Z_2X_1 \\ &= X_1X_1Z_1X_1 \cdot Z_2Z_2X_2Z_2 \cdot Z_3Z_4 \\ &= Z_1X_1X_2Z_2Z_3Z_4 \\ &= U. \end{aligned} \tag{6.12}$$

So, this choice of U did not work. However, we just have to determine if there exists $U \in S_{|\psi\rangle}, V \in AS_{|\psi\rangle}A^\dagger$ such that, $U = -V$. This means we can try a different selection of U and see if for that selection $V = AUA^\dagger = -U$. Also, we do not have to set V such that $V = AUA^\dagger$, V must be in $AS_{|\psi\rangle}A^\dagger$. For example, let $U_1, U_2 \in S_{|\psi\rangle}$ and $U_1 \neq U_2$, we can also prove orthogonality by showing $V = AU_1A^\dagger = -U_2$. For this example, we can still prove orthgonality using the same U in both.

Now, let $U = X_1Z_2Z_3 \cdot X_2Z_1Z_4 \cdot X_3Z_1Z_6 = X_1Z_2X_2Z_3X_3Z_4Z_6$, $A = X_1Z_2Z_6$. Then,

$$\begin{aligned} V &= AUA^\dagger \\ &= X_1Z_2Z_6 \cdot X_1Z_2X_2Z_3X_3Z_4Z_6 \cdot Z_6Z_2X_1 \\ &= X_1X_2Z_2Z_3X_3Z_4Z_6 \\ &= -X_1Z_2X_2Z_3X_3Z_4Z_6 \\ &= -U. \end{aligned} \tag{6.13}$$

Therefore, $\langle\psi| A |\psi\rangle = 0$.

Lemma 6.4
Let $|\psi\rangle \in \mathbb{C}^{2^n}$ be a graph state corresponding to a graph, $G = (V, E)$. Let $A = X_iZ_jZ_k \in \mathbb{C}^{2^n \times 2^n}$, where $i \neq j \neq k \neq i$. If there exists $U \in S_{|\psi\rangle}$ such that $UA = -AU$, then $\langle\psi| A |\psi\rangle = 0$.

Proof. We know from Lemma 4.34 that A is unitary, therefore $AA^\dagger = \mathbb{I}$. Let $AU = -UA$, and let $V = AUA^\dagger$. $V \in S_{A|\psi\rangle} = AS_{|\psi\rangle}A^\dagger$. Then $V = AUA^\dagger = -UAA^\dagger = -U$. If $AU = -UA$ then $V = -U$ is also true. And thus, according to Lemma 6.1, $\langle\psi| A |\psi\rangle = 0$. □

Theorem 6.5
Let $|\psi\rangle \in \mathbb{C}^{2^n}$ be a graph state corresponding to a graph, $G = (V, E)$. Let $A = X_iZ_jZ_k \in \mathbb{C}^{2^n \times 2^n}$, where $i \neq j \neq k \neq i$. But, $A \notin \mathcal{G}_{|\psi\rangle}$. Then, $\langle\psi| A |\psi\rangle = 0$.

Proof. One needs to show that a $U \in S_{|\psi\rangle}$ can always be constructed such that $AU = -UA$. Then, according to Lemma 6.4, $\langle\psi| A |\psi\rangle = 0$ for $|\psi\rangle$, a graph state, and A, a generator of a cycle graph state, but not a generator for $S_{|\psi\rangle}$.
Since, $A \notin \mathcal{G}_{|\psi\rangle}$, we know that either (ij) or (ik) is not an edge in $|\psi\rangle$. Without loss of generality, let $(ij) \notin E_{|\psi\rangle}$. Let $U = X_j \prod_{l\in L} Z_l$, where $L \subset \{1, \ldots, n\}$. L will be the set of vertices that vertex j is connected to by an edge. $U \in \mathcal{G}_{|\psi\rangle} \subseteq S_{|\psi\rangle}$. $j \notin L$, since a vertex cannot have an edge to itself in a graph state. $i \notin L$, since $(ij) \notin E_{|\psi\rangle}$. We will use Lemma (4.30) in this proof.
A and U anti-commute:

$$\begin{aligned} AU &= X_iZ_jZ_kX_j \prod_{l\in L} Z_l \\ &= -X_jX_iZ_jZ_k \prod_{l\in L} Z_l \\ &= -X_j \prod_{l\in L} Z_l \cdot X_iZ_jZ_k \\ &= -UA. \end{aligned} \tag{6.14}$$

Where, the second to last equality is due to the fact that $i \notin L$. Therefore, $\langle\psi| A |\psi\rangle = 0$. □

When $X_iZ_jZ_k$ is applied to the graph state, if vertex i has active edges $(ij), (ik)$, then the term will evaluate to half of the sum of the cost of those edges. However, each edge will be counted twice, which is the reason for the factor of $\frac{1}{2}$ in the Hamiltonian. For example, if (ij) is active it will contribute to the cost when the generator for vertex i is evaluated and when the generator for vertex j is evaluated. If $X_iZ_jZ_k$ is not a generator for the graph state, if vertex i does not have $(ij), (ik)$ both as active edges, the term will evaluate to zero and not contribute to the cost.

6.4 Penalty Hamiltonian

One needs to satisfy the hard feasibility constraint of TSP, that the graph state be a cycle graph state and thus the corresponding graph a tour.

6.4.1 Mixing and Graph States

One moves between graph states by 'toggling' edges with controlled-Z gates. One can easily enforce a mixer that stays in the subspace of graph state space. This will be a mixer of controlled-Z gates to move among graph states, as seen in Equation (6.2). But, it is much more difficult to create a mixer that will move among the space of cycle graph states, a mixer that stays in the feasible solution subspace. This difficulty is due to working with graph states instead of enumeration coding. With an enumeration encoding, there are several ways to enforce a mixer that moves among feasible cycles [Had+19]. But when working with graph states, the existing controlled-Z gates on the graph state determine the current configuration, so it is difficult to move among cycle states unless you know the current configuration. If you know the current configuration, one can easily specify the edge toggles necessary to move to all other feasible cycles from that graph state. The issue with alternating operator ansatz, is that the graph state will change throughout the process, so it is difficult to create a universal mixer that always takes the current cycle graph state to another cycle state. We are still pursuing methods for accomplishing this.

6.4.2 Penalty Method

The best approach, so far, is to use a penalty method to move among cycle states. Basically, move among graph states and try to penalize any non-cycle graph states. It is relatively easy to create a penalty that enforces the graph state must have two edges at every vertex. To do that we add penalty terms for vertices with a edges, where $a = 0, 1, 3, 4, \ldots, n-1$. The main issue with this, is that there are many penalty terms and that can be costly in the objective. For example, the penalty term for $a = 4$ will be

$$H_{A_4} = \sum_{r \in V} \sum_{j,k,l,m \in V/r} X_r Z_j Z_k Z_l Z_m.$$

The penalty term for penalizing vertices that have i edges will be

$$H_{A_i} = \sum_{r \in V} \sum_{K \subseteq V/r, |K|=i} X_r \prod_{m \in K} Z_m. \tag{6.15}$$

Our total penalty term for ensuring two edges are attached to each vertex will be

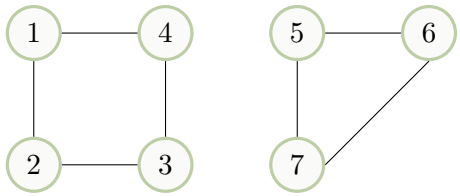
$$H_A = p_2 \cdot \left(H_{A_0} + H_{A_1} + \sum_{i=3}^{n-1} H_{A_i} \right), \tag{6.16}$$

where p_2 is a weighting constant.

6.4.3 Subtour Breaking

The more difficult problem in the penalty method is how to prevent subtours. Because, even if every vertex has two edges, there is still the possibility of disconnected subtours in the graph state, as seen in Figure 6.1. But, the feasible solutions for TSP are fully

Figure 6.1: Two disconnected subtours on a graph with seven vertices.



connected tours.
The most successful method for subtour breaking takes advantage of stabilizer theory. The method works as a penalty and will be included, along with the 2-vertex penalty, H_A, in the Penalty Hamiltonian, H_D, and thus also the Phase Hamiltonian, H_P.
If a graph on n vertices has $\deg(v) = 2, \forall v \in V$, then the graph must consist of a single tour or multiple subtours. Given every vertex in the graph has degree equal to two, then subtours in the graph will have at most length $n - 3$ and at least length 3.

Lemma 6.6
Let $|\psi\rangle \in \mathbb{C}^{2^n}$ be a graph state corresponding to a graph, $G = (V, E)$. If G has a tour or subtour, without loss of generality, defined as $1, \ldots, k$, with $k \in \{3, \ldots, n-3\}$ in the case of subtours, and $\deg(v) = 2, \forall v \in \{1, \ldots, k\}$, then $|\psi\rangle$ has a stabilizer operator of the form $(-1)^k \prod_{i=1}^{k} X_i$.

Proof. We will use Lemmata (4.30) and (4.31) in this proof. The terms in the generating set for the stabilizer group of $|\psi\rangle$ corresponding to the tour or subtour will all be of the form $X_i Z_j Z_k$, where $i \neq j \neq k \neq i$. This is because $\deg(v) = 2, \forall v \in \{1, \ldots, k\}$

and there can be no self edges or double edges in graph states. The other terms in the generating set will be of the form $X_i \prod_{k\in N(i)} Z_k$. These terms may or may not be a part of a tour or subtour, but they are not part of the tour or subtour defined by $1,\ldots,k$. Let T be defined as

$$T = \{X_1 Z_k Z_2, X_2 Z_1 Z_3, \ldots, X_{k-1} Z_{k-2} Z_k, X_k Z_{k-1} Z_1\}. \tag{6.17}$$

The generating set for the stabilizer group of $|\psi\rangle$ can be written as follows.

$$\mathcal{G}_{|\psi\rangle} = T \cup \bigcup_{\forall v\in V\setminus\{1,\ldots,k\}} \left\{ X_v \prod_{r\in N(v)} Z_r \right\} \tag{6.18}$$

The stabilizer group, $S_{|\psi\rangle}$, of the graph state, $|\psi\rangle$, is the abelian subgroup generated by $\mathcal{G}_{|\psi\rangle}$. One can see how to create our stabilizer operator $(-1)^k \prod_{i=1}^k X_i$ from the elements of T. Every subgroup is closed under products and inverses. Therefore, the product of any combination of elements, or their inverses, from the generating set will be in the stabilizer group.
Taking the product of all generating elements in T, the ones contributing to the tour or subtour, we obtain the following expression.

$$\begin{aligned}
\prod_{A\in T} A &= X_1 Z_k Z_2 \cdot X_2 Z_1 Z_3 \cdots X_{k-1} Z_{k-2} Z_k \cdot X_k Z_{k-1} Z_1 \\
&= X_1 Z_1 Z_1 \cdot Z_2 X_2 Z_2 \cdots Z_{k-1} X_{k-1} Z_{k-1} \cdot Z_k Z_k X_k \\
&= X_1 \cdot Z_2 X_2 Z_2 \cdots Z_{k-1} X_{k-1} Z_{k-1} \cdot X_k \\
&= (-1)^{k-2} X_1 \cdot X_2 Z_2 Z_2 \cdots X_{k-1} Z_{k-1} Z_{k-1} \cdot X_k \\
&= (-1)^{k-2} X_1 \cdot X_2 \cdots X_{k-1} \cdot X_k.
\end{aligned} \tag{6.19}$$

Since, $k-2$ and k are both even or both odd, which is what is relevant for the exponent, we can simplify further,

$$(-1)^k X_1 \cdot X_2 \cdots X_{k-1} \cdot X_k \in S_{|\psi\rangle}. \tag{6.20}$$

□

Lemma 6.7
Let $|\psi\rangle \in \mathbb{C}^{2^n}$ be a graph state corresponding to a graph, $G = (V, E)$, where $\deg(v) = 2, \forall v \in V$. If $|\psi\rangle$ has a tour and no subtours, then, $|\psi\rangle$ has no stabilizer operator of the form $(-1)^k \prod_{i=1}^k X_i$ for $k \in \{3,\ldots,n-3\}$.

Proof. Let $|\psi\rangle \in \mathbb{C}^{2^n}$ be a tour. Without loss of generality, let the tour ordering be described by $1,\ldots,n$. We will attempt to construct $(-1)^k \prod_{i=1}^k X_i$ and show it is not possible for $k \in \{3,\ldots,n-3\}$.

$$\mathcal{G}_{|\psi\rangle} = \{X_1 Z_k Z_2, X_2 Z_1 Z_3, \ldots, X_{n-1} Z_{n-2} Z_n, X_n Z_{n-1} Z_1\}. \tag{6.21}$$

$$S_{|\psi\rangle} = \langle X_1 Z_k Z_2, X_2 Z_1 Z_3, \ldots, X_{n-1} Z_{n-2} Z_n, X_n Z_{n-1} Z_1 \rangle. \tag{6.22}$$

There are two things to note about this generating set:

- X_i appears in exactly one generator $\forall i \in \{1, \ldots, n\}$,
- Every generator contains exactly one X_i term.

Let $J \subseteq \{1, \ldots, n\}$. Since X_i and Z_i are Hermitian and unitary, we have the following properties $\forall A, B \in S_{|\psi\rangle}$:

- $AA = \mathbb{I}$,
- $AB = \pm BA$.

For this reason, we only need to look at the possibilites from using each generator exactly once. To create the $(-1)^k \prod_{i=1}^k X_i$ term we must use each generator containing X_i, for $i = 1, \ldots, k$, exactly once. Let F be the product of these terms.

$$\begin{aligned} F &= X_1 Z_k Z_2 X_2 Z_1 Z_3 \cdots X_{k-1} Z_{k-2} Z_k X_k Z_{k-1} Z_1 \\ &= X_1 Z_n Z_1 Z_2 X_2 Z_2 Z_3 X_3 Z_3 \cdots Z_{k-1} X_{k-1} Z_{k-1} Z_k X_k Z_{k+1} \\ &= (-1)^k X_1 Z_n Z_1 X_2 X_3 \cdots X_{k-1} Z_k X_k Z_{k+1}. \end{aligned} \tag{6.23}$$

To obtain the desired term, $(-1)^k \prod_{i=1}^k X_i$, from F, one needs to eliminate the Z_n, Z_1, Z_k, and Z_{k+1} terms. To eliminate the Z_{k+1} term above we must apply a generator, or a product of generators, that contains Z_{k+1}. The only generators that contain Z_{k+1} are $X_{k+2} Z_{k+1} Z_{k+3}$ and $X_k Z_{k-1} Z_{k+1}$. Applying either of these terms on their own, or in a product of generators, will add an unwanted X_{k+2} or cancel a desired X_k. Therefore, it is impossible to create the operator $(-1)^k \prod_{i=1}^k X_i$ from the generators for this stabilizer group.

$$(-1)^k \prod_{i=1}^{k} X_i \notin S_{|\psi\rangle}, k \in \{3, \ldots, n-3\} \tag{6.24}$$

□

Theorem 6.8
Let $|\psi\rangle \in \mathbb{C}^{2^n}$ be a graph state corresponding to a graph, $G = (V, E)$, where $\deg(v) = 2, \forall v \in V$ and there are no subtours. Without loss of generality, let the tour be defined as $1, \ldots, n$. Let $k \in \{3, \ldots, n-3\}$. Then $\langle\psi| (-1)^k \prod_{i=1}^k X_i |\psi\rangle = 0$.

Proof. There exists $l, m \notin \{1, \ldots, k\}$, such that $X_l \cdot Z_m \cdot Z_k \in S_{|\psi\rangle}$. Since $|\psi\rangle$ is a cycle on n vertices and $k \in \{3, \ldots, n-3\}$, there exists an edge leaving vertex k, that is not attached to the vertices in $\{1, \ldots, k-1\}$. Without loss of generality, this edge is given by (kl), $l \notin \{1, \ldots, k-1\}$. l cannot be equal to k as that would imply a self

edge, which is not possible.
One proves the existence of m by contradiction. Let m be the vertex that the second edge attached to node l is connected to. Assume that $m \in \{1, \dots, k-1\}$, m cannot be equal to k as that would imply a double edge, which is not possible. Define the second edge attached to l as (lm). Then, $|\psi\rangle$ contains a subtour of length $k+1$ or smaller. Since, $|\psi\rangle$ has no subtours, we have $k+1 \geq n$, a contradiction to $k \leq n-3$. Thus, there exists $m \notin \{1, \dots, k\}$ such that the edge (lm) is active in $|\psi\rangle$.
Then, since $X_l Z_m Z_k \in S_{|\psi\rangle}$,

$$\begin{aligned}
\langle\psi| (-1)^k \prod_{i=1}^{k} X_i |\psi\rangle &= \langle\psi| X_l Z_m Z_k \cdot (-1)^k \prod_{i=1}^{k} X_i \cdot X_l Z_m Z_k |\psi\rangle \\
&= \langle\psi| X_l X_l Z_m Z_m Z_k \cdot (-1)^k \prod_{i=1}^{k} X_i \cdot Z_k |\psi\rangle \\
&= \langle\psi| Z_k \cdot (-1)^k \prod_{i=1}^{k-1} X_i \cdot X_k \cdot Z_k |\psi\rangle \\
&= \langle\psi| (-1)^k \prod_{i=1}^{k-1} X_i \cdot Z_k \cdot X_k \cdot Z_k |\psi\rangle \\
&= -\langle\psi| (-1)^k \prod_{i=1}^{k-1} X_i \cdot X_k \cdot Z_k \cdot Z_k |\psi\rangle \\
&= -\langle\psi| (-1)^k \prod_{i=1}^{k} X_i |\psi\rangle ,
\end{aligned} \tag{6.25}$$

Where, for the second equality we use that $l, m \notin \{1, \dots, k\}$ and for the fifth equality we use Lemma (4.30). This implies

$$\langle\psi| (-1)^k \prod_{i=1}^{k} X_i |\psi\rangle = 0. \tag{6.26}$$

□

Let $|\psi\rangle \in \mathbb{C}^{2^n}$ be a graph state corresponding to a graph, $G = (V, E)$, that has $\deg(v) = 2, \forall v \in V$ and subtours. By Lemma 6.6, $(-1)^k \prod_{i=1}^{k} X_i \in S_{|\psi\rangle}$ for some $k \in \{3, \dots, n-3\}$. Therefore, for the values of k corresponding to the subtours in $|\psi\rangle$,

$$\langle\psi| (-1)^k \prod_{i=1}^{k} X_i |\psi\rangle = \langle\psi|\psi\rangle = 1 > 0.$$

For $k \in \{3, \dots, n-3\}$, one can use $(-1)^k \prod_{i=1}^{k} X_i$ to penalize subtours in the penalty Hamiltonian. Graph states with $\deg(v) = 2, \forall v \in V$ and no subtours will have

$\langle\psi| (-1)^k \prod_{i=1}^{k} X_i |\psi\rangle = 0$, $\forall k \in \{3, \ldots, n-3\}$ by Theorem 6.8. Graph states with $\deg(v) = 2, \forall v \in V$ and subtours will have $\langle\psi| (-1)^k \prod_{i=1}^{k} X_i |\psi\rangle = 1$, for some $k \in \{3, \ldots, n-3\}$.
The penalty encoded as a Hamiltonian will be

$$H_M = p_2 \left(\sum_{K \subset \{1,\ldots,n\}, 3 \leq |K| \leq n-3} (-1)^{|K|} \prod_{i \in K} X_i \right). \tag{6.27}$$

H_M will be the sum of stabilizer operators for graph states which have subtours on nodes in K, for example $K = \{1, 2, \ldots, n-4\}$. Where K can be any subset of $\{1, .., n\}$ as long as its cardinality is less than $n-2$ and greater than 2.
Therefore, our final penalty Hamiltonian will be $H_D = H_A + H_M$, and the final phase Hamiltonian will be $H_P = H_D + H_C$.
Since H_D satisfies the feasibility constraints for TSP and H_C satisfies the optimality constraints for TSP, p_1 should be weighted less than p_2, such that it is never favourable to violate the feasibility constraints to satisfy the optimality constraints. One such constraint is $p_1 \sum_{uv \in E} W_{uv} < p_2$, where W is the distance matrix for the TSP instance.

6.5 Measurement

Multiple measurement approaches were considered. The first approach was to measure in a basis of cycle graph states. However, this is not feasible since cycle graph states are not necessarily orthogonal.

(a) $|\psi\rangle$, cycle graph state $(1, 2, 3, 4)$. **(b)** $|\phi\rangle$, cycle graph state $(1, 2, 4, 3)$.

Figure 6.2: Two cycle graphs on 4 vertices.

Example 6.9
We will give a counterexample to prove that cycle graph states do not always form an orthonormal basis. The calculations will be done in the computational basis to give more intuition. Let $n = 4$ and $|\psi\rangle, |\phi\rangle \in \mathbb{C}^{2^n}$. Recall Equation (4.34), the

computational basis representation for a graph state with four vertices and no edges is

$$\begin{aligned}|+\rangle^{\otimes 4} =& \frac{1}{\sqrt{2^4}}(|0\rangle + |1\rangle)^{\otimes 4} \\ =& \frac{1}{4}(|0000\rangle + |0001\rangle + |0010\rangle + |0100\rangle + |1000\rangle + |0011\rangle + |0101\rangle + |1001\rangle \\ &+ |1100\rangle + |0110\rangle + |1010\rangle + |1110\rangle + |1101\rangle + |1011\rangle + |0111\rangle + |1111\rangle).\end{aligned} \tag{6.28}$$

Let $|\psi\rangle$ be the cycle graph state corresponding to Figure 6.2a. The edge toggles necessary to to create $|\psi\rangle$ from $|+\rangle^{\otimes 4}$ are controlled-Z gates on edges $E_1 = \{(1,2),(2,3),(3,4),(4,1)\}$. To apply a controlled-Z gate on edge (i,j) to a quantum state, one examines every basis vector and if the ith qubit, the control qubit, is 1 then the jth qubit, the target qubit, has a Z gate applied to it. Meaning, that the basis vector is multiplied by negative one if the jth qubit is in state $|1\rangle$ and is left unchanged if the jth qubit is in state $|0\rangle$. Therefore,

$$\begin{aligned}|\psi\rangle =& \prod_{e\in E_1} CZ_e \frac{1}{4}(|0\rangle + |1\rangle)^{\otimes 4} \\ =& CZ_{(1,2)}CZ_{(2,3)}CZ_{(3,4)}CZ_{(4,1)}\frac{1}{4}(|0\rangle + |1\rangle)^{\otimes 4} \\ =& CZ_{(2,3)}CZ_{(3,4)}CZ_{(4,1)}\frac{1}{4}(|0000\rangle + |0001\rangle + |0010\rangle + |0100\rangle + |1000\rangle + |0011\rangle + |0101\rangle \\ &+ |1001\rangle - |1100\rangle + |0110\rangle + |1010\rangle - |1110\rangle - |1101\rangle + |1011\rangle + |0111\rangle - |1111\rangle). \\ =& \frac{1}{4}(|0000\rangle + |0001\rangle + |0010\rangle + |0100\rangle + |1000\rangle - |0011\rangle + |0101\rangle - |1001\rangle \\ &- |1100\rangle - |0110\rangle + |1010\rangle + |1110\rangle + |1101\rangle + |1011\rangle + |0111\rangle + |1111\rangle).\end{aligned} \tag{6.29}$$

Let $|\phi\rangle$ be the cycle graph state corresponding to Figure 6.2b. The edge toggles neceessary to to create $|\phi\rangle$ from $|+\rangle^{\otimes 4}$ are controlled-Z gates on edges $E_2 = \{(1,2),(2,4),(4,3),(3,1)\}$. Therefore,

$$\begin{aligned}|\phi\rangle =& \prod_{e\in E_2} CZ_e \frac{1}{4}(|0\rangle + |1\rangle)^{\otimes 4} \\ =& \frac{1}{4}(|0000\rangle + |0001\rangle + |0010\rangle + |0100\rangle + |1000\rangle - |0011\rangle - |0101\rangle + |1001\rangle - |1100\rangle \\ &+ |0110\rangle - |1010\rangle + |1110\rangle + |1101\rangle + |1011\rangle + |0111\rangle + |1111\rangle).\end{aligned} \tag{6.30}$$

Both $|\phi\rangle$ and $|\psi\rangle$ are cycle graph states, graph states whose corresponding graph is a tour, now one will show that they are not orthogonal. If $\langle\phi|\psi\rangle \neq 0$ the states are

not orthogonal. Let a and b be computational basis states, e.g., $a = |0000\rangle, b = |0001\rangle$. Since, the computational basis states are orthogonal to each other, $\langle a|b\rangle = 0$ if $a \neq b$, and $\langle a|b\rangle = 1$ if $a = b$.

$$\begin{aligned}
\langle\phi|\psi\rangle =& \frac{1}{4}(\langle 0000| + \langle 0001| + \langle 0010| + \langle 0100| + \langle 1000| - \langle 0011| - \langle 0101| + \langle 1001| \\
& - \langle 1100| + \langle 0110| - \langle 1010| + \langle 1110| + \langle 1101| + \langle 1011| + \langle 0111| + \langle 1111|) \\
& \cdot \frac{1}{4}(|0000\rangle + |0001\rangle + |0010\rangle + |0100\rangle + |1000\rangle - |0011\rangle + |0101\rangle - |1001\rangle \\
& - |1100\rangle - |0110\rangle + |1010\rangle + |1110\rangle + |1101\rangle + |1011\rangle + |0111\rangle + |1111\rangle) \\
=& \frac{1}{16}(\langle 0000|0000\rangle + \langle 0001|0001\rangle + \langle 0010|0010\rangle + \langle 0100|0100\rangle + \langle 1000|1000\rangle \\
& + \langle 0011|0011\rangle - \langle 0101|0101\rangle - \langle 1001|1001\rangle + \langle 1100|1100\rangle - \langle 0110|0110\rangle \\
& - \langle 1010|1010\rangle + \langle 1110|1110\rangle + \langle 1101|1101\rangle + \langle 1011|1011\rangle + \langle 0111|0111\rangle \\
& + \langle 1111|1111\rangle) \\
=& \frac{1}{16}(12 - 4) = \frac{1}{2} \neq 0.
\end{aligned} \tag{6.31}$$

Therefore, $|\phi\rangle$ and $|\psi\rangle$ are not orthogonal.

Theorem 6.10
There is no value of n, the number of vertices in the TSP, where the set of cycle graph states form an orthonormal basis in $\mathbb{C}^{2^n}$.

Proof. The number of possible cycle graph states for a given number of vertices, n, is equal to the number of unique tours for that value of n. Let K be the number of cycle graph states on n vertices,

$$K = \frac{(n-1)!}{2}. \tag{6.32}$$

A orthonormal basis in $\mathbb{C}^{2^n}$ must have exactly 2^n elements. For $n \leq 6$, $2^n > \frac{(n-1)!}{2}$. And, for $n \geq 7$, $2^n < \frac{(n-1)!}{2}$. Therefore, there is no value of $n \in \mathbb{Z}^+$ where $2^n = \frac{(n-1)!}{2}$. Thus, there is no value of n where the set of cycle graph states on n vertices forms an orthonormal basis in $\mathbb{C}^{2^n}$. □

Another approach was to map cycle states to a set of orthogonal states in order to measure. But, this is not feasible since unitary operators preserve orthogonality. The first measurement protocol was an approach using Bell states. During the coding implementation of the algorithm we found a flaw with this method and realized the measurement protocol was infeasible. This was disappointing because if the Bell measurement protocol had worked the algorithm would only need n qubits to run a

TSP on n vertices. Our solution was to use a POVM for the measurement in our algorithm. With a POVM, the algorithm will need to use ancillary qubits and the total number of qubits required to run a tsp on n vertices will be greater than n. Another disadvantage to using a POVM is that the distinguishability is imperfect, we cannot be certain we have measured the state we think we have measured. However, with increasing the number of measurements we can increase the probability we have measured the state correctly close to certainty. Three distinct POVM approaches were examined, each having different advantages and disadvantages. The third POVM is the best measurement protocol of these. These are the most successful measurement protocols for this algorithm so far. Although, future work on this algorithm could be improving the measurement protocol.
For the POVM, we want to find a set of positive operators we can use to measure and distinguish our graph states. One needs to find $\{E_m\}$ such that every E_m is a positive operator and the completeness relation is satisfied, $\sum_m E_m = \mathbb{I}$. When we apply a single POVM to a graph state, one of the $E_i \in \{E_m\}$ will be returned, based on the individual probabilities for each operator. The measurement protocol uses repeated measurements and these probabilities to unveil the graph state. We use the following properties in the calculation of these probabilities and in the proofs of the positivity of the operators, let $|\psi\rangle \in \mathbb{C}^{2^n}$:

- $\langle\psi|\,\mathbb{I}\,|\psi\rangle = \langle\psi|\psi\rangle = 1$.
- If edges (ij) and (ik) are active, $(ij) \wedge (ik)$, then $X_i Z_j Z_k \in \mathcal{G}_{|\psi\rangle}$.
- If edges (ij) and (ik) are not both active, $\neg(ij) \vee \neg(ik)$, then $X_i Z_j Z_k \notin \mathcal{G}_{|\psi\rangle}$.
- Let $A \in \mathcal{G}_{|\psi\rangle}$, then $A \in S_{|\psi\rangle}$.
- Let $A \in S_{|\psi\rangle}$, then $\langle\psi|\,A\,|\psi\rangle = \langle\psi|\psi\rangle = 1$.
- Let $A \notin \mathcal{G}_{|\psi\rangle}$, then $\langle\psi|\,A\,|\psi\rangle = 0$, by Theorem 6.5.
- The only eigenvalue of $\mathbb{I}$ is 1.
- The eigenvalues of the X and Z operators are ± 1.

Throughout the POVMs, we will make use of the following index set I,

$$I = \{(i,j,k) | i,j,k \in \{1,\ldots,n\}, i \neq j \neq k \neq i\}. \tag{6.33}$$

Also, throughout the POVM section, when a set of two edges are reffered to as inactive, this means both edges are not active, so either both edges are inactive or one of the two edges is inactive, not necessarily that both are inactive.

6.5.1 Measurement one

Let $|\psi\rangle \in \mathbb{C}^{2^n}, \gamma(\mathbb{I}+X_iZ_jZ_k), \gamma(\mathbb{I}-X_iZ_jZ_k) \in \mathbb{C}^{2^n \times 2^n}$. We define the following positive operators,

$$\{E_m\} = \bigcup_{(i,j,k)\in I} \gamma(\mathbb{I} + X_iZ_jZ_k) \cup \bigcup_{(i,j,k)\in I} \gamma(\mathbb{I} - X_iZ_jZ_k). \tag{6.34}$$

Let a be the total number of operators, $a = |\{E_m\}|$, then

$$a = n(n-1)(n-2) \cdot 2 \cdot \frac{1}{2} = n(n-1)(n-2). \tag{6.35}$$

The factor of 2 comes from the positive and negative version of the operator and the factor of $\frac{1}{2}$ comes from switching j and k. The number of ancillary qubits required to use this measurement method in the implementation of the algorithm is a. We set

$$\gamma = \frac{1}{a}. \tag{6.36}$$

This value for γ ensures the completeness relation is satisfied. Now, we will calculate the probabilities of the operators being returned in a POVM, given the relevant edges are active or inactive.
For operators $\gamma(\mathbb{I} + X_iZ_jZ_k)$:
The probability of receiving operator $\gamma(\mathbb{I} + X_iZ_jZ_k)$ from a POVM on graph state $|\psi\rangle$ which has active edges $(ij), (ik)$ is given by:

$$\begin{aligned}
\mathbb{P}(X_iZ_jZ_k|(ij) \wedge (ik)) &= \langle\psi|\, \gamma(\mathbb{I} + X_iZ_jZ_k)\, |\psi\rangle \\
&= \gamma(\langle\psi|\, \mathbb{I}\, |\psi\rangle + \langle\psi|\, X_iZ_jZ_k\, |\psi\rangle). \\
&= \gamma(1 + \langle\psi|\, X_iZ_jZ_k\, |\psi\rangle) \\
&= \gamma(1 + \langle\psi|\psi\rangle) \\
&= 2\gamma = \frac{2}{a}.
\end{aligned} \tag{6.37}$$

The probability of receiving operator $\gamma(\mathbb{I} + X_iZ_jZ_k)$ from a POVM on graph state $|\psi\rangle$ which has inactive edges $(ij), (ik)$ is given by:

$$\begin{aligned}
\mathbb{P}(X_iZ_jZ_k|\neg(ij) \vee \neg(ik)) &= \langle\psi|\, \gamma(\mathbb{I} + X_iZ_jZ_k)\, |\psi\rangle \\
&= \gamma(\langle\psi|\, \mathbb{I}\, |\psi\rangle + \langle\psi|\, X_iZ_jZ_k\, |\psi\rangle). \\
&= \gamma(1 + \langle\psi|\, X_iZ_jZ_k\, |\psi\rangle) \\
&= \gamma = \frac{1}{a}.
\end{aligned} \tag{6.38}$$

For operators $\gamma(\mathbb{I} - X_iZ_jZ_k)$:
The probability of receiving operator $\gamma(\mathbb{I} - X_iZ_jZ_k)$ from a POVM on graph state $|\psi\rangle$

which has active edges $(ij), (ik)$ is given by:

$$\begin{aligned}\mathbb{P}(X_iZ_jZ_k|(ij) \wedge (ik)) &= \langle\psi|\, \gamma(\mathbb{I} - X_iZ_jZ_k)\, |\psi\rangle \\ &= \gamma(\langle\psi|\, \mathbb{I}\, |\psi\rangle - \langle\psi|\, X_iZ_jZ_k\, |\psi\rangle). \\ &= \gamma(1 - \langle\psi|\, X_iZ_jZ_k\, |\psi\rangle) \\ &= \gamma(1 - \langle\psi|\psi\rangle) \\ &= 0.\end{aligned} \tag{6.39}$$

The probability of receiving operator $\gamma(\mathbb{I} - X_iZ_jZ_k)$ from a POVM on graph state $|\psi\rangle$ which has inactive edges $(ij), (ik)$ is given by:

$$\begin{aligned}\mathbb{P}(X_iZ_jZ_k|\neg(ij) \vee \neg(ik)) &= \langle\psi|\, \gamma(\mathbb{I} - X_iZ_jZ_k)\, |\psi\rangle \\ &= \gamma(\langle\psi|\, \mathbb{I}\, |\psi\rangle - \langle\psi|\, X_iZ_jZ_k\, |\psi\rangle). \\ &= \gamma(1 - \langle\psi|\, X_iZ_jZ_k\, |\psi\rangle) \\ &= \gamma = \frac{1}{a}.\end{aligned} \tag{6.40}$$

Positivity of operators

Proof. We must prove that the eigenvalues of $\gamma(\mathbb{I} - X_iZ_jZ_k)$ and $\gamma(\mathbb{I} + X_iZ_jZ_k)$ are real and non-negative. Let $B = X_iZ_jZ_k$, and $y \in \sigma(B)$. B is Hermitian according to Lemma 4.34.
First, we will prove that $y = \pm 1$.
Without loss of generality, let $i < j < k$. Using the fourth property from Lemma 4.8, we can write B as follows:

$$\begin{aligned}B &= X_iZ_jZ_k \\ &= \left(\bigotimes_{a=1}^{i-1} \mathbb{I} \otimes X \otimes \bigotimes_{a=i+1}^{n} \mathbb{I}\right)\left(\bigotimes_{a=1}^{j-1} \mathbb{I} \otimes Z \otimes \bigotimes_{a=j+1}^{n} \mathbb{I}\right)\left(\bigotimes_{a=1}^{k-1} \mathbb{I} \otimes Z \otimes \bigotimes_{a=k+1}^{n} \mathbb{I}\right) \\ &= \left(\bigotimes_{a=1}^{i-1} \mathbb{II} \otimes X\mathbb{I} \otimes \bigotimes_{a=i+1}^{j-1} \mathbb{II} \otimes \mathbb{I}Z \otimes \bigotimes_{a=j+1}^{n} \mathbb{II}\right)\left(\bigotimes_{a=1}^{k-1} \mathbb{I} \otimes Z \otimes \bigotimes_{a=k+1}^{n} \mathbb{I}\right) \\ &= \bigotimes_{a=1}^{i-1} \mathbb{I} \otimes X \otimes \bigotimes_{a=i+1}^{j-1} \mathbb{I} \otimes Z \otimes \bigotimes_{a=j+1}^{k-1} \mathbb{I} \otimes Z \otimes \bigotimes_{a=k+1}^{n} \mathbb{I}.\end{aligned} \tag{6.41}$$

B is the Kronecker product of multiple $\mathbb{I}$ operators with two Z operators and one X operator. By the second property of Lemma 4.8, any eigenvalue of $X_iZ_jZ_k$ arises as a product of the eigenvalues of $\mathbb{I}, X$ and Z. Therefore,

$$y = 1 \cdot \pm 1 \cdot \pm 1 = \pm 1. \tag{6.42}$$

Now, we will prove the positivity of our operators.

$$\begin{aligned} &\det(B - y\mathbb{I}) = 0, \\ &\det((B + \mathbb{I}) - (y+1)\mathbb{I}) = 0. \end{aligned} \tag{6.43}$$

This implies the eigenvalues of $\mathbb{I} + B$ are $y + 1$. Therefore, the eigenvalues of the operator $\mathbb{I} + X_i Z_j Z_k$ are

$$\lambda = y + 1 = \pm 1 + 1 = (0, 2). \tag{6.44}$$

Now, let $C = -B = -X_i Z_j Z_k$, and $x \in \sigma(C)$. When a matrix is multiplied by a constant, its eigenvalues are multiplied by the same constant. Therefore, $x = -y = \pm 1$.

$$\begin{aligned} &\det(C - x\mathbb{I}) = 0, \\ &\det((C + \mathbb{I}) - (x+1)\mathbb{I}) = 0. \end{aligned} \tag{6.45}$$

This implies the eigenvalues of $C + \mathbb{I}$ are $x + 1$. Therefore, the eigenvalues of the operator $\mathbb{I} - X_i Z_j Z_k$ are

$$\lambda = x + 1 = \pm 1 + 1 = (0, 2). \tag{6.46}$$

Therefore, $\mathbb{I} + X_i Z_j Z_k$ and $\mathbb{I} - X_i Z_j Z_k$ will have eigenvalues of zero or two. After multiplying by the constant $\gamma = \frac{1}{a}$, the resulting eigenvalues are zero and $\frac{2}{a}$. Both of which are non-negative, thus our operators are positive. □

Number of measurements

Next, we will outline the number of expected measurements to uncover the graph state using this method. For each measurement, one operator will be returned based on the probabilities detailed in the beginning of this section. Let r be the number of times we measure the quantum state, where $r \geq a$. Let $M+_{i,j,k}$ be the number of times $\gamma(\mathbb{I} + X_i Z_j Z_k)$ is returned in r measurements and let $M-_{i,j,k}$ be the number of times $\gamma(\mathbb{I} - X_i Z_j Z_k)$ is returned in r measurements. Let $\mathbb{N}(x|y)$ be the expected count of a variable, x, given conditions, y.

$$\mathbb{N}(r|M+_{i,j,k} = 1, (ij) \wedge (ik))) = \frac{a}{2} \tag{6.47}$$

$$\mathbb{N}(r|M-_{i,j,k} = 1, (ij) \wedge (ik)) = \infty \tag{6.48}$$

$$\mathbb{N}(r|M+_{i,j,k} = 1, \neg(ij) \vee \neg(ik)) = a \tag{6.49}$$

$$\mathbb{N}(r|M-_{i,j,k} = 1, \neg(ij) \vee \neg(ik)) = a. \tag{6.50}$$

So, if we measure the state a times, we should expect to see $\gamma(\mathbb{I} + X_i Z_j Z_k)$ twice for active edges and once for inactive edges. And, we should expect to see $\gamma(\mathbb{I} - X_i Z_j Z_k)$

once for inactive edges and never for active edges.
There are a couple strategies for how to approach unveiling the graph state given these probabilities, we will detail two approaches.
First approach:
Let $(ij) \wedge (ik)$, $\neg(lm) \vee \neg(ln)$. We expect

$$M+_{i,j,k} \approx 2 \cdot M+_{l,m,n} \approx 2 \cdot M -_{l,m,n} . \tag{6.51}$$

After r measurements, we will consider edges active if the corresponding positive version of the operator, $\gamma(\mathbb{I} + X_i Z_j Z_k)$, appears approximately twice as often as the negative version of the operator, $\gamma(\mathbb{I} + X_i Z_j Z_k)$, appears for other edges. For example, if $\gamma(\mathbb{I} - X_a Z_b Z_c)$, appears twice in r measurements and $\gamma(\mathbb{I} + X_i Z_j Z_k)$ appears five times in r measurements, then we will consider $(ij), (ik)$ active and $(ab)(ac)$ inactive.

$$\mathbb{E}(M +_{i,j,k} |(ij) \wedge (ik)) = \frac{2r}{a} \tag{6.52}$$

$$\mathbb{E}(M -_{i,j,k} |(ij) \wedge (ik)) = 0 \tag{6.53}$$

$$\mathbb{E}(M +_{i,j,k} |\neg(ij) \vee \neg(ik)) = \frac{r}{a} \tag{6.54}$$

$$\mathbb{E}(M -_{i,j,k} |\neg(ij) \vee \neg(ik)) = \frac{r}{a}. \tag{6.55}$$

One chooses a splitting value, l, such that l is halfway between Equations (6.52) and (6.55).

$$\begin{aligned} l &= \left\lfloor \frac{\mathbb{E}(M +_{i,j,k} |(ij), (ik)) + \mathbb{E}(M -_{i,j,k} |\neg(ij) \vee \neg(ik))}{2} \right\rfloor \\ &= \left\lfloor \frac{3r}{2a} \right\rfloor . \end{aligned} \tag{6.56}$$

If $M+_{i,j,k} > l$ then edges $(ij), (ik)$ are both considered active and if $M-_{i,j,k} \leq l$ then edges $(ij), (ik)$ are considered inactive.
We can use the binomial formula to calculate the probability of x successes in r measurements. Where a success means the corresponding operator is returned.

$$\mathbb{P}(M+_{i,j,k} = x|(ij) \wedge (ik)) = \binom{r}{x} \left(\frac{2}{a}\right)^x \left(1 - \frac{2}{a}\right)^{r-x} \tag{6.57}$$

$$\mathbb{P}(M-_{i,j,k} = x|\neg(ij) \vee \neg(ik)) = \binom{r}{x} \left(\frac{1}{a}\right)^x \left(1 - \frac{1}{a}\right)^{r-x} . \tag{6.58}$$

The probabilities that the procedure accurately identifies edge activity and inactivity are given below, respectively.

$$\mathbb{P}(M+_{i,j,k} > l|(ij) \wedge (ik)) = 1 - \sum_{x=0}^{l} \binom{r}{x} \left(\frac{2}{a}\right)^x \left(1 - \frac{2}{a}\right)^{r-x} \tag{6.59}$$

$$\mathbb{P}(M_{-i,j,k} \leq l | \neg(ij) \vee \neg(ik)) = \sum_{x=0}^{l} \binom{r}{x} \left(\frac{1}{a}\right)^{x} \left(1 - \frac{1}{a}\right)^{r-x}. \tag{6.60}$$

Given the a value, determined by the number of vertices in the graph, one must increase r until we are confident the measurement technique is successful, given the probabilities defined in Equations (6.59) and (6.60). Increasing r increases the certainty of correct identification of edge activity by increasing the probabilities in Equations (6.60) and (6.59). But we also do not want to increase r more than necessary. The best values for r can be determined by analyzing the probabilites for each a value.

Second approach:
One considers edges $(ij), (ik)$ active if $\gamma(\mathbb{I} - X_i Z_j Z_k)$ never appears after r measurements, i.e., $M_{-i,j,k} = 0$. In order for this to work, one wants to minimize the probability that $M_{-i,j,k} = 0$ when the corresponding edges, $(ij), (ik)$, are inactive.
The probability of this approach inaccurately identifiying an inactive edge as active is given by

$$\mathbb{P}(M_{-i,j,k} = 0 | \neg(ij) \vee \neg(ik)) = \left(1 - \frac{1}{a}\right)^{r}. \tag{6.61}$$

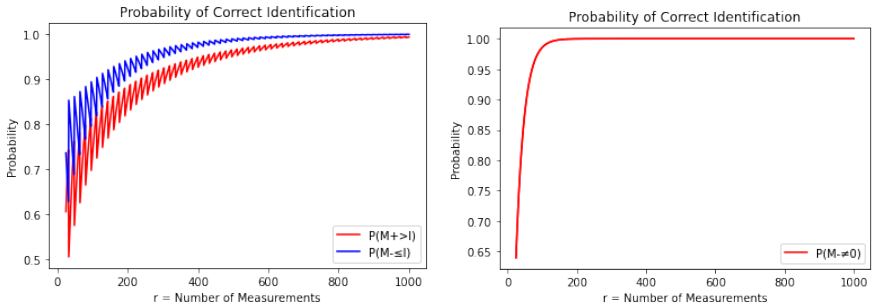
For this approach, the probability of inaccurately identifying an active edge as inactive is zero. Therefore, the probability that this approach accurately identifies the edges is given by

$$\begin{aligned} \mathbb{P}(M_{-i,j,k} \neq 0 | \neg(ij) \vee \neg(ik)) &= 1 - \mathbb{P}(M_{-i,j,k} = 0 | \neg(ij) \vee \neg(ik)) \\ &= 1 - \left(1 - \frac{1}{a}\right)^{r}. \end{aligned} \tag{6.62}$$

So, one should choose r such that the probability in Equation (6.62) is close to one.

Example 6.11
Let there be a TSP on 4 vertices. We want to measure the graph state $|\psi\rangle$ returned by our algorithm. Here, $a = 4 \cdot 3 \cdot 2 = 24$.
First approach:
Figure 6.3a displays the plot of Equations (6.59), in red, and (6.60), in blue, with respect to r for $a = 24$. The cyclic nature of both lines is due to the fact that the the approximations in Equation (6.51) will be closest to equal at certain r values, and thus the splitting strategy and identification is most accurate around those values for r. One can see both probabilities, of accurately identifying active edges (red) and of accurately identifying inactive edges (blue), are consistently above .9 for $r \geq 225$ and above .95 for $r \geq 387$. This approach requires a very high number of measurements to reach a good accuracy.
Second approach:
Figure 6.3b displays the plot of Equation (6.62) against r with $a = 24$. Figure 6.3b

displays the probability that $M_{-i,j,k} \neq 0$ for inactive edges, which is also the probability that the approach accurately identifies edge activity. The probability of accurate edge identification is above .9 for $r \geq 55$ and above .95 for $r \geq 71$. This approach requires much fewer measurements than the first approach to obtain the same accuracy, and is therefore superior and will be used instead of the first approach for this POVM. We can also continue to increase r to improve the probability. The probability of accurate identification is greater than .99 for $r \geq 109$.



(a) First approach: plot of Equations (6.59) (red) and (6.60) (blue) against r with $a = 24$.

(b) Second approach: plot of Equation (6.62) against r with $a = 24$.

Figure 6.3: Probability of accurate edge identification for n=4 vertices, for the first POVM, for the first approach (a) and the second approach (b).

6.5.2 Measurement two

Let $|\psi\rangle \in \mathbb{C}^{2^n}, \zeta(\mathbb{I} + X_i Z_j Z_k)) \in \mathbb{C}^{2^n \times 2^n}$. We define the following positive operators,

$$\{E_m\} = \bigcup_{(i,j,k)\in I} \zeta(\mathbb{I} + X_i Z_j Z_k) \cup \mathbb{I} - \zeta \sum_{(i,j,k)\in I} (\mathbb{I} + X_i Z_j Z_k). \tag{6.63}$$

The final operator is necessary to fulfill the completeness relation. Let b be the total number of operators, $b = |\{E_m\}|$, then

$$b = (n(n-1)(n-2) \cdot \frac{1}{2}) + 1 = \frac{n(n-1)(n-2)}{2} + 1 = \frac{a}{2} + 1. \tag{6.64}$$

Where the factor of $\frac{1}{2}$ comes from switching j and k and the $+1$ comes from the last operator. The number of ancillary qubits required to use this measurement method in the implementation of the algorithm is b. One advantage to this approach, over the first approach, is that we have fewer operators and thus fewer ancilla qubits are needed for the POVM. Let $\zeta = \frac{1}{2b}$. This choice of ζ will become clear later.
Now, we will calculate the probabilities of the operators being returned in a POVM,

given the relevant edges are active or inactive.
The probability of receiving operator $\zeta(\mathbb{I} + X_i Z_j Z_k)$ from a POVM on graph state $|\psi\rangle$ which has active edges $(ij), (ik)$ is given by:

$$\begin{aligned} P(X_i Z_j Z_k | (ij) \wedge (ik)) &= \langle\psi| \zeta(\mathbb{I} + X_i Z_j Z_k) |\psi\rangle \\ &= \zeta(1 + \langle\psi| X_i Z_j Z_k |\psi\rangle) \\ &= \zeta(1 + \langle\psi|\psi\rangle) \\ &= 2\zeta = \frac{1}{b}. \end{aligned} \tag{6.65}$$

The probability of receiving operator $\gamma(\mathbb{I} + X_i Z_j Z_k)$ from a POVM on graph state $|\psi\rangle$ which has inactive edges $(ij), (ik)$ is given by:

$$\begin{aligned} P(X_i Z_j Z_k | \neg(ij) \vee \neg(ik)) &= \langle\psi| \zeta(\mathbb{I} + X_i Z_j Z_k) |\psi\rangle \\ &= \zeta(1 + \langle\psi| X_i Z_j Z_k |\psi\rangle). \\ &= \zeta = \frac{1}{2b}. \end{aligned} \tag{6.66}$$

Positivity of operators

Proof. The positivity for all operators, except the last operator, follows from the proof for the operators in the first approach. $\mathbb{I} + X_i Z_j Z_k$ has eigenvalues of zero or 2. So for all but the last operator, the eigenvalues are

$$\lambda = (2\zeta, 0) = (\frac{1}{b}, 0). \tag{6.67}$$

The eigenvalues are non-negative. Therefore, these operators are positive. Let

$$\begin{aligned} Q &= \mathbb{I} - \zeta \sum_{(i,j,k)\in I} (\mathbb{I} + X_i Z_j Z_k) \\ &= \mathbb{I} - \sum_{(i,j,k)\in I} \zeta\mathbb{I} - \zeta \sum_{(i,j,k)\in I} X_i Z_j Z_k \\ &= \mathbb{I} - b\zeta\mathbb{I} - \zeta \sum_{(i,j,k)\in I} X_i Z_j Z_k \\ &= (1 - b\zeta)\mathbb{I} - R. \end{aligned} \tag{6.68}$$

Where,

$$R = \zeta \sum_{(i,j,k)\in I} X_i Z_j Z_k. \tag{6.69}$$

One still needs to prove the positivity of Q. We begin by examining what the eigenvalues of R must be.

$X_iZ_jZ_k$ has eigenvalues ± 1 and $X_iZ_jZ_k$ is Hermitian, by Lemma 4.34. The sum of Hermitian matrices is itself Hermitian, so R is Hermitian. We will use Lemma 4.9 to determine the bounds for the eigenvalues of R.
Now let, $A = X_iZ_jZ_k$ and $B = X_iZ_jZ_k$, where the selection of i, j, k is different and therefore $A \neq B$. Let $C = A + B$. The eigenvalues of A are $a_1 \geq \ldots \geq a_n$, the eigenvalues of B are $b_1 \geq \ldots \geq b_n$, and the the eigenvalues of C are $c_1 \geq \ldots \geq c_n$. Since A and B have eigenvalues of ± 1, we know $a_1 = b_1 = 1$ and $a_n = b_n = -1$. We use $c_{i+j-1} \leq a_i + b_j$ to put an upper bound on c_1. Since $i, j \in \{1, \ldots, n\}$, the only selection of i and j that allows $i + j - 1 = 1$ is $i = 1$ and $j = 1$. Therefore,

$$\begin{aligned} c_{i+j-1} &\leq a_i + b_j \\ c_1 &\leq a_1 + b_1 \\ c_1 &\leq 1 + 1 = 2. \end{aligned} \tag{6.70}$$

So, the largest eigenvalue of C will be less than or equal to two.
We use $c_{n-i-j} \geq a_{n-i} + b_{n-j}$ to put a lower bound on c_n. Since $i, j \in \{0, \ldots n-1\}$, the only selection of i and j that allows $n - i - j = n$ is $i = 0$ and $j = 0$. Therefore,

$$\begin{aligned} c_{n-i-j} &\geq a_{n-i} + b_{n-j} \\ c_n &\geq a_n + b_n \\ c_n &\geq -1 + (-1) = -2 \end{aligned} \tag{6.71}$$

So, the smallest eigenvalue of C will be greater than or equal to negative two. Therefore, for $\lambda \in \sigma(C)$,

$$-2 \leq \lambda \leq 2. \tag{6.72}$$

C will be Hermitian. Now, we repeatedly apply this procedure until we have summed over all operators and can bound the eigenvalues of R. Let the eigenvalues of R be $r_1 \geq \ldots \geq r_n$. Since there are b operators to sum over, the previous procedure will be performed $b - 1$ times. Every $X_iZ_jZ_k$ operator in the summation has eigenvalues ± 1, so the change in the bounds of the eigenvalues remain the same for each sum in the summation. The first $X_iZ_jZ_k$ operator in the summation starts with eigenvalues ± 1 and every time we add an operator the lower bound on the lowest eigenvalue decreases by 1 and the upper bound on the highest eigenvalue increases by 1. For example, in the second step of the procedure, we will have $D = C + E$, where the eigenvalues of E are $1 = e_1 \geq \ldots \geq e_n = -1$ and the eigenvalues of C are $2 \geq c_1 \geq \ldots \geq c_n \geq -2$. We have that,

$$\begin{aligned} r_1 &\leq \zeta(1 + [(b-1) \cdot 1]) = b\zeta \\ r_n &\geq -1 + [(b-1) \cdot -1] = -b\zeta. \end{aligned} \tag{6.73}$$

Let $x \in \sigma(R)$,

$$-b\zeta \leq x \leq b\zeta. \tag{6.74}$$

Let $y \in \sigma(-R)$, then

$$-b\zeta \leq y \leq b\zeta. \tag{6.75}$$

Now, we will determine the eigenvalues of Q. Let $z \in \sigma(Q)$.

$$\begin{aligned} &\det(-R - y\mathbb{I}) = 0 \\ &\det([-R + ([1 - b\zeta]\mathbb{I})] - [(y + [1 - b\zeta])\mathbb{I}]) = 0. \end{aligned} \tag{6.76}$$

Since $Q = -R + (1 - b\zeta)\mathbb{I}$, $z = y + (1 - b\zeta)$. Therefore,

$$\begin{aligned} -b\zeta + (1 - b\zeta) \leq z \leq b\zeta + (1 - b\zeta) \\ 1 - 2b\zeta \leq z \leq 1. \end{aligned} \tag{6.77}$$

For Q to be positive, $z \geq 0$.

$$\begin{aligned} 1 - 2b\zeta &\geq 0 \\ \frac{1}{2b} &\geq \zeta. \end{aligned} \tag{6.78}$$

We set $\zeta = \frac{1}{2b}$, and the operator Q is positive. □

Number of measurements

Next, we will outline the number of expected measurements to uncover the graph state using this method. For each measurement, one operator will be returned based on the probabilities detailed in the beginning of this section. Let r be the number of times we measure the quantum state, where $r \geq 2b$. Let $M{+}_{i,j,k}$ be the number of times $\gamma(\mathbb{I} + X_i Z_j Z_k)$ is returned in r measurements.

$$\mathbb{N}(r|M{+}_{i,j,k} = 1, (ij) \wedge (ik))) = b \tag{6.79}$$

$$\mathbb{N}(r|M{+}_{i,j,k} = 1, \neg(ij) \vee \neg(ik)) = 2b. \tag{6.80}$$

So, if we measure the state $2b$ times, we should expect to see $\gamma(\mathbb{I} + X_i Z_j Z_k)$ twice for active edges and once for inactive edges.
There is one strategy for how to approach unveiling the graph state given these probabilities, it is very similar to the first approach in the first POVM.
Let $(ij) \wedge (ik)$, $\neg(lm) \vee \neg(ln)$. We expect

$$M{+}_{i,j,k} \approx 2 \cdot M{+}_{l,m,n}\,. \tag{6.81}$$

After r measurements, we will consider edges active if the corresponding positive version of the operator, $\gamma(\mathbb{I} + X_i Z_j Z_k)$, appears approximately twice as often as that operator appears for other edge sets. For example, if $\gamma(\mathbb{I} + X_a Z_b Z_c)$, appears twice

in r measurements and $\gamma(\mathbb{I} + X_i Z_j Z_k)$ appears five times in r measurements, then we will consider $(ij), (ik)$ active and $(ab)(ac)$ inactive.

$$\mathbb{E}(M +_{i,j,k} |(ij) \wedge (ik)) = \frac{r}{b} \tag{6.82}$$

$$\mathbb{E}(M +_{i,j,k} |\neg(ij) \vee \neg(ik)) = \frac{r}{2b}. \tag{6.83}$$

One chooses a splitting value, l, such that l is halfway between Equations (6.82) and (6.83).

$$\begin{aligned} l &= \left\lfloor \frac{\mathbb{E}(M +_{i,j,k} |(ij) \wedge (ik)) + \mathbb{E}(M +_{i,j,k} |\neg(ij) \vee \neg(ik))}{2} \right\rfloor \\ &= \left\lfloor \frac{3r}{4b} \right\rfloor. \end{aligned} \tag{6.84}$$

If $M+_{i,j,k} > l$ then edges $(ij), (ik)$ are both considered active and if $M+_{i,j,k} \leq l$ then edges $(ij), (ik)$ are considered inactive.
We can use the binomial formula to calculate the probability of x successes in r measurements. Where a success means the corresponding operator is returned.

$$\mathbb{P}(M+_{i,j,k} = x|(ij) \wedge (ik)) = \binom{r}{x} \left(\frac{1}{b}\right)^x \left(1 - \frac{1}{b}\right)^{r-x} \tag{6.85}$$

$$\mathbb{P}(M-_{i,j,k} = x|\neg(ij) \vee \neg(ik)) = \binom{r}{x} \left(\frac{1}{2b}\right)^x \left(1 - \frac{1}{2b}\right)^{r-x}. \tag{6.86}$$

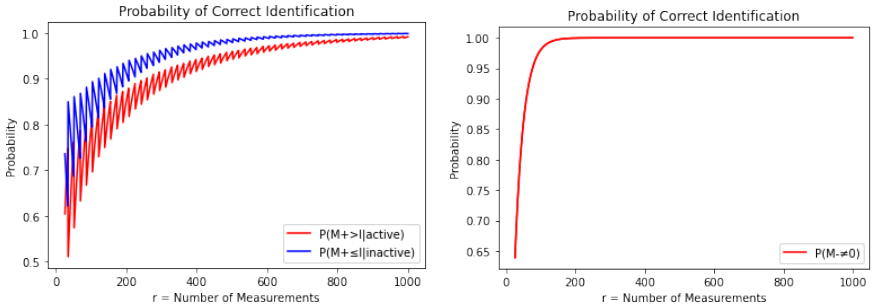
The probabilities that the procedure accurately identifies edge activity and inactivity are given below, respectively.

$$\mathbb{P}(M+_{i,j,k} > l|(ij) \wedge (ik)) = 1 - \sum_{x=0}^{l} \binom{r}{x} \left(\frac{1}{b}\right)^x \left(1 - \frac{1}{b}\right)^{r-x} \tag{6.87}$$

$$\mathbb{P}(M+_{i,j,k} \leq l|\neg(ij) \vee \neg(ik)) = \sum_{x=0}^{l} \binom{r}{x} \left(\frac{1}{2b}\right)^x \left(1 - \frac{1}{2b}\right)^{r-x}. \tag{6.88}$$

Given the b value, determined by the number of vertices in the graph, one must increase r until we are confident the measurement technique is successful, given the probabilities defined in Equations (6.87) and (6.88). Increasing r increases the certainty of correct identification of edge activity by increasing the probabilities in Equations (6.88) and (6.87). But we also do not want to increase r more than necessary. The best values for r can be determined by analyzing the probabilites for each b value.

Example 6.12
Let there be a TSP on 4 vertices. We want to measure the graph state $|\psi\rangle$ returned by our algorithm. Here, $b = \frac{4\cdot3\cdot2}{2} + 1 = 13$.
Figure 6.4a displays the plot of Equations (6.87), in red, and (6.88), in blue, with respect to r for $b = 13$. The cyclic nature of both lines is due to the fact that the the approximations in Equation (6.81) will be closest to equal at certain r values, and thus the splitting strategy and identification is most accurate around those values for r. One can see both probabilities, of accurately identifying active edges (red) and of accurately identifying inactive edges (blue), are consistently above .9 for $r \geq 349$ and above .95 for $r \geq 539$. This approach requires a very high number of measurements to reach a good accuracy.



(a) Second POVM: plot of Equations (6.87) (red) and (6.88) (blue) against r with $b = 13$.

(b) Third POVM: plot of Equation (6.105) against r with $b = 13$.

Figure 6.4: Probability of accurate edge identification for n=4 vertices, for the second POVM (a) and the third POVM (b).

6.5.3 Measurement three

Let $|\psi\rangle \in \mathbb{C}^{2^n}, \delta(\mathbb{I} - X_i Z_j Z_k)) \in \mathbb{C}^{2^n \times 2^n}$. We define the following positive operators,

$$\{E_m\} = \bigcup_{(i,j,k)\in I} \delta(\mathbb{I} - X_i Z_j Z_k) \cup \mathbb{I} - \delta \sum_{(i,j,k)\in I} (\mathbb{I} - X_i Z_j Z_k). \tag{6.89}$$

The final operator is necessary to fulfill the completeness relation. Let c be the total number of operators, $c = |\{E_m\}|$, then

$$c = (n(n-1)(n-2)\cdot\frac{1}{2}) + 1 = \frac{n(n-1)(n-2)}{2} + 1 = b = \frac{a}{2} + 1. \tag{6.90}$$

Where the factor of $\frac{1}{2}$ comes from switching j and k and the $+1$ comes from the last operator. The number of ancillary qubits required to use this measurement method in

the implementation of the algorithm is c. One advantage to this approach, over the first approach, is that we have fewer operators and thus fewer ancilla qubits are needed for the POVM. This approach uses the same number of ancillary qubits as the second approach. Let $\delta = \frac{1}{2c} = \frac{1}{2b}$. This choice of δ will become clear later.
Now, we will calculate the probabilities of the operators being returned in a POVM, given the relevant edges are active or inactive.
The probability of receiving operator $\delta(\mathbb{I} - X_i Z_j Z_k)$ from a POVM on graph state $|\psi\rangle$ which has active edges $(ij), (ik)$ is given by:

$$\begin{aligned} P(X_i Z_j Z_k | (ij) \wedge (ik)) &= \langle\psi| \delta(\mathbb{I} - X_i Z_j Z_k) |\psi\rangle \\ &= \delta(1 - \langle\psi| X_i Z_j Z_k |\psi\rangle) \\ &= \delta(1 - \langle\psi|\psi\rangle) \\ &= 0. \end{aligned} \tag{6.91}$$

The probability of receiving operator $\delta(\mathbb{I} + X_i Z_j Z_k)$ from a POVM on graph state $|\psi\rangle$ which has inactive edges $(ij), (ik)$ is given by:

$$\begin{aligned} P(X_i Z_j Z_k | \neg(ij) \vee \neg(ik)) &= \langle\psi| \delta(\mathbb{I} - X_i Z_j Z_k) |\psi\rangle \\ &= \delta(1 - \langle\psi| X_i Z_j Z_k |\psi\rangle). \\ &= \delta = \frac{1}{2b}. \end{aligned} \tag{6.92}$$

Positivity of operators

Proof. The positivity for all but the last operator, follows from the proof for operators in the first approach. $\mathbb{I} - X_i Z_j Z_k$ has eigenvalues of zero or 2. So for all but the last operator,

$$\lambda = (2\delta, 0) = (\frac{1}{b}, 0). \tag{6.93}$$

The eigenvalues are non-negative. Therefore, these operators are positive.
Let,

$$\begin{aligned} W &= \mathbb{I} - \delta \sum_{(i,j,k)\in I} (\mathbb{I} - X_i Z_j Z_k) \\ &= \mathbb{I} - \sum_{(i,j,k)\in I} \delta\mathbb{I} + \delta \sum_{(i,j,k)\in I} X_i Z_j Z_k \\ &= \mathbb{I} - b\delta\mathbb{I} + \delta \sum_{(i,j,k)\in I} X_i Z_j Z_k \\ &= (1 - b\delta)\mathbb{I} + R. \end{aligned} \tag{6.94}$$

Where,

$$R = \delta \sum_{(i,j,k)\in I} X_i Z_j Z_k. \tag{6.95}$$

One still needs to prove the positivity of W. This R is equal to the R in the proof of positivity from the second POVM, except for the change in constant. By Equation (6.74), for $x \in \sigma(R)$,

$$-b\delta \leq x \leq b\delta. \tag{6.96}$$

Now, we will determine the eigenvalues of W. Let $z \in \sigma(W)$.

$$\begin{aligned} &\det(R - x\mathbb{I}) = 0 \\ &\det([R + ([1 - b\delta]\mathbb{I})] - [(x + [1 - b\delta])\mathbb{I}]) = 0. \end{aligned} \tag{6.97}$$

Since $W = R + ([1 - b\delta]\mathbb{I})$, $z = x + (1 - b\delta)$. Therefore,

$$\begin{aligned} -b\delta + (1 - b\delta) \leq z \leq b\delta + (1 - b\delta) \\ 1 - 2b\delta \leq z \leq 1. \end{aligned} \tag{6.98}$$

For W to be positive, $z \geq 0$.

$$\begin{aligned} 1 - 2b\delta &\geq 0 \\ \frac{1}{2b} &\geq \delta. \end{aligned} \tag{6.99}$$

We set $\delta = \frac{1}{2b} = \zeta$, and the operator W is positive. □

Number of measurements

Next, we will outline the number of expected measurements to uncover the graph state using this method. For each measurement, one operator will be returned based on the probabilities detailed in the beginning of this section. Let r be the number of times we measure the state, where $r \geq 2b$. Let $M_{-i,j,k}$ be the number of times $\gamma(\mathbb{I} - X_i Z_j Z_k)$ is returned in r measurements.

$$\mathbb{N}(r|M_{-i,j,k} = 1, (ij) \wedge (ik)) = \infty \tag{6.100}$$

$$\mathbb{N}(r|M_{-i,j,k} = 1, \neg(ij) \vee \neg(ik)) = 2b. \tag{6.101}$$

So, if we measure the state $2b$ times, we should expect to see $\gamma(\mathbb{I} - X_i Z_j Z_k)$ once for inactive edges and never for active edges.

$$\mathbb{E}(M_{-i,j,k} | (ij) \wedge (ik)) = 0 \tag{6.102}$$

$$\mathbb{E}(M_{-i,j,k} \,|\neg(ij) \vee \neg(ik)) = \frac{r}{2b}. \tag{6.103}$$

Here the strategy for unveiling the graph state is the same as the second approach in the first POVM. One considers edges $(ij), (ik)$ active if $\gamma(\mathbb{I} - X_i Z_j Z_k)$ never appears after r measurements, i.e., $M_{-i,j,k} = 0$. In order for this to work, one wants to minimize the probability that $M_{-i,j,k} = 0$, when the corresponding edges $(ij), (ik)$ are inactive. The probability of this approach inaccurately identifiying an inactive edge as active is given by

$$\mathbb{P}(M_{-i,j,k} = 0 | \neg(ij) \vee \neg(ik)) = \left(1 - \frac{1}{2b}\right)^r. \tag{6.104}$$

For this approach, the probability of inaccurately identifying an active edge as inactive is zero. Therefore, the probability that this approach accurately identifies the edges is given by

$$\begin{aligned} \mathbb{P}(M_{-i,j,k} \neq 0 | \neg(ij) \vee \neg(ik)) &= 1 - \mathbb{P}(M_{-i,j,k} = 0 | \neg(ij) \vee \neg(ik)) \\ &= 1 - \left(1 - \frac{1}{2b}\right)^r. \end{aligned} \tag{6.105}$$

So, one should choose r such that the probability in Equation (6.105) is close to one.

Example 6.13
Let there be a TSP on 4 vertices. We want to measure the graph state $|\psi\rangle$ returned by our algorithm. Here, $c = b = \frac{4 \cdot 3 \cdot 2}{2} + 1 = 13$.
Figure 6.4b displays the plot of Equation (6.105) against r with $b = 13$. Figure 6.4b displays the probability that $M_{-i,j,k} \neq 0$ for inactive edges, which is also the probability that the approach accurately identifies edge activity. The probability of accurate edge identification is above .9 for $r \geq 59$ and above .95 for $r \geq 77$. We can also continue to increase r to improve the probability. The probability of accurate identification is greater than .99 for $r \geq 118$.
One can see the approach here requires almost as few measurements as the second approach in the first POVM. The only difference in the plotting of the equations is that the first POVM has $\frac{1}{a} = \frac{1}{24}$ for its probability and the third POVM has $\frac{1}{2b} = \frac{1}{26}$ for its probability.

6.5.4 Measurement Comparison

The first POVM, Section 6.5.1, uses almost double the number of ancillary qubits as the second and third POVMs, $b = \frac{a}{2} + 1$. The first POVM and the third POVM both have a measurement strategy that requires far fewer measurements than the second POVM. For this reason we can easily rule out using the second POVM, as its only advantage is the fewer number of qubits required and the third POVM also has that advantage.

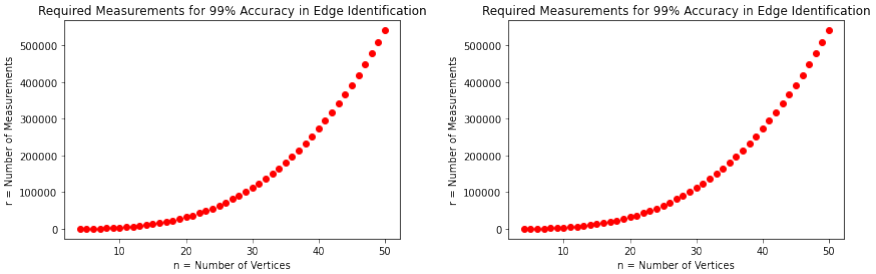
Now, we will investigate the scalability of the measurement procedure. We will compare the second measurement approach from the first POVM and the measurement approach from the third POVM. Say we want to ensure that our measurement technique has a probability of accuracy of p or greater. For the first POVM, using Equations (6.62) and (6.35), the minimum number of measurements, in terms of n, necessary to ensure a probability of p is

$$\begin{aligned} p &= 1 - \left(1 - \frac{1}{a}\right)^r \\ r &= \log_{1-\frac{1}{a}}(1-p) \\ r &= \log_{1-\frac{1}{n(n-1)(n-2)}}(1-p). \end{aligned} \tag{6.106}$$

For the third POVM, using Equations (6.105) and (6.64), the minimum number of measurements, in terms of n, necessary to ensure a probability of p is

$$\begin{aligned} p &= 1 - \left(1 - \frac{1}{2b}\right)^r \\ r &= \log_{1-\frac{1}{2b}}(1-p) \\ r &= \log_{1-\frac{1}{n(n-1)(n-2)+2}}(1-p). \end{aligned} \tag{6.107}$$

In Figure 6.5 we show the plots of Equations (6.106) and (6.107) against n for $n \in \{4, \ldots, 50\}$ and $p = .99$. One can see the number of measurements required to ensure a good accuracy does not scale well as n increases. For $n = 50$, for the first POVM, $r = 541,566$ measurements are required and for the third POVM, $r = 541,575$ measurements are required. The first POVM requires slightly fewer measurements



(a) First POVM: plot of Equation (6.106) against r with $p = .99$.

(b) Third POVM: plot of Equation (6.107) against r with $p = .99$.

Figure 6.5: The minimum number of required measurements necessary to ensure accurate edge identification with probability $p = .99$ for the first POVM (a) and the third POVM (b).

than the third POVM but they both scale similarly. So, we believe the fewer qubits

advantage of the third POVM makes it the best choice. However, for large n the number of measurements required is extremely large. Future work can be to pursue another measurement strategy for this algorithm, one that requires fewer measurements.

6.6 Implementation

In this section, we give an overview of how the circuit depth for the GSAOA scales and detail the implementation and benchmarking results for the GSAOA. The implementation of the GSAOA was done using the Python package Qiskit. We chose to omit the POVM implementation in our coding implementation of the algorithm. Instead, to measure, we use the Qiskit "statevector simulator" option for the simulator and use the Qiskit circuit function "snapshot" which takes a statevector snapshot of the internal simulator representation. The snapshot gives us the computational basis state representation of the graph state being measured. From this, we can ascertain the edge activity and thus the graph state itself. This option is only available when using a simulator of a quantum computer, if we were to implement the GSAOA on a quantum computer, we would need to use the POVM implementation to measure the graph state and ascertain the edge activity. For the coding implementation, we chose $p_1 = 1$ and $p_2 = 10 \cdot \sum_{uv \in E} W_{uv}$.

6.6.1 Circuit depth

Now, we will analyze the circuit depth of the GSAOA. The mixing Hamiltonian, H_B, seen in Equation (6.2), contributes

$$B = \sum_{i=1}^{n-1} n - i \tag{6.108}$$

terms. The phase Hamiltonian, H_P, has two parts, the cost Hamiltonian, H_C, and the penalty Hamiltonian, H_D. The cost Hamiltonian, seen in Equation (6.8), contributes

$$C = \frac{n(n-1)(n-2)}{2} \tag{6.109}$$

terms. The penalty Hamiltonian has two parts, the vertex penalty Hamiltonian, H_A, seen in Equation (6.16), and the subtour penalty Hamiltonian, H_M, seen in Equation (6.27). The vertex penalty Hamiltonian contributes

$$A = n + n(n-1) + \sum_{i=3}^{n-1} \frac{n}{i!} \prod_{j=1}^{i} (n-j) \tag{6.110}$$

terms. The subtour penalty Hamiltonian, which is only non-zero if $n \geq 6$, contributes

$$M = \sum_{i=3}^{n-3} \frac{1}{i!} \prod_{j=0}^{n-1} (n-j) \tag{6.111}$$

terms. The total number of terms for p applications of the unitary operators, H_B and H_P, will be

$$D = p \cdot (B + C + A + M). \tag{6.112}$$

The circuit depth scales constantly with D, as one will need to use a constant amount of gates to implement $e^{-i\beta H_B}$, Equation (6.3), and $e^{-i\gamma H_P}$, Equation (6.1).

6.6.2 Benchmarking

We use the same performance measures for the GSAOA as in Section 5.6.1. However, the objective function here is the objective function from the GSAOA, H_P. For our benchmarking we use the same six instances of the TSP and the same initial points for the parameters as in Section 5.6. This was done for $p \in \{1, \ldots, 20\}$. For the GSAOA, we ran the algorithm with four different initial states for the algorithm, to see if the initial state influenced the effectiveness of the algorithm. Throughout the figures that follow the colors of the lines depict the different initial states for the GSAOA. In green, the starting point is a graph state with no edges. In blue, the starting point is a graph state corresponding to the tour (0213). In red, the starting point is a graph state corresponding to the tour (0132). In purple, the starting point is a graph state corresponding to the tour (0123). Figure 6.6 displays the plot of Equation (5.6), R, against p for four different initial states for the GSAOA. Figure 6.7 displays the plot of

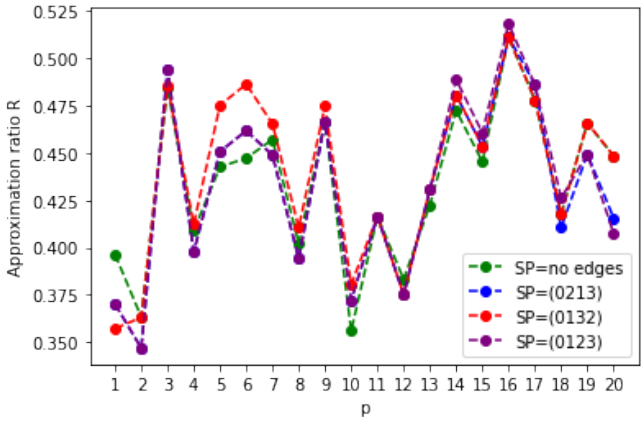


Figure 6.6: The approximation ratio of all solutions, R, plotted against p for four different initial states for the GSAOA.

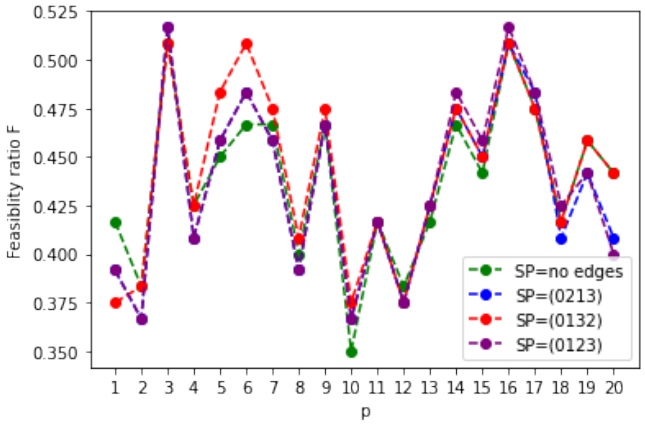


Figure 6.7: The feasibility ratio, F, of all solutions plotted against p for four different initial states.

Equation (5.7), F, against p for four different initial states for the GSAOA. Figure 6.8 displays the plot of the approximation ratio of feasible solutions, A, against p for four different initial states for the GSAOA.

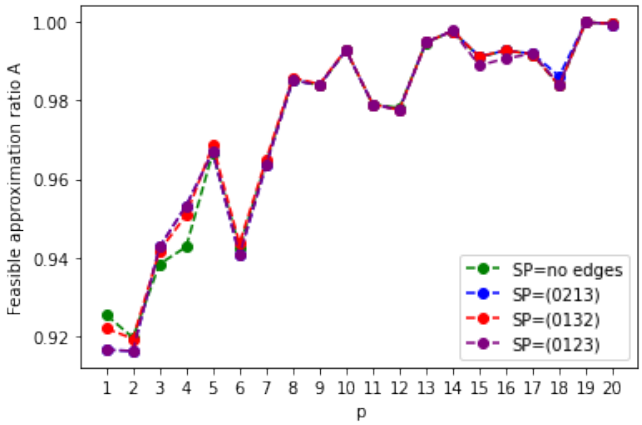


Figure 6.8: The approximation ratio of feasible solutions, A, plotted against p for four different initial states for the GSAOA.

One can see from Figures 6.6, 6.7, and 6.8 that the initial state seems to not have much influence on the success of the GSAOA. The algorithm is about as successful when run with an initial state of an empty graph state as it is when run with an initial state corresponding to a feasible solution. Both R and F do not seem to follow a definite pattern. However, they do have almost the exact same behavior. This is

not that surprising, as the feasibility ratio heavily determines how good the average approximation ratio is. How closely correlated these are could also indicate that we have set the penalty for the hard constraints, p_2, too high compared to the penalty for the optimality constraints, p_1. This is because, if the penalty is too large, there is a huge difference in the objective value, H_P, between feasible and infeasible solutions. This would lead the average approximation ratio to closely follow the behavior of the feasibility ratio, which is what one can see in Figures 6.6 and 6.7. Future work for this algorithm could be to determine better penalty settings. One can see a definite pattern in Figure 6.8. The approximation ratio among feasible solutions, A, steadily increases as p increases, which is promising for the future of the GSAOA.
Now we will benchmark the GSAOA against the algorithms benchmarked in Section 5.6. For this comparison, we choose to use the version of GSAOA with the starting point correpsonding to the tour (0132), seen in red in Figures 6.6, 6.7, and 6.8, as it seems to have slightly better performance than the other starting points. Figure 6.9 displays the plot of Equation (5.6), R, against p for the four algorithms. Figure 6.10 displays the plot of Equation (5.7), F, against p for the for the four algorithms. Figure 6.11 displays the plot of the approximation ratio of feasible solutions, A, against p for the four algorithms. The negative performance of the GSAOA in R, seen in

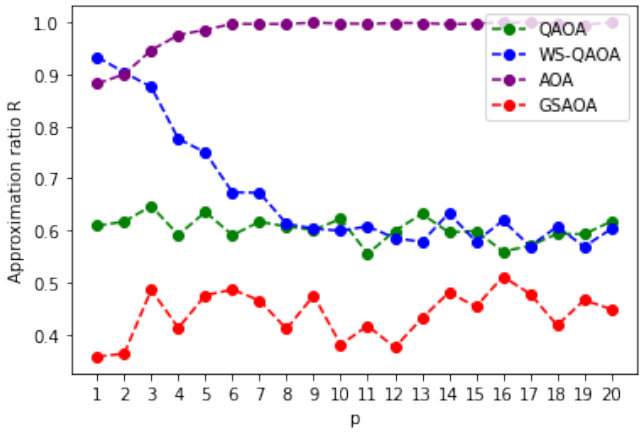


Figure 6.9: The approximation ratio of all solutions, R, plotted against p for the QAOA, the WS-QAOA, the AOA, and the GSAOA.

Figure 6.9, is most likely due to the penalty setting and the fact that the WS-QAOA, the QAOA, and the AOA are using a different objective than the GSAOA. For the feasibility ratio, F, seen in Figure 6.10, the GSAOA performs noticeably better than the QAOA and the WS-QAOA. However, it does seem to remain roughly constant and not be increasing with p. The GSAOA does not perform as well as the AOA, but that is to be expected, since with the GSAOA the solution returned is not guranteed to

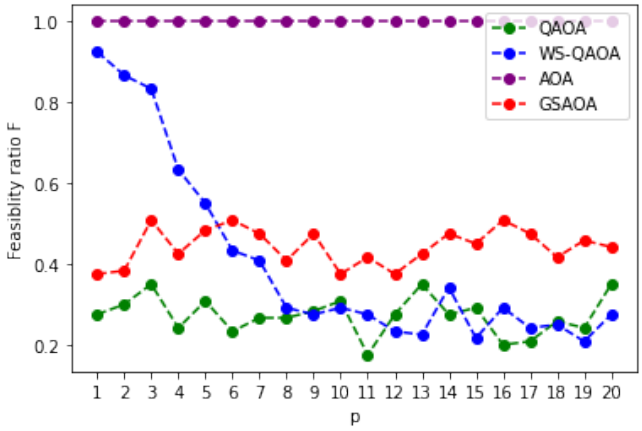


Figure 6.10: The feasibility ratio, F, of all solutions plotted against p for the QAOA, the WS-QAOA, the AOA, and the GSAOA.

be feasible as in the AOA. The GSAOA performs very well, compared to the other algorithms, with regards to the approximation ratio of feasible solutions, A, as seen in Figure 6.11. The GSAOA performs almost as well as the AOA with regards to A, and at higher p values, the GSAOA and the AOA seem to converge to $A = 1$, which is optimal. The behavior of A for the GSAOA is especially noticable in contrast with the QAOA and the WS-QAOA, which did not exhibit this increasing behavior with A. Future work on this algorithm could entail testing the GSAOA at higher values of p to see if F increases with higher p and to see if the GSAOA maintains its convergence for A.
The benchmarking results show that the GSAOA is quite promising as a quantum algorithm for the TSP. Based on this dataset, the GSAOA outperforms the QAOA and the WS-QAOA. Future work on the GSAOA could entail attempting a mixer such as the on in AOA to guarantee the feasibility of the solution. The feasibility ratio seems to be the main reason the GSAOA does not perform quite as well as the AOA.

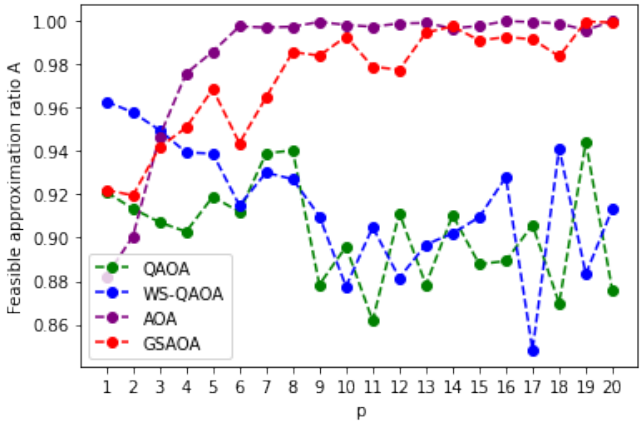


Figure 6.11: The approximation ratio of feasible solutions, A, plotted against p for the QAOA, the WS-QAOA, the AOA, and the GSAOA.

List of Figures

Bibliography

[ACG17] D. Aggarwal, V. Chahar, and A. Girdhar. "Lagrangian relaxation for the vehicle routing problem with time windows". In: 2017, pp. 1601–1606.

[AH92] S. Anily and R. Hassin. "The swapping problem". In: *Networks* 22 (1992), pp. 419–433.

[BCM08] R. Baldacci, N. Christofides, and A. Mingozzi. "An exact algorithm for the vehicle routing problem based on the set partitioning formulation with additional cuts". In: *Math. Program.* 115 (2008), pp. 351–385.

[BHM04] R. Baldacci, E. Hadjiconstantinou, and A. Mingozzi. "An Exact Algorithm for the Capacitated Vehicle Routing Problem Based on a Two-Commodity Network Flow Formulation". In: *Operations Research* 52 (2004), pp. 723–738.

[Ban+12] J. Bang, J. Ryu, C. Lee, S. Yoo, J. Lim, and J. Lee. "A quantum heuristic algorithm for the traveling salesman problem". In: *Journal of the Korean Physical Society* 61.12 (2012), pp. 1944–1949.

[BE20] A. Bartschi and S. Eidenbenz. "Grover Mixers for QAOA: Shifting Complexity from Mixer Design to State Preparation". In: *2020 International Conference on Quantum Computing and Engineering QCE*. IEEE, 2020.

[Ben+21] T. Bennett, E. Matwiejew, S. Marsh, and J. B. Wang. *Quantum walk-based vehicle routing optimisation*. 2021. arXiv: 2109.14907 [quant-ph].

[BE07] A. Billionnet and S. Elloumi. "Using a Mixed Integer Quadratic Programming Solver for the Unconstrained Quadratic 0-1 Problem". In: *Mathematical Programming Computation* 109.1 (2007), pp. 55–68.

[BV97] S. Boyd and L. Vandenberghe. *Communications, Computation, Control, and Signal Processing: a tribute to Thomas Kailath*. Springer US, 1997, pp. 279–287.

[Bra+22] S. Bravyi, A. Kliesch, R. Koenig, and E. Tang. "Hybrid quantum-classical algorithms for approximate graph coloring". In: *Quantum* 6 (2022), p. 678.

[CM99] P. Chalasani and R. Motwani. "Approximating Capacitated Routing and Delivery Problems". In: *SIAM Journal on Computing* 28.6 (1999), pp. 2133–2149.

[CKR01] M. Charikar, S. Khuller, and B. Raghavachari. "Algorithms For Capacitated Vehicle Routing". In: *SIAM Journal on Computing* 31 (2001).

[Chr22] N. Christofides. "Worst-Case Analysis of a New Heuristic for the Traveling Salesman Problem". In: *Carnegie Mellon University* 3 (2022), p. 10.

[EMW21] D. J. Egger, J. Mareček, and S. Woerner. "Warm-starting quantum optimization". In: *Quantum* 5 (2021), p. 479.

[FGG14] E. Farhi, J. Goldstone, and S. Gutmann. *A Quantum Approximate Optimization Algorithm*. 2014.

[Far+00] E. Farhi, J. Goldstone, S. Gutmann, and M. Sipser. *Quantum Computation by Adiabatic Evolution*. 2000.

[Fel+19] S. Feld, C. Roch, T. Gabor, C. Seidel, F. Neukart, I. Galter, W. Mauerer, and C. Linnhoff-Popien. "A Hybrid Solution Method for the Capacitated Vehicle Routing Problem Using a Quantum Annealer". In: *Frontiers in ICT* 6 (2019).

[Fit+21] D. Fitzek, T. Ghandriz, L. Laine, M. Granath, and A. F. Kockum. *Applying quantum approximate optimization to the heterogeneous vehicle routing problem*. 2021. arXiv: 2110.06799 [quant-ph].

[Fuk+06] R. Fukasawa, H. Longo, J. Lysgaard, M. Poggi, M. Reis, E. Uchoa, and R. Werneck. "Robust Branch-and-Cut-and-Price for the Capacitated Vehicle Routing Problem". In: *Math. Program.* 106 (2006), pp. 491–511.

[Ful99] W. Fulton. *Eigenvalues, invariant factors, highest weights, and Schubert calculus*. 1999. arXiv: 9908012 [math].

[GKZ20] A. Glos, A. Krawiec, and Z. Zimborás. *Space-efficient binary optimization for variational computing*. 2020.

[GK18] F. W. Glover and G. A. Kochenberger. "A Tutorial on Formulating QUBO Models". In: *CoRR* abs/1811.11538 (2018).

[GW95] M. X. Goemans and D. P. Williamson. "Improved Approximation Algorithms for Maximum Cut and Satisfiability Problems Using Semidefinite Programming". In: *J. ACM* 42.6 (1995), pp. 1115–1145.

[Gro96] L. Grover. *A fast quantum mechanical algorithm for database search*. 1996. arXiv: 9605043 [quant-ph].

[GS17] G. G. Guerreschi and M. Smelyanskiy. *Practical optimization for hybrid quantum-classical algorithms*. 2017.

[Had+19] S. Hadfield, Z. Wang, B. O'Gorman, E. Rieffel, D. Venturelli, and R. Biswas. "From the Quantum Approximate Optimization Algorithm to a Quantum Alternating Operator Ansatz". In: *Algorithms* 12.2 (2019), p. 34.

[Har+20] R. Harikrishnakumar, S. Nannapaneni, N. H. Nguyen, J. E. Steck, and E. C. Behrman. *A Quantum Annealing Approach for Dynamic Multi-Depot Capacitated Vehicle Routing Problem*. 2020. arXiv: 2005.12478 [quant-ph].

[Hei+17] B. Heim, E. W. Brown, D. Wecker, and M. Troyer. *Designing Adiabatic Quantum Optimization: A Case Study for the Traveling Salesman Problem*. 2017.

[Hei+06] M. Hein, W. Dür, J. Eisert, R. Raussendorf, M. V. d. Nest, and H. Briegel. *Entanglement in Graph States and its Applications*. 2006. arXiv: 0602096 [quant-ph].

[HK61] M. Held and R. M. Karp. "A Dynamic Programming Approach to Sequencing Problems". In: *Proceedings of the 1961 16th ACM National Meeting*. ACM '61. Association for Computing Machinery, 1961, pp. 71.201–71.204. ISBN: 9781450373883.

[HK71] M. Held and R. M. Karp. "The Traveling-Salesman Problem and Minimum Spanning Trees: Part II". In: *Math. Program.* 1.1 (1971), pp. 6–25.

[HS15] I. Hen and F. Spedalieri. "Quantum Annealing for Constrained Optimization". In: *Physical Review Applied* 5 (2015).

[Her+21] R. Herrman, P. C. Lotshaw, J. Ostrowski, T. S. Humble, and G. Siopsis. *Multi-angle Quantum Approximate Optimization Algorithm*. 2021.

[Luc14] A. Lucas. "Ising formulations of many NP problems". In: *Frontiers in Physics* 2 (2014).

[LLE04] J. Lysgaard, A. Letchford, and R. Eglese. "A New Branch-and-Cut Algorithm for the Capacitated Vehicle Routing Problem". In: *Mathematical Programming* 100 (2004), pp. 423–445.

[MSY20] A. Matsuo, Y. Suzuki, and S. Yamashita. *Problem-specific Parameterized Quantum Circuits of the VQE Algorithm for Optimization Problems*. 2020.

[MLM17] D. J. Moylett, N. Linden, and A. Montanaro. "Quantum speedup of the traveling-salesman problem for bounded-degree graphs". In: *Physical Review A* 95.3 (2017).

[NC10] M. A. Nielsen and I. L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, 2010.

[Pat+21] T. L. Patti, O. Shehab, K. Najafi, and S. F. Yelin. *Markov Chain Monte-Carlo Enhanced Variational Quantum Algorithms*. 2021.

[PRW95] S. Poljak, F. Rendl, and H. Wolkowicz. "A Recipe for Semidefinite Relaxation for (0,1)-Quadratic Programming". In: *Journal of Global Optimization* 7 (1995), pp. 51–73.

[Ral+03] T. Ralphs, L. Kopman, W. Pulleyblank, and L. Trotter. "On the Capacitated Vehicle Routing Problem". In: *Mathematical Programming* 94 (2003), pp. 343–359.

[Reg+11] C. Rego, D. Gamboa, F. Glover, and C. Osterman. "Traveling salesman problem heuristics: Leading methods, implementations and latest advances". In: *European Journal of Operational Research* 211.3 (2011), pp. 427–441. ISSN: 0377-2217.

[Rua+20] Y. Ruan, S. Marsh, X. Xue, Z. Liu, and J. Wang. "The Quantum Approximate Algorithm for Solving Traveling Salesman Problem". In: *Computers, Materials Continua* 63 (2020), pp. 1237–1247.

[Sho94] P. Shor. "Algorithms for quantum computation: discrete logarithms and factoring". In: *Proceedings 35th Annual Symposium on Foundations of Computer Science*. 1994, pp. 124–134.

[Sla+21] N. Slate, E. Matwiejew, S. Marsh, and J. B. Wang. "Quantum walk-based portfolio optimization". In: *Quantum* 5 (2021), p. 513.

[Sri+18] K. Srinivasan, S. Satyajit, B. K. Behera, and P. K. Panigrahi. *Efficient quantum algorithm for solving travelling salesman problem: An IBM quantum experience*. 2018. arXiv: 1805.10928 [quant-ph].

[TV02] P. Toth and D. Vigo. "Models, relaxations and exact approaches for the capacitated vehicle routing problem". In: *Discrete Applied Mathematics* 123.1 (2002), pp. 487–512.

[Vid+14] T. Vidal, T. G. Crainic, M. Gendreau, and C. Prins. "A unified solution framework for multi-attribute vehicle routing problems". In: *European Journal of Operational Research* 234.3 (2014), pp. 658–673. ISSN: 0377-2217.

[WHB19] D. Wang, O. Higgott, and S. Brierley. "Accelerated Variational Quantum Eigensolver". In: *Physical Review Letters* 122.14 (2019).

[Wat08] J. Watrous. *Quantum Computational Complexity*. 2008. arXiv: 0804.3401 [quant-ph].