Warehouse Optimisation Using Quadratic Assignment on New Hardware

Submitted by

Xu Jun

In partial fulfilment of the

requirements for the Degree of

Bachelor of Engineering (Computer Engineering)

National University of Singapore

B.Eng. Dissertation

Warehouse Optimisation Using Quadratic Assignment on New Hardware

By

Xu Jun

National University of Singapore

2019/2020

Project No: H074910
Advisor: A/P Wong Weng Fai, Dr Luo Tao
Deliverables:
Report: 1 Volume

Abstract

As quantum computing gains traction, there is a significant effort in applying it to real-world problems. One of the approaches is to use quantum annealing for the Quadratic Assignment Problem (QAP), which in turn has a wide range of applications. The strategy is to convert a QAP into Quadratic Unconstrained Binary Optimisation(QUBO) form which is compatible with hardware solver. However, the number of binary variables in QUBO is quadratic with respect to the size of the generating QAP. This leads to a lack of scalability in state-of-the-art quantum(-inspired) annealers. This project explores a novel heuristic that overcomes such scalability issue for a special class of QAPs satisfying a block-structural property. This property is frequently observed in practical settings such as warehouse allocation. The heuristic enables quantum(-inspired) annealers to solve block-structural QAPs of sizes linear to the number of physical qu-bits. The heuristic is explored on two platforms, the D-Wave 2000Q Quantum Annealer (QA) and the Fujitsu Digital Annealer (DA). Experiments are performed on standard datasets as well as in-house block-structural QAP instances, and performance comparison is made against traditional, software simulated annealing. Initial results demonstrate that the hardware platforms, with the new heuristic, produce solutions of comparable quality on standard datasets and are superior on in-house instances. Moreover, the hardware solutions achieve significant speedups. It is claimed that quantum(-inspired) annealers, with the help of the new heuristic, can be used to solve QAPs to good effect.

Subject Descriptors:
C5 Computer System Implementation

Keywords:
Quadratic Assignment Problem, Quadratic Unconstrained Binary Optimisation, Application-specific Hardware, Simulated Annealing

Implementation Software and Hardware:
Macintosh OS, Fujitsu Digital Annealer, D-Wave Quantum Annealer, Python 3.6

Acknowledgement

I would like to thank Dr Luo and Prof Wong for their invaluable guidance. I would also like to thank Dr Luo for generously supporting this project with resources from the Institute of High Performance Computing, A*STAR. Special thanks to Dr Anazawa from Fujitsu and Dr Xiaozhe Gu for their insights. This project would not be possible without them.

List of Figures

List of Tables

Table of Contents

Chapter 1

Introduction

1.1 QAP for Warehouse Optimisation

1.1.1 General warehouse assignment

Consider the problem of assigning items to locations in a warehouse. How should the assignment be done, so that when orders come in and items are picked, the travel distance of the picker is at a minimum? For a company, the reduction of such distance translates to the reduction of labour cost. That is the significance of the problem and the motivation for initiating this project.

The objective is to find a decent assignment that minimises travel distance for a given set of orders. There exist many assignment policies. An example from Tsige (Tsige, 2013). Consider a warehouse with a layout of Fig 1.1. It has three aisles and 30 locations in total. Each aisle can reach to the two columns on its left and right. There is an Input/Output (I/O) point at the bottom left corner through which items are transported in and out of the warehouse.

A set of orders is shown in Fig 1.2. There are 10 orders and 10 Storage Keep Units (SKUs). An SKU represents a unique type of item. The 10 SKUs are labelled 1 to 10. There are a total 30 items, each of which belonging to its particular SKU.

To pick an order, the picker starts from the I/O point. According to some routing strategy, he travels around the warehouse collecting the needed items and travels back to the I/O point

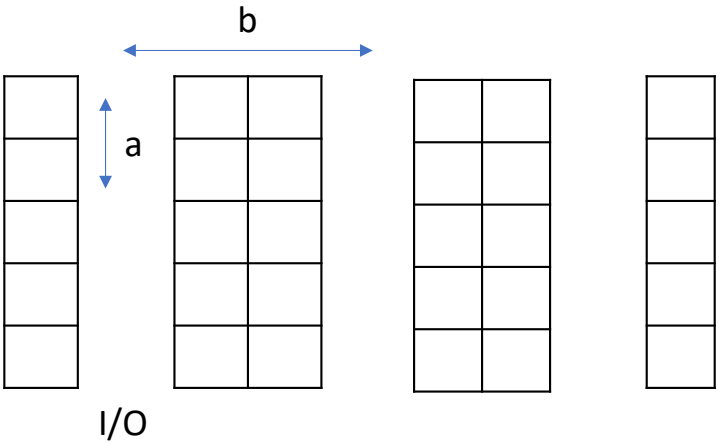


Figure 1.1: layout of an empty warehouse

Order #	SKUs				
1	1	2	3	4	5
2	1	6	7	8	
3	3	2	8		
4	1	4	5		
5	5	6	7		
6	1	5	6	7	
7	9				
8	10				
9	1	2	3	4	
10	5	4			

Figure 1.2: the set of orders

after all items in the order are collected. In this project the routing strategy is assumed to be S-shaped. This means that the picker goes from left to right, through the entire aisle if there is an item in that aisle to be picked, with no look-back. An example is illustrated in Fig 1.4b.

The assignment can be done in various ways. Below are a few of them.

Dedicated storage always assigns items to their own locations in the warehouse. This means every item has its dedicated location. This policy then relies on order pickers' memory power to minimise travel distance.

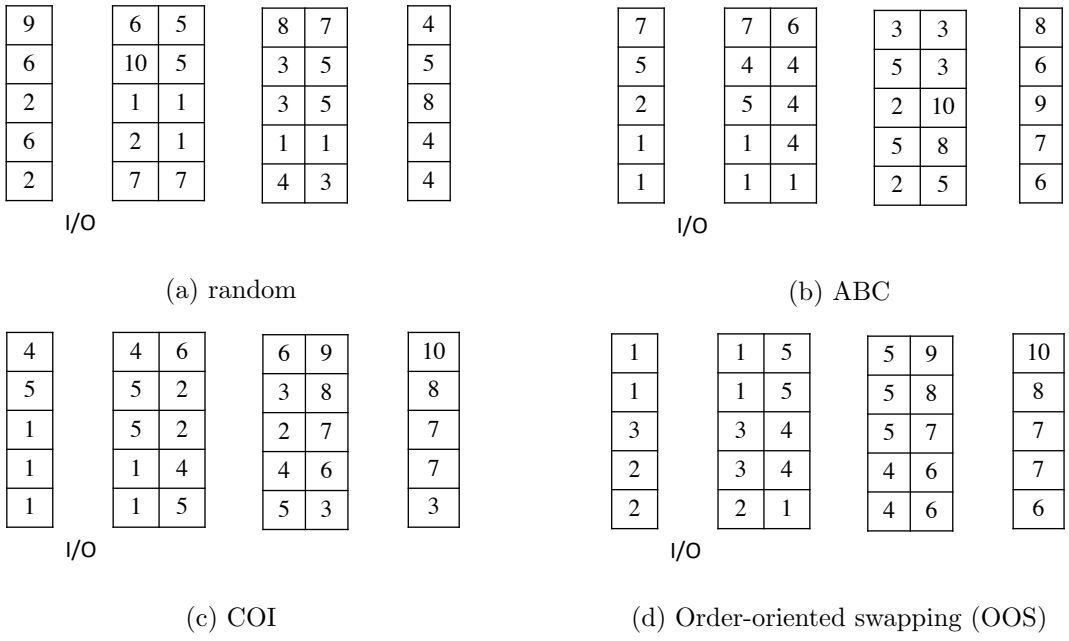
Random storage assigns items randomly. An instance is shown in Fig 1.3a.

ABC storage groups SKUs into classes according to popularity and allocates a dedicated area for each class. The number of popularity classes and the number of items in each class are arbitrary. For example, in Fig1.3b we have around 20%, 40% and 40% of items in classes A,B,C respectively, up to minor adjustments of discrete numbers. SKU 1 is the most popular and items of SKU 1 all fall into class A. Subsequently, SKUs 2,4,5 go to class B. The rest goes to class C. Next, count the number of locations required for each class and assign a dedicated area for the class. For example, class A requires 5 locations and gets an area at closest Euclidean distance to the I/O point.

Cube-per-order(COI) storage assigns items according to their ratio between volume and popularity. For the interest of this project, we consider all items to have identical volume so this reduces to arranging items according to popularity. An example is shown in Fig 1.3c.

Order-oriented-swapping (OOS) It starts with a random assignment and runs a certain number of iterations. In each iteration, two items in the assignment are swapped. The distance is recalculated and the new assignment is accepted if it reduces the distance, and is accepted with some probability otherwise. Effectively, this is the technique of **Simulated Annealing (SA)**. A generic introduction to SA is given in Appendix A.

To calculate the distance travelled in picking the order list of Fig 1.2, iterate through the order list. For each order, route through the warehouse in S-shape. For each item in the order, there will be multiple items in the warehouse of the same SKU, and the one in the leftmost aisle is chosen. Assuming the vertical distance $a = 1$, and the horizontal distance $b = 3$ in Fig 1.1,



(a) random

(b) ABC

(c) COI

(d) Order-oriented swapping (OOS)

Figure 1.3: storage layout after assignment

the distances of the various strategies (not including dedicated storage) is shown in Table 1.1.

Strategy	#vertical traversal	#horizontal aisle change	total distance
COI	28	34	270
Random	26	32	252
ABC	24	36	252
OOS	22	32	228

Table 1.1: Distances traversed in picking order set for various strategies

1.1.2 The QAP Strategy

The first QAP formalisation of warehouse assignment is due to Mantel (Mantel, Schuur, & Heragu, 2007). Following this, there are successful attempts in implementing QAP (de Ruijter & Schuur, 2007) for warehouse assignment and favourable results are observed. Below is a description of the QAP formalism.

In the QAP folklore, the language of "facilities" and "locations" are used to provide an easy

introduction. Consider n facilities and n locations. Between any two facilities there are products transported to and fro, and the amount of transport is called *flow*. Between two locations there is some *distance*. Define *traffic* as the product of flow and distance. How to assign facilities to locations such that the sum of traffic is minimised?

Formally, assume the following quantities:

n - #items = #locations

$F = (f_{ij})$ - The flow matrix for items. Each entry represents flow between items i and j.

$D = (d_{kl})$ - The distance matrix. Each entry represents distance between locations i and j.

$X = (x_{ij})$ - The binary decision matrix. Each entry represents whether item i is assigned to location j. Note that dimension of X is $n \times n$, which implies $O(n^2)$ binary decision variables with respect to the number of items.

Then, minimise:

$$\sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n}\sum_{l=1}^{n} f_{ij}d_{kl}x_{ik}x_{jl} \tag{1.1}$$

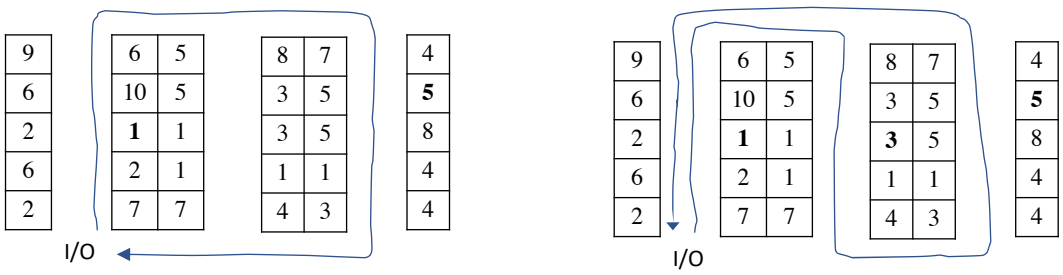
The answer is subject to the following constraints:

$$\sum_{k=1}^{n} x_{ik} = 1 \qquad \forall 1 \leq i \leq n \tag{1.2}$$

$$\sum_{i=1}^{n} x_{ik} = 1 \qquad \forall 1 \leq k \leq n \tag{1.3}$$

(1.2) imposes that each facility is assigned to exactly one location. (1.3) imposes that each locations contains exactly one facility. Solving for (1.1) yields a permutation from the facility set to the location set.

It is easy to apply QAP to the setting of warehouse assignment. To do so, simply replace "facilities" with "items". The flow between items is defined to be the frequency at which their respective SKUs appear in the same order. This can be computed based on the incoming order set. The distance between locations is *routing-specific*. With S-shaped routing, the distance between two items is roughly proportional to the number of columns between them, since the picker would in principle traverse those columns in a zig-zag manner. However, during actual picking the picker would skip a column if there is no item to be picked in that column, hence this distance is approximate. For example, in Fig 1.4, different situations result in different



(a) picking 1,5, skipping middle column

(b) picking 1,5, not skipping middle column

Figure 1.4: distances in different situations

distances between items 1 and 5. In 1.4a, there is no item to be picked in the middle so distance is approximately 2 columns' length. In 1.4b, item 3 has to be picked, so middle column has to be traversed and distance between 1 and 5 becomes 3 columns' length. It is assumed throughout the report that all columns in between are traversed when computing distance. We have arrived at the following formal formulation of QAP.

What inspires the application of QAP in warehouse assignment is the idea to assign items with high flows closer to each other. Intuitively, this would reduce the amount of *traffic* in picking orders because the picker is more likely to travel shorter distance in between items appearing in an order.

1.2 Quantum Annealer as QAP Solver

1.2.1 Motivation

QAP is NP-hard. A number of heuristic algorithms include Simulated Annealing, Genetic Algorithm, Tabu Search, etc. Even though such heuristic exist which return decent solution, the computation time for larger instances is still significant as in (de Ruijter & Schuur, 2007).

Quantum annealers, on the other hand, are new computing platforms inviting applications. The approach of quantum annealers is to solve optimisation problems in a generic Quadratic Unconstrained Binary Optimisation (QUBO) form. Many theoretical and real-world optimisation problems can be modelled as QUBO. Examples include portfolio optimisation in finance (Rosenberg, Haghnegahdar, Goddard, Carr, Wu, & De Prado, 2016), traffic flow management

(Neukart, Compostella, Seidel, von Dollen, Yarkoni, & Parney, 2017), least square solutions to polynomial mathematical equations (Chang, Lux, & Tipirneni, 2019), isomer search in chemistry (Terry, Akrobotu, Negre, & Mniszewski, 2020), and also traditional optimisation problems in computer science such as maximum clique (Naghsh, Javad-Kalbasi, & Valaee, 2019) and minimum multicut (Cruz-Santos, Venegas-Andraca, & Lanzagorta, 2019).

It is known that QAP can also be converted into QUBO, and this motivates the exploration of using quantum annealers to solve QAPs. The hope is to achieve a speed superior to conventional software heuristics and a solution quality comparable to conventional software heuristics.

1.2.2 The D-Wave Quantum Annealer

Below is a brief introduction to the inner workings of D-Wave Quantum Annealer (QA) provided in D-Wave's official technology overview.

Rather than store information using bits represented by 0s or 1s as conventional computers do, quantum computers use quantum bits, or qubits, to encode information as 0s, 1s, or both simultaneously. This superposition of states, along with the quantum effects of entanglement and quantum tunneling, enable quantum computers to consider and manipulate many combinations of bits simultaneously.

D-Wave's computers use quantum annealing to solve problems represented as mathematical functions (resembling a landscape of peaks and valleys) by harnessing the quantum mechanical effects, to find their global minima, corresponding to optimal or near optimal solutions. Computation is performed by initializing the quantum processing unit (QPU) into a ground state of a known problem and annealing the system toward the problem to be solved such that it remains in a low energy state throughout the process. At the end of the computation, each qubit ends up as either a 0 or 1. This final state is the optimal or near-optimal solution to the problem to be solved.

The following facts are needed for this report.

Firstly, the latest D-Wave 2000Q has over 2000 qubits. When such a machine is programmed, referring to the QAP expression (1.1), each binary variable corresponds with one qubit. For

each qubit there is a programmable bias term corresponding with the linear coefficient of (1.1), where linear coefficient arises from the fact that $x^2 = x$ for binary x. For each pair of qubits there is a programmable coupler strength corresponding to the quadratic coefficient.

Secondly, due to the above correspondence, a 2000-qubit machine would theoretically support a QAP with 2000 variables or, equivalently, a QAP of $\sqrt{2000}$ items and locations, because number of binary decision variables is $O(n^2)$. However, this theoretical limit is only achieved if a coupler exists between each pair of qubits (so that each quadratic coefficient can be programmed). This is not the case for D-Wave 2000Q because of its *Chimera* architecture, as in Fig 1.5 (D-Wave, 2019). 8 qubits form a unit cell, and there are 8 couplers between each pair of unit cells, whereas a fully connected graph would have 64. This is a hard technological constraint.

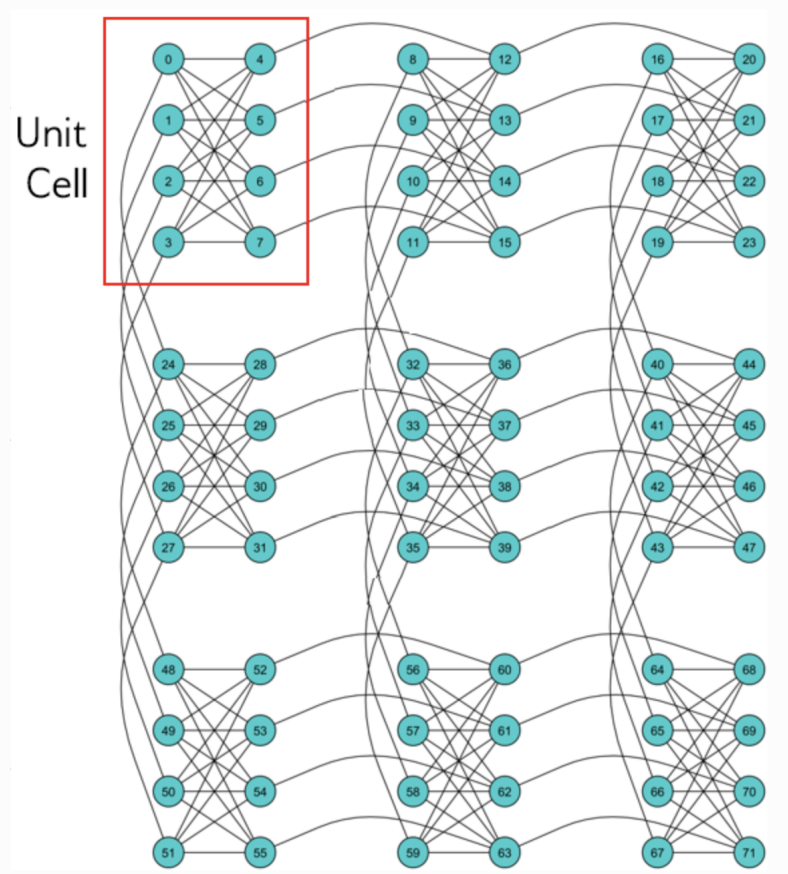


Figure 1.5: Chimera architecture

1.2.3 The Fujitsu Digital Annealer

The Fujitsu Digital Annealer (DA) is a CMOS architecture that is a parallel rendition of the SA algorithm. Its high-level architecture is shown in Fig 1.6 (Tsukamoto, Takatsu, Matsubara, & Tamura, 2017).

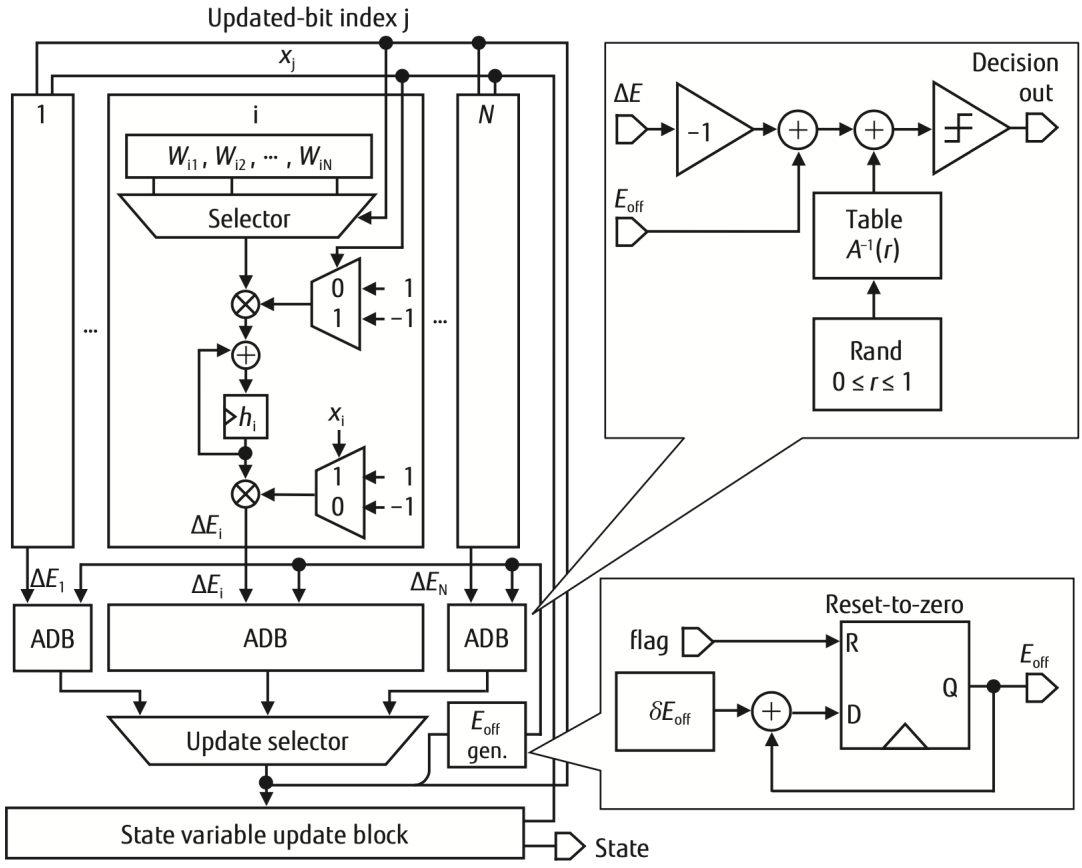


Figure 1.6: DA architecture

In each cycle there are two operations: trial and update. As in the figure, trial operation is parallelised as each bit x_i is considered for change in parallel, with respect to the current state X. Then, new objective function value X_i is calculated for each of the individual bit-flip x_i. The Acceptance Decision Block (ADB) processes the Metropolis criterion, similar to SA algorithm as in Appendix A. In the end of the trial phase, assuming there are N binary variables, N energy changes will be calculated and some of them will be accepted by ADB. Next, the update selector will select one of the acceptances and update the energy accordingly.

The benefit of doing parallel trial is that more bit-flips are considered and thus higher

probability of moving to a neighbouring state in each cycle. This effectively speeds up the annealing process, and with some techniques is able to achieve a tremendous speedup of 12,000X for 32-city Traveling Salesman Problem. Detail is available in (Tsukamoto et al., 2017).

1.2.4 Workflow

A QUBO is defined as an unconstrained optimisation problem whose objective function is a binary quadratic form.

It is apparent that expression (1.1) is a binary quadratic form. By linear algebra, a binary quadratic form is a polynomial of binary variables such that every term has degree of 2. It is then easy to convert the function to the following form:

$$\bar{x}^T Q \bar{x} \tag{1.4}$$

where $\bar{x}$ is the vector of binary decision variables, i.e. all the x_{ik}'s and x_{jl}'s. Q is a $n^2 \times n^2$ coefficient matrix that can be copied from the coefficients in (1.1). For example, if the coefficient $f_{ij}d_{kl} = c$ for $x_{ik}x_{jl}$, and x_{ik} and x_{jl} are at positions a and b respectively in $\bar{x}$, then the (a, b) entry of Q is c. Note that (b, a) of Q can also be c because expression (1.1) does not differentiate between variable orderings, thus Q can be expressed symmetrically or in upper triangular form depending on the situation. One can check that the vector product in (1.4) corresponds exactly with (1.1).

The next step is to input coefficients of (1.4) into QA or DA. Quantum annealers normally have an interface in a standard programming language. For example, D-Wave has a Python remote API. DA uses an in-house HTTPS WebAPI.

However, naively inputting coefficients does not yield a feasible solution because the problem in (1.4) is unconstrained. There needs to be a way to encode constraints (1.2) and (1.3), which by convention can be done via quadratic penalty. Quadratic penalty adds a non-negative term to (1.1) for each of the constraints, such that when a constraint is satisfied the term equals 0, and is positive otherwise. For example, if it is desired that $x_0 = 1$, then $P(x-1)^2$ is added, where P is some positive constant. Violating the constraint would imply increasing objective function value which is undesirable. In general, linear constraints in the form $Ax = b$ can be converted to

the usual inner product $(Ax-b)^T(Ax-b)$. Therefore, the final input to the quantum annealer enables solving for feasibility and optimality simultaneously, and it has been shown that optimal result of the modified objective function corresponds with that of the original function. The parts of this procedure relevant to this project is detailed in section 3.3. For a comprehensive generic tutorial on how to encode penalty, refer to (Glover, Kochenberger, & Du, 2019).

In this way, a full QUBO problem covering all constraints can be obtained. To further improve quality of solution, an exterior penalty algorithm can be employed which iteratively increases penalty weights starting from a random solution with zero weights. This procedure is used in our heuristic and is described in section 3.4.

1.3 Scalability Problem with Quantum Annealer

Quantum annealers cannot yet solve practical QAPs. Whereas QAPs in research literature are often small, ranging from several tens of items to at most over two hundred items, practical QAPs can be way larger. For instance, a warehouse can have hundreds of items, and the number of decision variables in a derived QUBO is easily in the range of 10^4.

Moreover, quantum annealers face a scalability issue. The number of binary decision variables in QAP is $O(n^2)$ with respect to the number of items. Meanwhile, as mentioned in section 1.2.2, *chimera* architecture limits the maximum number of nodes in a fully-connected problem graph to around 64. DA supports over 8000 variables with full connectivity. This means a QAP of maximum size of nearly 90 items is directly supported on DA. However, this is still too small compared to 10^4. It would be ideal if the maximum supported number of items in QAP is linear to the number of physical (qu)bits.

1.4 Contribution and Report Organization

This project attempts to overcome the aforementioned scalability issue in quantum(-inspired) annealers for certain special cases of QAPs. In particular, it deals with QAPs having a block-structure. In this block-structure, the set of locations can be partitioned such that distance

between locations in the same partition is relatively small, and distance between locations in different partitions is larger. A real-life instance of this special QAP is exactly warehouse optimisation, where locations are organised into columns which correspond with intervals. It is easy to travel within a column, but crossing over to another column will require traversing longer distances.

This project presents a novel divide-and-conquer heuristic that takes advantage of the said block-structure. Using the heuristic, the original QAP of n items can be meaningfully decomposed into sub-QAPs and each sub-QAP will have number of items that is $\sqrt{n}$ at best. This reduces problem size by a sqaure root and makes the resulting sub-QAPs amenable to solving on a quantum device. Solutions to the sub-QAPs can be pieced back together to form a solution for the original QAP.

The report is organised as follows. Chapter 2 discusses related work. Chapter 3 elaborates on the QAP decomposition heuristic, including a proof of optimality under certain simplifications. Chapter 4 outlines the implementation of the heuristic and reports computational results. Chapter 5 concludes and points to future work.

Chapter 2

Related Work

2.1 QUBO decomposition

While there is a paucity of research that deals with directly decomposing QAP, a few published techniques aim at decomposing QUBO. These techniques almost always revolve around the concept of conditioning, which means fixing a subset of variables (or, nomenclatively, "conditioning" them) so that the rest of the variables can be solved on hardware. The condition on the subset of variables is then lifted and the space of the subset explored as an outer loop in order to obtain an overall good solution.

In cut-set conditioning, a cut-set of the problem graph is taken out and the remaining disconnected subgraphs are solved separately. The separate solutions are then considered together with the conditioned cut-set to obtain a final solution. This would work for sparser graphs, but block-structural QAP instances in this paper are fully connected, which means the only cut-set is the whole set. This renders cut-set conditioning unusable.

In large neighbourhood local search, a random feasible solution is taken as the starting point and a small subset of variables solvable on hardware are chosen in each iteration. The subproblem generated by this subset is solved while variables outside of the subset are conditioned. This yields an exploration of a very large neighbourhood, whose end result can be optionally plugged into a complementary software local search that gets to a local minimum. Overall, this approach has two complementary steps: the perturbation/jump step to find a point with a good

neighbourhood to work with, and the local search step for finding local optimum within such a neighbourhood. Aside from *qbsolv*, the canonical D-wave implementation (Booth & Reinhardt, 2017) that uses subsets of variables for perturbation, there is another implementation by Oracle (Mihic, Ryan, & Wood, 2018) which inverts the ways by solving random subsets of variables for local search and updating randomly chosen variables for perturbation. The approach described in this paper falls into this category, but is a special case without perturbation. A neighbourhood is a-priori determined and subsets of variables are chosen heuristically on a higher abstraction level instead of at random.

2.2 Theories on specially structured QAP

An interesting line of research deals with specially structured QAPs. Those are QAPs with block structures and have been studied by Cela (Çela, Deineko, & Woeginger, 2014; Çela, Deineko, & Woeginger, 2015; Çela, Deineko, & Woeginger, 2018) and shown to be well-solvable when the frequency matrix is anti-Monge and when the distance matrix is a specially-defined block matrix. While the QAP instances in this paper have similarly structured distance matrices, numerical values in this paper are more complex, and property of the frequency matrix being anti-Monge is not satisfied. Nevertheless, inspiration can be drawn from this line of work that motivates further theoretical study of the QAP at hand.

2.3 Use of Quantum(-inspired) Devices in solving QAP

Much of the quantum annealer application work cited in chapter 1 is experimental in scale and does not deal directly with QAP, much less QAP of large scale. There is only one instance of QAP being solved on quantum device with application in flight gate assignment (Stollenwerk, Lobe, & Jung, 2019). Reminiscent of what is mentioned in chapter 1, the QAP instance in this work is too big for DA. The scaling method used in this paper is to randomly divide a graph into disconnected components if it is too large, and solve each component separately. This approach, while scalable, is a brute-force one based on randomness and does not give significant insight

as to how a QAP can be meaningfully decomposed.

Chapter 3

QAP Decomposition Heuristic

3.1 The Block Structure

Firstly, the formal description of QAP is copied below.

Given:

$F = (f_{ij})$ $n \times n$ interaction frequency matrix.

$D = (d_{ij})$ $n \times n$ distance matrix.

(x_{ij}) $n^2 \times n^2$ binary decision matrix.

Each entry x_{ij} represents whether indexed item i is assigned to indexed location j.

Minimise:

$$\sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n}\sum_{l=1}^{n} f_{ij}d_{kl}x_{ik}x_{jl}$$

Subject to:

$$\sum_{k=1}^{n} x_{ik} = 1 \qquad \forall 1 \leq i \leq n$$

$$\sum_{i=1}^{n} x_{ik} = 1 \qquad \forall 1 \leq k \leq n$$

Before describing the block-structure and the decomposition heuristic, it is necessary to introduce a graph formulation of QAP equivalent to the above.

In the graph formulations, think of items and locations as two graphs. The nodes of the graphs represent items and locations themselves, indexed the same way as they are in F and D. Formally,

$G = (V(G), E(G))$ $|V(G)| = n$ is the graph of items.

$H = (V(H), E(H))$ $|V(H)| = n$ is the graph of locations.

Define the edge weights of G to be interaction frequencies, and those of H to be distances. Formally,

$$f : E(G) \to \mathbb{R}$$

$$f(i, j) = f_{ij}$$

$$d : E(H) \to \mathbb{R}$$

$$d(i, j) = d_{ij}$$

An assignment is given by a bijection $\varphi : V(G) \to V(H)$. The goal is to solve the following minimisation:

$$\min_{\varphi} \sum_{(i,j) \in E(G)} f(i, j) d(\varphi(i), \varphi(j)) \tag{3.1}$$

One can check that 1.1 and 3.1 are equivalent.

The block structure of QAP occurs in the distance matrix (d_{ij}). There is a partition of $V(H)$ into K subsets and a function that maps each location to its respective subset: $c : V(H) \to K$, where $K = \{1, 2, ..., k\}$, such that

$$d(x, y) = \begin{cases} \delta(x, y) & c(x) = c(y) \\ M(x, y) & c(x) \neq c(y) \end{cases}$$

where δ and M are both functions. The concrete functions would depend on d, but with block structure δ would "usually" return smaller values than M. A special case is when δ and M are constants with $\delta < M$, and (d_{ij}) will be a "block matrix".

Consider an example with the following warehouse of 8 locations, with each location labeled as in Fig 3.1. Assume that the vertical distance is 1 and horizontal distance is 3 as in the example in section 1.1.1.

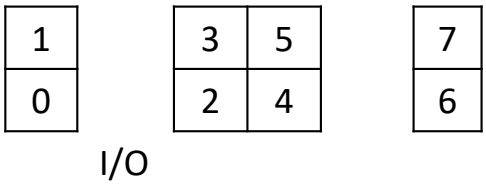


Figure 3.1: layout of an empty warehouse

The distance matrix looks like the following (only upper triangular part is shown):

$$\begin{bmatrix} 1 & 1 & 0 & 1 & 7 & 6 & 7 & 6 \\ 0 & 2 & 1 & 0 & 6 & 5 & 6 & 5 \\ 0 & 0 & 1 & 1 & 7 & 6 & 7 & 6 \\ 0 & 0 & 0 & 2 & 6 & 5 & 6 & 5 \\ 0 & 0 & 0 & 0 & 7 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 8 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 7 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 8 \end{bmatrix}$$

Noticeably, the 4×4 sub-matrix in the upper right corner, corresponding to the distances in between the 1st and the 2nd aisles, is uniformly larger in number than the sub-matrices near the diagonal. In general, we could relabel the indices of locations such that the columns of a warehouse form intervals, and that when we move away from the diagonal, the matrix entries become larger because the columns are further away.

Since the above block-structure is observed, one simplification is that all locations within the same column have distance 1 (i.e. $\delta \equiv 1$) in between, and all locations in different columns have a larger distance, say 8 (i.e. $M \equiv 8$). Then the block-structure becomes more pronounced:

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 8 & 8 & 8 & 8 \\ 0 & 2 & 1 & 1 & 8 & 8 & 8 & 8 \\ 0 & 0 & 1 & 1 & 8 & 8 & 8 & 8 \\ 0 & 0 & 0 & 2 & 8 & 8 & 8 & 8 \\ 0 & 0 & 0 & 0 & 7 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 8 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 7 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 8 \end{bmatrix}$$

In the next section, a QAP decomposition is outlined. It is proven that under the above simplification, the decomposition leads to an optimal solution.

3.2 Decomposition

Similar to the way we perform partitioning on locations, the item set $V(G)$ can also be partitioned into K subsets of equal size, where K is fixed to be $|K|$. Intuitively, we want items that appear together in high frequency to be assigned closer to each other. Therefore, we maximise the sum of interaction frequency within the subsets, and then assigning each subset to a subset in the partition of $V(H)$, the set of locations. Note that K divides n is assumed, so that locations and items can be divided evenly. Formally,

Given: (x_{ij}) $\quad n \times k$ decision matrix

x_{ij} denotes whether item i goes to subset j.

Maximise:

$$\sum_{i,j=1}^{n} \sum_{l=1}^{k} f_{ij} x_{il} x_{jl} \tag{3.2}$$

which denotes the sum of interaction frequencies within the subsets. Note that each subset can have at most $s = \frac{n}{k}$ items, and thus the following constraints:

$$\sum_{l=1}^{k} x_{il} = 1 \quad \forall i, 1 \leq i \leq n \tag{3.3}$$

$$\sum_{i=1}^{n} x_{il} \leq s \quad \forall l, 1 \leq l \leq k \tag{3.4}$$

(3.3) means an item can belong to exactly one subset.(3.4) means each subset must not exceed its capacity s.

Note that (3.2) is in fact a "graph partitioning" problem that tries to partition G while maximising sum of edge weights within partitions. This can be translated into a graph formulation. In the graph formulation, (x_{ij}) is equivalent to a function $g : V(G) \to K$, which maps an element of $V(G)$ into its subset, such that

$$g(i) = a \iff x_{ia} = 1$$

This function is well-defined due to constraint 3.3. Expression (3.2) can be converted to the

equivalent graph formulation as follows:

$$\begin{aligned}\sum_{i,j=1}^{n}\sum_{l=1}^{k} f_{ij}x_{il}x_{jl} &= \sum_{l=1}^{k}\sum_{\substack{i,j=1\\ x_{il}=x_{jl}=1}}^{n} f_{ij} \\ &= \sum_{l=1}^{k}\sum_{\substack{i,j=1\\ g(i)=g(j)=l}}^{n} f_{ij} \\ &= \sum_{\substack{(i,j)\in E(G)\\ g(i)=g(j)}} f(i,j)\end{aligned}$$

which, intuitively, is the sum of flows within subsets. This is the setup for the theorem below, which states that with a solution to (3.2), an optimal solution to the overall QAP (3.1) can be constructed.

Theorem 1. *Let* $d(x,y) = \begin{cases}\delta & c(x)=c(y)\\ M & c(x)\neq c(y)\end{cases}$ *for some positive* ***constants*** δ, M*. Then there exists a* φ_0 *for which* (3.1) *is achieved, and* φ_0 *can be constructed from a solution to* (3.2) *denoted by* g_0*.*

Proof. Suppose $\varphi : V(G) \to V(H)$ is some assignment. Then expression (3.1) can be written as

$$\begin{aligned}&ll \sum_{(i,j)\in E(G)} f(i,j)d(\varphi(i),\varphi(j)) \\ =& \sum_{\substack{(i,j)\in E(G)\\ c(\varphi(i))=c(\varphi(j))}} f(i,j)d(\varphi(i),\varphi(j)) \\ +& \sum_{\substack{(i,j)\in E(G)\\ c(\varphi(i))\neq c(\varphi(j))}} f(i,j)d(\varphi(i),\varphi(j)) \\ &= \delta\left(\sum_{\substack{(i,j)\in E(G)\\ c(\varphi(i))=c(\varphi(j))}} f(i,j)\right) \\ &+M\left(\sum_{\substack{(i,j)\in E(G)\\ c(\varphi(i))\neq c(\varphi(j))}} f(i,j)\right) \\ &= \delta\mathcal{A} + M(\mathcal{F}-\mathcal{A}) \\ &= M\mathcal{F} + (\delta - M)\mathcal{A}\end{aligned}$$

where $\mathcal{F} = \sum_{(i,j)\in E(G)} f(i,j)$, which is the sum of all edge weights in graph G, and $\mathcal{A} = \sum_{\substack{(i,j)\in E(G) \\ c(\varphi(i))=c(\varphi(j))}} f(i,j)$. Since $\delta - M < 0$, the original expression is minimised iff $\mathcal{A}$ is maximised.

Note that $\varphi : V(G) \to V(H)$ and $c : V(H) \to K$, therefore we can view $c \circ \varphi : V(G) \to K$ and $\mathcal{A} = \sum_{\substack{(i,j)\in E(G) \\ c\circ\varphi(i))=c\circ\varphi(j))}} f(i,j)$. But then $g_0 : V(G) \to K$ is by definition the solution to

$$\min_{g} \sum_{\substack{(i,j)\in E(G) \\ g(i)=g(j)}} f(i,j)$$

which means

$$\sum_{\substack{(i,j)\in E(G) \\ g_0(i)=g_0(j)}} f(i,j) \geq \mathcal{A}, \quad \forall\varphi$$

.

Now we need to construct a φ such that $c(\varphi(i)) = c(\varphi(j)) \iff g_0(i) = g_0(j)$. This is equivalent to saying that after splitting items into subsets according to g_0, assign the items such that items within the same subset are in the same interval. The relative positions of items within an interval does not matter. Therefore, there are many possible answers. □

The above method assumes a strong simplification that (d_{ij}) only has two distinct values, δ for locations within a column and M in between columns. In reality, (d_{ij}) is usually more complex. For example, in the case of warehouses, a warehouse might not even have a rectangular shape, or its columns might have a more complicated layout. In such cases, it is difficult to partition the locations by hand.

However, not all is lost. The concept of columns can be generalised. Think of columns as clusters of locations that have short distances amongst themselves. This way, the formulation of columns does not depend on a specific warehouse shape. In fact, it no longer depends on anything other than the QAP distance matrix D. When δ and M are non-constant, a function c can still be defined on $V(H)$ such that the sum of distances within subsets is minimised. Note that this definition automatically specialises to the formulation of columns when the QAP at hand is about warehouses, and the definition of function c in section 3.1 applies. Formally,

Given: (x_{ij}) $\quad n \times k$ decision matrix

x_{ij} denotes whether item i goes to location group j.

Minimise:

$$\sum_{i,j=1}^{n} \sum_{l=1}^{k} d_{ij} x_{il} x_{jl} \tag{3.5}$$

Subject to:

$$\sum_{l=1}^{k} x_{il} = 1 \quad \forall i, 1 \leq i \leq n \tag{3.6}$$

$$\sum_{i=1}^{n} x_{il} \leq s \quad \forall l, 1 \leq l \leq k \tag{3.7}$$

Note that (3.5) is dual to (3.2), with maximisation changed to minimisation. The result is that items in a subset will be assigned to locations in a corresponding subset. The final step is to produce such a correspondence the subsets of items and the subsets of locations, and for each pair of subsets $(items, locs)$ there is a sub-QAP of size $O(\sqrt{n})$, and therefore a QUBO of size $O(n)$, where n is the total number of items. The first intuition would be to sort the interaction frequencies of subsets of items from low to high, and assign them to subsets of locations in reverse order, such that the best column gets the subset with lowest interaction frequency and so on. However, there is no guarantee that this produces the best combination, so in this project they are matched randomly.

Note that this is subject to n being a perfect square; in practical situations where n is not a perfect square, compromise has to be made in either finding the nearest smaller perfect square and do the optimisation on the smaller set, or use integer divisors of n other than $\sqrt{n}$, and correspondingly deal with sub-QAPs of sizes other than $\sqrt{n}$. For example, if $n = 3600$, it is possible to divide it into 40 groups of 90, where each sub-QAP will have size 90.

3.3 QAP-to-QUBO conversion

It is mentioned in section 2 that any QAP needs to be converted to QUBO before it is solvable by a quantum annealer. This is also true of each of the sub-QAPs generated by the decomposition procedure described in section A. Here the procedures of QAP-to-QUBO conversion is described. Readers familiar with square penalty may safely skip this section.

For the linear constraint

$$\sum_{k=1}^{n} x_{ik} = 1 \qquad \forall 1 \leq i \leq n$$

there is one constraint for each i. Thus, for each i the quadratic penalty term $P(\sum_{k=1}^{n} x_{ik} - 1)^2$ is added to the original objective function, where P is some positive constant to be determined a-posteriori. Note that this penalty term, when expanded, is also a quadratic form, and thus can be represented as an addition to the QUBO matrix Q.

We can express this constraint in a more general form. For the decision vector $\bar{x}$, locate the position of x_{ik} and instantiate a 0-1 bit-vector with length n, with the corresponding positions having 1 and the other positions having 0. For all n constraints, collect the respective bit-vectors as rows of a $n^2 \times n^2$ matrix A (note that not all rows of A has to be used as a constraint, in which case the row is left as zero). All constraints are therefore in the form $A\bar{x} = \bar{b}$, where $\bar{b}$ is just a length n^2 vector of all 0's except $n + n$ 1's (corresponding to the $n + n$ constraints we have) in the corresponding rows of A. To convert the constraints into a sum of squares we take the inner product

$$(Ax - b)^T (Ax - b)$$

We can embed the penalty P easily into A and b by scaling. Subsequently, the expanded QUBO coefficient matrix can be obtained by matrix manipulation. It could be shown that the final coefficient matrix to be augmented to Q is $Q' = (A^T A - 2D)$, where $D = diag(b^T A)$. The complete algorithm is described below.

Algorithm 1 Conversion from QAP to QUBO

//Given: $FindIndex(i, j)$ that maps quadratic index of x_{ij} to its corresponding position in the decision vector $\bar{x}$

procedure CONVERT(F,D,P)
  $Q = ZeroMatrix(n^2 \times n^2)$
  for i in $[1..n]$ **do**
    for j in $[1..n]$ **do**
      for k in $[1..n]$ **do**
        for l in $[1..n]$ **do**
          $x_{ik} = FindIndex(i, k)$
          $x_{jl} = FindIndex(j, l)$
          $Q[x_{ik}][x_{jl}] = F[i][j] \times D[k][l]$
        end for
      end for
    end for
  end for
  $Q' = AtoQ(preparematrixA(), preparevectorB(), P)$
  return $Q + Q'$
end procedure

procedure PREPAREMATRIXA
  $A = ZeroMatrix(n^2 \times n^2)$
  $i = 1$
  for r in $[1..n]$ **do**
    for k in $1..n$ **do**
      $idx = FindIndex(i, k)$
      $A[r][idx] = 1$
    end for
    $i = i + 1$
  end for
  $k = 1$
  for r in $[(n + 1)..2n]$ **do**
    for i in $1..n$ **do**
      $idx = FindIndex(i, k)$
      $A[r][idx] = 1$
    end for
    $k = k + 1$
  end for
  return A
end procedure

procedure PREPAREVECTORB
  $b = ZeroVector(n^2)$
  for r in $[1..2n]$ **do**
    $b[r] = 1$
  end for
  return b
end procedure

44: **procedure** AtoQ(A,b,P)
45: Scale each row of A by $\sqrt{P}$
46: Scale each entry of b by $\sqrt{P}$
47: $D = b^T A$
48: return $A^T A - 2D$
49: **end procedure**

3.4 Exterior penalty method

We use exterior penalty method to iteratively solve a QUBO as a series of QUBO's with increasing penalty weights. This is called the exterior penalty method, whose algorithm is described below.

The algorithm starts with a random permutation matrix as the initial solution. Note that the procedure is equipped with a $isValid()$ function that tests if a solution satisfies constraints (1.2) and (1.3).

Algorithm 2 Exterior Penalty Method

1: **procedure** solve(F,D,P_0,α)
2: $P = P_0$
3: $solution = randomPermutationMatrix(n \times n)$
4: **repeat**
5: $solution = annealer.solve(convert(F, D, P), solution)$
6: $P = P \times \alpha$
7: **until** $isValid(solution)$
8: **end procedure**

3.5 Overall procedure

The overall procedure first runs the decomposition in section 3.2, and then solves each individual sub-QAP using exterior penalty method of section 3.4. Each sub-QAP utilises the conversion procedure in 3.3 to transform it into QUBO solvable by quantum annealer. The overall procedure could be abstractly outlined as Algorithm 3.

Algorithm 3 Solving QAP by decomposition

1: **procedure** RUN(F,D,n)
▷ n-#items/locations
▷ k-#groups
2: $s = \sqrt{n}$ ▷ assume group size is $\sqrt{n}$
3: $Fs, Ds, FtoD = decompose(F, D, s)$ ▷ FtoD is the map between subsets and intervals
4: $P_0 = 10000$
5: $\alpha = 1.5$
6: $subsolutions = []$
7: **for** i in $[1..k]$ **do**
8: $items = F[i]$
9: $locations = D[FtoD[i]]$
10: $F' = emptyMatrix(s \times s)$
11: $D' = emptyMatrix(s \times s)$
12: **for** p, q in $items$ **do**
13: $F'[localindex(p)][localindex(q)] = F[p][q]$
14: **end for**
15: **for** k, l in $locations$ **do**
16: $D'[localindex(k)][localindex(l)] = D[k][l]$
17: **end for**
18: $subsolutions.append(solve(F', D', P_0, \alpha))$
19: **end for**
20: $combineSubsolutions(subsolutions)$
21: **end procedure**

Note that P_0 and α are adjustable parameters. Instead of statically determined, they are often estimated and adjusted dynamically.

Chapter 4

Implementation

4.1 Objectives

The experiment has three objectives, namely 1) to evaluate the performance of QAP decomposition heuristic (on hardware) in solving block-structural QAPs of various sizes, 2) to evaluate the effectiveness of the heuristic in minimising warehouse picking distance.

4.2 System setup

As the heuristic in chapter 3 contains hybrid tasks, the hardware setup is hybrid. There is a on-campus remote server carrying a general-purpose Xeon Silver 4116 at 2.1GHz with 512GB DDR4 memory and 960GB SSD storage. Both the D-Wave quantum annealer and the Fujitsu Digital Annealer are hosted remotely. Data has to be sent back and forth over the network in order to carry out annealing computations.

Only *annealer.solve*() in Algorithm 3 is run on annealing hardware. The rest of the computation is done entirely on CPU. Note that *decompose*(), the QAP decomposition process as outlined in section 3.2, is also done on CPU.

CPU computations are implemented entirely in Python.

4.3 The Dataset

Block-structural QAP instances of size 8, 90, 180, 270, 3600 and 8100 are synthesised using randomly generated order sets, using the procedure described in section 1.1.2. In addition, for objective 2), input data distribution is perturbed to given modified versions of datasets of size 270 and 800. This is to match the findings in (Tsige, 2013) that interaction-based methods such as QAP work well when 80% of ordered items are concentrated in 20% of the SKUs. In other words, there is a small set of commonly ordered products.

The following assumptions are made about the order sets as well as the shape of the respective warehouses. xu270b below is for objective 2), whereas others are for objective 1).

name	problem size	#items/#locations	#subsets	warehouse #rows	warehouse #columns
xu8	8	8	1	4	2
xu90	90	90	1	45	2
xu180	180	180	2	45	4
xu270	270	270	3	45	6
xu270b	270	270	3	45	6
xu3600	3600	3600	40	45	80
xu8100	8100	8100	90	45	180

As DA supports problems of up to size 90, most datasets are divided into groups of 90 for solving. For example, the order set of size 3600 can be solved as 40 sub-QAPs, each of size 90. All except size 8 warehouses are assumed to comprise columns of size 45, and there is an aisle running in between columns. The respect layout is similar to that in Fig 1.1.

4.4 Experiments and Results

At the current stage, two experiments have been conducted. The first experiment aims for objective 1), and the second experiment targets objective 2).

4.4.1 Experiment on QAP

For objective 1), the heuristic is applied to the datset as described in 4.3. At the same time, an open-source simulated annealing software library is directly applied to the dataset without decomposition heuristic. This serves as standard reference for comparison. Solution quality (in terms of the value of objective function (1.1)) and time to solution(s) are measured and compared.

In applying the heuristic to the dataset, D-Wave quantum annealer can only handle the problem of size 8 due to hardware restrictions. Therefore, D-Wave data are only available in size 8, whereas DA has been used for all the instances.

To ensure a fair comparison between the heuristic and the software library, critical parameters are set as identical. In particular, both are set to run for 10,000,000 iterations, which is the determining factor of the speed of any simulated annealing algorithm. Note that in the heuristic, our hardware has to perform 10,000,000 iteration for *each* of the sub-QAPs. Other annealing parameters such as maximum temperature and temperature interval are left to the default settings.

The results obtained for each of the datasets are average values over a certain number of runs. The software library runs 3 times for each dataset. The number of runs for the decomposition heuristic varies across datasets due to overhead. For xu8, xu90, xu180, xu270 the heuristic is run 3 times, so the readings are averages of 3. For xu3600 and xu8100 the heuristic is run 1 time.

name	#sub-QAP:k	hardware total annealing time:t1(s)	hardware annealing time per sub-QAP:t1/k(s)	software total annealing time:t3(s)	speedup:t2/t1
xu8(D-Wave)	1	0.016	0.016	683	42687.5X
xu8(DA)	1	3.70	3.70	683	184.6X
xu90	1	23	23	556	24X
xu180	2	60	30	716	11.9X
xu270	3	68	34	594	9X
xu3600	40	983	24	3447	3.5X
xu8100	90	2080	23	6884	3.21X

Table 4.1: Annealing time comparison

name	hardware result (A)	software result (B)	random	%worse $\frac{A-B}{B}$
xu8(D-Wave)	119	81	-	47%
xu8(DA)	81	81	-	0%
xu90	737625	643051	757057	14.7%
xu180	2205183	1956870	2578729	12.7%
xu270	15907384	14418020	17539173	10.3%
xu3600	12947375054	10604494000	13117154976	22.1%
xu8100	114727417613	91677402330	115321812305	19.3%

Table 4.2: QAP solution quality comparison

From Table 4.1, it can be observed immediately that either D-Wave or DA is in general faster than software annealing. D-Wave outperforms software by a considerable amount, while speedup of DA is also significant. However, as problem size increases DA's speed advantage is attenuated.

As for solution quality, the software library serves as a golden standard because it is an established technique and is used directly on the original QAP without decomposition. As evidenced by Table 4.2, solution quality is expectedly worse in heuristic on hardware as compared to direct solving in software. For D-Wave, a noticeable gap of 47% is observed. DA performs as well as software in small problems such as xu8, but when it reaches full capacity as in xu90 there is a gap of slightly higher than 10%. Beyond the size of 90, the heuristic starts to take effect and there is a noticeable worsening of an additional 10% on solution quality for large instances of 3600 and 8100. However, for moderate sizes of 180 and 270, the gap is around 10%.

The heuristic fails directly at size 90, 3600 and 8100 because a sanity check against random permutations does not show more than 5% difference. It seems that a random solution would fall at somewhat 20% higher than software annealed solution. For sizes 180 and 270, the heuristic performs better than random.

4.4.2 Experiment on Warehouse

xu270b is used in this experiment to test for the effectiveness of decomposition heuristic in reducing warehouse travel distance. A simulation framework is built to calculate the distance travelled given an order and an assignment. Various assignment strategies are run as references, and solution quality is compared across the board. Specifically, COI, ABC, and OOS in section 1.1.1 are implemented and compared against the heuristic. The heuristic is implemented on DA, which solves a block-structured QAP generated from xu270b.

Except for DA, which are run 2 times, the other methods are run multiple times and 5 readings are chosen and averaged.

name	ABC	COI	OOS	Random	DA
xu270b	3246	4524	3084	3510	2868
	3318			3864	3048
	3786			3324	3156
	3606			3888	3282
	3330			3510	3138
Avg	3457.2	4524	3084	3619.2	3098.4

Table 4.3: Warehouse solution quality comparison

From the above table, travel distance for the xu270b is second shortest for OOS. This is followed by algorithmic DA, and then ABC, random and COI.

4.5 Discussion

4.5.1 Objective 1: Heuristic for QAP

The relevant experiment for this objective is in section 4.4.1.

Both DA and D-Wave have merits of being much faster than software annealing, but only within their respective capacities. Observably, whilst our heuristic enables scaling to $O(n)$ variables, speedup does not scale accordingly. The speedup provided by DA attenuates as problem size increases, presumably due to the increasing line of sub-QAPs generated by the heuristic, which must be solved sequentially. Expectedly, as problem size continues to increase, DA will eventually take even longer than software. Nonetheless, DA retains an approximately 3X speedup for size 8100.

Solution quality is not reliable for either DA or D-Wave. For D-Wave, after a correspondence with D-Wave engineer, there is the understanding that such unreliability is perhaps due to the nature of QAP, where the entries of the converted QUBO has a large range that exceeds the granularity of the numerical range of D-Wave machine. Note that D-Wave is an analog solver and cannot guarantee good solutions for arbitrary QAPs.

DA has a similar issue with regards to solution quality. At the limit of its capacity which is 90, there is a deviation of about 14.7%. A correspondence with Fujitsu reveals a few potential causes: 1) parameter tuning 2) lack of preprocessing 3) numerical values of QAP. To tackle 1) is a matter of advanced trial and error beyond the basic technique in section 3.4 and will require additional expertise perhaps in machine learning. 2) is confidential to Fujitsu, so details cannot be revealed here, but additional trials of preprocessing does not seem to improve results in the author's case. Again, according to Fujitsu, preprocessing is necessary but not sufficient to get good solutions to QAP. 3) reveals the ad-hoc nature of DA. Even though there are a few benchmarks showing good results for DA for some standard QAPLIB (Burkard, Karisch, & Rendl, 1997) instances, the overall DA quality seems to be subject to careful paramter tuning and numerical trials except for small problems. Such has been the general impression of the author as a user of DA.

Due to the above issues of DA, it cannot be concluded that the heuristic solves block-structure QAP with good quality, even though the quality seems to improve a little with the heuristic for moderate sizes of 180 and 270.

There are many reasons for which the heuristic does not perform well for larger instances. Firstly, DA could have performed poorly for the individual sub-QAPs, as discussed above. Secondly, the heuristic may be inaccurate as size increases. It is possible that when too many columns of 90 are stacked together, the effect of assigning item pairs in one column is superseded by the vast shape of the warehouse, such that even items from different columns, which have relatively low interaction frequency, contribute significantly to the overall *traffic* because distance becomes large. Therefore, it is not advisable to split items and locations into too many subsets.

4.5.2 Objective 2: Heuristic on Warehouse assignment

The relevant experiment for this objective is in section 4.4.2.

Firstly, the result is initial and there is a paucity of datasets. More datasets will need to be added later.

An important assumption in the comparison of Table 4.3 is that QAP is a good way of modelling warehouse assignment. That is, if the QAP corresponding to a particular warehouse has a good solution, then that solution will also be good to reduce travel distance for the warehouse. That is something which needs to be verified in future work.

With that assumption in mind, it is observed that OOS has shorter picking distance than DA. A straightforward answer to this is that OOS is manually tuned and optimised for xu270b, whereas QAP and its heuristic on DA is completely generic, modulo the initial estimation of penalty weights. A better solution to the QAP on DA will then yield a better solution for the warehouse assignment problem, which brings the discussion back to section 4.5.1.

Another observation is that DA's heuristic solution to the QAP has some fluctuation in its values. One may then question the reproducibility of the results in Table 4.3, in particular, whether it will still be good over some additional number of runs, or over larger datasets. This is acknowledged and a fuller experiments will need to be conducted to verify this point. Note that ABC assignment strategy has inherent randomness and the deviation across runs is normal, as is random assignment.

Overall, due to the above points of critique, it cannot be said conclusively that the decomposition heuristic as applied to DA solves the warehouse assignment problem with good quality, although initial results show promise.

Chapter 5

Conclusion

New quantum and quantum-inspired annealers surely display promise in solving QAP. In turn, such solution will have a wide range of applications, including scalable warehouse assignment as presented in this report. The caveat is that the current generation of hardware is not mature enough to reliably give generic QAP solutions of high quality, and the hope is that better results can be obtained as technology matures.

5.1 Future Work

For future work, there are a number of possibilities. Advancement in any of these could be of great help to the current project.

5.1.1 QAP-to-QUBO Conversion

On the theoretical side, the crux of the issue now is the conversion of QAP to QUBO. A good conversion should have the characteristic that solution to the QUBO has matching quality with solutions to the original QAP. This may manifest in the estimation of penalty weights, or alternative penalty functions. There is a related current research topic which explores how new penalty functions can be more suitable for quantum hardware, with literature such as (Vyskocil & Djidjev, 2018)(Vyskocil & Djidjev, 2019). In this project, the most basic methods are used, but it will be valuable to include a variety of other methods to potentially improve

the efficacy of DA and D-Wave.

5.1.2 Partitioning and subset matching

In section 3.2, the heuristic partitions sets of items and locations according to an aggregate interaction-frequency measure and distance measure. It turns out that this is a special case of the more generalised randomised decomposition technique in (Mihic et al., 2018). In the paper the items and locations are randomly taken to form sub-QAPs and impressive results are reported. It is therefore valuable to change the decomposition technique to random instead of using interaction frequency measure. The author has run some speculative experiments on this random method but the observation is that only doing random subsets is much worse than using the aggregate measures. Therefore, one of the possibilities is to add a random layer on top of the current decomposition layer. This could be easily done with a random number generator.

In a similar spirit, the subsets are matched randomly in this project. Instead, there could be other organised ways as mentioned in section 3.2. When the number of bijections is small, the matchings can be considered exhaustively to determine which one is the best. For example, for xu180, there are only 2 item and location subsets. There are only 2 ways to match the subsets. However, a more interesting but much more difficult question is how a matching would be good a-priori.

Of course, the overall decomposition heuristic is only proven to sustain optimality for a very restricted simplification as in section 3.2. It would be certainly encouraging if further advancements can be made in this direction, expanding the solution quality guarantee to a larger class of block-structural QAPs. However, it is so far unclear how that could be done because that would involve analysing the distance matrix case-by-case.

5.1.3 Model for warehouse assignment

The current QAP model for warehouse assignment is the primitive one from (Mantel et al., 2007).

It is certainly a necessity to plug the holes in assumptions as mentioned in section 4.5.2.

Namely, the standard QAPs need to be solved with standard tools, which give a reliable solution quality, with solutions simulated a-posteriori in order to verify its effectiveness in modelling warehouse assignment.

Other than the aforementioned verification exercise, the problem of modelling warehouse assignment is more subtle. There exists more measures of interaction-frequency other than the basic one, each of different numerical ranges and purported effectiveness, as reviewed in (Kofler, 2015). This is another aspect that carries potential for improvement.

5.1.4 Use of hardware platforms

Better ways to use hardware platforms such as DA or D-Wave can be acquired through more experience. A number of inherent problems in the hardware are already highlighted in section 4.5.1. Understanding these problems in greater detail might lead to a better chance at getting high quality solutions for generic QAPs. In particular, parameter tuning is a task of finding magic numbers; preprocessing is not an open technique but seems to be important for DA; numerical ranges of QAP might be scaled to be better suited to D-Wave, although this would be a-priori challenging unless with the help of advancements in aforementioned interaction-frequency measure in 5.1.3.

References

Booth, M., & Reinhardt, S. P. (2017). Partitioning optimization problems for hybrid classical / quantum execution technical report, , 2017.

Burkard, R. E., Karisch, S. E., & Rendl, F. (1997). Qaplib - a quadratic assignment problem library. *Journal of Global Optimization*, *10*(4), 1997, 391–403.

Çela, E., Deineko, V. G., & Woeginger, G. J. (2015). A new tractable case of the qap with a robinson matrix. In Z. Lu, D. Kim, W. Wu, W. Li, & D.-Z. Du (Eds.), *Combinatorial Optimization and Applications* (pp. 709–720), Cham, , 2015: Springer International Publishing.

Chang, T. H., Lux, T. C. H., & Tipirneni, S. S. (2019). Least-squares solutions to polynomial systems of equations with quantum annealing. *Quantum Information Processing*, *18*(12), , 2019, 374.

Cruz-Santos, W., Venegas-Andraca, S. E., & Lanzagorta, M. (2019). A qubo formulation of minimum multicut problem instances in trees for d-wave quantum annealers. *Scientific Reports*, *9*(1), , 2019, 17216.

D-Wave (2019). D-wave system documentation. , 2019. Accessed: 2020-04-06.

de Ruijter, H., & Schuur, P. C. (2007). Improved storage in a book warehouse. *NA*, The Netherlands, October, 2007.

Glover, F., Kochenberger, G., & Du, Y. (2019). Quantum bridge analytics i: a tutorial on formulating and using qubo models. *4OR*, *17*(4), 1, 2019, 335–371.

Kofler, M. (2015). *Optimising the storage location assignment problem under dynamic conditions*. PhD thesis.

Mantel, R., Schuur, P., & Heragu, S. (2007). Order oriented slotting: A new assignment strategy for warehouses. *European Journal of Industrial Engineering*, *1*, 02, 2007, 301–316.

Mihic, K., Ryan, K., & Wood, A. (2018). Randomized decomposition solver with the quadratic assignment problem as a case study. *INFORMS Journal on Computing*, *30*(2), , 2018, 295–308.

Naghsh, Z., Javad-Kalbasi, M., & Valaee, S. (2019). Digitally annealed solution for the maximum clique problem with critical application in cellular v2x. Vol. 2019-May, , 2019.

Neukart, F., Compostella, G., Seidel, C., von Dollen, D., Yarkoni, S., & Parney, B. (2017). Traffic flow optimization using a quantum annealer. *Frontiers in ICT*, *4*(DEC), , 2017.

Rosenberg, G., Haghnegahdar, P., Goddard, P., Carr, P., Wu, K., & De Prado, M. (2016). Solving the optimal trading trajectory problem using a quantum annealer. *IEEE Journal on Selected Topics in Signal Processing*, *10*(6), , 2016, 1053–1060.

Stollenwerk, T., Lobe, E., & Jung, M. (2019). Flight gate assignment with a quantum annealer. In S. Feld, & C. Linnhoff-Popien (Eds.), *Quantum Technology and Optimization Problems* (pp. 99–110), Cham, , 2019: Springer International Publishing.

Terry, J. P., Akrobotu, P. D., Negre, C. F. A., & Mniszewski, S. M. (2020). Quantum isomer search. *PLOS ONE*, *15*(1), 01, 2020, 1–21.

Tsige, M. T. (2013). Improving order-picking efficiency via storage assignments strategies. *NA*, The Netherlands, February, 2013.

Tsukamoto, S., Takatsu, M., Matsubara, S., & Tamura, H. (2017). An accelerator architecture for combinatorial optimization problems. *Fujitsu Scientific and Technical Journal*, *53*, 09, 2017, 8–13.

Vyskocil, T., & Djidjev, H. (2018). Simple constraint embedding for quantum annealers. *2018 IEEE International Conference on Rebooting Computing (ICRC)* (pp. 1–11), , 2018.

Vyskocil, T., & Djidjev, H. (2019). Embedding equality constraints of optimization problems into a quantum annealer. *Algorithms*, *12*(4), Apr, 2019, 77.

Çela, E., Deineko, V., & Woeginger, G. J. (2018). New special cases of the quadratic assignment problem with diagonally structured coefficient matrices. *European Journal of Operational Research*, *267*(3), , 2018, 818 – 834.

Çela, E., Deineko, V. G., & Woeginger, G. J. (2014). Well-solvable cases of the qap with block-structured matrices. *Discret. Appl. Math.*, *186*, , 2014, 56–65.

Appendix A

Simulated Annealing

There are many ways to solve combinatorial optimisation problems. Exact solution methods such as branch-and-bound exist, but normally take too long to be useful. Practically, heuristics are employed to discover suboptimal but decent solutions. Normally, a heuristic is only developed for a specific problem instance and cannot be used across the board. However, a special kind of heuristics, the metaheuristics, arise in recent years and shed light on the generic way the solution space of any combinatorial optimisation problem is searched. One of them is Simulated Annealing (SA), which imitates the physical process of how a piece of hot metal anneals, i.e. cools down naturally to minimal energy. In the optimisation problem, the energy of metal corresponds with the formula that is to be minimised. It is postulated that after several iterations, the solution yielded by SA would be of decent quality.

The high level idea of SA is that at each temperature, several randomly generated, neighbouring new *states* are explored. If the new *state* is better (with lower energy), it is accepted and replaces the current one. Otherwise, the new state is accepted with a probability given by the Metropolis criterion:

$$p = exp^{\frac{-\Delta E}{T}}$$

where ΔE is the change in energy and T is the current temperature. This occasional acceptance of a worse state (with higher energy) is crucial because the overall state would then avoid being trapped in a local minimum. After a number of trials, the temperature is lowered by multiplying

with a cooling factor α. The entire procedure ends when some target temperature is reached. The pseudocode of SA is shown below.

Algorithm 4 Near optimal solution

```
1: param TARGET_T{target temperature}
2: param NT{number of attempts within a single temperature}
3: param α {cooling factor}
4: t = t_0{starting temperature}
5: p = p_0{starting state}
6: e = E(p_0){starting energy}
7: p_best = p
8: e_best = e
9: k = 0
10: while t > TARGET_T do
11:     for n from 1 to NT do
12:         p_new = random(p)
13:         e_new = E(p_new)
14:         ΔE = e_new − e
15:         prob = exp^(−ΔE/t)
16:     end for
17:     if ΔE is less than zero then
18:         p = p_new
19:         e = e_new
20:     else if prob > random(0, 1) then
21:         p = p_new
22:         e = e_new
23:     else if e < e_best then
24:         p_best = p
25:         e_best = e
26:     end if
27:     t = t × α
28: end while
29: return(p_best, e_best)
```

One may wonder how the parameters are obtained for a specific problem. In fact, the best parameters are determined empirically. The set of parameters is called a *cooling schedule*. There exist cooling schedule proposals that seem to work well for combinatorial optimisation problems.