UNIVERSITY OF EXETER

Computer Science Department

MSci. Computer Science
Computer Science Department

Experimenting with the QAOA on the TSP problem

Candidate

Sam Shailer

Student ID 690019000

Supervisor

Dr. Alberto Moraglio

University of Exeter

Academic Year
2022/2023

Abstract

The Traveling Salesman Problem (TSP) is a well-known optimisation problem, with many applications in a variety of different industries. Methods for approximating solutions to the TSP have been around for many years now, but very little progress has been made in finding better solutions recently. The progress of technology and further developments in quantum computing offers a new paradigm for solving optimisation problems like the TSP. An experimental quantum optimisation algorithm that might be promising is the Quantum Approximate Optimisation Algorithm (QAOA). The QAOA uses a hybrid of classical and quantum technology to solve problems. This report and project aims to provide an analysis of the the TSP and feasibility of using the QAOA to solve it.

	Yes	No
I certify that all material in this dissertation which is not my own work has been identified.	☑	☐
I give the permission to the Department of Computer Science of the University of Exeter to include this manuscript in the institutional repository, exclusively for academic purposes.	☑	☐

Contents

List of Figures

List of Tables

List of Algorithms

List of Code Snippets

List of Acronyms

CSV Comma Separated Values

1 Introduction

While the field of Quantum Computing and Quantum Technology is still in its infancy, with the current progress of technology and considerable contributions to the fields of Physics, Mathematics and Engineering we are beginning to see not only the opportunity that Quantum Computing will bring to us but also a glimpse of it coming to realisation. The vast opportunities that Quantum Systems might offer have led to a considerable push for research into the area recently. One of the applications of Quantum Computing greatly anticipated in industries is its potential for solving optimisation problems [14].

Optimisation problems are prevalent in many industries, from shortening production times and cost to finding the best weight of assets in a portfolio, the appearance of these problems is extensive. Unfortunately using current classical computing technology, a lot of these problems have proven to be infeasible to solve exactly for large instances. One such problem is the Traveling Salesman Problem (TSP)[18], which originally proposed the task of finding the shortest route a Salesman should take to visit a number of cities. The TSP has become a rather famous optimisation problem due to its simple nature, numerous applications and extreme difficulty to solve for even a relatively small number of cities. Due to the originally simple nature of the TSP, many derivations of the TSP have been created, such as; the Asymmetric TSP, where a path between cities costs different amounts depending on the direction travelled; the Dynamic TSP, where the cost of paths change with time; and the Vehicle Routing Problem (VRP) where there are multiple salesman/vehicles.

One of the current best solutions to this type of problem is to use heuristic/metaheuristic solvers, which aim to find an approximate answer to the problem in an acceptable amount of time. The solutions of these solvers are only approximate, and as the size of the problem grows, either the time to find a good solution increases or the reliability of the solution decreases.

Quantum Computing might hold the solution to such problems. The Quantum Approximate Optimisation Algorithm (QAOA) uses a set of angular parameters to apply an adiabatic evolution. These parameters are calculated using classical computation. This hopes to use fewer quantum resources. The QAOA then uses a process known as Trotterization [46] to apply the time evolution of the mixing Hamiltonian into the cost Hamiltonian. The QAOA has been shown to perform relatively well in solving optimisation problems [14]. Due to this is has gained appeal in the research space for offering a quantum method for solving combinatorial optimisation problems.

It has therefore been made clear why further research into solving the TSP is important. The purpose of this project and paper is to complete further research into the feasibility of using current quantum systems to improve solutions to the TSP with a focus on the QAOA algorithm. As discussed previously the QAOA algorithm is able to use the QUBO model as an input function. Within this report, I will take different QUBO formulations of the TSP and attempt to solve them using the QAOA algorithm. I will then compare the results found on the QAOA to a classical solver to determine whether the QAOA might be a promising alternative or whether the technological limitations are still too high.

2

Literature Review and Project Report

2.1 The Traveling Salesman Problem

The basic concept of the Traveling Salesman Problem (TSP) is finding the shortest route a salesman can take between a set of cities. Using more mathematical terminology we can describe the problem as a graph $G(N, E)$. G comprises a set of n nodes N corresponding to each city and a set of weighted edges E corresponding to the route between each node. In reality, the weight of an edge would correspond to a number of factors such as time, distance fuel cost or many others depending on the application of the problem. The goal is then to minimise the cost of the edges taken when traversing all nodes in the graph. The definition, history and applications of the TSP are expanded upon in the book titled Travelling salesman problem: Theory and applications by D. Davendra [18]. Proof that the TSP is NP-Complete is usually done via a reduction from the Hamiltonian cycle problem and has been provided by multiple authors, for example, M. Held & R. M. Karp [26] and G. Laporte [34].

Since the TSP in nature is quite an abstract problem, many different variants have been created to better model real-world scenarios. The original version of the TSP is known as the Symmetric TSP (STSP) since the weight of the edge from one node to another is the same both ways. A popular version of the TSP is the Asymmetric TSP (ATSP) which models the TSP where edge might have different costs depending on the direction travelled. This is quite common in practice, with a very simple example being an incline over the edge between nodes leading to higher fuel costs travelling one way than the other. The ATSP is described further by J. Cirasella et al[13]. Another well-known variation of the TSP is the Dynamic TSP (DTSP)[38], where the location of the nodes and edge weights change over time. A real-world example of this would be trying to find the optimal path between a set of ships in the sea. Solving the DTSP is a bit different to solving the STSP or ATSP since it must be broken down into a sequence of static TSP sub-problems[45].

2.1.1 TSP Formulations

As with many optimisation problems the method used to model the TSP has a large impact on the solver's ability to find good solutions. One of the most common ways to model the TSP mathematically is using Integer Programming (IP). This method uses integer variables within the mathematical model to describe the problem. Two of the most popular formulations of the TSP using IP are the Miller, Tucker & Zemlin (MTZ) formulation [41] and the Dantzig, Fulkerson & Johnson (DFJ) formulation [17].

Both the MTZ, DFJ and many other formulations use a relatively standard method for enforcing that the overall cost of the tour is meant to be minimized (eq:2.1), all the nodes have one edge going to them (eq:2.3) and one edge going away from them (eq:2.4). The equations for this model are given below:

$$\min \sum_{i=1}^{n} \sum_{j \neq i, j=1}^{n} d_{ij} x_{ij} : \tag{2.1}$$

$$x_{ij} \in \{0, 1\} \qquad i, j = 1, ..., n; \tag{2.2}$$

$$\sum_{i=1, i \neq j} x_{ij} = 1 \qquad j = 1, ..., n; \tag{2.3}$$

$$\sum_{j=1, i \neq j} x_{ij} = 1 \qquad i = 1, ..., n; \tag{2.4}$$

The main difference between the MTZ and DFJ formulations is their methods of preventing sub-tours in the solution. The MTZ formulation uses a set of integer variables to determine the order the nodes are visited in. It then makes sure that for every edge selected (except for the final edge to the start node) that the subsequent node has a higher order. The DFJ formulation on the other hand uses a list of all possible sub-tours and then makes sure that at least one edge in each sub-tour is not selected.

2.1.2 QUBO Formulations of the TSP

The Quadratic Unconstrained Binary Optimisation (QUBO) model has shown to be able to model many different combinatorial optimisation problems[32]. The main equation of the QUBO model is given below:

$$f(x) = x^T Q x = \sum_{i}^{n} \sum_{j}^{n} Q_{ij} x_i x_j \tag{2.5}$$

In this equation, x is a 1-dimensional matrix of binary variables of length n and Q is a 2-dimensional matrix representing the coefficients of those binary variables and their relationships with each other. It's important to understand that the Q matrix is what predominantly defines the original problem. It does this by attributing weights to each of the relationships between variables, either encouraging the selection of those variables with the objective of minimization or penalizing the cost of the solution with the objective of following constraints. As can be seen from the QUBO equation it is very simple and yet providing the Q matrix is set up correctly offers the ability to solve numerous combinatorial optimisation problems. One of the benefits of using the QUBO model to formulate problems is its portability. Since an entire problem can be formulated into the Q matrix, any solver with the ability to solve QUBOs will be able to solve that formulation. This also allows heuristic QUBO algorithms to be able to solve many different combinatorial optimisation problems.

Another advantage of the QUBO model is the similarity between it and the Ising model used to represent energy systems in quantum mechanics. This makes it relatively easy to convert problems formulated as QUBOs into problems which can be solved using quantum methods such as Quantum Annealing.

Just like with the IP formulations of the TSP, there are a number of different ways to formulate the TSP using the QUBO model. The first formulation, known as native TSP, is widely used for modelling the TSP as a QUBO. It is an extension of the QUBO formulation of the Hamiltonian cycle problem and is discussed fully in the paper by A. Lucas [36]. This formulation uses a set of binary variables $x_{i,j}$, where the value i represents the node and the value j represents the time that node is visited. The native formulation then uses a relatively simple but effective set of penalties to make sure that the optimal solution will be one where all nodes are selected, there are no sub-tours and the cost of the tour will be minimised.

Another method for formulating the TSP as a QUBO is a relatively recent approach known as the GPS formulation [23]. This approach uses a similar method to the MTZ formulation by enforcing a node ordering system. This formulation uses a set of binary variables $x_{i,j,r}$, where the values i and j represent

nodes and $r \in \{0, 1, 2\}$ encodes the relationship between i and j. The benefits of the GPS method over the MTZ QUBO formulation are that the MTZ model requires $N^2 \log_2(N)$ variables, whereas this formulation only requires $3N^2$. The paper introducing the GPS model also claims that the MTZ formulation has been shown to produce inaccurate results. This is allegedly due to the solver getting stuck trying to minimize part of the sub-tour constraints.

2.2 Classical Solvers

2.2.1 Exact Solvers

Although exact solvers may not be efficient in finding solutions for large problems, it's important to understand their approach to solving instances of the TSP. For example, Concorde, an exact solver, takes a reasonably good solution from an approximation algorithm and utilizes it to achieve even better solutions. In a survey conducted by C. Chauhan et al. [11], the main exact solvers discussed were the Branch and Bound method[33], the Cutting Plane method[22], the Branch and Cut method[43], and Dynamic programming[4][5]. The Branch and Bound, Cutting Plane, and Branch and Cut methods use linear programming formulations of the TSP, while Dynamic programming adopts a different approach. Concorde, one of the best exact solution algorithms for the TSP problem, utilizes a combination of these methods.

2.2.2 Hueristic Solvers

Numerous algorithms have been developed to solve the Traveling Salesman Problem (TSP) due to the various mathematical formulations available, as stated in a study by Matai et al. [37]. Since solving large instances using an exact solver is impractical, many heuristic solvers have been developed. Some heuristic solvers have been detailed in the survey by Yahia et al. [49]. The survey highlights several prominent algorithms, including Ant Colony Optimization [19], Particle Swarm Optimization [27][48], Genetic Algorithms [24][35], Tabu Search [31] and Simulated Annealing[30]. Other algorithms in the survey are mainly based on these.

Christofides algorithm [12], developed in 1976, was considered the best algorithm for approximating TSP until recently. It's important to note that while named Christofides algorithm it was also discovered separately by Serdyukov [8] in 1978. To solve the TSP problem, this algorithm first chooses the minimum spanning tree of the graph, and then links all nodes with an odd number of vertices to their closest unlinked node. This algorithm is significant because it provides an approximation ratio of $3/2$, which for a long time was the best-known approximation ratio. Karlin et al have made a recent improvement to this of 10^{-36}, as noted in their publication [29].

2.3 Quantum & Hybrid Solvers

It is shown in research that many of the methods used to solve the TSP are outdated. Although there are still efforts to enhance classical solutions, most new studies tend to focus on specific cases of the problem. For example, the Christofides algorithm has shown that making minor improvements to the problem can be quite time-consuming if approached in the same way. In 1980, Paul Benioff introduced the concept of quantum computing[6], which was further developed by Benioff and Richard Feynman in 1981[7][21]. This developing technology aims to provide an alternative solution to computational problems, such as the TSP.

There are numerous quantum and hybrid algorithms that have been developed, but some stand out as innovative solutions to the TSP and combinatorial optimization problems. Some of the noteworthy algorithms in the field include Quantum Annealing[1], Digital Annealing[42], Coherent Ising Machines[40],

and the Quantum Approximate Optimization Algorithm[20]. These algorithms have been extensively studied and have shown promising results in various applications.

2.3.1 The Quantum Approximate Optimisation Algorithm

Although some of the algorithms mentioned previously have gained more attention and research, the Quantum Approximate Optimization Algorithm (QAOA) is the sole approach that utilizes a conventional quantum gate-based structure for performing computations. This makes it possible to solve optimisation problems using a quantum approach without the need for optimisation-specific hardware. To put it simply, the QAOA algorithm operates as follows:

1. A system is put into a superposition state.
2. Two Hamiltonians are then applied to the system: The mixing Hamiltonian which has a clear minimum energy state, and the cost Hamiltonian, which defines the problem to optimise.
3. Using parameters β and λ, the weighting of these Hamiltonians in the overall system can be varied.
4. Repeating the circuit with new adjusted β and λ values causes the system to gradually transform into the cost Hamiltonian while staying in the minimum energy state. The number of times the circuit is repeated is generally denoted with the variable p.
5. Based on the average final energy reached by the system when measured multiple times, the values of β and λ are then optimised using a classical linear solver.
6. The previous steps are run until a convergence criterion is met.

Testing the effectiveness of the QAOA on large or medium-scale problems is difficult due to the infancy of quantum computing. Nonetheless, research has been conducted on the algorithm's effectiveness, revealing that in certain cases it performs competitively with classical solvers[14]. Furthermore, for a time, QAOA demonstrated a superior approximation ratio to any classical algorithm for a specific problem[20]. However, a more effective classical algorithm was later introduced[3], invalidating this advantage.

2.4 Project Specification

In conclusion, after reviewing the literature, it is evident that the TSP is an important problem that requires solving. It has also been shown that the TSP has an abundance of practical uses. While classical algorithms can provide satisfactory solutions, there has been slow progress in improving their approximation ratios. Therefore, it is imperative to explore different perspectives and alternative approaches, such as quantum computing, to enhance the accuracy of solutions.

2.4.1 Project Aims and Objectives

The main aim of this project is to experiment with solving the TSP using a variety of new methods. These methods include using the Native and GPS QUBO formulations of the TSP and trying to solve them with the QAOA. This will also test the feasibility of solving the TSP using the QAOA using current quantum technology.

2.4.2 Research Questions

- Can the QAOA be used to solve the Native and GPS formulations of the TSP with current quantum technology?
- When it comes to solving these problems, how does the performance of QAOA compare to classical algorithms?

2.4.3 Methodology

After having completed a literature review on the TSP and the current state of algorithms designed to solve it. I will now implement and test the QAOA algorithm using state-of-the-art quantum simulators and hardware available to me. I will then compare the performance of the QAOA to a classical algorithm such as simulated annealing or tabu search on a range of Native and GPS TSP instances of varying sizes and structures. I will also investigate the effect of the QAOA's parameters, including the number of layers and angles, on its performance.

2.4.4 Expected Outcomes

I expect to be able to provide a comprehensive analysis of the performance of the QAOA for solving the TSP on current quantum technology. I will aim to compare the performance of the QAOA to classical algorithms on a range of TSP instances of varying sizes and determine whether the QAOA provides competitive solutions. I will also investigate the effect of the QAOA's parameters, including the β, λ and p values, on its performance. The motivation of this research project is to help contribute to the growing field of quantum computing and its potential applications in optimization problems.

3

Design, Methods and Implementation

3.1 Design

3.1.1 Native TSP QUBO

In the literature review, we were introduced to the Native TSP QUBO formulation. In this section, I will detail the mathematical formulation of the Native TSP from the paper by A. Lucas[36]. In their paper, the TSP problem is first formulated as the Hamiltonian cycle problem (HCP), and then an additional weighting Hamiltonian is added to make the optimal solution relate to the tour with a minimum cost.

Hamiltonian Cycle Problem

First let's consider a graph $G = (V, E)$, where V is a list of vertices, E is a list of edges and N equals the number of vertices in the graph. The formulation used in the paper works for both directed and undirected graphs which means both the STSP and ATSP can be represented with this formulation. The Hamiltonian cycle problem is described as determining whether it is possible to travel between a number of edges in a graph with the purpose of visiting every node in the graph only once and then returning to the start node.

Given a set of all edges in the graph (uv) it is quite easy to see that if the graph is directed then you only need to add (vu) to the edge set in order to model the problem. This formulation uses N^2 bits $x_{v,i}$, where v represents a vertex and i represents the order that the vertex appears in the expected cycle. The constraints for this formulation and their respective mathematical formula are given below:

- *Constraint 1:* Every vertex can only appear once in the solution cycle.

$$c_1 = \sum_{v=0}^{N-1} \left(1 - \sum_{j=0}^{N-1} x_{v,j}\right)^2 \tag{3.1}$$

- *Constraint 2:* For every potential i value in the solution there must be a corresponding i^{th} node.

$$c_2 = \sum_{j=0}^{N-1} \left(1 - \sum_{v=0}^{N-1} x_{v,j}\right)^2 \tag{3.2}$$

- *Constraint 3:* If for the nodes in the expected cycle $x_{u,j}$ and $x_{v,j+1}$ both equal 1, then there should be a penalty if $(uv) \notin E$.

$$c_3 = \sum_{(u,v)\notin E} \sum_{j=0}^{N-1} x_{u,j} x_{v,j+1} \tag{3.3}$$

$$H_A = A(c_1(x) + c_2(x) + c_3(x)) \tag{3.4}$$

The final Hamiltonian for the Hamiltonian cycle problem is therefore shown in equation (3.4). Where $A > 0$ is a constant. I should also note that I have adapted the exact formulation in the paper to account for the fact that array data structures start at 0, so the first vertex is 0. Another note is that when $j + 1 = N \equiv j + 1 = 0$. Since the constraints will only ever be greater than 0 if they're broken, it's easy to see that the system has a ground state where $H = 0$ when none of the constraints are broken. Being in the ground state, therefore, means there must be a Hamiltonian cycle.

TSP

The TSP differs from the HCP as each edge (uv) in the graph G has a weight W_{uv} associated with it. The goal of the TSP is therefore to find a Hamiltonian cycle where the sum of the edge weights in the cycle is minimized. Generally, the TSP assumes the graph is complete however by setting the weight of an edge to infinity it is possible to solve incomplete graphs using this formulation.

To convert the HCP formulation to the TSP formulation, the paper simply adds the Hamiltonian shown in equation (3.5), so $H_{\text{TSP}} = H_A + H_B$.

$$H_B = B \sum_{(uv)\in E} W_{uv} \sum_{j=0}^{N-1} x_{u,j} x_{v,j+1} \tag{3.5}$$

H_B will cause the energy of the system to be minimised when the cost of the cycle is minimised. The constant B must be small enough that it is never favourable to violate the constraints of H_A, the paper therefore sets out a constraint for B as follows: $0 < B\max(W_{uv}) < A$. It is also assumed that $W_{uv} \geq 0$ for each $(uv) \in E$. Both the formulation for the HCP and the TSP use N^2 variables, however, it is shown in the paper to be possible to reduce this to $(N-1)^2$ by fixing the first node. This involves setting the variable where $i = 0$ and $j = 0$ to 1 indicating the $0th$ vertex is first, then setting all variables where $i = 0$ or $j = 0$ to 0.

3.1.2 GPS TSP QUBO

This section shows the design of the GPS formulation from the paper by S. Bermejo et al.[23]. The GPS formulation again uses the same graph G and edge weights W_{uv} as the Native formulation. The first difference is that the GPS formulation uses a different set of binary variables denoted $x_{i,j,r}$. It should be noted that the variables where $i = j$ don't have to be included since we don't want the cycle to stay at the same node. The values of i and j give two vertices in the graph where $i, j \in \{0, ..., N+1\}$, and the value of r encodes the relationship between these two vertices. The encoding for these variables is as follows:

- $x_{i,j,0} = 1$: The edge (i,j) is not in the tour, and the node i is reached earlier than j.
- $x_{i,j,1} = 1$: The edge (i,j) is in the tour, and the node i is reached earlier than j.
- $x_{i,j,2} = 1$: The edge (i,j) is not in the tour, and the node $\boldsymbol{j}$ is reached earlier than $\boldsymbol{i}$.

From the definition of the variables the paper was then able to define the distance travelled through the objective function in equation (3.6).

$$H_D = \sum_{i=0}^{N+1} \sum_{j=0}^{N+1} W_{i,j} x_{i,j,1} \tag{3.6}$$

Below I have defined the constraints for this formulation and their mathematical formula. I should also note that I've adjusted the constraints to represent penalties in the final formulation. This doesn't affect the method they've used in the paper.

- *Constraint 1:* Every vertex can only appear once in the solution cycle.

$$c_1 = \sum_{i=0}^{N+1} \sum_{j=0}^{N+1} \left(1 - \sum_{r=0}^{2} x_{i,j,r}\right)^2 \tag{3.7}$$

- *Constraint 2:* Each node must be exited once.

$$c_2 = \sum_{i=0}^{N} \left(1 - \sum_{j=0}^{N+1} x_{i,j,r}\right)^2 \tag{3.8}$$

- *Constraint 3:* Each node must be reached once.

$$c_3 = \sum_{j=1}^{N+1} \left(1 - \sum_{i=0}^{N} x_{i,j,r}\right)^2 \tag{3.9}$$

- *Constraint 4:* If vertex i is reached before vertex j, then vertex j is reached after vertex i. This only has to be specified for $r = 2$ due to constraint 1. It also does not apply when $i = j$.

$$c_4 = \sum_{i=0}^{N+1} \sum_{j=0}^{N+1} \left(1 - x_{i,j,2} - x_{j,i,2}\right)^2 \tag{3.10}$$

- *Constraint 5:* This constraint is used to prevent sub-tours. It is only valid if vertex i is reached before vertex j, vertex j is reached before vertex k and vertex i is reached before vertex k. Excluding the cases where $i = j$, $i = k$ and $j = k$.

$$c_5 = \sum_{i=1}^{N} \sum_{j=1}^{N} \sum_{k=1}^{N} (x_{j,i,2} x_{k,j,2} - x_{j,i,2} x_{k,i,2} - x_{k,j,2} x_{k,i,2} + x_{k,i,2}) \tag{3.11}$$

$$H = B \times H_D + A(c_1 + c_2 + c_3 + c_4 + c_5) \tag{3.12}$$

Equation (3.12) describes the final Hamiltonian of the QUBO formulation. The values of the constants A and B should have the same effect as they do on the Native formulation. This also means the same constraints apply for A and B.

This formulation uses $3N^2$ variables, which as previously stated is an improvement over the MTZ formulations $N^2 \log_2(N)$. Another benefit of using the GPS formulation, as shown in the paper, is that adding further constraints makes it possible to model the Vehicle Routing Problem (VRP) using the same number of variables. It is also possible to reduce the number of variables used in the GPS formulation when solving the TSP by removing the variables where $i = j$. This yields a final number of variables for the GPS as $3N^2 - 3N$.

For both the Native and TSP approach, we're multiplying the distance Hamiltonian by a constant B. This means the energy of the optimal solution will be equal to B multiplied by the cost of the minimum tour.

3.1.3 Conversion from QUBO to Ising model

The Ising model was originally proposed by William Lenz in 1920 and later developed by Ernest Ising in 1925[9]. One example of its use is to model a lattice of magnetically charged particles in space. Its

purpose is to represent the interaction between the spin of these particles and an external magnetic field acting on the particles themselves. The spin of a particle in the Ising model is represented by a variable with values $\sigma_i \in \{-1, +1\}$. The energy of the system can be given by the following Ising Hamiltonian:

$$H(\sigma) = -\sum_i \sum_j J_{ij}\sigma_i\sigma_j - \mu \sum_j h_j\sigma_j \tag{3.13}$$

Where $J_{i,j}$ represents the interaction between neighbouring particles, h_j represents the interaction of an external magnetic field with particle j. The magnetic moment is given by μ.

It is then relatively easy to convert the Ising model to the QUBO model by doing the following replacement, $\sigma_i = 2x_i - 1$.

3.1.4 QAOA

In the literature review section, I gave a brief overview of the QAOA algorithm. Now I would like to provide a more detailed explanation of its design and a bit about why it works.

The QAOA is a variational algorithm, which means that it attempts to find the minimum of a cost function by optimising a set of variational parameters. The QAOA uses a unitary operator $U(\beta_p, \gamma_p)$ with variational parameters (β_p, γ_p), to prepare a quantum state $|\psi(\beta, \gamma)\rangle$. The goal of the QAOA is to find the optimal values of (β_p, γ_p) such that the final quantum state $|\psi(\beta, \gamma)\rangle$ encodes the solution to the problem. Where the ground state of $|\psi(\beta, \gamma)\rangle$ represents the optimal solution to the problem, and the probability of measuring the ground state is maximised.

The unitary operator $U(\beta_p, \gamma_p)$ is comprised of two unitary operators $U(\beta_p) = \exp^{-i\beta H_B}$ and $U(\gamma_p) = \exp^{-i\gamma H_P}$, where H_B is the mixing Hamiltonian and H_P is the problem Hamiltonian. The quantum element of the QAOA works by applying alternating blocks, where $U(\beta_p)U(\gamma_p)$ is a block, to an initial superposition state $|\psi_0\rangle$, p times. The formula for this is given below:

$$|\psi(\beta, \gamma)\rangle = \underbrace{U(\beta_1)U(\gamma_1)...U(\beta_p)U(\gamma_p)}_{\text{p times}} |\psi_0\rangle \tag{3.14}$$

This process uses an approach known as Trotterization[46]. The cost Hamiltonian is typically a complex and difficult Hamiltonian to model, Trotterization is used to split the cost and mixing Hamiltonians into a sum of simpler Hamiltonians. It is then possible to use the simpler Hamiltonians to construct a sequence of unitary operators, which approximate the time evolution of the full Hamiltonians.

The values of β and γ change the strength of the time evolution. If they are large then the initial state will rapidly take on the attributes of the cost and mixing Hamiltonians. Doing so may lead to better performance in modelling of the problem in the final quantum state, however, it can also lead to more errors and due to the increase in energy might cause the system to jump out of the ground state. If the parameters are too small then the final quantum state may not represent the optimisation problem very well. By optimising the values of β and γ, the algorithm attempts to reduce errors and keep the system in its ground state while still modelling the problem relatively well.

This also accounts for why it's recommended to apply multiple blocks of the unitary operators. By varying the values of β and γ between layers it is possible to model more of the solution space, produce more complex states and capture intricate features of the problem in the final quantum state. A higher value of p leads to more depth in the circuit, which has the advantage mentioned previously but comes at the cost of adding further complexity to the circuit.

A good way of viewing the QAOA in comparison to classical optimisation algorithms is similar to surrogate optimisation. Both attempt to build an approximate model for the optimisation problem to more efficiently traverse the solution space and approximate an optimal solution. This also shows why the QAOA is an approximation algorithm and offers no guarantee of finding the optimal solution. It can be said however that as $p \to \infty$, QAOA is guaranteed to find the optimal solution.

3.1.5 Experimental Setup

This section details the design of the experimental setup for the project.

Software

All experiments conducted in this project have been run using Python 3.10.5, this is due to the vast amount of libraries available and useability. The program files have been edited, run and debugged through VS Code. The following libraries have been used for experimentation: Matplotlib [28], Networkx [25], Pandas [39], Qiskit [44], D-Wave Ocean [15], Scipy [47].

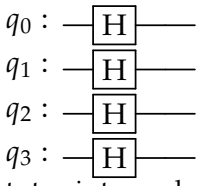
Hardware

Locally I have had access to an 8th gen Intel i7 cpu, GTX 1080 graphics card and 16GB or RAM. This is the hardware that's been used to run the Qiskit "qasm-simulator", which is able to simulate quantum systems on a classical architecture. I decided to use this simulator as it integrates easily with other Qiskit functionality, such as building and solving quantum circuits and using the IBMQ api to run circuits using IBMs systems.

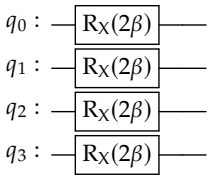
For external hardware I have had access to IBMs free tier quantum systems. These include gate-based quantum systems up to 7 qubits, and simulators up to 32 qubits. I chose to use IBMQ as it offers a number of systems with their free tier and unlimited access to those systems.

QAOA Circuit Design

The main premise of the QAOA circuit design used in this project comes from the Qiskit textbook [2]. In the textbook, the QAOA has been implemented to solve the Max-Cut problem, however, when understanding the functionality of the QAOA it is not too difficult to adapt this circuit to solve QUBO problems. The circuits shown below all use only 4 variables and have a p value of 1. If the number of variables increases, it's easy to see how the circuit can extend. For different p values we can also just apply the mixing unitary circuit and cost unitary circuit p times. The first step in the creation of the QAOA circuit is to place the qubits in the initial state, the circuit for this is given in the textbook as:

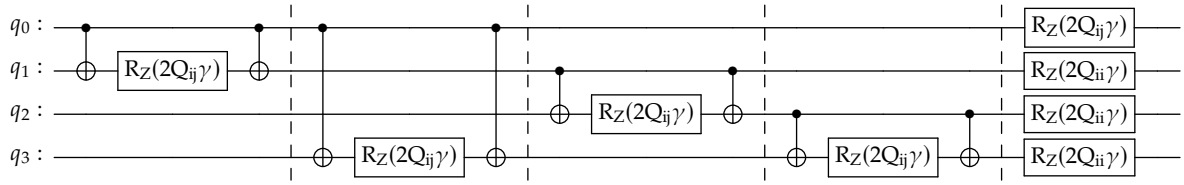


The next step is to apply the mixing unitary. In the textbook, the circuit describing the mixing unitary is as follows:

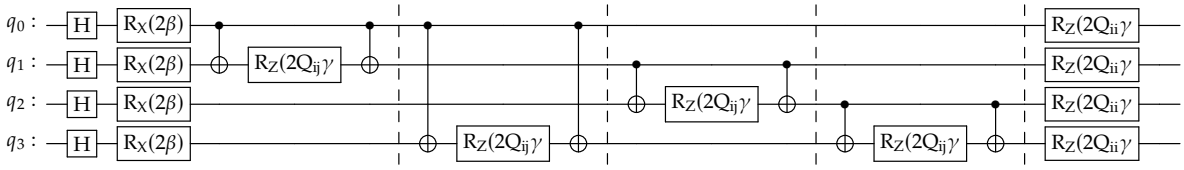


The final component of the circuit implements the cost unitary for a QUBO. In the textbook, the implementation of the Max-Cut problem uses the RZZ gate to entangle the phase of two qubits. This is used to represent the interaction between two nodes in the graph. Since we're looking at solving the QUBO formulation, we can use the same gate to represent the interaction between non-diagonal elements in the Q matrix, where $i \neq j$. When prototyping this project I also found that the RZZ gate was not implemented on all of the IBM machines. In my design, I've opted to use the decomposition of the RZZ gate, which is the following combination of gates CNOT RZ CNOT, as these gates are implemented on a broader range of machines. What the RZZ gate does is entangle 2 qubits and then change the phase of the qubits only when they're in the state $|11\rangle$. This represents the Q_{ij} value. The RZ gate can be used to apply a phase change to the state $|1\rangle$ of a qubit. I can, therefore, apply just an RZ gate to each qubit

in order to represent the diagonal terms of the Q matrix, where $i = j$. It's important to note that the Q matrix must be converted to its Ising form when creating this circuit as the value of each qubit will be either -1 or $+1$. The circuit for the cost unitary of a QUBO is as follows:



The complete circuit for the QAOA implementation designed to solve QUBOs is, therefore, given below:



Calibration

A lot of the parameters in this project were varied for testing purposes. There is one parameter relating to the QUBO formulations that was kept constant.

When formulating both the Native and GPS QUBO formulations they both used constants A and B in order to enforce penalties while maintaining that the shortest path will have the lowest energy. Due to the large number of parameters to be varied in the testing phase, I decided to use a value of B based on the implementation of the native formulation in the D-Wave ocean package [16]. In this package, they calculated a Lagrange point which was an estimate of the average tour length. Below is the equation for B both used in D-Wave's and my implementation:

$$B = \frac{(\sum_i^N \sum_j^N W_{ij}) \times N}{|E|} \tag{3.15}$$

where N is the total number of vertices in the graph, $|E|$ is the total number of edges and W_{ij} is the weight of edge (i, j).

Due to the constraint that $0 < B \max(W_{uv}) < A$, I decided to set the value of A to be $A = B \max(W_{uv})+1$. Doing so will make sure the constraint is always valid.

Execution

The execution of tests was conducted on both the local and external quantum systems. These tests used standard measurement techniques and protocols.

3.2 Methods

3.2.1 Classical Algorithms for Comparison

In order to compare the performance of the QAOA algorithm I selected a classical optimisation algorithm to also run the QUBOs on. The algorithm I selected for testing was Simulated Annealing, due to its similarity to the adiabatic nature of the QAOA.

I decided to use the D-Wave samplers package as it has built-in functionality for Simulated Annealing. I was also using the D-Wave BQM model for representing QUBOs which made the process of solving the QUBOs using their samplers a lot simpler.

3.2.2 Pre-Processing and Post-Processing

In the design of the Native and GPS QUBO formulations, it is easy to see that the variable encoding schemes are not the same. The variables are also stored in a 1-dimensional array meaning that indexing has to be linear. For this reason, it was necessary to implement formulation-specific encoding schemes. These encoding schemes also provide a method for decoding the solution variables as will be shown in the implementation.

The encoding scheme for the Native formulation is as follows:

$$x_{i,j} = x_k \tag{3.16}$$

$$where\ k = (i \times N) + j \tag{3.17}$$

The encoding scheme used for the GPS formulation is as follows:

$$x_{i,j,r} = x_k \tag{3.18}$$

$$where\ k = \begin{cases} (i \times N) + (j \times |r|) + r - (|r| * (i)), & \text{if } i > j \\ (i \times N) + (j \times |r|) + r - (|r| * (i+1)), & \text{if } i < j \end{cases} \tag{3.19}$$

In the case of the GPS formulation, the value of $|r|$ would be 3. The GPS encoding scheme also assumes the removal of the variables where $i = j$ as they're not necessary for the TSP problem and doing so helps to reduce the number of variables used.

3.3 Implementation

The first step when implementing the project was deciding which programming language to use. While languages like C offer very high performance, I decided to use Python due to its ease of implementation and vast package library. A lot of these packages also implement a C backend, so performance could still be maintained.

3.3.1 TSP QUBO Formulations

Since both formulations would be QUBOs, a lot of functionality around handling these QUBOs could be shared. For this reason, I decided to use an Object-Oriented Programming (OOP) paradigm when implementing the different formulations. Another benefit of using OOP is the reusability of code.

Since a number of the functionality around handling different QUBO formulations would overlap, I decided to implement an abstract class "TSPBQMFormulator". This class implements all common methods between the formulations and then has a number of abstract methods which must be implemented by a child class when inheriting "TSPBQMFormulator". I then created two child classes, the "NativeBQMFormulator" class and the "GPSBQMFormulator" class which implement the functionality of their individual formulation. The class diagrams for these are shown in figure **??**.

In the implementation of the QUBO formulations, I decided to use the D-Wave Dimod package. This is due to the ease of modelling quadratic functions using their built-in BinaryQuadraticModel datatype and extensive functionality surrounding this class. Using the BQM class it was possible to create lists of binary variables and then apply the same exact mathematical models as presented in the design section to implement the different QUBO formulations and store them as a BQM. It should also be noted that the BQM model is very similar to the QUBO model but also includes with the model the offset value for the energy. The BQM class has built-in functions for converting from a BQM to a QUBO so reverting to this format later would be simple.

As mentioned in the methods section, I decided to use the D-Wave samplers package to solve the BQMs using different classical algorithms. I also implemented the solve_bqm() function, in the

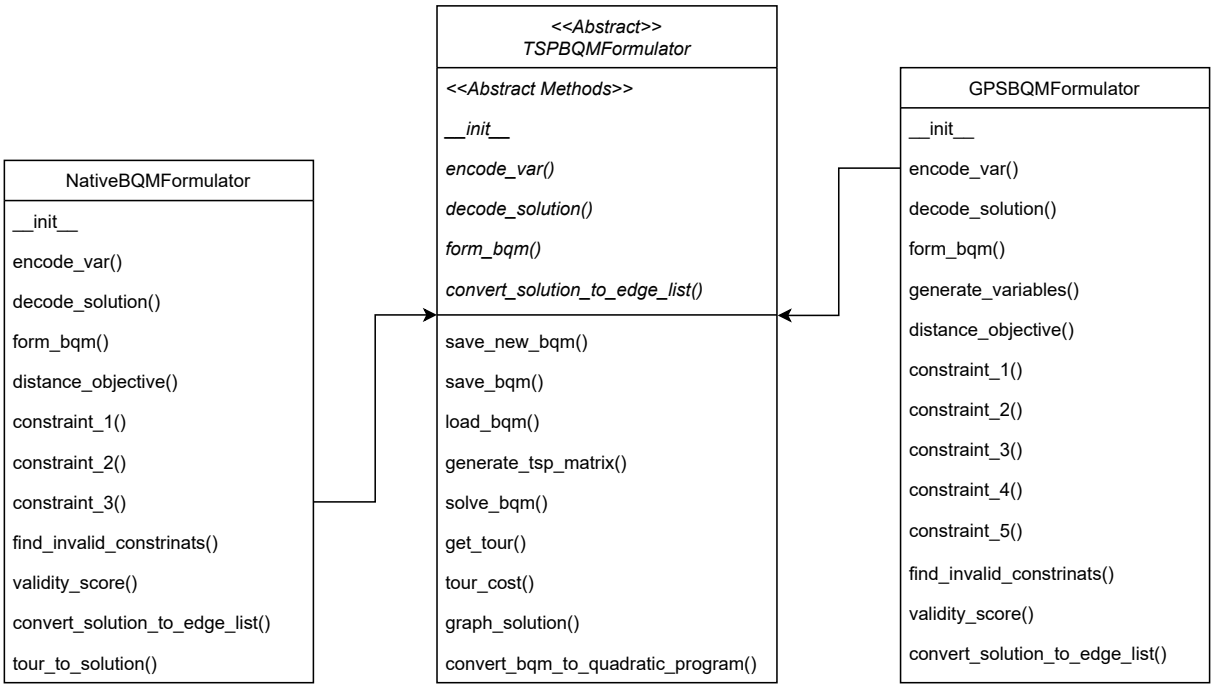


Figure 3.1: QUBO Implementation Class Diagram

"TSPBQMFormulator" class, to take a "dimod.Sampler" object as a parameter which is then used to solve the BQM. This makes it a lot easier to change which algorithm is being used to solve the BQM, while also making it simple to use other algorithms in the D-Wave samplers package if future research using this project was undertaken.

3.3.2 Native Formulation

The implementation of the Native formulation was relatively simple, and only required adapting the mathematical model into Python. The main difference with this is the use of "for-loops" in the place of summations.

The entire formulation of the BQM is done within the form_bqm(), distance_objective(), constraint_1(), constraint_2() and constraint_3() functions, with the name of each function representing which part of the problem it formulates. The reason I decided to split these into separate functions was to make it possible to test each component separately. It also means that the constraints could be used elsewhere in the future. Other functions within the "NativeBQMFormulator" class, just offer additional functionality for testing purposes.

As described in the design section its possible to reduce the number of variables used to $(N-1)^2$ by setting the values of the variables so that the $0th$ node must come first. I implemented this in the form_bqm() function by using the bqm.fix_variables() function which is part of the Dimod BQM class. This function sets the values of variables in a BQM object to a specified value, updates the necessary coefficients and offsets for the model and then removes the variables. The function maintains the labeling of the variables so I then remapped the labels to be linear values from 0 to $(N-1)^2$.

3.3.3 GPS Formulation

The implementation of the GPS formulation turned out to be significantly harder than expected. Initially, I implemented the mathematical models used in the original paper directly. When running a few preliminary tests however this led to completely incorrect solutions being found. It was evident something was wrong with the formulation since the optimal solution should have an energy value equal to the cost of the shortest cycle. With the original implementation however breaking certain constraints led to a lower energy than a solution without any broken constraints.

Initially it was possible to tell constraints were broken through a process of plotting a graph of the entire TSP graph and solution cycle. This was done using the graph_solution() function I had implemented in the "TSPBQMFormulator" class. This wasn't very practical for finding the exact constraints that were broken and which variables were breaking them. I, therefore, implemented the find_invalid_constraints(), and validity_score() functions, to return the number of constraints invalidated, which constraints were invalidated and which variables caused the constraints to be broken.

The next stage involved looking over the mathematical models of the constraints carefully, adapting my implementation of the constraints and then using the functions mentioned previously to check the solutions' energy values and if any constraints were being broken. For this, I also created a known valid solution to the problem by running solving the Native formulation on the same TSP instance and converting the Native solution into a GPS solution.

As a result of my experiments, I would like to update the mathematical formulation of the GPS formulation used in this project. The implementation of the GPS in this project also follows these models directly. The new formulations are given below:

- *Distance Objective:*

$$H_D = \sum_{i=0}^{N-1}\sum_{j=0}^{N-1} W_{i,j}x_{i,j,1} \tag{3.20}$$

- *Constraint 1:* Every vertex can only appear once in the solution cycle.

$$c_1 = \sum_{i=0}^{N-1}\sum_{j=0}^{N-1}\left(1 - \sum_{r=0}^{2} x_{i,j,r}\right)^2 \tag{3.21}$$

- *Constraint 2:* Each node must be exited once.

$$c_2 = \sum_{i=0}^{N-1}\left(1 - \sum_{j=0}^{N-1} x_{i,j,r}\right)^2 \tag{3.22}$$

- *Constraint 3:* Each node must be reached once.

$$c_3 = \sum_{j=0}^{N-1}\left(1 - \sum_{i=0}^{N-1} x_{i,j,r}\right)^2 \tag{3.23}$$

- *Constraint 4:* If vertex i is reached before vertex j, then vertex j is reached after vertex i. This only has to be specified for $r = 2$ due to constraint 1. It also does not apply when $i = j$.

$$c_4 = \sum_{i=1}^{N-1}\sum_{j=i}^{N-1}\left(1 - x_{i,j,2} - x_{j,i,2}\right)^2 \tag{3.24}$$

- *Constraint 5:* This constraint is used to prevent sub-tours. It is only valid if vertex i is reached before vertex j, vertex j is reached before vertex k and vertex i is reached before vertex k. Excluding the cases where $i = j$, $i = k$ and $j = k$.

$$c_5 = \sum_{i=1}^{N-1}\sum_{j=1}^{N-1}\sum_{k=1}^{N-1}(x_{j,i,2}x_{k,j,2} - x_{j,i,2}x_{k,i,2} - x_{k,j,2}x_{k,i,2} + x_{k,i,2}) \tag{3.25}$$

With these constraints, the new range of values i and j can take is $i, j \in \{0, ..., N-1\}$. Where N is the total number of vertices in the graph. Using these constraints applies the correct coefficients to the objective function so that the minimum energy occurs when the cost of the solution cycle is minimised and none of the constraints are invalidated.

3.3.4 QAOA

In the implementation of the QUBO formulations section, I stated that the solve_bqm() function solves a BQM by utilizing a solver from the "dimod.Samplers" class. Since there is no Dimod QAOA sampler, I decided to create a "QiskitQAOASampler" class which extends the "dimod.Samplers" class, for my

implementation of the QAOA algorithm. While this might add more complexity to the implementation of the QAOA algorithm, in the scope of the project, maintaining standardised formats where possible helps to minimise the overall complexity. It also makes the implementation clearer, which helps greatly when debugging. Implementing the QAOA as a class also makes it widely reusable, for use in potential future applications. The class diagram for the "QiskitQAOASampler" class is given in figure (3.2).

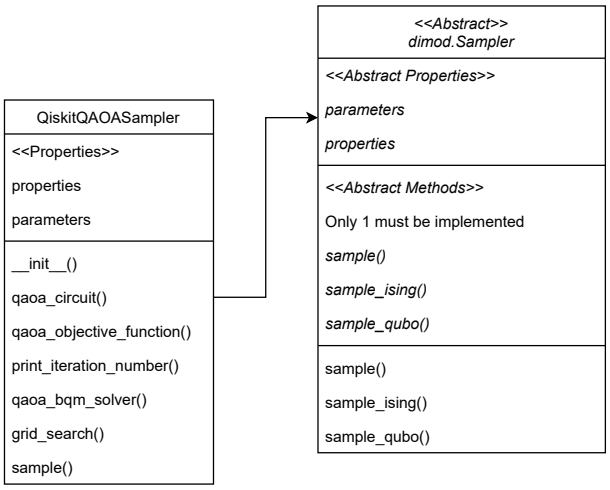


Figure 3.2: QAOA Implementation Class Diagram

For the implementation of the QAOA algorithm I also decided to use the Qiskit package. The Qiskit package offers a wide range of functionality for creating and running quantum circuits. It also provides support for running quantum circuits on both Qiskits' Aer quantum simulator and IBMQ machines.

The main method I needed to implement from the "dimod.Samplers" class is the sample() method. This method takes a BQM object and solves it using the specified algorithm (in this case the QAOA), it then returns a "dimod.Sampleset" object which contains a set of solutions to the problem from different samples. A sample in the "dimod.Sampleset" object consists of a solution, the solution energy and the variable type of the solution. The implementation of the QAOA must therefore return these values. Another advantage of inheriting the "dimod.Sampler" class is that it has functionality for sampling Ising models and QUBOs, it does this by converting the Ising model or QUBO to a BQM and then running the sample() method.

The first function for my implementation of the QAOA algorithm is the qaoa_bqm_solver() function. This function is implemented to run each stage of the QAOA algorithm and then return the optimal solution found. This function takes in a BQM with binary variables and generates a BQM with spin variables using a built-in method of the BinaryQuadraticModel. The initial β_p and γ_p are then selected from a random uniform distribution between 0 and 2π. I then use the SciPy minimize function to optimise the parameters for the QAOA algorithm. The SciPy minimize function is a fast, flexible optimisation function that implements a range of optimisation methods. The SciPy uses the qaoa_objective_function() method to calculate a fitness value for the selected parameters and try to vary them to improve this value. When the optimal parameters have been found, the circuit is run once again to find the best solution.

The qaoa_objective_function() calls the qaoa_circuit() method to create a quantum circuit and then executes the circuit on a user-specified backend. The results of the execution are stored and the average solution energy is returned from the function.

The qaoa_circuit() function creates the circuit for the QAOA. It creates a Qiskit QuantumCircuit() object with the relevant number of qubits. Using the QuantumCircuit class it is then possible to apply the gates as shown in the experimental set-up of the QAOA to apply each layer of the circuit. The quadratic and linear coefficients of the BQM are easily accessed using the bqm.quadratic.items() and bqm.linear.items() method.

Finally it's important to discuss the selection of the backend used to execute the quantum circuit. The __init__() function has been implemented to allow the user to set the backend. The backend options are

as follows: 'qasm_simulator', which is set as default, will use Qiskit Aers qasm_simulator, the exact name of the IBMQ machine to use or 'least_busy' which will find the least busy IBMQ machine. **Important:** before using the IBMQ machines you must save and load the API key for your IBMQ account. This can be done at the top of the "QiskitQAOASampler" class or by running the following code:

```
from qiskit import IBMQ
IBMQ.save_account('YOUR_API_KEY')
IBMQ.load_account()
```

Code 3.1: Saving IBMQ API Key

3.3.5 Experiments and Benchmarks

For running experiments and benchmarks, I implemented a simple test program. This program provides a few functions to run each experiment while automatically changing parameters and storing the results.

4

Project Methods and Results

4.1 TSP Instances

Before any testing could be completed it's important to describe the TSP instances used. To begin with, I should specify that the TSP instances are all relatively small, between 3 and 5 nodes. The reason for this is that the hardware currently available to run the QAOA takes an exceedingly long time. Currently, any instances above 5 nodes would take too long to test with significance.

The TSP instances I tested on are as follows:

- **tsp_3_0** A TSP instance I generated myself, with 3 nodes. The optimal tour cost of this instance is 12.
- **tsp_4_0** Another TSP instance I generated myself, with 4 nodes. The optimal tour cost of this instance is 18.
- **five_d** This TSP instance is from the Florida State University, Travelling Salesman Problem (TSP) Data[10]. The optimal tour cost of this instance is 19.

I couldn't find any libraries with 3 and 4 node instances as they're quite trivial to solve. This meant generating my own. I then calculated the optimal tour cost for these instances by hand.

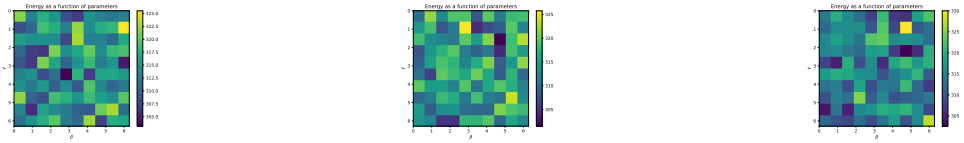
4.2 Testing the QUBO Formulations

In order to test the different QUBO formulations, I ran each of the instances 10 times on each model and then collected their average data. For each run of the instance, I used the simulated annealing algorithm. The simulated annealing algorithm was set up to run 10 times with 10000 iterations, and then the best solution from these runs was taken as the current solution for that instance. The data I decided to collect was: the average time taken to run the instance, the average cost of the instance, the average energy of the instance and the average number of constraints broken in the formulation of the instance (validity).

The results from these experiments are shown in Table (4.1). We can see from the results that for all instances the Native model takes less time to solve, generally finds a lower energy and doesn't violate any

Model	Instance	Time(ms)	Cost	Energy	Validity
GPS	five_d	3.172000	22.000000	26.020833	0.500000
	tsp_3_0	1.856000	12.000000	12.000000	0.000000
	tsp_4_0	2.248500	17.800000	21.825000	0.500000
Native	five_d	2.004300	21.800000	21.800000	0.000000
	tsp_3_0	1.155300	12.000000	12.000000	0.000000
	tsp_4_0	1.271800	18.800000	18.800000	0.000000

Table 4.1: QUBO Formulation Test



(a) Energy Landscape 1: Shots = 1000 (b) Energy Landscape 2: Shots = 1000 (c) Energy Landscape 2: Shots = 1000

Figure 4.1: Energy Landscapes: Method=Native, Instance=tsp_3_0, Shots=1000

constraints. The GPS model performs relatively well on the smaller problems but it's already apparent that as the size of the instance grows so does the chance for constraints to be violated and sub-optimal tours to be found.

The reason that the GPS model performs worse is most likely due to a couple of reasons. Firstly, there are more variables and the number of variables grows faster than the Native model. This leads to a larger search space and makes it harder to find optimal solutions. Secondly, the objective function of the model has more quadratic terms. This can lead to greater hamming distances in the solution space and make it much harder to converge upon the optimal solution.

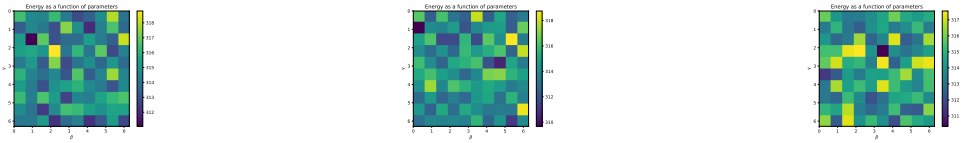
4.3 Testing the QAOA

Unless stated otherwise the following tests are run on Qiskits qasm_simulator.

4.3.1 Testing Angle Parameters

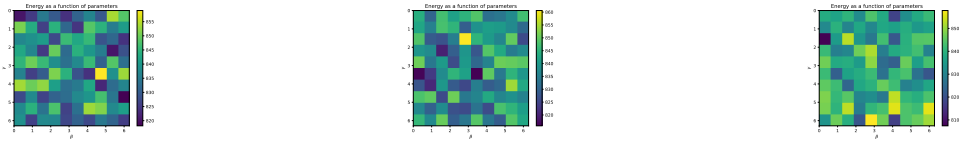
As discussed in the design section some of the most important parameters of the QAOA are the β and γ values. The following test is utilised by Google in its implementation of the QAOA. They use a grid search method to determine the energy of a range of parameter values and then plot the results as a colour map. This helps to show the landscape of the parameter optimisation.

To set up this test we use a p value of 1 so that we're only looking at 2 parameter values. I also decided to develop upon Google's test by applying it 3 times for 1000 shots and 10000 shots. Doing this will allow us to see whether the energy landscape is affected by quantum noise and if so whether increasing the number of shots helps to reduce this effect. Due to the time it takes to run the QAOA this test was only run on the 3-node and 4-node tsp instances. It was run using both the Native and GPS model to see if the model also had any effect on the energy fitness landscape.



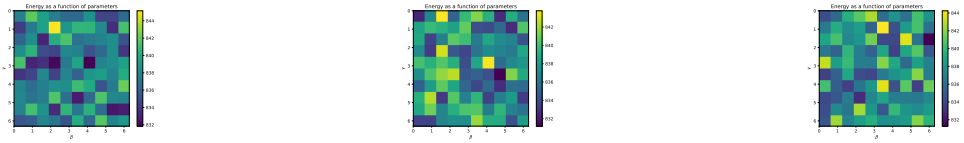
(a) Energy Landscape 1: Shots = 10000(b) Energy Landscape 2: Shots = 10000 (c) Energy Landscape 2: Shots = 10000

Figure 4.2: Energy Landscapes: Method=Native, Instance=tsp_3_0, Shots=10000



(a) Energy Landscape 1: Shots = 1000 (b) Energy Landscape 2: Shots = 1000 (c) Energy Landscape 2: Shots = 1000

Figure 4.3: Energy Landscapes: Method=GPS, Instance=tsp_3_0, Shots=1000



(a) Energy Landscape 1: Shots = 10000(b) Energy Landscape 2: Shots = 10000 (c) Energy Landscape 2: Shots = 10000

Figure 4.4: Energy Landscapes: Method=GPS, Instance=tsp_3_0, Shots=10000

We can see that in figures (4.1) - (4.4), the energy landscape was very noisy. We can also see that the energy landscape was not consistent over multiple runs, which indicates the noise was due to quantum interference. It also isn't apparent that increasing the number of shots helped to decrease the level of noise.

4.3.2 Testing P Values

To test the effect of the p value I used 100 iterations, 1000 shots and the 'Nelder-Mead' search method as other parameters to the QAOA. This test is used to see how varying the p-value affects the time taken to run the algorithm, the energy of the solution found and the validity of that solution. This test was run on the 3-node and 4-node instances. Each value of p was only run once due to time constraints. In this test, I used p values of 3 and 11.

In table (4.2) we can see the results of the test. These results show that for the Native formulation of a 3-node problem, the p-value does not seem to have a significant impact on the solution found. With the GPS formulation of the 3-node problem, there appears to be a slight improvement with a larger p-value.

	Model	Instance	p	Time(ms)	Cost	Energy	Validity
1	Native	tsp_3_0	3	8534.007000	12.000000	12.000000	0.000000
2	Native	tsp_3_0	11	26867.112000	12.000000	12.000000	0.000000
3	GPS	tsp_3_0	3	44632.667000	12.000000	36.333333	4.000000
4	GPS	tsp_3_0	11	123478.407000	8.000000	20.166667	2.000000

Table 4.2: Testing p values on QAOA

4.3.3 Comparison of QAOA

Using the parameters found in the tests above (iterations=100, shots=1000, method='Nelder Mead', p=3), I then tested the QAOA on solving the Native formulation of the 5-node TSP instance. This was only run once for each formulation method due to time constraints. The results are shown in table (4.3).

Looking at the results of this test and the previous one we can partially compare the performance of the QAOA algorithm to that of the simulated annealing algorithm. It should be noted due to the QAOA tests only being run once they are not statistically sound however they can still provide insight on the difference between the algorithms.

Comparing these two algorithms we can see that currently, simulated is more powerful in almost every aspect. Since quantum systems are still highly experimental the QAOA is not able to perform very well compared with an alternative classical optimiser using a quantum simulator.

	Model	Instance	p	Time(ms)	Cost	Energy	Validity
1	Native	five_d	3	82127.823000	0.000000	28.083333	2.000000

Table 4.3: Testing the QAOA on an 5 node instance

4.3.4 Running the QAOA on an actual Quantum Machine

While the technology is very limited, I was able to run the QAOA algorithm on one of IBM's quantum machines. The specific machine was 'ibmq_quito'. Due to restrictions in circuit size, I was only able to run the 3-node instance using the Native model. I also used the same parameters as the previous test. The circuit took far longer to run not because of the compute time but due to the queue time of each job on the machine. Below are the results of the test.

- *Solution Energy:* 12
- *Solution Cost:* 12
- *Time:* 24minutes 25.245749seconds

As can be seen the algorithm correctly solved the 3-Node problem on a quantum machine. This shows that despite the disadvantages, it is possible to successfully use the QAOA algorithm running on a quantum machine to solve the TSP problem. This also means that as quantum machines become more powerful it will be possible to take these tests further. Below is a graph of the solution to the 3-node problem found by this test.

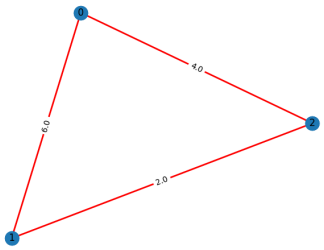


Figure 4.5: QAOA Run On IBMQ

5 Project Discussion & Conclusion

5.1 Project Discussion

In the original project aims and objectives section I stated 2 research questions for the project and the expected outcomes. I will now discuss the answer to those questions and the actual outcomes of the project.

5.1.1 Question 1

Can the QAOA be used to solve the Native and GPS formulations of the TSP with current quantum technology? Based on the implementation and testing conducted in this project, I would like to state that yes it is possible to use the QAOA to solve the Native and GPS formulations. It has been shown both on quantum simulators and actual quantum machines that it is possible.

5.1.2 Question 1

When it comes to solving these problems, how does the performance of QAOA compare to classical algorithms? Again based on the implementation and testing in this project, I would like to state that the QAOA is not comparable to even Simulated Annealing at this current point in time. One of the main factors for this is the current state of quantum technology. Quantum computing is still in a very early stage so access to viable systems is difficult to obtain, if not impossible depending on what you class as viable. I would like to state however that the QAOA has shown its ability to solve problems and therefore further testing needs to be conducted when the level of quantum technology develops.

5.1.3 Project Outcomes

In this project, I have run many experiments regarding both different models of the TSP and the use of the QAOA. I think that this project has fulfilled the outcome of providing a comprehensive analysis of the QAOA for solving the TSP on current quantum technology very well.

I have also conducted a number of experiments regarding the analysis of the QAOA parameters, so again I think that this project outcome has been achieved well.

Lastly the analysis between the performance of the QAOA and classical algorithms has been somewhat achieved. Unfortunately due to time constraints a proper statistical analysis of the QAOA could not be completed. This makes the comparison to the Simulated Annealing results relatively unfair, although it does provide some insight.

5.2 Conclusion

In conclusion, I think the development, implementation and testing of this project was a success. All the project research questions have been answered and all the expected outcomes have been achieved. It has been made clear through this project that the QAOA offers a unique method for solving optimisation problems and possesses the ability to scale in the future. At this current time however, the technology required to properly utilise and test the QAOA is not available and therefore the use of the QAOA in a practical setting is not advised.

5.3 Future Directions

Since the project is implementing very new and experimental technology, there is an abundance of future research directions that could be taken. Below I will list a few that I think have been made especially evident throughout the development of this project.

- The Native formulation is able to reduce the total number of variables by hard coding the variables for the start node. Could this same method be applied to the GPS formulation, and if so what effect would it have?
- The GPS formulation seems like an interesting approach and further research could show ways to improve it even further. A mathematical analysis of the prospective landscape of the GPS formulation might help to provide insight into how it can be improved.
- It is possible, through research and education programs, to get access to much larger quantum machines with IBM. It would be interesting to test how well these machines perform when running the QAOA, and how well the QAOA scales with access to larger machines.
- It would also be interesting to further analyse how the QAOA runs on the different formulations. With considerably more time it might be possible to find new methods to tune the QAOA algorithm in order to solve these problems.

References

[1] B Apolloni, N Cesa-Bianchi, and D De Falco. *A numerical implementation of "quantum annealing"*. Tech. rep. Bielefeld: Bielefeld TU. Bielefeld-Bochum-Stochastik, 1988. URL: https://cds.cern.ch/record/192546.

[2] Abraham Asfaw et al. *The Quantum Approximate Optimization Algorithm*. https://qiskit.org/textbook/ch-applications/qaoa.html. 2021.

[3] Boaz Barak et al. "Beating the random assignment on constraint satisfaction problems of bounded degree". In: *arXiv preprint arXiv:1505.03424* (2015).

[4] Richard Bellman. "On the theory of dynamic programming". In: *Proceedings of the national Academy of Sciences* 38.8 (1952), pp. 716–719.

[5] Richard E. Bellman. *Dynamic programming*. en. Princeton, NJ: Princeton University Press, 1957. ISBN: 9780691079516.

[6] Paul Benioff. "The computer as a physical system: A microscopic quantum mechanical Hamiltonian model of computers as represented by Turing machines". In: *Journal of statistical physics* 22.5 (1980), pp. 563–591.

[7] Paul A Benioff. "Quantum mechanical Hamiltonian models of discrete processes that erase their own histories: Application to Turing machines". In: *International Journal of Theoretical Physics* 21.3 (1982), pp. 177–201.

[8] René van Bevern and Viktoriia A Slugina. "A historical note on the 3/2-approximation algorithm for the metric traveling salesman problem". In: *Historia Mathematica* 53 (2020), pp. 118–127.

[9] Stephen G Brush. "History of the Lenz-Ising model". In: *Reviews of modern physics* 39.4 (1967), p. 883.

[10] John Burkardt. *Travelling Salesman Problem (TSP) Data*. Online. Accessed on: May 3, 2023. 1999. URL: https://people.sc.fsu.edu/~jburkardt/datasets/tsp/tsp.html.

[11] Chetan Chauhan, Ravindra Gupta, and Kshitij Pathak. "Survey of methods of solving tsp along with its implementation using dynamic programming approach". In: *International journal of computer applications* 52.4 (2012).

[12] N Christofides. "Worst-case analysis of a new heuristic for the traveling salesman problem". In: *Sympos. on New Directions and Recent Results in Algorithms and Complexity* (1976).

[13] Jill Cirasella et al. "The asymmetric traveling salesman problem: Algorithms, instance generators, and tests". In: *Workshop on Algorithm Engineering and Experimentation*. Springer. 2001, pp. 32–59.

[14] Gavin E Crooks. "Performance of the quantum approximate optimization algorithm on the maximum cut problem". In: *arXiv preprint arXiv:1811.08419* (2018).

[15] D-Wave Systems Inc. *D-Wave Ocean*. Version 3.3.0. [Software]. 2021. URL: https://docs.ocean.dwavesys.com/en/latest/.

[16] D-Wave Systems Inc. *D-Wave Ocean TSP*. Version 3.3.0. [Software]. 2021. URL: https://docs.ocean.dwavesys.com/en/latest/docs_dnx/reference/algorithms/generated/dwave_networkx.algorithms.tsp.traveling_salesperson_qubo.html.

[17] George Dantzig, Ray Fulkerson, and Selmer Johnson. "Solution of a large-scale traveling-salesman problem". In: *Journal of the operations research society of America* 2.4 (1954), pp. 393–410.

[18] Donald Davendra. *Traveling salesman problem: Theory and applications*. BoD–Books on Demand, 2010.

[19] M Drigo. "The Ant System: Optimization by a colony of cooperating agents". In: *IEEE Transactions on Systems, Man, and Cybernetics-Part B* 26.1 (1996), pp. 1–13.

[20] Edward Farhi, Jeffrey Goldstone, and Sam Gutmann. "A quantum approximate optimization algorithm". In: *arXiv preprint arXiv:1411.4028* (2014).

[21] Richard P. Feynman. "Simulating physics with computers". In: *International Journal of Theoretical Physics* 21.6–7 (June 1982), pp. 467–488. DOI: 10.1007/bf02650179. URL: http://dx.doi.org/10.1007/BF02650179.

[22] Ralph E Gomory. "Outline of an algorithm for integer solutions to linear programs and an algorithm for the mixed integer problem". In: *50 Years of Integer Programming 1958-2008*. Springer, 2010, pp. 77–103.

[23] Saul Gonzalez-Bermejo, Guillermo Alonso-Linaje, and Parfait Atchade-Adelomou. "GPS: A new TSP formulation for its generalizations type QUBO". In: *Mathematics* 10.3 (2022), p. 416.

[24] John Grefenstette et al. "Genetic algorithms for the traveling salesman problem". In: *Proceedings of the first International Conference on Genetic Algorithms and their Applications*. Vol. 160. Lawrence Erlbaum. 1985, pp. 160–168.

[25] Aric Hagberg, Pieter Swart, and Daniel S Chult. *Exploring network structure, dynamics, and function using NetworkX*. Tech. rep. Los Alamos National Lab.(LANL), Los Alamos, NM (United States), 2008.

[26] Michael Held and Richard M Karp. "The traveling-salesman problem and minimum spanning trees". In: *Operations Research* 18.6 (1970), pp. 1138–1162.

[27] Tim Hendtlass. "Preserving diversity in particle swarm optimisation". In: *International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems*. Springer. 2003, pp. 31–40.

[28] John D Hunter. "Matplotlib: A 2D graphics environment". In: *Computing in science & engineering* 9.03 (2007), pp. 90–95.

[29] Anna R Karlin, Nathan Klein, and Shayan Oveis Gharan. "A (slightly) improved approximation algorithm for metric TSP". In: *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*. 2021, pp. 32–45.

[30] Scott Kirkpatrick, C Daniel Gelatt Jr, and Mario P Vecchi. "Optimization by simulated annealing". In: *science* 220.4598 (1983), pp. 671–680.

[31] John Knox. "Tabu search performance on the symmetric traveling salesman problem". In: *Computers & Operations Research* 21.8 (1994), pp. 867–876.

[32] Gary A Kochenberger and Fred Glover. "A unified framework for modeling and solving combinatorial optimization problems: A tutorial". In: *Multiscale optimization methods and applications* (2006), pp. 101–124.

[33] Ailsa H Land and Alison G Doig. "An automatic method for solving discrete programming problems". In: *50 Years of Integer Programming 1958-2008*. Springer, 2010, pp. 105–132.

[34] Gilbert Laporte. "The traveling salesman problem: An overview of exact and approximate algorithms". In: *European Journal of Operational Research* 59.2 (1992), pp. 231–247.

[35] Fei Liu and Guangzhou Zeng. "Study of genetic algorithm with reinforcement learning to solve the TSP". In: *Expert Systems with Applications* 36.3 (2009), pp. 6995–7001.

[36] Andrew Lucas. "Ising formulations of many NP problems". In: *Frontiers in physics* 2 (2014), p. 5.

[37] Rajesh Matai, Surya Prakash Singh, and Murari Lal Mittal. "Traveling salesman problem: an overview of applications, formulations, and solution approaches". In: *Traveling salesman problem, theory and applications* 1 (2010).

[38] Michalis Mavrovouniotis, Mien Van, and Shengxiang Yang. "Pheromone modification strategy for the dynamic travelling salesman problem with weight changes". In: *2017 IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE. 2017, pp. 1–8.