University of London
Imperial College of Science, Technology and Medicine
Department of Computing

Investigating the Feasibility of Solving the Quadratic Assignment Problem using Quantum Computing

Joel Choo

Supervisors:
Dr Ruth Misener
Dr Luigi Nardi

Submitted in part fulfilment of the requirements for the degree of
Master of Research in Advanced Computing at Imperial College, January 2015

Abstract

This paper investigates the feasibility of solving the quadratic assignment problem (QAP) on the D-Wave quantum computer. We describe a method to convert the QAP to a quadratic unconstrained binary optimisation problem and embed that on to the D-Wave hardware. We show that in the worst case, a QAP of size n requires $O(n^4)$ qubits to represent, so the current D-Wave hardware would only be able to solve a QAP with 7 variables.

To circumvent that limitation, we show that we can reduce the number of qubits required for a QAP with duplicate rows. Using this method, we can reduce the number of qubits required to $O(n^2)$ if there are many duplicate rows. We also propose a decomposition approach to solve generic QAPs.

Acknowledgements

I would like to thank my supervisors Dr Ruth Misener and Dr Luigi Nardi for their invaluable guidance and support and for always believing in me.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In recent years, there has been a significant amount of effort put into determining the feasibility of quantum computation and comparing their performance against classical computers. In particular, the effort has been focused on the quantum computers built by D-Wave Systems Inc., BC, Canada.

The quadratic assignment problem (QAP) is one of the problems that has been used to compare the D-Wave quantum computers against classical computers [33] [4] [39]. In 2013, a comparison by McGeoch and Wang found that the D-Wave Two platform performed better than the CPLEX software solver when solving the QAP [33]. However, the D-Wave Two appears to perform worse compared to a dedicated QAP solver on a classical computer [39]. This is in part because the QAP is solved on the D-Wave Two using a hybrid approach on a generic solver, Blackbox, instead of being solved directly on the quantum hardware.

This paper presents a study on the feasibility of solving the QAP on the quantum computers built by D-Wave Systems. We investigate the challenges in solving the quadratic assignment problem directly on the D-Wave quantum hardware and present a decomposition approach to overcome the challenges.

1.1 Motivation

In 2011, D-Wave Systems announced D-Wave One, described as "the world's first commercially available quantum computer" with 128 qubits [3]. Since then, D-Wave Systems has improved their chipset, and in 2015 announced the availability of the D-Wave 2X with 1152 qubits [5]. The increasing number of qubits potentially allows us to compute solutions to difficult problems that are currently intractable for classical computers.

The D-Wave hardware allows quadratic unconstrained binary optimisation problems (QUBO) to be encoded natively, and has been shown to be able to solve them faster than classical solvers [33]. Solving QUBO problems quickly is an exciting beginning, but it is unclear whether the D-Wave quantum computers offer any advantage for practical optimisation problems.

The QAP is a type of optimisation problem which may benefit from quantum computing. QAP is a problem which is important in engineering and decision making, and has been applied to designing building layouts [28], testing of self-testable sequential circuits [20] and analysing patient movement in hospitals [19]. The problem statement is simple and intuitive: you have n locations and n facilities and each location has to be assigned one and only one facility. A cost function determines how good an assignment is.

Despite the simple description, the QAP is very hard. There are plenty of instances as small as $n = 30$ that still have not been solved such as the Tai30a instance on the QAPLIB [11].

1.2 Objectives

The main aim of this project is to find a way to encode the QAP in the form of QUBOs that can be solved efficiently on the D-Wave quantum computers. In particular, we want to focus on:

- QAP $\rightarrow$ QUBO: Find a transformation that minimises the size and connectivity of the QUBO obtained from the QAP
- QUBO $\rightarrow$ D-Wave: Find an efficient way to embed the QUBOs obtained from the initial transformation on to the quantum hardware

1.3 Contributions

In this project, we show how to transform the QAP into a QUBO that can be encoded on the quantum hardware natively. We also discover what makes the QAP difficult to solve on the D-Wave quantum computers, and describe the bottleneck that is preventing the QAP from being solved natively.

We show a method to represent a specific type of QAP using fewer qubits.

We also present a method of decomposing the QAP into smaller instances that can be solved on the quantum hardware natively.

1.4 Organisation

The rest of this paper is organised as follows:

- Chapter 2: Background Reading
- Chapter 3: Solving the QAP directly on the D-Wave
- Chapter 4: Representing constraints for a certain type of QAP
- Chapter 5: Decomposition approach to solve larger QAPs on the D-Wave

Chapter 2

Background

In this chapter, we provide an introduction to the two main parts of this project: the QAP and quantum computation. We will explain those concepts and look into existing research on solving the QAP as well as solving problems on the D-Wave quantum computers.

Section 2.1 will cover the background to the QAP, Section 2.2 will describe the D-Wave quantum computer hardware, and Section 2.3 will cover existing work on how the QAP has been solved on the D-Wave quantum computers.

2.1 Quadratic Assignment Problem

Section 2.1.1 will be a basic introduction to the QAP, including various formulations of the problem. Section 2.1.2 will cover the existing research on solving the QAP.

2.1.1 Introduction

The QAP is a combinatorial optimisation problem that is fundamental in the field of optimisation. The basic idea behind the QAP is simple: we want to assign n facilities to n different locations in a way that minimises cost.

For each pair of facilities, there is a flow between them; for each pair of locations, there is a distance between them. The cost of an assignment is the sum over all the pairs of facilities of the distances between their assigned locations multiplied by the corresponding flows between them.

Problem Formulations

To more formally describe the QAP, we define the following:

- Let the facilities and locations be numbered 1 to n
- Let $N = \{1, 2, \ldots, n\}$
- Then an assignment of facilities to locations can be seen as a permutation $\phi : N \to N$
- Let S_n be the set of all such permutations
- Let f_{ij} be the flow between facility i and facility j
- Let d_{kl} be the distance between location k and location l
- Let b_{ij} be the cost of placing facility i at location j

Then, the Koopmans-Beckmann version of the QAP is [27]:

$$\min_{\phi \in S_n} \sum_{i=1}^{n} \sum_{j=1}^{n} f_{ij} \cdot d_{\phi(i)\phi(j)} + \sum_{i=1}^{n} b_{i\phi(i)} \tag{2.1}$$

Another formulation which makes the quadratic nature of the problem more obvious is based on permutation matrices.

Consider the $n \times n$ matrix formed (where x_{ij} refers to the element in the i^{th} row and j^{th} column) such that:

$$x_{ij} = \begin{cases} 1 & \text{if } \phi(i) = j \\ 0 & \text{otherwise} \end{cases} \tag{2.2}$$

Such a matrix is known as a permutation matrix, and is equivalently characterised by these constraints:

$$\begin{aligned} &\forall j \in N \quad \sum_{i=1}^{n} x_{ij} = 1 \\ &\forall i \in N \quad \sum_{j=1}^{n} x_{ij} = 1 \\ &\forall i, j \in N \quad x_{ij} \in \{0, 1\} \end{aligned} \tag{2.3}$$

Then the QAP can be formulated as the following [12]:

$$\min_{X \in \Pi} \sum_{i=1}^{n} \sum_{j=1}^{n} f_{ij} \cdot d_{\phi(i)\phi(j)} \cdot x_{ij} \cdot x_{kl} + \sum_{i=1}^{n} b_{i\phi(i)} \tag{2.4}$$

where Π is the set of all permutation matrices.

Then we can generalise the problem a bit more by introducing C, a four dimensional array, and obtaining [12]:

$$\min_{X \in \Pi} \sum_{i,j,k,l=1}^{n} c_{ijkl} \cdot x_{ik} \cdot x_{jl} + \sum_{i=1}^{n} b_{i\phi(i)} \tag{2.5}$$

Note that in most QAPs, $c_{ijkl} = f_{ij} \cdot d_{kl}$, and in that case equation 2.4 is equivalent to 2.5.

In fact, for many QAPs it is also true that the b_{ij} are ignored. The comprehensive library of QAP problems, QAPLIB, only has pure quadratic instances of the problem, and problem instances are described only using the flow and distance matrix [11].

This paper only makes use of the formulation in equation 2.4, but it may be helpful to realise that there are other equivalent formulations. The QAP can also be formulated as the following [12]:

- Quadratic convex minimisation problem
- Kronecker product
- Trace formulation

2.1.2 Existing Research

In this section, we will first explore the complexity of QAP, surveying efforts to show how difficult the QAP is. Following that, we will describe the current best methods of solving the QAP.

Complexity

The QAP has been shown to be NP-hard and that no poly time k-approximation algorithm exists unless P=NP [38].

QAPs have been extremely difficult to solve to optimality, with instances of size $n = 30$ that have not been solved to optimality [11].

The QAP is at times also difficult to solve with heuristic based methods; there are some QAP instances of size up to 75 which are relatively difficult for some heuristic based methods and in fact are easier to solve to optimality [18].

Solving the QAP

There are two broad categories of efforts to solve the QAP: Exact/optimal solution algorithms, and Heuristic based algorithms.

Exact Solution Algorithms

Many of the exact solution algorithms are using the Branch and Bound (B&B) method [12] [8] [12]. The Branch and Bound method is simply a divide and conquer algorithm, and works by recursively dividing the problem into sub-problems where the solutions to the sub-problems can be combined to compute a solution to the original problem [15]. The branch and bound method has three main steps: bounding, branching, and selection [12].

Bounding is used to prune the search space and eliminate computation that can be proved to be not optimal. Assume that we want to find the minimum, $m \in \mathbb{R}$, of some objective function. An upper bound to the problem is $u \in \mathbb{R}$ where $m \leq u$. For example, the cost of any feasible solution would be an upper bound to the problem. A lower bound to the problem is $l \in \mathbb{R}$ where $m \leq l$.

In the bounding part of B&B, first we compute an upper bound to the problem. Then for any generated sub-problem, we compute a lower bound inclusive of any cost to get to the sub-problem. If the lower bound is greater than the upper bound of the original problem, then the optimal solution cannot be found in that sub-problem. Then we can cease computation for that sub-problem.

For the QAP, the computational cost and quality of the bound are important considerations in choosing which bounds to use. Commonly used bounds include the Gilmore-Lawler bound [23] [31], linear programming bound [36], and semidefinite programming bounds [44].

In the branching part of B&B, we divide the problem into several sub-problems. For the QAP, the commonly used branching strategies are single assignment branching [23] [31], pair assignment branching [22], [35], and relative positioning branching [34]. The single assignment strategy works by assigning one facility to all locations or all facilities to one location. This gives us n branches with problem size $n - 1$.

Extending the idea of B&B algorithms, note that after branching we end up with n recursive calls that are independent of each other. This has potential for parallelisation, and distributed computing has been used to augment B&B algorithms. The Master-Worker metacomputing system has been used to solve some previously unsolved problems.

Heuristic Based Methods

The various heuristic based methods used on the QAP include simulated annealing [43], tabu search [25], genetic algorithms [18], GRASP [32], and ant systems [42, 21]. As you would expect from heuristic based methods, the performance tends to vary with different problem characteristics [41]. However, as mentioned in the complexity section, heuristic searches are not necessarily fast, and may even be slower than some exact solution algorithms.

2.2 D-Wave Quantum Computer Hardware

In classical computers, the hardware consists of registers and memory, and instructions deterministically change the contents of the hardware. In the D-Wave quantum computers, there are no registers or memory. Instead, the instructions available can affect two things: qubits and couplers.

A qubit is simply a variable $q \in \{0, 1\}$. Rather than setting the value of a qubit directly, the quantum machine instructions (QMI) allows us to set the weight of a qubit [24]. To refer to different qubits, we use a subscript to label the qubits, i.e. q_i. Let the weight of qubit q_i be defined as a_i.

A coupler is a link between two qubits, and allows us to control how qubits influence each other. Similar to setting the weight of a qubit, the QMI allows us to set the strength of the coupler. Since each coupler is described by the two qubits that it connects, we identify the strength of the coupler between qubits q_i and q_j as b_{ij}.

After setting the weights of the qubits and strengths of the couplers, running the D-Wave quantum computer will make it attempt to find the qubit values that minimise the following objective function [24]:

$$O(a, b; q) = \sum_{i=1}^{N} a_i \cdot q_i + \sum_{<i,j>} b_{ij} \cdot q_i \cdot q_j \tag{2.6}$$

where $< i, j >$ represents all the pairs of qubits that are connected by a coupler. Note that this objective function is equivalent to a quadratic unconstrained binary optimisation (QUBO) problem with the exception of having a limited set of pairs $< i, j >$.

One important difference between classical computers and the D-Wave quantum computer is that while instructions for classical computers are deterministic, the QMI return probabilistic results. Instead of reading results off registers or memory in a classical computer, running the QMI results in a probability distribution that we can obtain samples from [24].

The values of a_i and b_{ij} result in a probability distribution based on the objective function

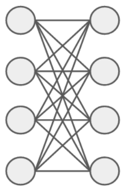
Figure 2.1: Unit Cell

in equation 2.6. The distribution is an equal weighting across all samples that give the minimum value of the objective function [24].

As such, solving a problem on the D-Wave quantum computer is to find a way to encode the possible solutions to the problem in terms of qubits, and then find a way to choose weights a_i and strengths b_{ij} such that when the objective is minimised, the qubits will satisfy the constraints and provide a solution to the problem.

2.2.1 Chimera Graph

Note that the D-Wave quantum hardware described above can be abstracted as a graph, where each qubit is a vertex and each coupler is an edge in the graph. This section describes what the D-Wave quantum hardware graph is like, and what limitations it puts on the problems that can be encoded on the D-Wave.

The D-Wave quantum hardware graph is known as the Chimera graph. To describe the Chimera graph, we need to define a few structures. First, we define a unit cell. A unit cell is a $2 \times L$ grid of qubits, forming a complete bipartite graph with L qubits in each set, i.e. $K_{L,L}$. In the D-Wave quantum hardware thus far, $L = 4$ and so the unit cell is the $K_{4,4}$ graph. Figure 2.1 shows the unit cell where $L = 4$.

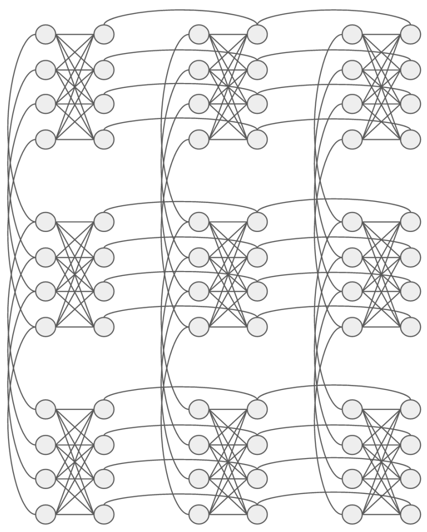

Figure 2.2: Chimera graph with $M = 3$, $N = 3$, $L = 4$

The unit cell forms the basic building block of the Chimera graph. The Chimera graph consists of many unit cells connected in a $M \times N$ grid. Each unit cell is connected horizontally in the grid through the 4 qubits on the right, and connected vertically in the grid through the 4 qubits on the left. This can be better visualised in Figure 2.2.

2.2.2 Limitations of Quantum Hardware

The Chimera graph layout of the quantum hardware results in some limitations on the problems that can be encoded on it natively. The two biggest limitations related to the QAP are qubit number and connectivity.

Qubit Number

The number of qubits in the D-Wave quantum computers have been relatively small so far. Note that for a $M \times N \times L$ Chimera graph, there are a total of $2 \cdot L \cdot M \cdot N$ qubits. Currently, the D-Wave quantum computer with the largest number of qubits is the D-Wave 2X system which has 2048 qubits. Out of those 2048 qubits, only 1152 qubits are turned on.

This means that only problems with fewer than 1152 variables can be solved on the D-Wave right now. In the body of this paper, we will describe how this limits the size of a QAP that can be solved.

To solve this limitation, the D-Wave documentation [24] suggests that there is no choice but to break the large problem into smaller subproblems, solve them, and reconstruct the solution to the larger problem. The various methods suggested include cut-set conditioning, branch and bound, and large neighbourhood local search.

Connectivity

Each vertex in the Chimera graph is adjacent to at most six other vertices. This means that if a problem has variables that are connected to each other in a way that does not exist in the Chimera graph, then it cannot be embedded on the hardware without a work around. In graph theoretic terms, if the problem has a primal graph which is not a subgraph of the Chimera graph then we require a work around to solve the problem.

To solve this limitation, we use multiple qubits in the Chimera graph to represent a qubit in our problem. We define a physical qubit as one that exists in the quantum hardware, and a logical qubit as one that we use in the problem formulation.

For example, using the objective function in equation 2.6, we can make two physical qubits, q_1, q_2, represent the same logical qubit, we can set $b_{12} = -2$ and $a_1 = a_2 = 1$. Then the objective function will be minimised exactly when $q_1 = q_2 = 1$ and $q_1 = q_2 = 0$. This is illustrated in Table 2.1.

q_1	q_2	$O(a,b;q)$
0	0	0
0	1	$a_2 = 1$
1	0	$a_1 = 1$
1	1	$a_1 + a_2 + b_{12} = 1 + 1 - 2 = 0$

Table 2.1: Representing one logical qubit with two physical qubits

Embedding Complete Graphs

For a Chimera graph with parameters L, M, N, the D-Wave documentation [24] states that the largest complete graph with n vertices is where $n = 1 + L \cdot \min(M, N)$.

Embedding Arbitrary Graphs

Embedding an arbitrary problem with graph structure H on to the Chimera graph is equivalent to the problem of finding H as a minor of the Chimera graph. Currently, the best known algorithm for finding H as a minor of a graph G has a runtime of $O(2^{(2k+1)\log k}\,|H|^{2k}\,2^{2|H|^2}\,m)$, where k is the branchwidth of G and m is the number of edges in G [6].

As the exponential runtime is undesirable, D-Wave Systems uses a heuristic search for graph minors [13]. The heuristic has a runtime of $O(n_H\ n_G\ e_H\ (e_G + n_G \log n_G))$ where n_H, e_H is the number of vertices and number of edges in H respectively and same for n_G, e_G. In practice, the heuristic typically runs in $O(n_H\ n_G)$. They tested the heuristic by embedding complete graphs, grid graphs, and cubic graphs on to the 512 vertex Chimera graph. The largest complete graph that fits in the 512 vertex Chimera graph (K_{33}) embeds successfully with high probability in less than two seconds.

2.2.3 Controversy

There has been some controversy and debate about whether the D-Wave quantum compter is actually using quantum mechanics in a significant way and if it actually offers any benefits over classical computers.

The major studies that suggest that the D-Wave quantum computer is actually quantum are:

- An article in the May 12, 2011 edition of *Nature* in which academics claim that the D-Wave chips have some of the quantum mechanical properties required for quantum computing [26].
- In 2011, there was an article in *Science* which claimed that the D-Wave is "at least a little quantum mechanical" [14].
- In March 2014, researchers published a paper comparing experimental data from the D-Wave with three explanations from classical physics and one explanation from quantum physics. They claim that the quantum model fits the experimental data better [7].
- In May 2014, researchers published a paper containing experimental results that suggest the entanglement of qubits in the D-Wave [30].
- In December 2015, researchers from Google published a paper comparing the D-Wave against simulated annealing and quantum Monte Carlo algorithms. They found that quantum annealing on the D-Wave is faster and scales better than simulated annealing. They also found that quantum annealing on the D-Wave scales similarly to quantum Monte Carlo algorithms, but is around 10^8 times faster [16].

The major studies that suggest that the D-Wave quantum computer may not be quantum are:

- In 2013, in reply to the McGoech and Wang paper [33] on how the D-Wave is faster than a few classical algorithms, researchers showed that other classical algorithms are faster then the D-Wave [39] [4].

- In January 2014, researchers published a classical model to explain the D-Wave's experimental data, suggesting that it may not be a quantum computer [40].
- In June 2014, a study published in *Science* found that the D-Wave produced no quantum speedup and that there was no quantum evidence. This includes the fact that simulated annealing on a classical computer was faster than quantum annealing on the D-Wave [37].

2.2.4 Analysis and Critique

Quantum Hardware

The limitations of the quantum hardware are very significant to this project. In particular, the limited connectivity will cause the number of qubits required to solve a QAP to increase significantly. We will discuss that more thoroughly in the body of the paper.

Another important thing to note is that the quantum hardware may not always be the full Chimera graph. Due to certain problems with the hardware, some of the qubits may be turned off [10]. As such, ideally any method developed would be somewhat resistant to small changes in the structure of the graph. Therefore, we need to ensure that we formulate the QAP in a way that allows us to find an embedding quickly with high probability. The question of finding H as a minor of G seems to be a relatively new one that is inspired by the D-Wave [13].

Controversy

There are two main questions surrounding the controversy:

- Is the D-Wave actually using quantum mechanics?

- Can the D-Wave actually compute anything faster than classical computers?

For the first question, most research agrees that the D-Wave chips are capable of some quantum mechanics at least on a small number of qubits. It is unclear whether the quantum effect is still present when a large number of qubits are used.

For the second question, it is important to realise that there are two aspects to comparing computational speed. The first is actual runtime of the machines, and the second is how the runtime scales as the input size increases.

A comparison of the actual runtime on the D-Wave and on a classical computer can tell us if there is currently any speedup that can be obtained for solving problems. Most research suggests that while the D-Wave may be faster than specific algorithms and generalised solvers, there are some classical algorithms that are faster than the D-Wave. As such, there are no practical problems that we can solve faster on the D-Wave right now.

A comparison of how the runtime scales as the input size increases is strongly linked to the first question. If the runtime scales better on the D-Wave, then it suggests that there has to be something quantum occuring that causes it to scale better than classical algorithms. The strongest result that suggests that this might be true is the one from December 2015 [16], where quantum annealing is shown to scale better than simulated annealing. However, it still scales similarly to other classical algorithms like quantum Monte Carlo.

In addition, increasing the input size and therefore the number of qubits required on the D-Wave potentially might require some error correction due to the difficulties in engineering quantum chips with many qubits. Some researchers think that the error correction would make the D-Wave more inefficient and cause it to perform worse than how it appears to scale right now.

Note that this project does not rely on the D-Wave being quantum. We are interested in contributing to the debate by exploring how the QAP can be solved on the D-Wave, and through this understand more about the two questions above.

2.3 Solving the QAP Using Quantum Computation

In this section, we explore the current efforts to solve the QAP using the D-Wave quantum computers.

McGeoch and Wang [33] made a comparison between the D-Wave Blackbox (described in detail in Section 2.3.1), CPLEX [1], METSlib tabu search (TABU) [2], and a branch-and-bound solver called Akmaxsat (AK) [29].

In their paper, they mention that a constrained QAP of size n can be transformed into an unconstrained QUBO problem with n^2 variables and using a penalty function to enforce 1-to-1 assignments. They note that this causes problem sizes to increase to ranges that cannot be represented on the D-Wave. Therefore, they chose to use the Blackbox solver when solving the QAP. In doing so, they used a more compact transformation with an objective function that cannot be represented by a QUBO.

They ran the software solvers with a 30 minute timeout and the Blackbox solver with a limit of 10^7 objective function evaluations. They found that out of 33 test cases, Blackbox obtained the best solution 24 times, TABU obtained the best solution 5 times, Blackbox and Tabu tied for the best solution 4 times. The other two solvers performed much worse. Blackbox and Tabu found the optimal solution 6 and 5 times respectively.

In another study, Alex Selby [39] found that a simple QAP solver, Prog-QAP, is at least 12000 times faster than the Blackbox solver. Prog-QAP starts off with a random permutation and keeps trying transpositions until there is no more benefit. It then repeats the process until it runs out of time.

2.3.1 D-Wave Blackbox Solver

The D-Wave Blackbox is a hybrid solver developed by D-Wave Systems that alternates a Tabu heuristic search with quantum hardware queries. The Blackbox solver takes as input a general problem instance P with an objective function defined on n binary variables (where n is limited by the number of qubits in hardware). The solver starts with a random initial solution, and then iterates towards a solution that minimises the objective function for P. In each iteration, the solver builds a set of neighbouring solutions based on a query to the quantum hardware. The solver selects a minimum cost solution that is not on its Tabu list. The solver iterates until it reaches a preset limit on the total number of objective function evaluations.

2.3.2 Analysis and Critique

The major difference between the two studies is the way the experiments were conducted. In the McGeoch and Wang study, they set a seemingly arbitrary limit on the number of objective function evaluations and time limit for the classical algorithms. The approach clearly favours heuristic based solvers as solvers like CPLEX attempt to find and prove an optimal solution. In addition, they provided Blackbox with the more compact transformation, and it can be argued that that affected the timing and quality of solutions found by Blackbox.

It is this same approach that causes the large difference in timing for the second study. The second study estimated the time taken for the D-Wave part of the Blackbox solver to solve the QAP based on the number of objective function evaluations. As the number of objective function evaluations is seemingly arbitrary (no explanation given in the paper), it is difficult to conclude exactly how the D-Wave compares to Prog-QAP.

Through this, we can see how the approach chosen by McGeoch and Wang is not suited to compare computational timing.

Note also that the McGeoch and Wang paper did not provide any details on how the QAP was transformed into input for the Blackbox solver.

2.4 Summary

In this chapter, we explored the essential background for this project.

We began by introducing the QAP and the formulation of the problem that we will refer to in the project. We discussed the difficulty of the QAP by analysing it from a complexity theory point of view, noting that it is NP-hard and has no polytime k-approximation (unless P=NP). We also note that problem sizes as small as $n = 30$ have not been solved to optimality.

Next, we looked at the various methods of solving the QAP currently. We explored exact solution algorithms such as the branch and bound method and using distributed computing to parallelise the solving. We also looked at heuristic based methods such as Tabu search, simulated annealing.

We then introduced the D-Wave quantum hardware. We looked at how the quantum machine instructions allow us to set the weight of a qubit and the strength of a coupler and how that translates into an objective function. We presented how the result from the machine are samples from a probability distribution based on the objective function.

We then introduced the layout of the quantum hardware graph (the Chimera graph). From this, we discussed the limitations that are introduced because of the limited number of qubits and limited connectivity. We also presented the current methods of finding a graph minor.

We also discussed the controversy over the D-Wave quantum computer, presenting the debate on whether the D-Wave is actually quantum, and if it can provide any meaningful speed up over classical computers and algorithms.

Next, we discussed the current studies on solving the QAP using the D-Wave quantum computer. There have been only two major studies, one by McGeoch and Wang and the other by Alex Selby. McGeoch and Wang found that the Blackbox solver was able to find better solutions than the other solvers 23 out of 33 time and tied for best five times. On the other hand, Alex Selby found that a simple QAP solver, Prog-QAP, was able to find equal or better solutions much faster. Ultimately, the two studies had approaches with different goals, which is why the results vary so much.

Chapter 3

Solving the QAP

In this chapter, we describe how to solve the QAP natively on the D-Wave quantum hardware. We show that in the worst case, in order the solve a QAP of size n we need $O(n^4)$ qubits, and therefore it seems that it is currently infeasible to solve generic QAPs of significant sizes natively on current generation D-Wave quantum computers.

We attempt to show why it is difficult to reduce the number of qubits required to embed even non-worst case QAPs, and what needs to be solved in order to reduce the number of qubits required.

3.1 Overview

The high level idea of solving the QAP directly on the D-Wave quantum hardware is to first convert it into a QUBO problem, and then embed that QUBO problem directly on to the hardware. In the next section, we start by showing how to embedding the QUBO in the D-Wave hardware, as that places some restrictions on how we want to transform the QAP to QUBO.

3.2 QUBO to D-Wave hardware

A quadratic unconstrained binary optimisation (QUBO) problem is the following:

$$\min_{X \in \{0,1\}^n} \sum_{i=1}^{n} \sum_{j=1}^{n} Q_{ij} \cdot x_i \cdot x_j \tag{3.1}$$

Note that the QUBO can be represented as a graph, with each x_i as a vertex and Q_{ij} being the weight of the edge between x_i and x_j. In this way, the QUBO is exactly what the D-Wave quantum hardware solves, and is described as a problem native to the D-Wave. The only difference is that a QUBO problem may have variables connected to each other that do not exist on the D-Wave hardware based on the Chimera graph. Recall that we can use multiple physical qubits to represent one variable in the QUBO, and that finding the embedding is equivalent to finding the QUBO graph as a minor of the Chimera graph.

Recall also that the largest complete graph with n vertices that is a minor of a $L \times M \times N$ Chimera graph is $n = 1 + L \cdot \min(M, N)$. The latest generation of D-Wave quantum computers have 1152 qubits, and so assuming $L = 4$ and $M = N = 12$, the largest complete graph that is a minor of that graph has $n = 1 + 4 \cdot 12 = 49$ vertices. The equations to calculate the number of qubits required given a complete graph with n vertices is as follows (assuming $M = N$):

$$\begin{aligned} M &= \frac{n-1}{L} \\ \text{Number of qubits } &= 2 \cdot L \cdot M \cdot N \\ &= 2 \cdot 4 \cdot \left(\frac{n-1}{4}\right)^2 \\ &= \frac{n^2 - 2n + 1}{2} \end{aligned} \tag{3.2}$$

Recall that the heuristic based search finds a graph minor for a complete graph with high probability quickly. As presented in the next section, the QAP to QUBO transformation generates a QUBO that is completely connected and therefore finding the embedding is unlikely to present much of a problem.

3.3 QAP to QUBO

Recall that a QAP is the following problem:

$$\min_{X \in \Pi} \sum_{i,j,k,l=1}^{n} c_{ijkl} \cdot x_{ij} \cdot x_{kl} \tag{3.3}$$

where Π is the set of all permutation matrices. A formulation of the QUBO is in Equation 3.1. In this section, we present a way to convert the QAP into a QUBO.

To visualise the transformation more easily, we use a pair of numbers to index into the variables in the QUBO problem. So we get:

$$\min_{X \in \{0,1\}^{n^2}} \sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{k=1}^{n} \sum_{l=1}^{n} Q_{(i,j)(k,l)} \cdot x_{(i,j)} \cdot x_{(k,l)} \tag{3.4}$$

Then, ignoring the constraints on X, we can see that a QAP of size n can be represented as a QUBO of size n^2 where $Q_{(i,j)(k,l)} = c_{ijkl}$ and $x_{(i,j)} = x_{ij}$.

3.3.1 Worst Case QAP Transformation

Before considering how to represent the constraints, note that in the worst case where all $c_{ijkl} \neq 0$ in the QAP, we always end up with a fully connected QUBO. This means that a QAP of size n transforms into a fully connected QUBO of size n^2, which then requires $O((n^2)^2) = O(n^4)$ physical qubits in the Chimera graph to be able to embed it. For the latest generation of D-Wave quantum computers with 1152 physical qubits, that means that it can solve a QUBO of size 49 and therefore a QAP of size 7.

A QAP of size 7 is very small and can be solved to optimality on a classical computer very quickly. As such, currently in the worst case it is infeasible to solve a QAPs with size greater than 7 on the D-Wave.

However, there are many QAP problems that are not fully connected, with many $c_{ijkl} = 0$. In this case, it may be possible to transform them into a QUBO that is not fully connected and allow us to solve larger QAPs.

In the following section, we investigate how to represent the constraints of a QAP in the QUBO, and attempt to prevent generating a densely connected QUBO when doing so.

3.3.2 Representing the Constraints

We want to represent the constraints of the QAP in the QUBO. Note that the QUBO is unconstrained, with the only constraint being that the variables are binary. On the other hand, X in the QAP needs to satisfy:

$$\begin{aligned} &\forall j \in N \quad \sum_{i=1}^{n} x_{ij} = 1 \\ &\forall i \in N \quad \sum_{j=1}^{n} x_{ij} = 1 \\ &\forall i,j \in N \quad x_{ij} \in \{0,1\} \end{aligned} \tag{3.5}$$

The third constraint is that all the variables are binary, which is already present in the QUBO so we do not need to explicitly represent it. The first two constraints can be represented in the QUBO using a penalty function. There are many ways to represent a penalty function, and we investigate three ways of doing so.

Squaring

The penalty function can be encoded by squaring the constraints and obtaining:

$$\min_{X\in\{0,1\}^{n^2}} X^TQX + \lambda \cdot \left(\sum_{i=1}^{n} \left(\sum_{j=1}^{n} x_{ij} - 1 \right)^2 + \sum_{j=1}^{n} \left(\sum_{i=1}^{n} x_{ij} - 1 \right)^2 \right) \tag{3.6}$$

where λ is the penalty for breaking any constraints, and is just a large number so that minimising the objective will occur when the expression that λ is multiplied with equal to zero.

This approach is straight forward and can be represented directly as a QUBO. Unfortunately, this results in a very densely connected QUBO which we want to avoid. This approach results in an edge between x_{ij} and $x_{ij'}$ for all $1 \leq i, j, j' \leq n$, as well as between x_{ij} and $x_{i'j}$ for all $1 \leq i, i', j \leq n$. This results in every x_{ij} having $2n-2$ neighbours in the graph without including edges from Q.

Absolute Value

$$\min_{X\in\{0,1\}^{n^2}} X^TQX + \lambda \cdot \left(\sum_{i=1}^{n} \left| \sum_{j=1}^{n} x_{ij} - 1 \right| + \sum_{j=1}^{n} \left| \sum_{i=1}^{n} x_{ij} - 1 \right| \right) \tag{3.7}$$

Let $a_i = \sum_{j=1}^{n} x_{ij} - 1$ and $b_j = \sum_{i=1}^{n} x_{ij} - 1$ to obtain:

$$\min_{X\in\{0,1\}^{n^2}} X^TQX + \lambda \cdot \left(\sum_{i=1}^{n} |a_i| + \sum_{j=1}^{n} |b_j| \right) \tag{3.8}$$

For simplicity, just consider:

$$\min \sum_{i=1}^{n} |a_i| \tag{3.9}$$

One way to deal with the absolute value is to introduce a new variable a'. Note that $|a'_i|$ is the smallest number that satisfies both $a_i \le a'_i$ and $-a_i \le a'_i$. So we can form a new equation with those constraints:

$$\begin{aligned} \min \sum_{i=1}^{n} a'_i \\ \text{subject to: } a_i \le a'_i \quad \forall\, 1 \le i \le n \\ -a_i \le a'_i \quad \forall\, 1 \le i \le n \end{aligned} \tag{3.10}$$

Note that in removing an equality constraint, we introduced an inequality constraint with the same variables. If there is a way to represent the inequality constraint in the QUBO without coupling all the variables, then we will be able to generate a QUBO that is not completely connected.

One method to represent the inequality constraint is to turn it into an equality constraint by introducing another two non-negative variables γ, γ'. The constraints can then be represented as:

$$\begin{aligned} a_i - a'_i + \gamma_i = 0 \quad \forall\, 1 \le i \le n \\ -a_i - a'_i + \gamma'_i = 0 \quad \forall\, 1 \le i \le n \end{aligned} \tag{3.11}$$

However, this results in an equality constraint with more variables than we started off with, and therefore is not very useful. Unfortunately, we are unaware of any other method that transforms the inequality constraint into a contraint with fewer variables.

We are aware of a few other ways to represent the absolute value. However, all those methods involve treating $\sum_{j=1}^{n} x_{ij} - 1$ as a single variable or entity and using it in a few other equalities or inequalities. In doing so, we return to the same problem with size no smaller than what we started off with. Therefore, we think that it seems unlikely that using absolute values in the penalty function will allow us to avoid generating a QUBO that is at least as densely connected as the one using squares.

Maximum

$$\begin{aligned}\min_{X\in\{0,1\}^{n^2}} X^TQX + \lambda \cdot \Bigg[&\sum_{i=1}^{n} \max\left(\left(\sum_{j=1}^{n} x_{ij} - 1\right), -\left(\sum_{j=1}^{n} x_{ij} - 1\right)\right) \\ &+ \sum_{j=1}^{n} \max\left(\left(\sum_{i=1}^{n} x_{ij} - 1\right), -\left(\sum_{i=1}^{n} x_{ij} - 1\right)\right)\Bigg]\end{aligned} \tag{3.12}$$

This formulation is exactly equivalent to the absolute value formulation and can be transformed in the same way.

Bottleneck

We explored three options to represent the constraints: Squaring, Absolute value, Maximum. All three options produce the same connectivity of the generated QUBO, with each x_{ij} having at least $2n - 2$ neighbours. Out of those three options, squaring is the most straight forward and since it results in the same connectivity, we use it to represent the penalty function.

When dealing with a QAP where many $c_{ijkl} = 0$, representing the QAP constraints in the QUBO is what causes the number of qubits required to increase significantly. Finding an alternative way to represent the constraints would increase the size of QAPs that can be solved to increase significantly.

3.4 Lower Bound on Qubits Required

Using the squaring method of representing the constraints on the QAP, we can come up with a lower bound on the number of qubits required to embed a QAP in the Chimera graph.

The squaring method generates a QUBO with n^2 variables, with each variable connected to $2n-2$ other variables: the ones in the same row and column. To lower bound the number of qubits required, we calculate the number of qubits required for a less connected structure.

Consider the graph G with n complete graphs with n vertices each. Note that the edges in G are a subset of the edges in the generated graph, equivalent to implementing the constraints on rows and not columns. Then each complete graph of n vertices requires $\frac{n^2-2n+1}{2}$ qubits to represent, so n such complete graphs would require $\frac{n^3-2n^2+n}{2}$ qubits. So on a current generation D-Wave quantum computer with 1152 qubits, the largest possible QAP that can be solved if the squaring method is used is of size 13.

3.5 Summary

In this chapter, we presented a transformation from QAP to QUBO, and discussed embedding the generated QUBO on the D-Wave quantum hardware. We showed that in the worst case where all $c_{ijkl} \neq 0$, the generated QUBO is fully connected and therefore a QAP of size n would require $O(n^4)$ physical qubits to embed.

We then showed how even for QAPs with many $c_{ijkl} = 0$, representing the QAP constraints results in a densely connected graph. We presented three alternatives for representing the constraints, all resulting in the same connectivity of the generated QUBO. We then computing a lower bound for the number of qubits required assuming that we have to use the squaring method of representing constraints.

Chapter 4

Representing Constraints for Certain QAPs

In the previous chapter, we discussed how representing constraints in the QAP causes the generated QUBO to be densely connected. In this chapter, we present a way to reduce the connectivity for a certain type of QAP.

4.1 QAPs with Rows of Zeroes

The subset of QAPs that we investigate are QAPs with multiple rows of all zeroes in the flow and/or distance matrix. These QAPs were introduced by Eschermann and Wunderlich [20] and can be found in the QAPLIB [11] with names with prefix `Esc`. These QAPs stem from the testing of self-testable sequential circuits trying to minimise the amount of additional hardware required.

4.2 Representing the Constraints

Without loss of generality, assume that the last two rows of the flow matrix are all zero. This means that placing the last two facilities in any location does not change the cost at all. As such, this is equivalent to placing the last facility twice, and not placing the second last facility at all.

In terms of constraints, it is easier to consider the permutation matrix than the sums. When we have two rows of zeroes in the flow matrix, we can relax the constraint and allow a slightly modified permutation matrix. We want each location to still have something mapped to it, so each column should still have exactly one 1. For the first $n - 2$ rows, we also want to ensure that we have exactly one 1 in each row. However, for the last two rows, both facilities are equivalent in that they will never affect the cost. As such, we can allow the last two rows to have one 1 in each row OR two 1s in one row and zero 1s in the other row.

Then note that given the constraints on the columns and on the first $n - 2$ rows, we are guaranteed that the last two rows have either one 1 each or two 1s in one row and zero 1s in the other. As such, we can remove the constraints on the last 2 rows.

Alternatively, we can reduce the number of variables required and remove all the variables corresponding to the last row of the permutation matrix (i.e. x_{n1}, x_{n2}, $\ldots$, x_{nn}). We let the $n - 1^{\text{th}}$ row represent the last two rows and in this way reduce both the number of variables and the connectivity of the generated QUBO problem.

Note that we can extend this to more than two rows and to any row by removing the variables corresponding to all the rows that contain all zeroes.

Concrete examples of these types of QAPs are provided in section 6.2.

Duplicate Columns

In fact, we can extend this to any QAP with duplicate rows and remove the constraints for all duplicate rows. However, this type of QAP rarely seems to occur outside of rows of all zeroes.

We can also extend this to work with duplicate columns instead of rows. In the case where we have a QAP that is not symmetric, it is possible that we encounter duplicate columns but not duplicate rows. If so, note that the argument above holds if you switch rows and columns, therefore we can remove constraints for duplicate columns while keeping the constraints for duplicate rows.

Multiple Types of Duplicate Rows

In the transformation above, we assume that there is only one row being duplicated. It is possible that the QAP has more than one row with duplicates. In this case, removing the variables representing the rows will no longer work because we need to ensure that the different duplicated rows have the right number of 1s in them.

We can solve this problem by adding a constraint for the collapsed rows. For example, if the QAP had one row duplicated x times and another distinct row duplicated y times, then we can collapse those into two rows i and j respectively. Then we add a constraint on the sum of row i equals to x and the sum of row j equals to y. In this way, we reduce the variables required for $(x-1)+(y-1)$ rows, while the connectivity for row i and j does not increase.

Chapter 5

Decomposition Approach

In Chapter 3, we see how the the Chimera graph layout of the D-Wave quantum computer limits the size of QAPs that we can embed. In this chapter, we present a decomposition approach to circumvent the limit and solve QAPs of arbitrary sizes.

5.1 Branch and Bound

The branch and bound technique is one that is widely used to solve the QAP on classical computers. The decomposition approach that we present uses branch and bound until the QAP is sufficiently small and can be solved on the D-Wave quantum computer.

The reason for choosing branch and bound instead of other decomposition techniques for the QAP is that we can exploit well established techniques such as distributed computing that help to reduce computation and parallelise the computation.

So the decomposition approach is as follows:

- Perform standard branch and bound.
- When we decompose to QAPs that can be solved on the D-Wave quantum computer, use a call to to the D-Wave to solve that QAP.

Here is some pseudo code for the branch and bound algorithm.

Algorithm 1 Decomposing QAP

```
procedure SOLVEQAP(Q)
    if lowerBound(Q) $>$ upperBound then return suboptimal
    else
        if $|Q| \leq$ max_QAP_size then return solve_QAP_on_DWave(Q)
        else
            best = upperBound
            for all subproblem in branch(Q) do
                if solveQAP(subproblem) = suboptimal then continue
                else
                    if solveQAP(subproblem) $<$ best then
                        best = solveQAP(subproblem)
            return best
```

5.2 Deciding When to Use the D-Wave

Note that in the branch and bound idea above, deciding when to use the D-Wave to solve the QAP is important. Attempting to solve a QAP that cannot be embedded on the D-Wave is a waste of computation. We also want to avoid decomposing the problem too much, as that underutilises the D-Wave. As such, how we choose when to use the D-Wave is important. However, determining if a QAP can be embedded on the D-Wave quantum computer is not a trivial task. In this section, we discuss a few options to do so.

One option is to decide when to use the D-Wave based on the size of the QAP. At this point, we have an upper bound on the number of qubits used as shown in the worst case analysis in Chapter 3. Using that upper bound, we can calculate the maximum size of a worst case QAP and whenever we decompose the QAP to that size, then we use the D-Wave.

The benefits to using this scheme is that there is almost no computational cost required in making the decision. Note that we can also do some pre-computation to find the embedding, reducing the overhead required when making the call to the D-Wave. The disadvantage of using this scheme is that we may have lots of unnecessary computation decomposing QAPs that do not need to be decomposed.

Another option is to decide when to use the D-Wave based on computing an upper bound on the number of qubits for the specific QAP being considered. The benefits to using this option is that we can potentially save on a lot of unnecessary computation decomposing QAPs that can already be solved directly. However, this option relies on developing a method to find good quality upper bounds quickly. Also, the overhead for the call to the D-Wave is greater as we need to find an embedding every time we use the D-Wave, which is very computationally expensive.

Another option is to attempt to find an embedding, and proceed to solve it on the D-Wave directly when an embedding is found or give up after a certain amount of time. One additional advantage this has over the upper bound option is that once we decide to use the D-Wave, we already have the embedding and so the overhead for using the D-Wave is reduced.

Note that all the options can be used in conjunction with the others and so a heuristic based method to choose when to use which option is another option that can be used.

Chapter 6

Evaluation

In this chapter, we evaluate the results of Chapters 3 to 5:

- Bound on the size of the QAPs that can be solved on the D-Wave quantum computer.
- Bound on the size of QAPs with duplicate rows that can be solved on the D-Wave quantum computer.
- Performance of the decomposition approach.

6.1 Embedding Generic QAPs on the D-Wave

In this section, we evaluate the our algorithm for solving the QAP on the D-Wave by attempting to bound the number of qubits required to solve a QAP of size n. Finding a good bound is important as it affects our branch and bound algorithm on when it decides to switch to solve the QAP on the D-Wave.

In the worst case transformation, a QAP of size n gets transformed into a QUBO with n^2 variables, with each variable connected to all the other variables, forming a complete graph with n^2 variables. Embedding this graph would require $\frac{n^4-2n^2+1}{2}$ on the Chimera graph and is a tight bound. For the current generation of D-Wave quantum computers with 1152 qubits, the largest QAP that it can solve has size 7.

Assuming we use the method of squaring the constraints in the QAP to represent them in the QUBO, we computed a lower bound for the number of qubits required to represent a QAP. Recall from section 3.4 that the QUBO obtained after squaring the constraints has n^2 variables, with each one connected to $2n - 2$ other variables. The lower bound obtained is based on the subset of edges representing n complete graphs with n vertices.

The lower bound obtained is not a tight bound, and it is difficult to estimate the quality of the bound. One hint at the quality of the bound is that number of edges considered for the lower bound is exactly half the total number of edges in the graph.

6.2 Embedding QAPs with Duplicate Rows

In this section, we evaluate the effectiveness of the solution for QAPs with duplicate rows. We bound the number of qubits needed to solve such QAPs and the reduction in the number of edges in the generated QUBO.

We can lower bound the number of qubits required when we have duplicate rows in a similar way to how we obtained the lower bound for generic QAPs. Assume that we have k duplicate rows. Then we remove the constraints and variables for those k rows. The generated QUBO then has $n \cdot (n - k)$ variables, with n column constraints each on $n - k$ variables and $n - k$ row constraints on n variables.

Consider the subgraph representing only the row constraints. There are $n - k$ complete graphs with n vertices. As shown before, each complete graph with n vertices requires $\frac{n^2-2n+1}{2}$ qubits, so in total we require at least $\frac{(n^2-2n+1)(n-k)}{2} = \frac{n^3-2n^2+n-kn^2+2kn-k}{2}$ qubits.

Alternatively, we can find a lower bound using the subgraph representing only the column constraints. In this case, we get n complete graphs with $n-k$ vertices. In this case, each complete graph would require $\frac{(n-k)^2-2(n-k)+1}{2} = \frac{n^2+k^2-2nk-2n+2k+1}{2}$ qubits. Then in total we would require $\frac{n^3+k^2n-2kn^2-2n^2+2kn+n}{2}$ qubits.

In the case with multiple duplicate rows, the number of qubits required increases slightly as we have to use row constraints for each distinct duplicate row. So if there are k distinct rows that have been duplicated $m > k$ times, we end up with n column constraints, and $n-(m-k)$ row constraints.

In all the cases above, as the number of duplicated rows increases, the number of qubits required decreases. However, as long as at least one row constraint is required, then we have at least one copy of the complete graph with n vertices, and therefore the number of qubits required is at least $O(n^2)$.

6.2.1 Performance on Practical Problems

In this section, we investigate the performance of this method of representing constraints on practical problems: the set of problems by Eschermann and Wunderlich [20]. In Table 6.1, we show some examples of problems and the number of duplicate rows they have.

By collapsing the rows with all zeroes, we manage to reduce the size and number of variables required for some of these problems significantly. In particular, `Esc32g` has 27 rows with all zeroes and as a result we only need to enforce 5 row constraints, reducing the number of vari-

Name	Size	Rows of Zeroes	Qubits required (before)	Qubits required (after)
Esc16a	16	6	32513	12641
Esc16b	16	2	32513	24865
Esc16e	16	7	32513	10225
Esc16f	16	16	32513	0
Esc16g	16	8	32513	8065
Esc16i	16	7	32513	10225
Esc16j	16	9	32513	6161
Esc32a	32	8	523265	294145
Esc32b	32	8	523265	294145
Esc32c	32	12	523265	204161
Esc32d	32	14	523265	165313
Esc32e	32	25	523265	24865
Esc32g	32	27	523265	12641
Esc64a	64	42	8384513	989825
Esc128	128	97	134201345	7868545

Table 6.1: QAPs with large number of rows of zeroes

Size (n)	Number of non-zero rows
1 to 7	Embedded fully
8	6
9	5
10 to 12	4
13 to 16	3
17 to 24	2
25 to 49	1
≥ 50	0

Table 6.2: Maximum number of non-zero rows for QAPs that can be embedded on the D-Wave

ables needed from 1024 to 160 and the number of edges between variables from 31744 to 2800.

Using the worst case situation as described in section 3.3.1, we can calculate an upperbound on the number of qubits required before and after the transformation. Recall that based on 3.2, the current generation D-Wave machine with 1152 qubits can embed a complete graph of 49 vertices. Therefore, if there are enough rows of zeroes such that the number of remaining variables is less than or equal to 49 then the QAP can be embedded on to the D-Wave. Table 6.2 states the maximum number of non-zero rows for QAPs of various sizes that can still be embedded on the D-Wave.

This shows that there exists some practical QAPs with a large number of rows of zeroes and that can benefit significantly from this method of representing the constraints.

6.3 Performance of Decomposition Approach

In this section, we evaluate the performance of the decomposition approach and compare it against classical QAP solvers. Unfortunately we have not been able to test our solver on the D-Wave directly, but we can calculate the time taken based on a formula [33]. The time taken is $T = t_1 + kt_2$ where t_1 is the setup time per instance and t_2 is the sampling time, and k is the number of solutions sampled.

We currently do not know the number of samples required to obtain good solutions, but we can lower bound the time taken by assuming that we only take one sample. In the V5 D-Wave Two platform, $t_1 = 201$ms and $t_2 = 0.29$ms and therefore the time taken for one sample is 201.29ms.

However, solving to optimality for QAPs of size 7 can be done much faster on a classical computer. A simple solver using Python (not known for its speed) takes about 50 to 60ms to solve a QAP of size 7 on a laptop. Therefore, we can see that the overhead for initialising the D-Wave is currently too large to be useful for solving QAPs.

However, as the D-Wave hardware matures and the number of qubits increases, we will be able to solve larger QAPs and assuming that the setup time and sampling time do not increase significantly, the D-Wave quantum computer will be able to solve the decomposed QAPs faster than classical algorithms can.

Chapter 7

Conclusion

7.1 Summary of Achievements

This project presents an in-depth investigation at solving the QAP on the D-Wave quantum computer.

We showed that in the worst case, embedding a QAP of size n requires $O(n^4)$ qubits on the D-Wave quantum computer. We then showed that even for QAPs that are not the worst case, representing the constraints causes the generated QUBO to be very densely connected, and therefore is the bottleneck in solving significant sized QAPs on the D-Wave quantum computer. We investigated multiple ways of representing the constraints, all ending up with the same connectivity of the generated QUBO.

Then we offer a solution in two ways: removing some variables and constraints for a certain subset of QAPs, and a general decomposition approach to solve QAPs of any size.

For QAPS with duplicate rows, we present a method that involves removing the contraints for the duplicate rows. We also show that for multiple distinct duplicate rows, we can collapse each duplicate row into one row constraint. We show that for a set of practical problems, this can help to reduce the number of variables and number of edges in the generated QUBO significantly.

We present a branch and bound decomposition approach to QAPs, with the D-Wave quantum computer being used when the QAP has been decomposed sufficiently. Unfortunately, it appears that this method is currently not faster than just solving it on a classical computer as the size of the QAP that can be solved on the D-Wave is limited. There is potential for a speed up in the future if the number of qubits increases sufficiently. We also present a few options on how to decide when to use the D-Wave quantum computer.

7.2 Future Work

We would like to highlight future work that could be done which would make solving the QAP on the D-Wave quantum computer more feasible:

Representing the Constraints
As we have shown, finding a way to represent the constraints of the QAP in the QUBO that reduces the connectivity of the QUBO is essential in being able to solve larger QAPs on the D-Wave quantum computer.

Better Upper Bound for Qubits Required
Given a specific QAP instance, being able to compute a good quality upper bound on the qubits required cheaply is important in the branch and bound algorithm.

Finding Other QAPs with Reduced Constraint Requirements

Finding practical QAPs that have reduced constraint requirements could help in constructing a better upper bound and better decision making scheme in the branch and bound algorithm.

Bibliography

[1] IBM ILOG CPLEX Optimizer.

[2] METSlib: An Open Source Tabu Search Metaheuristic framework in modern C++.

[3] Forbes: D-Wave Sells Quantum Computer to Lockheed Martin, May 2011.

[4] D-Wave vs CPLEX Comparison. Part 1: QAP, June 2013.

[5] D-Wave Systems Announces the General Availability of the 1000+ Qubit D-Wave 2X Quantum Computer, August 2015.

[6] Adler, I., Dorn, F., Fomin, F. V., Sau, I., and Thilikos, D. M. Faster parameterized algorithms for minor containment. In *Algorithm Theory-SWAT 2010*. Springer, 2010, pp. 322–333.

[7] Albash, T., Vinci, W., Mishra, A., Warburton, P. A., and Lidar, D. A. Consistency tests of classical and quantum models for a quantum annealer. *Physical Review A 91*, 4 (2015), 042314.

[8] Anstreicher, K., Brixius, N., Goux, J.-P., and Linderoth, J. Solving large quadratic assignment problems on computational grids. *Mathematical Programming 91*, 3 (2002), 563–588.

[9] Anstreicher, K. M. Recent advances in the solution of quadratic assignment problems. *Mathematical Programming 97*, 1-2 (2003), 27–42.

[10] Bian, Z., Chudak, F., Israel, R., Lackey, B., Macready, W. G., and Roy, A. Discrete optimization using quantum annealing on sparse ising models. *Frontiers in Physics 2* (2014), 56.

[11] Burkard, R., Çela, E., Karisch, S. E., and Rendl, F. Qaplib quadratic assignment problem library: Problem instances and solutions.

[12] Burkard, R., Çela, E., and Pardalos, P. *The Quadratic Assignment Problem.* Kluwer Academic Publishers, 1998, pp. 241–338.

[13] Cai, J., Macready, W. G., and Roy, A. A practical heuristic for finding graph minors. *arXiv preprint arXiv:1406.2741* (2014).

[14] Cho, A. Controversial computer is at least a little quantum mechanical. *Science 13* (2011).

[15] Clausen, J. Branch and bound algorithms-principles and examples. *Department of Computer Science, University of Copenhagen* (1999), 1–30.

[16] Denchev, V. S., Boixo, S., Isakov, S. V., Ding, N., Babbush, R., Smelyanskiy, V., Martinis, J., and Neven, H. What is the computational value of finite range tunneling? *arXiv preprint arXiv:1512.02206* (2015).

[17] Drezner, Z. A new genetic algorithm for the quadratic assignment problem. *INFORMS Journal on Computing 15*, 3 (2003), 320–330.

[18] Drezner, Z., Hahn, P. M., and Taillard, E. A study of quadratic assignment problem instances that are difficult for meta-heuristic methods.

[19] Elshafei, A. N. Hospital layout as a quadratic assignment problem. *Operational Research Quarterly* (1977), 167–179.

[20] Eschermann, B., and Wunderlich, H.-J. Optimized synthesis of self-testable finite state machines.

[21] Gambardella, L. M., Taillard, . D., and Dorigo, M. Ant colonies for the quadratic assignment problem. *JOURNAL OF THE OPERATIONAL RESEARCH SOCIETY 50* (1997), 167–176.

[22] Gavett, J. W., and Plyter, N. V. The optimal assignment of facilities to locations by branch and bound. *Operations Research 14*, 2 (1966), 210–232.

[23] Gilmore, P. C. Optimal and suboptimal algorithms for the quadratic assignment problem. *Journal of the society for industrial and applied mathematics* (1962), 305–313.

[24] Inc, D.-W. S. Programming with QUBOs, 2014.

[25] James, T., Rego, C., and Glover, F. A cooperative parallel tabu search algorithm for the quadratic assignment problem. *European Journal of Operational Research 195*, 3 (2009), 810–826.

[26] Johnson, M. W., Amin, M. H. S., Gildert, S., Lanting, T., Hamze, F., Dickson, N., Harris, R., Berkley, A. J., Johansson, J., Bunyk, P., Chapple, E. M., Enderud, C., Hilton, J. P., Karimi, K., Ladizinsky, E., Ladizinsky, N., Oh, T., Perminov, I., Rich, C., Thom, M. C., Tolkacheva, E., Truncik, C. J. S., Uchaikin, S., Wang, J., Wilson, B., and Rose, G. Quantum annealing with manufactured spins. *Nature 473*, 7346 (05 2011), 194–198.

[27] Koopmans, T. C., and Beckmann, M. Assignment problems and the location of economic activities. *Econometrica: journal of the Econometric Society* (1957), 53–76.

[28] Krarup, J., and Pruzan, P. M. Computer-aided layout design. In *Mathematical programming in use*. Springer, 1978, pp. 75–94.

[29] Kügel, A. akmaxsat solver.

[30] Lanting, T., Przybysz, A., Smirnov, A. Y., Spedalieri, F., Amin, M., Berkley, A., Harris, R., Altomare, F., Boixo, S., Bunyk, P., et al. Entanglement in a quantum annealing processor. *Physical Review X 4*, 2 (2014), 021041.

[31] Lawler, E. L. The quadratic assignment problem. *Management science 9*, 4 (1963), 586–599.

[32] Li, Y., P.M., P., and Resende, M. A greedy randomized adaptive search procedure for the quadratic assignment problem. *DIMACS Series on Discrete Mathematics and Theoretical Computer Science 16* (1994), 237–261.

[33] McGeoch, C. C., and Wang, C. Experimental evaluation of an adiabiatic quantum system for combinatorial optimization. In *Proceedings of the ACM International Conference on Computing Frontiers* (New York, NY, USA, 2013), CF '13, ACM, pp. 23:1–23:11.

[34] Mirchandani, P. B., and Obata, T. *Locational Decisions with Interactions Between Facilities: The Quadratic Assignment Problem; a Review.* Laboratory for Computer Methods in Public Systems, 1979.

[35] Nugent, C. E., Vollmann, T. E., and Ruml, J. An experimental comparison of techniques for the assignment of facilities to locations. *Operations research 16*, 1 (1968), 150–173.

[36] Resende, M. G., Ramakrishnan, K., and Drezner, Z. Computing lower bounds for the quadratic assignment problem with an interior point algorithm for linear programming. *Operations Research 43*, 5 (1995), 781–791.

[37] Rønnow, T. F., Wang, Z., Job, J., Boixo, S., Isakov, S. V., Wecker, D., Martinis, J. M., Lidar, D. A., and Troyer, M. Defining and detecting quantum speedup. *Science 345*, 6195 (2014), 420–424.

[38] Sahni, S., and Gonzalez, T. P-complete approximation problems. *J. ACM 23*, 3 (July 1976), 555–565.

[39] Selby, A. D-wave: comment on comparison with classical computers, 2014.

[40] Shin, S. W., Smith, G., Smolin, J. A., and Vazirani, U. How" quantum" is the d-wave machine? *arXiv preprint arXiv:1401.7087* (2014).

[41] Taillard, E. D. Comparison of iterative searches for the quadratic assignment problem. *Location science 3*, 2 (1995), 87–105.

[42] Talbi, E.-G., Roux, O., Fonlupt, C., and Robillard, D. Parallel ant colonies for the quadratic assignment problem. *Future Gener. Comput. Syst. 17*, 4 (Jan. 2001), 441–449.

[43] Wilhelm, M. R., and Ward, T. L. Solving quadratic assignment problems by simulated annealing. *IIE transactions 19*, 1 (1987), 107–119.

[44] Zhao, Q., Karisch, S. E., Rendl, F., and Wolkowicz, H. Semidefinite programming relaxations for the quadratic assignment problem. *Journal of Combinatorial Optimization 2*, 1 (1998), 71–109.