UNIVERSITY OF EXETER

Computer Science Department

B.Sc. Computer Science and Mathematics
Computer Science Department

Comparing the Efficiency of QUBOs with Standard Methods for Solving Classical Routing Problems

Candidate

Charlie Wells

Student ID 690005393

Supervisor

Dr. Alberto Moraglio

University of Exeter

Academic Year
2022/2023

Abstract

Quadratic Unconstrained Binary Optimization (hereafter QUBO) is a combinatorial optimisation problem used to model and solve many classical problems within the computer science field. In my project, I will focus on routing problems: a set of problems that aim to find optimal routes from a starting location to each other pre-determined destination before returning to the starting location. This problem set has numerous direct applications in supply chain management, transportation, and manufacturing. In a real-world scenario with many destinations or vehicles, the complexity of these problems becomes too large to solve through brute-force methods. Instead, we typically implement either heuristics (solutions that estimate the optimal answer) or specialised algorithms to find an optimal solution from a set of solutions. This report will discuss prior progress in modeling these problems and compare these classical methods for solving routing problems with methods that involve deriving and utilising QUBOs.

	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

Chapter 1

Introduction

Solutions to many real-world problems can be obtained by transforming them into a binary quadratic model [1] (hereafter BQM), an equation in the correct form to be parsed by a processing unit for a quantum computer. The variables in a BQM are binary-valued, meaning they each have two possible values. In this project, I will focus on QUBOs, a BQM for which any variable can take values 0 or 1. It is also possible to transform between a QUBO and a different model, say the Ising model, which uses the variables -1 and 1, as discussed by Lewis and Glover. [2], for which some problems already have an established solution.

This report will focus on the QUBO formulations of routing problems (specifically the vehicle routing problem): a combinatorial optimisation problems which attempts to find the "optimum routing of a fleet . . . between a bulk terminal and a large number of service stations", as explained by Dantzig and Ramser [3], and the travelling salesman problem, a generalisation where we consider a single vehicle fleet. One can understand the most general application of this problem by considering a delivery business with a fleet of trucks leaving a warehouse/depot, each visiting multiple predetermined locations to make deliveries, then returning to the depot they started from. Each of these trucks should follow an optimal route to reduce the time spent travelling, which decreases the number of hours a driver is working and reduces total fuel expenditure across the fleet. These problems also have more abstract applications in computer-generated art, astronomy, and agriculture.

These problems do not have an algorithm or method to solve them in polynomial time [4]. As a result, as the scope of the problem increases, the amount of time taken to solve the problem will also increase. As such, brute-force methods, which involve exhausting every combination until we find the optimal solution, will be hugely inefficient and typically not feasible when working with many destinations. Instead, we must rely on other algorithms or mathematical models to solve them.

QUBOs and their formulations

A QUBO (Quadratic Unconstrained Binary Optimization) is a class of mathematical problems expressed in a specific form, which we will use to model both the travelling salesman and vehicle routing problems. Quadratic implies that each term is at most quadratic (the product

of two independent variables). Unconstrained means that we have no formal constraints and instead use penalty functions to constrain the problem, which increases the energy level of our QUBO whenever a condition is broken. Binary means that each term is expressed with binary values (which can only take 0 or 1), and optimisation describes that we are attempting to maximise or minimise the energy level of our solution.

For our travelling salesman problem, we will attempt to minimise the route length that fits the necessary criteria (visiting each location, then returning to the starting point). Similarly, for our vehicle routing problem, we will attempt to minimise the maximum distance travelled by any single vehicle (considering similar constraints as our TSP). We can represent a QUBO as the optimisation problem:

$$\text{QUBO} : \text{minimize/maximize } y = x^T Q x$$

where Q is a square matrix of constants, and x is a vector of binary variables [5]. By representing them in this form, we can model and solve numerous combinatorial optimisation problems, including our previously mentioned routing problems, as demonstrated by Kochenberger and Glover [6].

Quantum Computing

In a classical computer, we encode and store information as a bit, with each bit's state representing a single binary value. This state is typically 0 or 1 (however, we may represent a state as on/off or yes/no). We often base classical computing algorithms on the assumption that one can examine the stored information (or bits) to determine a value without changing the contents [7].

In a quantum machine, we may use the direction of spin for a particle or the polarization of a photon (horizontal or vertical) to represent information. Due to quantum mechanics, the system will not necessarily be in the 0 or 1 state. It is instead in a superposition (linear combination) of both states [8]. We refer to this quantum unit of information as a qubit; the distinction between the quantum and classical machine (the possibility of creating superposition) allows one to perform many operations in parallel. In a sense, a quantum computer with N qubits can perform 2^N calculations in parallel.

In 1994, Shor demonstrated that prime factoring on a quantum computer will always be faster than the classical computer if the prime number is sufficiently large [9]. This discovery leads us to question whether other quantum solutions (such as quantum solutions for the travelling salesman or vehicle routing problems) are faster than their classical computing counterparts.

The remainder of this report will compare a selection of classical algorithms and the QUBO formulation for our travelling salesman and vehicle routing problem, detailing the correctness and runtime of each algorithm to investigate this proposed question.

Chapter 2

Project Specification

This project aims to compare a selection of classical algorithms with the QUBO formulations for the travelling salesperson problem and the vehicle routing problem. To accomplish this, we will compare the run-time and correctness of three classical algorithms and the QUBO formulation and solution.
Additionally, the QUBO formulation will be explained mathematically and take a form similar to:

$$\sum_{i=1}^{n} b_i x_i + \sum_{i=1}^{n} \sum_{j=1}^{n} q_{i,j} x_i x_j$$

where the variables $x_i \in \{0, 1\}$ and the coefficients $b_i, q_{i,j} \in \mathbb{R}$. This form matches the form $x^T Q x$ previously mentioned, as explained by Papalitsas et al [10]. The solution to the QUBO will be obtained through simulated annealing, implemented using the dwave-neal package provided in the pyqubo library (which will generate our QUBO given a list of constraints and a cost function).

2.1 Project Scope

- The project will implement a brute-force algorithm to solve the travelling salesman and vehicle routing problem. Significant effort should be made to optimise the brute-force algorithm and reduce run-time, and any optimisations should be detailed in the design section. This algorithm should always give the optimal output to a given graph.
- Implement a basic greedy algorithm (nearest neighbour), which should provide a deterministic answer, and a probabilistic algorithm (ant colony optimisation), which provides an answer using a random number generator.
- Formulate a QUBO for both the travelling salesman and vehicle routing problem. Both of these QUBOs should be solved locally through simulated annealing for a given input, which should take the form of an adjacency matrix for a complete graph.
- The efficiency and effectiveness of the above methods should be compared and graphed. We should analyse the run-time and the error from the optimal solution for each algorithm. We will test this by changing the number of vehicles, destinations, and the weighting between each node, repeating the tests where necessary to get averages for our probabilistic algorithms.

2.2 Project Criteria

A successful version of this project will produce a data set for the travelling salesman and vehicle routing problem, comparing the efficiency of the brute force algorithm, nearest neighbour method, ant colony optimisation, and QUBO solutions for various inputs of different sizes. We will compare each algorithm's run time and the error compared to the optimal minimal path.

A successful QUBO formulation should be at least more efficient than brute-force methods (the run time and time complexity will be lower). Alternatively, a valid justification will be given for any QUBO formulation slower than brute-force methods. An explanation for how I formulate the QUBO will be provided for each problem, which should be approachable by someone with little background knowledge.

The code produced should be sufficiently commented and tested. It should also be extensible, allowing for additional algorithms or routing problems. It should allow for user inputs (preferably through a text file) which are parsed and validated, and the output of the code should be easily understandable by the user. The project should include a readme file explaining how the program works and a requirements text file that documents all the libraries necessary to run the program.

Each algorithm should work as expected, producing sufficient practical results. In the case of the brute-force and nearest neighbour algorithms, these are deterministic, so given a specific input, the output should be consistent. In the case of the ant colony and simulated annealing algorithms, these should produce results that reliably estimate the optimal solution with little error. Where necessary, the algorithms should be optimised, with explanations given on how this has been achieved (this is especially necessary for the brute-force algorithm to reduce memory and run time for large numbers of destinations).

The project report should explain all the design choices, including why each algorithm was chosen and details about the specific algorithms, and, where necessary, the time complexity of these algorithms. It will also detail why the particular QUBO formulation was chosen for the travelling salesman and vehicle routing problem, highlighting any solutions to issues faced when implementing both the algorithms and the QUBO.

The results of our testing should be displayed in a readable and understandable format, with conclusions being drawn regarding both the correctness and run-time of each algorithm. Data should be produced for multiple test cases with differing numbers of destinations and distances. Each test should be run multiple times to cater to our non-deterministic algorithms (such as ant colony optimisation or the simulated annealing algorithm implemented to obtain QUBO solutions).

Chapter 3

Design and Implementation

3.1 Travelling Salesman and Vehicle Routing Problem

Dantiz and Ritter proposed that the objective of the vehicle routing problem was to "assign stations to trucks in such a manner that station demands are satisfied and total mileage covered by the fleet is a minimum," [3] however, the objectives of the vehicle routing problem are often amended depending on the particular application. The most notable objectives attempt to minimise global transportation costs based on the distance travelled and fixed costs associated with the used vehicles, minimise the number of vehicles required to reach all destinations, or reduce the variation in travel time between vehicles, as noted by Toth and Vigo [11].
For this project, the objective of the vehicle routing problem will be to minimise the length of the longest single route among all vehicles. This chosen method aims to reduce the maximum distance a single vehicle would travel, which should also reduce the variance between vehicles. For the travelling salesman problem, the fleet consists of only one vehicle. Therefore, reducing the distance of the longest single route is equivalent to reducing the total distance, matching the expected objective. I decided against Dantzing and Ritter's solution, as it may result in solutions where a single vehicle travels most of the total distance. Additionally, a single vehicle travelling the best route (generated by solving the travelling salesman problem) would be the optimal total regardless of fleet size (so long as each vehicle is not required to leave the depot).

3.2 Brute Force Design

The brute force solution consists of iterating through every possible route for each vehicle, then selecting the solution which yields the optimal solution (minimising the longest route amongst vehicles). In the TSP case, this would consists of generating every possible permutation of given destinations, then running the distance-finding algorithm on each permutation. Suppose an input consists of n destinations a vehicle must visit; then the number of possible permutations is $n!$, which is extremely large for large n. One of the main concerns for the brute force algorithm is memory usage for large numbers of cities, which will need to be mitigated through the use of yield statements and generators to make use of lazy evaluation [12].

One optimisation that has been implemented notices that, given the weights are bi-directional, a given path is identical to its reverse. As such, a permutation can be removed if its reverse already exists. Also, since we know that the starting depot will not be visited until all other nodes have been visited and the vehicle must start at this node, we can append it to our path after generating the permutations. These optimisations reduce the number of permutations and thus the number of times we must run the distance-finding algorithm, to $\frac{1}{2}(n-1)!$ given n destinations (including the starting depot).

In the case of VRP, a similar method is implemented, utilising set partitioning to generate possible combinations of routes for each vehicle [13]. This method inherently accounts for routes being reversible and that a vehicle taking a given path is identical to any other vehicle taking the same path, which hugely optimises the algorithm when compared to simply generating all possible route permutations for each vehicle (a method which we have seen is at least factorial). The number of partitions for a given set is determined by the bell number, calculated by using the following formula: [14]

$$B(n) = \frac{1}{e}\sum_{r\geq 0}\frac{r^n}{r!}$$

This formula demonstrates that, as expected, the brute-force solution produces large amounts of partitions, each of which we must calculate the distance of. A slight further optimisation has been made, only considering cases where the number of elements within the partition is equivalent to the number of vehicles. Doing so reduces the number of valid partitions to the Stirling partition number [15], which is always less than or equal to the respective Bell number. This number can be calculated using the following formula, where n is the number of destinations, and k is the number of vehicles:

$$S(n,k) = \frac{1}{k!}\sum_{i=0}^{k}(-1)^i\binom{k}{i}(k-i)^n$$

We must then calculate every permutation of each partition, meaning that the runtime of this algorithm will also grow factorially. Since the brute-force algorithm guarantees an optimal solution, I will compare the output of all other algorithms to the brute-force solution when determining their error. I will also be using the results of the brute-force algorithm to assist in testing the algorithms (specifically to ensure that the minimum value of the QUBO is the optimal solution, or in other words, that it matches the solution yielded by the brute-force algorithm). As a result, the brute force solution must be tested rigorously, ensuring that the correct number of permutations/partitions are generated for a given input and that the optimal solution is output. The brute force algorithm is also one of the slowest (ignoring the noted optimisations) feasible algorithms, so provides an upper bound for the run time, which we can use to compare with our other implemented algorithms and generated QUBO solution.

3.3 Nearest Neighbour Design

The Nearest Neighbour algorithm is a greedy algorithm that consists of a vehicle taking the shortest path to a non-visited node from its current node. It is one of the simplest algorithms to approximate solutions to TSP and, in this project, has been extended to also approximate VRP solutions. It was chosen over other greedy algorithms due to its simplicity, speed, and ease of implementation while still providing a good estimation for the optimal solution. This should be one of the fastest algorithms as it only has a time complexity of $O(N^2)$, providing a lower bound for runtime. The algorithm is also deterministic, allowing for consistent output for any given input.

In the TSP case, we iteratively select the closest neighbour (the node with the least distance from our current node) which has not been visited and visit that node. Once all nodes have been visited, we return to the starting node. Gutin et al. showed that this method is better than at least $\frac{N}{2} - 1$ possible tours [16], which means that this method should provide a satisfactory estimate for the optimal route.

In the VRP case, we expand on this idea, accounting for the distance travelled by each vehicle, iteratively selecting the vehicle with the lowest total distance traveled, which moves to its nearest non-visited node. We repeat this until all nodes are visited, then all vehicles return to the starting depot. By selecting the vehicle with the least distance travelled, we attempt to minimise variations of distances between vehicles, which assists in achieving our chosen VRP objective. This algorithm also has time complexity $O(N^2)$ as selecting the vehicle with the least distance traveled is done in constant time through the use of a custom vehicle data class, meaning that in the VRP case, this algorithm can similarly act as a lower bound for run time.

3.4 Ant Colony Optimisation Design

Any Colony Optimisation is a swarm intelligence algorithm providing a heuristic for the travelling salesman problem, first proposed by Dorigo, Maniezzo, and Colorni [17]. This algorithm attempts to mimic an ant's ability to locate the shortest path through pheromones and, in a TSP sense, will simulate an ant (or a cluster of ants) randomly selecting a route and leaving a trail of pheromones, the strength of which depends on the length of the path. Latter ants will be more likely to travel paths with more traces of pheromones, which should, after a predetermined number of iterations, yield an estimate for the optimal solution.

Ant colony optimisation was chosen as it is a probabilistic algorithm (similar to our simulated annealing algorithm for solving QUBOs), whereas the brute force and nearest neighbour are deterministic. As a result, we can make a better comparison between it and the QUBO solution (comparing the lower-bound, mean average, and upper-bound when considering the error). It should also provide a middle-ground for runtime and reliability compared to the other two algorithms. The time complexity is approximately $O(N^4)$, which means it should be faster than brute force but slower than the nearest neighbour algorithm (with the benefit of providing a more accurate estimate).

The algorithm requires parameter tuning in both the TSP and VRP implementation, controlling: the relative importance of pheromones, the amount of evaporation during each iteration, the relative importance of edge weighting, and the number of ants. For this, I used the work of Dorigo, summarised by X. Wei [18], which suggests setting the number of ants proportional to the number of destinations and provides experimental data for different parameter values. As the implementation of TSP and VRP was very similar (differing only in the dimension of the array used), I used the same parameter values for both algorithms for convenience, which may have the consequence of yielding slightly sub-optimal results for the VRP case.

3.5 QUBO Design

When designing the QUBO, I wanted to ensure that the VRP formulation was an extension of the TSP formulation, which required selecting a design that could be easily extended into further dimensions and for which constraints apply to the travelling salesman and vehicle routing problem. I also wanted the formulation's design to be easily understandable, meaning the idea behind the formulation should seem natural, allowing me to effectively debug and add constraints when necessary. The paper published by Gonzalez-Bermejo, Alonso-Linaje, and Atchade-Adelomou [19] details three formulations for the traveling salesman problem, extending one of these into the vehicle routing case. I decided against any noted implementations and instead opted for the TSP formulation included in the paper's appendix.

The native formulation utilises variable $x_{i,j,t}$, which has value one if given a time t we traverse the edge between city i and j. This variable formulation is natural. However, it requires an unnatural penalty term to ensure that a vehicle immediately leaves a city once visited (this ensures that there are no cycles within the route and that the vehicle visits every destination in the given time frame):

$$x_{u,v,t}\left(1-\sum_{w=1}^{N+1} x_{v,w,t+1}\right)=0 \qquad \text{For each } t\in\{0,\ldots,N-1\}, u, v\in\{0,\ldots,N\}$$

It is also not immediately obvious how this penalty term will extend to the vehicle routing case. Comparatively, the selected formulation does not need these penalty terms as the cost function retroactively removes cycles, meaning it can be easily extended into further dimensions for the vehicle routing problem. This formulation also takes N^3 variables (i, j, t), whereas the chosen formulation only uses N^2 variables, reducing memory usage and hopefully increasing the speed of execution when solving the QUBO.
The GPS formulation suggested in the paper by Gonzalez-Bermejo et al. [19] has the disadvantage of being unnatural and not immediately obvious with extremely complex penalty and cost functions. This would reduce the ease of debugging and could lead to many issues not being caught during the implementation or testing of the QUBO. It also uses $3N^2$ variables in the TSP case, which is more than our selected formulation (which only uses N^2).

The formulation I have selected uses the variable $x_{i,t}$, the value of which is one if destination i is reached at position t. This only requires N^2 variables ($i \in \{0, ..., N+1\}$ and $t \in \{0, ..., N+1\}$), which means it will take less memory than the other suggested formulations, and should hopefully execute faster. The cost function is given by the sum:

$$\sum_{i=0}^{N+1} \sum_{j=0}^{N+1} \sum_{t=0}^{N} d_{i,j} x_{i,t} x_{j,t+1}$$

where $d_{i,j}$ represents the distance between node i and node j. This cost function determines the total distance travelled and is similar to the distance-finding algorithm used for the brute-force algorithm. Summing each element in this way also means that a row and column with fewer 1's will have a lower distance, which retroactively reduces cycles, still yielding the correct result. Combined with the penalty functions, this leads to rows and columns that consist of a singular one being preferable. For the travelling salesman formulation, three constraints must be considered. Firstly, each node should be visited exactly once:

$$\sum_{t=0}^{N} x_{i,t} = 1 \qquad \text{For each } i \in \{0, ..., N\}$$

Secondly, each node should have an order that it was visited (denoted by t), and this order should be unique for the node:

$$\sum_{i=0}^{N} x_{i,t} = 1 \qquad \text{For each } t \in \{0, ..., N\}$$

Finally, we should finish at the node we designated as the starting node. While this is not strictly necessary for the travelling salesman case, it has been included to ease the transition to the vehicle routing case.

$$x_{s,N} = 1 \qquad \text{For given starting node } s$$

The sum of these constraints multiplied by the penalty term makes up the penalty function for our TSP formulation. The value of the penalty term must be carefully considered to ensure it is appropriate for the constraints. A penalty term that is too low may result in constraint violations as the energy level is not sufficiently penalised. However, a penalty term that is too high may lead to the formation of local minima in the solution space. This may result in sub-optimal solutions as the solution may be trapped in local minima instead of the global minimum of the objective function.

In M. Ayodele's paper [20], she summarises and evaluates the most common methods for selecting valid penalty weights, showing that for the travelling salesman problem, the Maximum QUBO Coefficient (MQC) yields the smallest, valid penalty weights.

This penalty value corresponds to the maximum distance between any two cities in the TSP defined by:

$$MQC = \max\{d_{i,j}\} \quad \text{For } i, j \in \{0, ..., N\}$$

This method of obtaining a penalty value will be used for the travelling salesman and the vehicle routing problem. The formulation can be easily extended from TSP to VRP by adding an additional variable to $x_{i,t}$ producing the variable $x_{v,i,t}$ (where i, t are the same previously defined, and v represents the vehicle number), inspired by Parizy et al. [21] Our cost function remains similar to the previous formulation, however, we will need to determine the route of each vehicle, so must iterate through v). Thus, for M vehicles, our cost function is given by:

$$\sum_{i=0}^{N} \sum_{j=0}^{N} \sum_{t=0}^{N} d_{i,j} x_{v,i,t} x_{v,j,t+1} \qquad \text{For } v \in \{0, ..., M\}$$

Where, as before, $d_{i,j}$ represents the distance between node i and node j. We eventually compare each vehicle's path length to solve the objective for VRP once we have performed simulated annealing to generate a solution, as we are attempting to minimise the maximum distance travelled, not the total distance travelled. We also require a set of constraints grouped into static and dynamic. The first constraint ensures that each node (except for the starting node) is visited only once:

$$\sum_{t=0}^{N} \sum_{v=0}^{M} x_{v,i,t} = 1 \qquad \text{For } i \in \{1, ..., N\}$$

It is also required to constrain the number of times the starting node is visited. Each vehicle should visit the starting node twice, so for a starting node, our constraint is given by: S

$$\sum_{t=0}^{N+1} x_{v,S,t} = 2 \qquad \text{For } v \in \{0, ..., M\}$$

It is also required that each vehicle starts at the starting node, and they must immediately leave the node. These constraints are given respectively by:

$$x_{v,S,0} = 1 \qquad \text{For } v \in \{0, ..., M\}$$

$$x_{v,S,1} = 0 \qquad \text{For } v \in \{0, ..., M\}$$

The remaining constraints all change depending on the number of destinations each specific vehicle visits. As there is no way to tell the optimal distribution beforehand, this can be solved in multiple ways, for example: assuming that each vehicle will visit the same number of destinations, using slack variables, or by generating each case and selecting the optimal. In this implementation, I chose the third option. By determining all possible partitions, we solve for each case, then select the case which yields the best results. This method increases the run time as we perform our simulated annealing algorithm multiple times. However, doing so ensures

that our solution is more accurate. If we let L represent the number of destinations a specific vehicle will visit, we can generate the following dynamic constraints. Firstly, we must constrain that each vehicle returns to the starting node after visiting L locations.

$$x_{v,S,L} = 1 \qquad \text{For } v \in \{0, ..., M\}$$

Finally, we require that a single vehicle cannot visit two nodes simultaneously and must move to a new node during each action (unless it has already returned to the starting node). This is similar to constraint two in the TSP case, but we must also account for a vehicle returning early:

$$\sum_{i=0}^{N} x_{v,i,t} = 1 \qquad \text{For } v \in \{0, ..., M\}, t \in \{0, ..., L+2\}$$

As before, the sum of these constraints multiplied by the penalty term makes up the penalty function for our formulation. We could consider the penalty terms of the static and dynamic constraints separately. However, we do not want our solution to violate any of these constraints, so this implementation instead uses a singular penalty term generated using the MQC method discussed previously (modified slightly to be the square of the edge weight, to account for multiple vehicles).

3.6 Python Implementation

In order to obtain a solution from the formulated QUBO, I have used the PyQUBO library [22], which allows the creation of QUBO (and Ising) models through Pythonic syntax. Originally this library was chosen over others due to it being well-documented and having direct integration with D-Wave, which would allow me to interface and solve on D-Wave's suite of quantum computers through cloud services. However, since I am now solving the QUBOs locally, this is not as relevant. The library also provides a simple implementation of simulating annealing through the dwave-neal package, which allows for the local solving of the formulated QUBO expression. The simulated annealing method requires various parameters such as the beta range, number of sweeps, and number of reads to be tuned to produce reliable results. In my implementation, these values were determined using the documentation and various papers to decide on a starting value, then amending through experimentation until accurate results were output.

The program also allows for simple user input, allowing the user to determine how many vehicles will be traversing a graph, which algorithms to use, and also input a custom graph (if they instead choose for the graph to be randomly generated, they can also input the size of the graph). The output of each algorithm will be printed to the console, showing the user a formatted version of the resulting path and the length of the route.

Chapter 4

Project Results and Testing

4.1 Unit Testing

As we are using the brute force algorithm to compare the correctness of the other heuristic algorithms, it is necessary to extensively unit test these methods, ensuring that the correct number of permutations (or partitions in the VRP case) are produced and that the distance function is producing the expected result for a given input. It is also necessary to test whether the QUBO solution is a minimum when given the optimal route, which we can only do once we can ensure the brute force algorithm works as intended.

In the TSP case, testing the brute force algorithm consisted of passing in different quantities of cities and ensuring the number of generated permutations is as expected (according to the formula calculated earlier). To test the remainder of the algorithm, a pre-made graph, for which the optimal route is pre-calculated, is passed in. If the brute force algorithm is working as intended, the output should be the same as the known value. In the VRP case, a similar methodology is implemented, verifying that the number of set permutations is equivalent to the respective Stirling partition number and passing in a pre-made graph to ensure the output value matches the optimal value.

To test the QUBO formulation, we can pass, as an argument, a minimal path to the constraint and cost methods to ensure that the output is as expected. For a minimal path each constraint should be met, so the total of the penalty function should equal 0. Also the cost function should equal the length of the path, meaning the weight of the QUBO should match the minimum path length. To achieve this, I could use paths generated by the brute force algorithm or known graphs. I again chose to use pre-calculated graphs as the other option would have our QUBO test relying on the brute force tests, unfavorably coupling these together.

In addition, the parsing functions, which convert a user's input to a graph, make use of unit tests; This is because the graph must meet numerous conditions to be considered valid for TSP or VRP (it must be symmetrical and complete with zeroes on the leading diagonal and non-zeroes everywhere else). To test this, valid and invalid graphs are passed into the function, and the output is verified against the expected output, ensuring the user can input their data without receiving an erroneous output due to formatting issues or an invalid graph.

4.2 Comparisons between algorithms

To perform my comparisons, I randomly generated graphs of varying lengths and executed each algorithm, noting their runtime and output, allowing me to accurately tabulate and comment on trends in the data. Additionally, for the heuristic methods (all except for brute force), I ran each algorithm against publically available TSP data sets, where optimal routes are already provided, which allows me to comment on runtime and correctness when the number of cities is greater and brute force would be too slow to feasibly obtain a solution.
Because ant colony and simulated annealing are probabilistic, I ran these fifteen times on each graph, obtaining the best, worst, and average results for each. Doing this should remove some variance that may have occurred due to these algorithms using random numbers to generate an output; This was not necessary for the nearest neighbour or brute force algorithm as they are both deterministic (so the same graph will yield the same answer). This is also comparable with a real use case, as one would likely run each probabilistic algorithm multiple times for a singular graph (taking the lowest result of these runs) to determine the solution for the travelling salesman or vehicle routing problem.

4.3 Results

Cities	Output path length for TSP							
	Brute Force	Nearest Neighbour	Ant Colony Optimisation			QUBO		
			Low	Average	High	Low	Average	High
3	163	163	163	163	163	163	163	163
4	178	178	178	180	186	178	178	178
5	244	259	244	254	259	244	244	244
6	233	233	233	244	271	233	233	233
7	321	322	322	335	370	321	323	326
8	249	293	249	284	340	249	263	279
9	323	323	323	342	374	339	359	363
10	248	374	248	301	417	280	308	343
11	253	349	273	324	392	289	294	318
12	358	445	382	427	465	423	444	458
13	374	468	374	438	498	444	473	508
14	379	437	414	454	520	426	489	528

Table 4.1: Table showing the optimal path length generated by each algorithm for varying numbers of cities for the Travelling Salesman Problem

For the TSP case, we can observe that the lower-bound results generated by the QUBO are often better than the optimal path generated by the nearest neighbour algorithm, and in some cases (for eight and ten cities), the average and upper-bound results of the QUBO are both noticeably better than the path output by the nearest neighbour algorithm. In all cases, the QUBO successfully generated a valid path that did not violate any of the constraints, meaning

that the MQC method was generating optimal penalty values.
In comparison to ant colony optimisation, the QUBO appears worse on average. However, the upper bound is often lower, which implies that there is less variance in the solutions produced by the QUBO compared to the answer produced by ant colony optimisation. It is also worth noting that the simulated annealing algorithm takes the "number of reads" as a parameter (which was set to 500). Changing this parameter would likely increase the reliability and accuracy of the QUBO's output but would increase runtime.
These discrepancies may also be due to the parameters used for the algorithms themselves. As previously mentioned, in addition to the number of ants (which is determined by using the MQC method), ant colony optimisation uses three further parameters to control the relative importance of pheromones and pheromone evaporation. These parameters were decided both through various papers and experimentation. However, if these parameters were chosen poorly, this may have caused ant colony optimisation to perform worse than expected. Similarly, for the simulated annealing algorithm, I made use of the documentation which suggested using a specific beta range. However, this range may not have been the best for my use case, leading to the algorithm performing worse than anticipated.

When further compared using both the publicly available "Dantzing42" and "Fri26" data sets (supplied within the TSPLIB library [23]), the QUBO performs much worse than both the nearest neighbour and ant colony optimisation algorithms (in some cases being unable to find a valid route without violating constraints); This may further reinforce that the parameters chosen for the simulated annealing algorithm may have needed further tuning.

Number of Cities	Time Taken (s)			
	Brute Force	Nearest Neighbour	Ant Colony Optimisation	QUBO
3	0	0	0	0.141
4	0	0	0	0.219
5	0	0	0.047	0.328
6	0	0	0.094	0.484
7	0	0	0.156	0.734
8	0.016	0	0.219	1.047
9	0.109	0	0.328	1.36
10	1.108	0	0.891	1.755
11	11.703	0	1.188	2.344
12	138.219	0	1.563	3.109
13	1773.844	0	2.05	4.08
14	25082.891	0	2.641	5.125

Table 4.2: Table showing the runtime for each algorithm for varying numbers of cities for the Travelling Salesman Problem

Comparing the runtime between each algorithm, we can see that the nearest neighbour algorithm is the most efficient, executing almost instantly regardless of the number of cities. This

coincides with our expected result as the nearest neighbour is an $O(N^2)$ algorithm, meaning it will take very few operations to return an answer. Comparatively, the brute force algorithm has a factorial complexity, requiring many more operations and hence a longer runtime.

Observing our QUBO, we can see that for a smaller number of cities, it is the slowest algorithm of the set. This may be a result of keeping the simulated annealing parameters (more specifically, the number of reads) constant regardless of the number of cities, resulting in unnecessary operations when an answer could be found with fewer reads. However, for a larger number of cities, the brute force algorithm has a vastly worse runtime than all of the other algorithms, taking almost eight hours to yield a solution for the fourteen-city case. This is a result of the algorithm having a factorial time complexity, meaning that the runtime increased multiplicatively whenever a new city was added. In comparison, the QUBO algorithm took about five seconds to calculate an answer for the fourteen-city case, meaning it is much more feasible to use the QUBO or other heuristic algorithms for larger numbers of cities than the brute-force algorithm.
In comparison to the ant colony optimization, the QUBO has a slower runtime for every number of cities, which can again be mitigated by reducing the number of reads and iterations in the simulated annealing algorithm. As we further increase the number of cities, the simulated annealing algorithm used to solve the QUBO does, however, become more efficient than the ant colony optimisation algorithm in terms of runtime (however, as shown earlier, the result becomes less accurate). In all cases, the nearest neighbour algorithm remains the fastest, regardless of the number of cities we are considering. This makes it extremely useful if your use case requires a fast but not precise answer.
For the VRP case, the same experiments were run, performing each algorithm on a randomly generated graph, gradually increasing the number of destinations. In this case, I considered a fleet of three vehicles traversing the graph and, as before, ran the probabilistic algorithms fifteen times, obtaining an algorithmic upper-bound, lower-bound, and mean average for each graph.

Number of Cities	Time Taken (s)			
	Brute Force	Nearest Neighbour	Ant Colony Optimisation	QUBO
4	0	0	0.016	0.391
5	0	0	0.062	1.252
6	0	0	0.109	1.891
7	0	0	0.156	4.281
8	0.031	0	0.234	5.281
9	0.266	0	0.328	9.921
10	2.484	0	0.891	13.517
11	27.578	0	1.219	21.375
12	327.047	0	1.529	29.078

Table 4.3: Table showing the runtime for each algorithm for varying numbers of cities for the Vehicle Routing Problem

From this table, we can see that the trends of runtime for each algorithm in the vehicle routing problem match the runtimes in the travelling salesman problem. At a fewer number of cities, the QUBO provides the slowest runtime. However, as we increase the number of cities, the runtime of the brute force algorithm overtakes the simulated annealing algorithm, coinciding with both our expected outcome and the outcome seen in the travelling salesman problem. This means that for the travelling salesman and vehicle routing problem, for most practical use cases, it is more efficient to perform the QUBO algorithm than the brute force algorithm (if runtime is the only priority and precision is unnecessary).
In comparison to the ant colony optimisation algorithm, the result of the QUBO is much slower for all inputs and remains slower as we continue increasing the number of cities. This differs from our travelling salesman case, where the simulated annealing algorithm eventually overtook the ant colony optimisation in terms of runtime. I theorize that this might be a result of the way the dynamic constraints were implemented in the QUBO formulation. As the implementation does not utilise a slack variable or approximation for the number of destinations each vehicle visits (instead making use of partitions to generate all possibilities), we perform the simulating annealing operation multiple times, which may be the cause of this increased runtime. Dividing the runtime by the number of generated partitions (and hence the number of times we perform simulated annealing in a single run) provides a runtime similar to the TSP case, further reinforcing that this may be the cause of the increased runtime.

Cities	Output path length							
	Brute Force	Nearest Neighbour	Ant Colony Optimisation			QUBO		
			Low	Average	High	Low	Average	High
4	138	138	138	138	138	138	138	138
5	94	111	94	109	126	94	94	94
6	137	144	137	152	196	137	137	137
7	128	164	128	133	164	128	128	128
8	159	170	174	204	241	159	160	168
9	148	150	162	189	228	149	155	166
10	167	174	188	207	250	171	184	188
11	141	172	164	184	223	145	169	183
12	168	236	183	217	269	192	199	213

Table 4.4: Table showing the optimal path length generated by each algorithm for varying numbers of cities for the Vehicle Routing Problem

From this data, we can observe that the lower-bound solutions provided by the QUBO are more accurate (closer to the optimal brute force solution) than either the nearest neighbour or ant colony optimisation algorithms for the same number of cities. Similar to the TSP case, we can observe that the average solution of the QUBO is also better than the nearest neighbour path for almost every number of cities, meaning that if a use case prefers accuracy over runtime, it would be preferable to use simulated annealing to generate a QUBO.
Comparing the ant colony optimisation algorithm to our QUBO, in this case, shows that the

QUBO will provide a better solution on average. We can also see that in the majority of cases, the lower bound of the QUBO provides a better solution than the ant colony optimisation algorithm, and the upper bound of the QUBO is lower in every case, meaning the algorithm is less susceptible to variance. The reason for this discrepancy may be because of the parameters used for both algorithms. In the ant colony optimisation algorithm, I used the same parameters as the travelling salesman problem (and a similar method for calculating the number of ants), which may have required further tuning in the VRP case. Furthermore, in ant colony optimization's case, the algorithm randomly decides which vehicle to move at each step but ensures that each vehicle moves a similar number of times; This may lead to a case where optimal paths are missed as each vehicle is encouraged to travel the same number of times, regardless of efficiency. Comparatively, in the QUBO case, the algorithm performs simulated annealing over every partition to find the optimal number of times each truck should move. This may be the primary cause of the improvement in solutions, however, is also the main reason the runtime of the QUBO solver is slower in comparison to the ant colony optimisation algorithm.

4.4 Summary

In both the TSP and VRP cases, the runtime of the brute force algorithm increased drastically in comparison to the simulated annealing algorithm used to provide a solution to the QUBO, becoming less efficient when considering more than ten cities in both cases. Obtaining a QUBO solution is also more efficient than the ant colony optimisation algorithm when considering a much larger network for the travelling salesman problem. However, due to the choices taken when designing the QUBO, this does not hold for the vehicle routing problem.
In both cases, we also conclude that the lower-bound solutions provided by the QUBO solver are more accurate than those output by the nearest neighbour algorithm and, in general, are better than solutions yielded by the ant colony optimisation algorithm. However, as we increase the number of cities, this does not hold. In the TSP case, the simulated annealing algorithm struggles to find a viable solution that does not violate the constraints, and those found that do not violate constraints are often very inefficient (compared to the ant colony or nearest neighbour solutions). In the VRP case, the algorithm will find valid solutions that do not violate any constraints, but this is still not feasible for large numbers of cities due to the long run time compared to the other heuristic algorithms.
From the data, we can conclude that the simulated annealing algorithm provides an accurate result and reliably solves the QUBO. We can also conclude that the formulation of the QUBO must have been correct, as the weight matched the brute force algorithm in both the travelling salesman and vehicle routing case when considering a small number of cities; This also means that the methods for choosing parameters for the QUBO (specifically the value for the penalty function) and those chosen for the simulated annealing algorithm were accurate and all the parameters were successfully tuned for both the QUBO solver and ant colony optimisation algorithm.

Chapter 5

Project Discussion

The results we obtained from testing match our expected results. The runtime of the simulated annealing algorithm is better than brute force for more than ten cities, and the solutions provided by the QUBO are, on average, more optimal than those yielded by the other two heuristic algorithms that it was compared with (nearest neighbour and ant colony optimisation); This conclusion holds for both the travelling salesman problem and vehicle routing problem. From this, one may conclude that given a quantum computer with enough processing power (a sufficiently large number of qubits), formulating the problem as a QUBO and solving it will provide a more accurate solution than current classical methods for solving TSP and VRP.

If I were to redo the project, I would slightly amend my QUBO formulation for the vehicle routing problem, as the runtime is inconsistent with what one would expect (as a result of how the dynamic variables were handled). Instead of generating all partitions to obtain the number of times each vehicle travels and running the simulated annealing algorithm multiple times, I would instead utilise slack variables to generate these dynamically in the simulated annealing process. Doing this should drastically reduce the runtime for obtaining a QUBO solution in the VRP case, making it consistent with our results for the travelling salesman. I may also consider utilising and comparing different algorithms for solving the QUBO, as the current program only uses simulated annealing. It perhaps makes more sense to solve the QUBO using multiple algorithms (simulated annealing, tabu search, genetic algorithms, etc.), then compare the efficiency and correctness of these methods. The project also presented the opportunity to utilise d-wave's suite of quantum computers through the quantum cloud in addition to solving the QUBO locally. If I were to redo the project, I may further investigate the quantum solvers provided and attempt to solve the QUBO through both quantum and hybrid methods, comparing each of these to the solutions obtained locally.
I would also further investigate the other Python libraries for implementing QUBOs, as pyqubo was mainly chosen due to its popularity and integration capabilities with d-wave. However, since the QUBOs were solved locally, the other available libraries, such as dimod or qubovert, may have provided more flexibility and better fit my use case.

I consider this project successful as it met all the criteria described in section 2.2. In the design section, the steps to optimise the brute force algorithm were also explained, detailing how each optimisation was achieved and the resulting benefit to the number of permutations considered. Where appropriate, each algorithm's asymptotic notation was noted and used to hypothesise the expected outcomes of each test (for example, I concluded that the nearest neighbour algorithm has the smallest asymptotic approximation $O(N^2)$, so has the smallest runtime). Parameters were chosen through the use of pre-existing papers and parameter tuning during the testing phase, resulting in the QUBO providing solutions that violated no constraints, even as the number of cities was increased; This also meant that both the ant colony optimisation algorithm and the QUBO produced routes that were better than those generated by the nearest neighbour algorithm.

Data sets were produced for the travelling salesman and vehicle routing problem, which compared the efficiency of each algorithm with the simulated annealing algorithm used to generate the solution for the QUBO. From these data sets, we interpreted and compared trends in the runtime and efficiency of each algorithm, explaining why these matched our expected outcomes and highlighting areas that could be improved, with any inconsistencies in the data being explained.
The project also detailed the formulation of the QUBO, including why the specific design was chosen and the benefits of the method compared to other common formulations. The formulation was explained mathematically and in a way that could be approached by someone with little background knowledge, explaining each constraint in the context of the problem being solved, which should allow experts in the field and those not familiar with QUBOs to understand the design choices made. The difficulties faced while formulating the QUBO (such as generating the dynamic constraints in the VRP case) were explained, and information about how these problems were overcome was given. In addition to this, details on future improvements and changes in design were given, with the benefit of each change being explained in terms of performance and reliability.

Chapter 6

Conclusion

The primary aim of this project was to understand how one would formulate QUBOs (mathematical equations in the correct form to be parsed by a quantum processing unit) and compare the results produced by solving the QUBOs with those yielded by classical computing algorithms for the travelling salesman and vehicle routing problems; This was achieved through the successful design of a QUBO for both TSP and VRP, for which I demonstrated that given an optimal path, the QUBO solution would violate no constraints, in addition to the cost function producing the expected value (the length of the path).

The project demonstrated that the solutions produced by locally solving a QUBO using simulated annealing are better than those generated by classical computing methods, such as the nearest neighbour algorithm and ant colony optimisation for both the travelling salesman problem and the vehicle routing problem. This project also demonstrated that for large numbers of destinations, solving a QUBO is more efficient (in terms of execution time) than brute force and ant colony optimisation algorithms for solving the travelling salesman problem.
From this, we may conclude that given a quantum computer with enough qubits, it is more efficient to solve routing problems by formulating a QUBO and solving it using a quantum processing unit than it is to solve it using classical methods.

In the future, it may be worth investigating whether the simulated annealing algorithm used to locally solve the QUBO is optimal or whether a different method, such as tabu search or genetic algorithms, is more efficient, considering both the runtime and accuracy of the solution produced.

References

[1] Andrew Lucas. "Ising formulations of many NP problems". In: *Frontiers in Physics* 2 (Feb. 2014), p. 5. DOI: 10.3389/fphy.2014.00005.

[2] Mark Lewis and Fred Glover. "Quadratic Unconstrained Binary Optimization Problem Preprocessing: Theory and Empirical Analysis". In: *Networks* 70 (May 2017), p. 2. DOI: 10.1002/net.21751.

[3] G. B. Dantzig and J. H. Ramser. "The Truck Dispatching Problem". In: *Management Science* 6.1 (1959), pp. 80–91. (Visited on 11/15/2022).

[4] Christos H. Papadimitriou. "The Euclidean travelling salesman problem is NP-complete". In: *Theoretical Computer Science* 4.3 (1977), pp. 237–244. ISSN: 0304-3975. DOI: https://doi.org/10.1016/0304-3975(77)90012-3. URL: https://www.sciencedirect.com/science/article/pii/0304397577900123.

[5] Fred Glover et al. "Quantum bridge analytics I: a tutorial on formulating and using QUBO models". In: *Annals of Operations Research* 314.1 (2022), pp. 141–183.

[6] Gary A. Kochenberger and Glover Fred. "A Unified Framework for Modeling and Solving Combinatorial Optimization Problems: A Tutorial". In: *Multiscale Optimization Methods and Applications*. Boston, MA: Springer US, 2006, pp. 101–124.

[7] Arthur O. Pittenger. *An Introduction to Quantum Computing Algorithms*. USA: Birkhauser Boston, Inc., 1999. ISBN: 0817641270.

[8] Joachim Stolze, Dieter Suter, and David Divincenzo. "Quantum Computing: A Short Course from Theory to Experiment". In: *American Journal of Physics - AMER J PHYS* 73 (Aug. 2005), pp. 799–800. DOI: 10.1119/1.1938953.

[9] Peter W. Shor. "Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer". In: *SIAM Journal on Computing* 26.5 (1997), pp. 1484–1509. DOI: 10.1137/S0097539795293172.

[10] Evangelos Stogiannos, Christos Papalitsas, and Theodore Andronikos. *Experimental analysis of quantum annealers and hybrid solvers using benchmark optimization problems*. 2022. arXiv: 2202.08939 [quant-ph].

[11] Paolo Toth and Daniele Vigo. *The Vehicle Routing Problem*. Ed. by Paolo Toth and Daniele Vigo. Society for Industrial and Applied Mathematics, 2002. DOI: 10.1137/1.9780898718515. eprint: https://epubs.siam.org/doi/pdf/10.1137/1.9780898718515. URL: https://epubs.siam.org/doi/abs/10.1137/1.9780898718515.

[12] Paul Hudak. "Conception, evolution, and application of functional programming languages". In: *ACM Comput. Surv.* 21 (1989), pp. 359–411.

[13] P.R. Halmos. *Naive Set Theory*. Dover Books on Mathematics. Dover Publications, 2017. ISBN: 9780486821153. URL: https://books.google.co.uk/books?id=bWu9DgAAQBAJ.

[14] Herbert S. Wilf. *Generatingfunctionology.* English. 2nd ed. Boston, MA: Academic Press, 1994. ISBN: 0-12-751956-4.

[15] Ronald L. Graham, Donald E. Knuth, and Oren Patashnik. *Concrete Mathematics: A Foundation for Computer Science*. 2nd. USA: Addison-Wesley Longman Publishing Co., Inc., 1994. ISBN: 0201558025.

[16] Gregory Gutin, Anders Yeo, and Alexey Zverovich. "Traveling salesman should not be greedy: domination analysis of greedy-type heuristics for the TSP". In: *Discrete Applied Mathematics* 117.1 (2002), pp. 81–86. ISSN: 0166-218X. DOI: https://doi.org/10.1016/S0166-218X(01)00195-0. URL: https://www.sciencedirect.com/science/article/pii/S0166218X01001950.

[17] Marco Dorigo, Vittorio Maniezzo, and Alberto Colorni. "Positive Feedback as a Search Strategy". In: *Tech rep., 91-016, Dip Elettronica, Politecnico di Milano, Italy* (Apr. 1999).

[18] Xianmin Wei. "Parameters Analysis for Basic Ant Colony Optimization Algorithm in TSP". In: *International Journal of u- and e-Service, Science and Technology* 7 (Aug. 2014), pp. 159–170. DOI: 10.14257/ijunesst.2014.7.4.16.

[19] 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). ISSN: 2227-7390. DOI: 10.3390/math10030416. URL: https://www.mdpi.com/2227-7390/10/3/416.

[20] Mayowa Ayodele. *Penalty Weights in QUBO Formulations: Permutation Problems*. June 2022.

[21] Whei Yeap Suen, Matthieu Parizy, and Hoong Chuin Lau. "Enhancing a QUBO Solver via Data Driven Multi-Start and Its Application to Vehicle Routing Problem". In: GECCO '22. Boston, Massachusetts: Association for Computing Machinery, 2022, pp. 2251–2257. ISBN: 9781450392686. DOI: 10.1145/3520304.3533988. URL: https://doi.org/10.1145/3520304.3533988.

[22] *PyQUBO*. URL: https://pyqubo.readthedocs.io/en/latest/.

[23] *TSPLIB*. URL: http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/index.html.

Acknowledgments

Special thanks to my supervisor Alberto Moraglio, and Fujitsu representatives Mayowa Ayodele, Mattheiu Parizy, and Marcos Diez García for offering advice and suggesting relevant papers for my project.

I would also like to extend my gratitude to lecturers, friends and family who have supported me along the way.