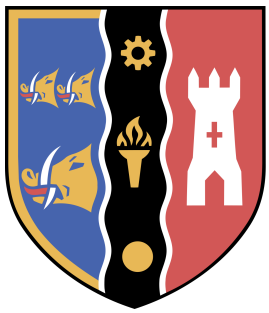
Estimating Penalty Coefficients For Quadratic Unconstrained Binary Optimisation Problems

Raufs Dunamalijevs



BSc (Hons) in Computer Science
School of Computing
Robert Gordon University
Aberdeen, Scotland

May 2022

Supervised by Professor John McCall

Abstract

The Quadratic Unconstrained Binary Optimisation (QUBO) is a model that can represent many Combinatorial Optimisation problems. Problems in such formulation consist of two functions: objective and constraint. The constraint function worsens the solution energy if it is not feasible. A penalty coefficient scalar controls the degree to which the constraint function will affect the overall energy of the solution. Our work focuses on a state-of-the-art approach that tends to overestimate the penalty coefficient. We attempt to design algorithms that would produce more optimal coefficients and experimentally demonstrate if such improvements would increase the speed of finding feasible solutions.

Acknowledgements

Although all the work that follows has been solely done by myself, I owe it to the great people surrounding me. If I were to list all of them, the focus of this dissertation would quickly shift into social sciences. That is why I will briefly thank only a few: my family, which has greatly supported me ever since I started university, Professor John McCall, who has shared his optimistic and beautiful view on science and guided me through the vastness of this project, and Dr Ayodele, whose knowledge of the industry proved to be indispensable. I also express my gratitude to Robert Gordon University for developing my critical thinking - an irreplaceable tool that will help me throughout my professional life.

Declaration

I confirm that the work contained in this BSc project report has been composed solely by myself and has not been accepted in any previous application for a degree. All sources of information have been specifically acknowledged and all verbatim extracts are distinguished by quotation marks.

Signature:  Date: 11 May 2022

Contents

List of Tables

List of Figures

List of Algorithms

Acronyms

CO Combinatorial Optimisation. 1–6, 9, 10, 12, 14, 15, 19, 20, 59, 62, 69, 70, 75

DA Digital Annealer. 1–4, 6, 8, 9, 15, 58, 60, 70, 73, 76

EC Expected Constraint. 29, 30, 32, 35–41, 46–48, 50–54, 56, 62

GS Greedy Search. 27, 41, 42, 44

MKP Multiple Knapsack Problem. ix, 28, 31, 32, 35, 36, 47–51, 53–55, 62

ML Minimum Lazy. 29, 30, 32, 36, 39, 41, 46–48, 50, 51, 53, 54, 62

ML Machine Learning. 4, 18–20, 48, 53, 54, 56

MVC Minimum Vertex Cover. 12, 13

QA Quantum Annealing. 1–4, 7–9, 15, 16, 58, 62, 73

QAP Quadratic Assignment Problem. ix, 28, 32, 47–50, 52–55, 62

QROSS QUBO Relaxation Parameter Optimisation via Learning Solver Surrogates. 4

QUBO Quadratic Unconstrained Binary Optimisation. 1–5, 9, 10, 12–21, 25–29, 31–33, 37–39, 41, 42, 44, 46, 48, 50, 55–64, 69, 70, 72–76

RC Reality Check. 29, 30, 32, 36, 40, 41, 46–48, 50–54

RLD Run-Length Distribution. 22, 25–27, 30, 36, 44, 45, 51–54, 59, 61–63

RTD Run-Time Distribution. 22, 25, 27, 30, 36, 44, 45, 51–54, 59, 61–63

SA Simulated Annealing. 1, 3, 6–8, 15, 20, 27, 41, 42, 44, 50–54, 60, 61

SS Solver Surrogate. 19

TS Tabu Search. 16, 20, 27, 41, 42, 44, 51–54, 61

TSP Travelling Salesman Problem. ix, 28, 32, 42, 47–50, 52–55, 62

VL Verma and Lewis. 27–32, 35–44, 46–48, 50, 51, 53–55, 61, 62

Chapter 1

Project Overview

1.1 Background

The Quadratic Unconstrained Binary Optimisation (QUBO) is a model that can be used to represent many Combinatorial Optimisation (CO) problems, including the Travelling Salesman Problem (Lucas 2014) and Quadratic Knapsack (Glover et al. 2019). It is widely applied in the quantum computing area as such models can be mapped onto a network of qubits and then solved in a process called Quantum Annealing (QA) (Lewis & Glover 2017). The QA is similar to an algorithm called Simulated Annealing (SA); however, quantum fluctuations cause state transitions instead of thermal fluctuations (Kadowaki & Nishimori 1998). To emulate quantum effects with classical algorithms, Fujitsu Laboratories has developed a Digital Annealer (DA) – CMOS hardware designed to solve fully connected QUBO problems (Aramon et al. 2019).

The QUBO models are used to represent constrained optimisation problems in a single unconstrained function of the following form: $min\ x^tQx$, where x is a binary vector and Q is a square matrix of constants (Glover et al. 2019). Since the function is unconstrained, the constraints of the original problem are transformed into quadratic penalties. This penalty function is added to the objective function and will worsen it if the configuration of the binary decision vector is infeasible. The amount by which the objective function will be affected depends on two factors:

1. The severity of the broken constraints. Therefore, if no constraints are broken, the penalty function will be equal to zero, leaving the original objective function unimpaired.

2. Penalty coefficient, M, multiplied by the penalty function to control the degree to which the penalty function will influence the overall fitness.

If the penalty coefficient is too low, the broken constraints will be undervalued, and the solution produced by the optimiser will be infeasible. On the other hand, if the penalty coefficient is too large, the solution process will be negatively impacted as the penalties will overwhelm the objective function making it harder to differentiate between good and bad solutions (Glover et al. 2019). In some cases, it is possible to use domain expertise to select an acceptable M (Glover et al. 2019), but it has also been shown that some problems can have solutions of better quality if you choose the penalty coefficient carefully (Seker et al. 2020). Finding an optimal penalty value is not trivial and different techniques have been proposed to estimate it (Glover et al. 2019, Verma & Lewis 2020, Huang et al. 2021).

This project will focus on studying QUBO, algorithms used for solving problems formulated in QUBO, reviewing the existing techniques for the penalty coefficient generation, implementing one of them, designing and implementing a new algorithm for M generation, and, finally, comparing the quality of the solutions produced using penalties generated by the existing and the new methods with different algorithms.

1.2 Motivation

There are many CO problems from industry, government and science that can be reformulated in QUBO (Glover et al. 2019). Subsequently, they can be solved on one of the QUBO solvers like QA devices developed by D-Wave Systems (Johnson et al. 2011) or DA developed by Fujitsu Laboratories (Aramon et al. 2019). All the problems being solved on D-Wave QA devices are formulated as an objective function in QUBO format or an equivalent Ising model (Cruz-Santos et al. 2019), while the problems solved on DA are formulated only in QUBO (Aramon et al. 2019). Both types of devices are commercially available and are actively used to solve real-world problems, some of which include:

1. Finding optimal routes for collecting delivery parts in warehouses (Fujitsu n.d.).
2. Searching for molecules with desired properties within a predefined chemical space to design new drugs (Snelling et al. 2020).
3. Optimising travel routes in real-time in response to traffic conditions (D-Wave Systems 2021).
4. Optimising the process of painting car bodies as they travel down the assembly line to minimise the number of times the workers need to switch between colours (D-Wave Systems 2021).

Estimating better penalty coefficients will help produce higher quality solutions and break fewer constraints. The new method, if it proves to be more effective than the existing one,

can be used to find more optimal penalty scalars for problems yet to be solved and for the ones that have been solved using QUBO formulation. It can be applied in industry to potentially improve solutions at no significant costs. And since better solutions can lead to greater utilisation of resources and save money, it is highly desirable to use a better M estimation algorithm if such is found.

1.3 Aims & Objectives

Aim Create a new method of estimating penalty coefficients for QUBO problems and compare it to an already existing method.

Objective 1 Study QUBO and the algorithms used for solving problems in such formulation.

Objective 2 Study the existing methods of estimating M and critically evaluate them.

Objective 3 Implement one of the studied methods.

Objective 4 Propose and implement a new method for estimating M.

Objective 5 Formulate QUBOs for instances of a single CO problem and run optimisation algorithms with M coefficients generated by an existing method (implemented in *Objective 3*) and by a new method (implemented in *Objective 4*).

Objective 6 Compare the quality of the produced solutions. Disseminate.

1.4 Key Techniques

The QUBO formulation Many CO problems can be modelled as QUBOs. Some of the methods used to reformulate different types of such problems are described by Glover et al. (2019).

Algorithms for solving QUBOs There are different algorithms for solving QUBOs, and some of them are tabu search and SA (Beasley 1998*a*), QA (Kadowaki & Nishimori 1998) and the DA's algorithm (Aramon et al. 2019).

PyQUBO A Python library that can formulate QUBOs out of the objective functions and the constraints of the CO problems (Zaman 2021). It allows to solve the defined QUBOs with SA or using a D-Wave QA machine if one is available.

qbsolv A Python library developed by D-Wave Systems (Booth et al. 2017). It can be used to find a solution to an already formulated QUBO problem by splitting it into smaller subQUBOs, solving them separately and combining the results. This is

useful when QUBO size is too large, meaning it cannot be mapped onto a D-Wave QA device. The decomposed QUBO can then be solved with a QA device or using tabu search on a classical device.

Analytical approach to estimating M The analytical approach to estimating M was described by (Glover et al. 2019). It involves using domain knowledge to approximate an average-case value of the original objective function (without taking the penalties or constraints into consideration). A percentage (75% to 150%) of this value is used to set an initial penalty coefficient, which can be further improved by running the solver and iteratively updating M until a satisfactory solution quality is reached.

Noisy Intermediate-Scale Quantum approach to estimating M Verma & Lewis (2020) have proposed a more systematic method of estimating M. Their approach involves calculating the maximum positive change that a single bit-flip can do to the original objective function and setting the penalty scalar to that number.

Machine Learning approach to predicting M QUBO Relaxation Parameter Optimisation via Learning Solver Surrogates (QROSS) uses Machine Learning (ML) to build surrogate models of QUBO solvers (Huang et al. 2021). This allows capturing the common structure shared by different instances of the same CO problem. The trained model can predict good penalty coefficients for the yet unseen problem instances.

1.5 Legal, Social, Ethical, Professional and Security issues

This project involves Fujitsu Laboratories. The penalty coefficients estimated by both the proposed and existing solutions will be tested using multiple optimisation algorithms. One of such algorithms will run on a DA hardware that belongs to Fujitsu. To minimise legal and security issues, the transformed QUBO problem with proposed penalty values will be sent to the industry advisor, Dr Ayodele, who works at the company. Dr Ayodele will run the optimisations on the DA and send back the results.

Chapter 2

Literature Review

2.1 Introduction

The focus area of this project is penalty coefficient estimation for QUBO models. The QUBO is a formulation of CO problems, which can be used to solve them quicker. For example, with algorithms that belong to the class of annealing metaheuristics.

Section 2.2 introduces the concept of annealing. It explains annealing metaheuristics that can run on classical devices, quantum devices and quantum-inspired classical devices.

Section 2.3 considers QUBO models themselves: what they are, how they are formulated, what other algorithms can solve them, how penalties arise and what is the meaning and importance of penalty coefficients.

Section 2.4 evaluates some of the algorithms that can be used to solve problems in QUBO formulation.

Section 2.5 critically evaluates the existing penalty coefficient estimation techniques, including state-of-the-art methods.

Section 2.6 summarises this review and sets the course for the project by forming a research question.

2.2 Annealing in Combinatorial Optimisation

This section will discuss a class of metaheuristics inspired by physical *annealing*. We will explain what it is and talk about optimisation metaheuristics relevant to this work.

Finally, we will look at the DA – specialised hardware that can efficiently run one of such metaheuristics.

2.2.1 Annealing

Annealing is a heat treatment process of matter like glass or metal (Collins 1921, Wu & Fan 2020). It involves heating the material and carefully cooling it down at a controlled rate to achieve a stable crystal structure. If this rate is exceeded, the resulting crystal will have defects as only locally optimal structures were allowed to be formed (Kirkpatrick et al. 1983). The lack of equilibrium will make such crystal metastable.

2.2.2 Simulated Annealing

The SA metaheuristic uses the temperature concept to approximate the global optimum of the CO problem (Kirkpatrick et al. 1983, Bertsimas & Tsitsiklis 1993). The temperature (T) is a variable that controls the likelihood of making locally non-optimal uphill moves. As in physical annealing, optimisation starts with a high temperature and decreases at a controlled rate. If the rate is low enough, SA is guaranteed to find a global optimum (Liang et al. 2014). It is possible to find good approximations of the optimum faster at higher cooling rates.

The temperature concept allows the algorithm to explore the search space to find the optimal solution (Poole & Mackworth 2017). The SA starts with a high temperature and acts as a random walk, choosing random moves with high probability, which prevents it from getting stuck at a local optimum. As the temperature decreases, the algorithm prioritises advantageous moves, acting more like a greedy algorithm. As SA shifts from higher to lower temperatures, it gradually moves from exploring the search space to finding the optimum.

Let c be the current node, n be the next node (neighbour of node c) and $evaluation(x)$ be the cost function. ΔE is an energy transition (negative when moving downhill) that the proposed node will introduce.

$$\Delta E = evaluation(n) - evaluation(c) \tag{2.1}$$

ΔE and T are the variables that affect the probability of choosing node n, where T controls the influence of ΔE (Johnson et al. 1989).

$$x = \frac{\Delta E}{T} \tag{2.2}$$

The *acceptance probability function* maps the current node, proposed node, and temperature to a probability. Many functions suit this purpose, the original being exponential (Kirkpatrick et al. 1983).

$$P(c, n, T) = e^{-\frac{\Delta E}{T}} \tag{2.3}$$

If acceptance probability is higher than a uniformly generated random number, node c is rejected, favouring node n. However, when $\Delta E \leq 0$ (downhill movement), the proposed node will always be selected (Kirkpatrick et al. 1983).

$$\Delta E \leq 0 \begin{cases} False & P(c, n, T) > random(0, 1) \begin{cases} False & continue \\ True & c \leftarrow n \end{cases} \\ True & c \leftarrow n \end{cases} \tag{2.4}$$

The pseudocode of the SA can be observed in the Algorithm 1.

Algorithm 1: Simulated Annealing (Johnson et al. 1989)

```
1 c ← n
2 while (T > threshold) do
3 |   c ← generate_next_node(c) ΔE ← evaluate(n) − evaluate(c)
4 |   if ΔE ≤ 0 or P(c, n, T) ≥ random(0, 1) then c ← n
5 |   T ← cooling(T)
6 end
```

2.2.3 Quantum Annealing

The QA metaheuristic uses quantum-mechanical fluctuations to find a global minimum of a function (Ohzeki & Nishimori n.d., Tanaka & Tamura 2012). The QA starts with a strong quantum field that gradually decreases (quantum fluctuation). It is analogous to SA, where the temperature was reduced (thermal fluctuation) to achieve the same result.

The QA begins in a uniform superposition of all possible states, but as the strength of the quantum field decreases, a single solution state is approached (Farhi et al. 2001). In SA, neighbouring nodes lie one move away from the current node. In QA, the neighbouring nodes are in a particular range defined by the strength of the quantum field, slowly ceasing in the process of annealing. The transition from one state to another can happen within that range. If there is a short wall between two states, the SA algorithm will need to make a series of disadvantageous moves to reach a better solution or, in other words, *"climb"* over the wall. This is avoided in QA as there is a capacity to tunnel through short walls if the quantum field is strong enough, making QA faster than SA when dealing with search spaces with many thin barriers (Crosson & Harrow 2016). This effect is called *quantum tunnelling*. A comparison of thermal hopping and quantum tunnelling

is shown in the Figure 2.1.

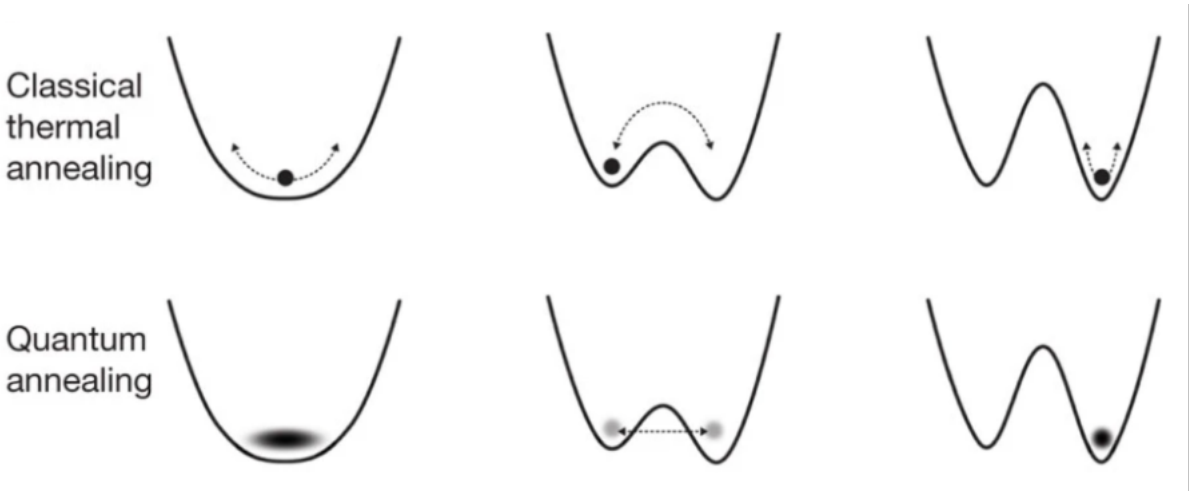


Figure 2.1: Comparison of node transitions in Simulated and Quantum Annealings (Johnson et al. 2011).

The main benefit of QA is that it rapidly samples a wide range of configurations when exploring the energy landscape (Higham & Bedford 2021). The QA devices have been developed and commercialised by D-wave (Johnson et al. 2011). They are used to solve real-world problems (D-Wave Systems 2021).

2.2.4 Digital Annealer

The DA is quantum-inspired custom CMOS hardware developed by Fujitsu Laboratories (Aramon et al. 2019). Its algorithm is based on SA but has three significant differences:

1. Instead of considering a single neighbouring node and choosing it if the acceptance probability is higher than a uniform random number, DA looks at all neighbouring nodes in parallel. If DA accepts more than one node, it will select one of them uniformly at random. Such a procedure is advantageous as in SA if the node was rejected, it needs to generate and consider a new one, which takes time. In DA, however, all the nodes are considered simultaneously, making the acceptance probability per "turn" higher.

2. The DA uses *dynamic offset*: if no nodes have been accepted, it will artificially increase the following acceptance probability to help the algorithm overcome narrow barriers.

3. All runs begin with the same random state to save time, preventing DA from evaluating initial nodes and their neighbours multiple times. Parallel SA generates the initial states for each run separately.

These differences can be observed in the Algorithm 2.

Algorithm 2: Digital Annealer's algorithm (Aramon et al. 2019)

```
initial_state ← random_state
for each run do
    current_state ← initial_state
    E_offset ← 0
    for each step do
        update temperature if needed
        for each neighbour (in parallel) do
            calculate ΔE_neighbour
            consider neighbour using ΔE_neighbour − E_offset
            if neighbour accepted then record
        end
        if accepted neighbours ≥ 1 then
            choose one accepted neighbour uniformly at random
            update current state and the energy landscape (in parallel)
            E_offset ← 0
        end
        else increase E_offset
    end
end
```

Ohzeki et al. (2019) have found that DA can be faster and produce more precise solutions than D-Wave 2000Q (penultimate generation of D-Wave's QA devices) in certain scenarios. DAs are being used in industry and science to solve CO problems (Fujitsu n.d., Snelling et al. 2020).

Currently, QA devices are expensive and challenging to run as the technology is still under development (Seker et al. 2020). D-Wave devices suffer from different problems, including qubit noise that results in lower precision (Katzgraber & Novotny 2018)). These problems are actively worked on and improved from one generation of devices to another (Dattani et al. 2019). As QA technology develops, it will likely outperform DA by a considerable amount due to its quantum parallelism (Boyd 2018).

2.3 The QUBO

Quadratic Unconstrained Binary Optimisation is a mathematical formulation that can be applied to many CO problems (Kochenberger et al. 2014). Many problems from industry, government, and science can be reformulated in QUBO (Glover et al. 2019). The QUBO solvers include D-Wave QA devices, which can also solve equivalent Ising models (Lucas 2014, Cruz-Santos et al. 2019), and Fujitsu DAs (Aramon et al. 2019). The Section 2.2

describes both. This section will discuss the structure of QUBO, how to reformulate constrained CO problems into unconstrained QUBO models using penalties and slack variables, and algorithms used to solve them.

2.3.1 The QUBO model

The QUBO model has been described in detail by Glover et al. (2019). The Equation 2.5 expresses it.

$$\begin{gathered} x \in \{0,1\}^n \\ minimise/maximise\ y = x^t Q x \end{gathered} \tag{2.5}$$

Where y is the value to be optimised for, x is a vector of decision variables, and Q is a square matrix with coefficients. The QUBO models are unconstrained; the only restriction is that variables in decision vector x should be 0 or 1. All the information needed for the optimisation is inside the Q matrix. The QUBO problems are NP-hard. The minimisation problem 2.6 will be used to demonstrate how to convert a Boolean function into a QUBO.

$$y = -5x_1 - 3x_2 - 8x_3 - 6x_4 + 4x_1x_2 + 8x_1x_3 + 2x_2x_3 + 10x_3x_4 \tag{2.6}$$

The function 2.6 has two parts: linear and quadratic. As x_j belong to $\{0,1\}$, linear part can be easily made quadratic too ($0^2 = 0$ and $1^2 = 1$).

$$y = -5x_1^2 - 3x_2^2 - 8x_3^2 - 6x_4^2 + 4x_1x_2 + 8x_1x_3 + 2x_2x_3 + 10x_3x_4 \tag{2.7}$$

The equation 2.7 can be expressed using matrices in the form of a QUBO model.

$$y = x_1x_2x_3x_4 \begin{bmatrix} -5 & 2 & 4 & 0 \\ 2 & -3 & 1 & 0 \\ 4 & 1 & -8 & 5 \\ 0 & 0 & 5 & -6 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} \tag{2.8}$$

2.3.2 Constraints and Penalties

The previous subsection shows how to reformulate unconstrained minimisation problems into QUBO. Constrained problems can also be reformulated into this model (Glover et al. 2019) by expressing the constraints as quadratic penalties and adding them to the original objective function (the function that is being optimised). The larger penalty is imposed when more constraints are broken and when they are important. When no constraint is violated, the penalty is equal to 0. Since a minimisation problem is considered, solutions

that break constraints and impose a penalty on the objective function will be avoided.

$$y = original\ objective\ function + M(quadratic\ penalties) \tag{2.9}$$

M is a positive penalty scalar, also called *the penalty coefficient* (Verma & Lewis 2020). It controls the effect that the broken constraints will have on the objective function. If a constraint is soft and can be broken, M may be decreased, which will reduce the impact of the quadratic penalty on the optimised function. It is possible to use multiple scalars for various constraints if they have different importance, but usually, only a single M is used (Glover et al. 2019).

If the penalty coefficient is too low, the solver will undervalue broken constraints, and the solution produced by the optimiser will be infeasible. On the other hand, if the penalty coefficient is too large, the penalties will overwhelm the objective function making it harder to differentiate between good and bad solutions (Glover et al. 2019). It is possible to use domain expertise to select an acceptable M (Glover et al. 2019), but carefully chosen penalty coefficients can produce solutions of better quality (Seker et al. 2020). Finding optimal M is not trivial and different techniques have been proposed to estimate it (Verma & Lewis 2020, Huang et al. 2021). Quadratic penalties for certain constraints are already known. A few are shown in the Table 2.1.

Classical Constraint	Equivalent Penalty
$x + y \leq 1$	$M(xy)$
$x + y \geq 1$	$M(1 - x - y + xy)$
$x + y = 1$	$M(1 - x - y + 2xy)$
$x \leq y$	$M(x - xy)$
$x_1 + x_2 + x_3 \leq 1$	$M(x_1x_2 + x_1x_3 + x_2x_3)$
$x = y$	$M(x + y - 2xy)$

Table 2.1: Some of the known constraint/penalty pairs (Glover et al. 2019).

Consider a binary problem where either x_1 or x_2 should be equal to one, but they cannot be identical. Using the equivalent penalty from the Table 2.1, the objective function can be expressed as shown in 2.10. The penalty is larger than zero only when both decision variables are the same.

$$y = original\ objective\ function + M(1 - x_1 - x_2 + 2x_1x_2) \tag{2.10}$$

2.3.3 Natural QUBO Formulation

Some CO problems can be naturally expressed in QUBO. This subsection will summarise one such problem described by Glover et al. (2019) to demonstrate the 'natural' QUBO formulation. The Minimum Vertex Cover (MVC) is a CO problem, where we are given an undirected graph with vertexes and edges represented by sets V and E. A vertex 'covers' the edges that it is incident to. Vertex cover is a subset of V covering all the edges. The Figure 2.2 demonstrates when a subset is a vertex cover. The objective is to find a vertex cover with the minimum number of vertices.

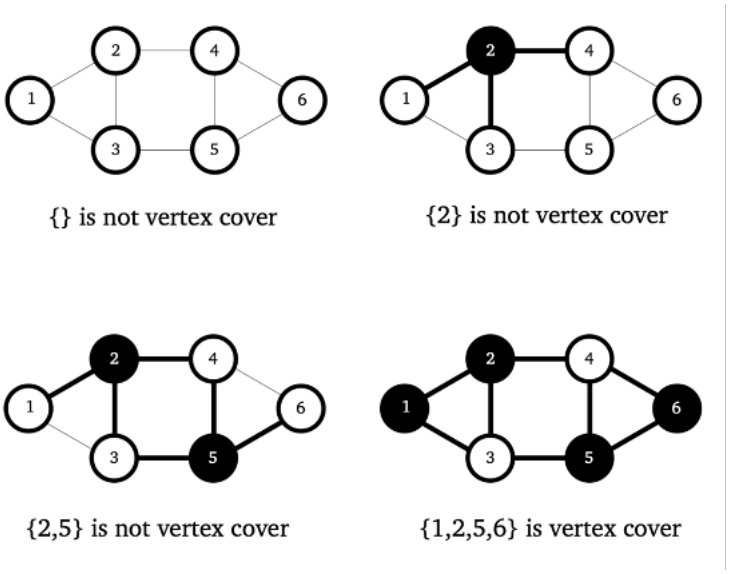


Figure 2.2: Visual representation of generic subsets and a vertex cover.

Decision vector x can be expressed in the following way: if vertex j is in the vertex cover $x_j = 1$, otherwise $x_j = 0$. Then minimisation function before penalties are added is the sum of all vertices in the cover.

$$\begin{aligned} x &\in \{0,1\}^n \\ f(x) &= \sum_{j \in V} x_j \end{aligned} \tag{2.11}$$

To get a feasible solution, all the edges need to be covered. In other words, every edge should have at least one vertex that belongs to the vertex cover.

$$\forall (i,j) \in E, x_i + x_j \geq 1 \tag{2.12}$$

Table 2.1 contains a penalty equivalent to this constraint. It is used to make the following

minimisation function:

$$y = \sum_{j \in V} x_j + M \sum_{(i,j) \in E} 1 - x_i - x_j + x_i x_j \tag{2.13}$$

This formula can be expressed as $y = x^t Q x + c$, where c is a constant that has no influence on the optimisation and can be removed. This leaves us with an unconstrained QUBO model.

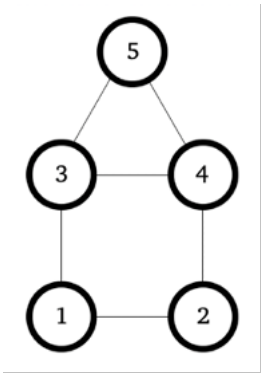


Figure 2.3: Example MVC graph.

MVC to be solved is shown in the Figure 2.3. Using the Equation 2.13, this problem can be expressed and then transformed in the following way:

$$\begin{aligned} y = {} & x_1 + x_2 + x_3 + x_4 + x_5 + \\ & M\,(1 - x_1 - x_2 + x_1 x_2) + \\ & M\,(1 - x_1 - x_3 + x_1 x_3) + \\ & M\,(1 - x_2 - x_4 + x_2 x_4) + \\ & M\,(1 - x_3 - x_4 + x_3 x_4) + \\ & M\,(1 - x_3 - x_5 + x_3 x_5) + \\ & M\,(1 - x_4 - x_5 + x_4 x_5) \end{aligned} \tag{2.14}$$

$$\begin{aligned} y = {} & (1 - 2M)x_1 + (1 - 2M)x_2 + (1 - 3M)x_3 + (1 - 3M)x_4 + (1 - 2M)x_5 \\ & + M x_1 x_2 + M x_1 x_3 + M x_2 x_4 + M x_3 x_4 + M x_3 x_5 + M x_4 x_5 + 6M \end{aligned} \tag{2.15}$$

If the constant that does not affect optimisation, $6M$, is removed and 2.15 is represented

using matrices, the following QUBO is formed:

$$y = x^t \begin{bmatrix} 1-2M & M & M & 0 & 0 \\ M & 1-2M & 0 & M & 0 \\ M & 0 & 1-3M & M & M \\ 0 & M & M & 1-3M & 4 \\ 0 & 0 & M & M & 1-2M \end{bmatrix} x \tag{2.16}$$

By assigning an arbitrary value of 8 to the M, we get the following solution:

$$V = \{2, 3, 5\},\ y = -45,\ x = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 1 \end{bmatrix} \tag{2.17}$$

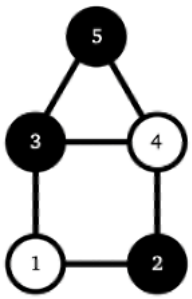


Figure 2.4: Solution to the example MVC.

2.3.4 Non-Natural QUBO Formulation and Slack Variables

Not all CO problems have a natural QUBO formulation, and equivalent penalties for some constraints are unknown. Such problems can be reformulated in QUBO too. In the first part of this subsection, we will show a general approach to QUBO formulation. The second part will focus on how inequality constraints can be expressed as quadratic penalties by introducing slack variables. Both the general approach and the slack variables have been accurately described by Glover et al. (2019).

Consider a constrained minimisation problem. Without quadratic penalties, it will have the following form:

$$\begin{aligned} x &\in \{0,1\}^n \\ y &= x^t C x \end{aligned} \tag{2.18}$$

Where x is a binary decision vector and C is a square matrix. If the constraints consist of

integers, they can be expressed as $Ax = b$, which is shown later in this subsection. Then, the quadratic penalty added to the original objective function is $M(Ax-b)^t(Ax-b)$. The resulting function with penalties is demonstrated in 2.19.

$$\begin{aligned} y &= x^tCx + M(Ax-b)^t(Ax-b) \\ y &= x^tCx + x^tDx + c \\ y &= x^tQx + c \end{aligned} \tag{2.19}$$

The constant, c, is removed as it does not affect the optimisation, and we get a standard x^tQx QUBO formulation.

Now consider an inequality constraint. It can be transferred into an equality constraint by adding a slack variable, s, as shown in 2.20.

$$\begin{aligned} 4x_1 + 5x_2 - x_3 &\leq 6 \\ 4x_1 + 5x_2 - x_3 + s &= 6 \end{aligned} \tag{2.20}$$

By using binary expansion on s, we can achieve an equality (possibly not the only) that satisfies all the configurations of the decision variable x.

$$\begin{aligned} s &\in \{0,1\}^n \\ 4x_1 + 5x_2 - x_3 + s_1 + 2s_2 + 4s_3 &= 6 \end{aligned} \tag{2.21}$$

As both x_j and s_j are binary, they can be contained in a single vector x, and their coefficients can be held in a single matrix A. The inequality is then in a standard $Ax = b$ form, which can be integrated into QUBO as shown in 2.19.

2.4 Algorithms for Solving QUBOs

Many algorithms can effectively solve problems in QUBO formulation (Kochenberger & Glover 2006). We have already described QA and DA algorithms in the Section 2.2. This section will collect some of the algorithms that are available in programming libraries dedicated to solving QUBO models.

2.4.1 Simulated Annealing

The SA has already been touched upon in 2.2.2. However, it was about solving CO problems in general, not particularly in QUBO formulation. The only noticeable feature when solving QUBOs is that the neighbouring nodes are located n bit-flips away from the current node, where n is usually 1 (Alkhamis et al. 1998). It is implemented in the *dwave-neal* (2015) library.

2.4.2 Tabu Search

The tabu search has been adapted to be used with QUBO models (Palubeckis 2004). It can quickly find minimums in neighbourhoods but may sometimes struggle to escape them. It is implemented in the *dwave-tabu* (2018) library.

2.4.3 Decomposing Solver

Decomposing solver splits large QUBO into smaller subQUBOs, solves them separately and combines the results (Booth et al. 2017). The subQUBOs can be solved using a D-Wave QA device or Tabu Search (TS) with a classical machine. It is implemented in the *qbsolv* (2017) library. This approach is practical when using quantum devices because larger QUBOs cannot be mapped onto them, but smaller subQUBOs can be, allowing us to solve bigger problems. On classical devices, a speedup is achieved by decomposing large QUBOs before applying the TS.

2.4.4 Steepest Descent

This is a greedy algorithm similar to gradient descent, but instead of making moves based on the gradient, it uses local minimisation. At each step, a move to the state one bit-flip away that causes the highest energy drop is made. It is implemented in the *dwave-greedy* (2019) library.

2.5 Penalty Coefficient Optimisation Techniques

In this section, we will present four techniques of penalty coefficient optimisation, two of which are state-of-the-art. This area of research is relatively new, and the number of unique approaches to this problem is limited. Penalty coefficient optimisation is not trivial (Verma & Lewis 2020). If the chosen coefficient is too low, the found solution may be infeasible. If it is too large, penalties will dominate the search space, which erases the difference between good and bad nodes, making the optimisation problem numerically unstable and decreasing the accuracy of the produced solutions (Vyskočil et al. 2019).

2.5.1 Analytical

The first approach involves using domain knowledge to set suitable M values (Glover et al. 2019). We need to make a rough estimate of the original objective function and set the initial penalty coefficient to about 75% to 150% of this estimate. The resultant problem needs to be solved using a QUBO solver. The solution should be checked for feasibility: if feasible, we can reduce the M and try again; if not, M needs to be increased. We keep

solving, checking for feasibility, and updating M until a satisfactory penalty coefficient is achieved.

This approach is simple and can produce good penalty coefficients. However, estimating the original objective function "by eye" is not always easy, especially when the problem is large. Also, this trial-and-error procedure requires at least multiple calls to the solver, which is expensive and impractical for performance-critical applications (Huang et al. 2021). This method of penalty estimation is not automated. It would be more efficient to use an algorithm to estimate the initial M, for example, the one proposed by Verma & Lewis (2020), and then adjust the coefficient in a loop using the solver.

2.5.2 Numerical 1

This method involves choosing an arbitrary value of matrix Q and making the penalty coefficient M larger (Verma & Lewis 2020). Although this approach is fast and straightforward, it does not always work. It is possible to improve the estimated M value using the methodology described by Glover et al. (2019) and summarised in 2.5.1, but that would bring the downsides of the analytical approach, inefficiency being one of them, to this method too.

2.5.3 Numerical 2

Verma & Lewis (2020) use the general formulation of QUBO described in 2.3.4 to estimate a lower bound of M. They consider a problem where the next node (x_{to}) has lower energy than the current node (x_{from}). However, x_{from} is feasible, and x_{to} is not. This is shown in 2.5.1.

$$x_{from}^t Q x_{from} > x_{to}^t Q x_{to},\ Ax_{from} = b,\ Ax_{to} \neq b \tag{2.22}$$

In this scenario, penalty coefficient M should be large enough to impose a sufficient penalty that will prevent transition, as shown in 2.5.2. Otherwise, the algorithm would choose an infeasible solution over a feasible one.

$$x_{from}^t Q x_{from} < x_{to}^t Q x_{to} + M\left(Ax_{to} - b\right)^2 \tag{2.23}$$

Equation 2.23 can be rearranged to get a lower bound of M:

$$M > \frac{x_{from}^t Q x_{from} - x_{to}^t Q x_{to}}{\left(Ax_{to} - b\right)^2} \tag{2.24}$$

To calculate the exact lower bound, we need to find a transition that will give us the maximum energy change with the minimum penalty imposed. Since there are no efficient methods for minimising the denominator, they make an optimistic assumption that it is

always equal to 1. They then propose an algorithm that finds the maximum transition for a 1-flip solver in $O(2n^2)$ steps, which is the estimated lower bound of a penalty coefficient as shown in 2.25.

$$M_{est} = \max\left(x_{from}^t Q x_{from} - x_{to}^t Q x_{to}\right) \tag{2.25}$$

Both $0 \to 1$ and $1 \to 0$ flips can cause energy drop. For example, flipping x_1 from 0 to 1 will bring a positive transition to $y = -6x_1$ and flipping 1 to 0 will bring a positive transition to $y = 6x_1$. Thus, every x_i needs to be flipped twice to find the maximum transition:

$$\max \text{transition} = \max \forall x_i \left(\Delta\left(x^t Q x\right)\big|_{x_i:0\to1}, \Delta\left(x^t Q x\right)\big|_{x_i:1\to0}\right) \tag{2.26}$$

The maximum transition of $0 \to 1$ flips can be found by doing the following. For every x_i, sum the coefficient of its linear part (q_{ii}) and all the positive coefficients of its quadratic part ($q_{ij} > 0$). Negative coefficients are ignored as the second variable in quadratic (x_j) will be 0. The maximum transition of $1 \to 0$ flips is found similarly, but since it brings positive changes by nullifying the negative coefficients, coefficient negatives need to be summed ($-6x_1|_{x_1:1\to0}$ will bring a positive change of 6, which is the negative of the coefficient q_1). For the same reasons, the negative coefficients of the quadratic part ($q_{ij} < 0$) need to be summed instead of positive ones. The maximum transition encountered in this process is the estimate of M. All these steps are expressed in 2.27. The estimates with feasible x_{to} are discarded as we only care about feasible to infeasible moves. The next largest M_{est} is selected instead.

$$M_{est} = \max\left\{q_{ii} + \sum_{q_{ij}>0}^{j\neq i} q_{ij} \forall i \in \{1,n\}, -q_{ii} - \sum_{q_{ij}<0}^{j\neq i} q_{ij} \forall i \in \{1,n\}\right\} \tag{2.27}$$

This method is fast and produces good estimates of the lower bound of M, but it does not consider the quadratic penalties, $(Ax_{to}-b)^2$, when estimating the penalty coefficient. That is why this approach can produce M values that are too large. One promising approach is to use Walsh analysis (Brownlee 2009), which explicitly attaches energies to variable interactions in a way natural to the QUBO model. These energies may be used to estimate what values the quadratic penalties can take to decrease the lower bound estimate further.

2.5.4 Machine Learning

Huang et al. (2021) proposed using ML to optimise penalty coefficients. They have reviewed the analytical approach from 2.5.1, where different penalty coefficients are tested on the QUBO solver to find the optimal value. Calling a QUBO solver multiple times to solve a single problem is expensive and is not suitable for applications where performance

is critical. As an alternative, they propose using solutions obtained from the QUBO solver in the past to train an ML model called Solver Surrogate (SS). Then instead of calling a QUBO solver, a faster SS will be called, adjusting the coefficient repeatedly until a satisfactory result is achieved.

As conventional QUBO solvers usually return a batch of solutions with their energies, feasible solutions can be counted to calculate the probability of feasibility (P_f) with the penalty coefficient used. The mean energy of the solutions (E_{avg}) and the standard deviation (E_{std}) can also be calculated. The SS will be trained to predict these three values. During the training, features of the CO problem (g) and the penalty coefficient (A in this paper) from the dataset will be inputted into SS to make a prediction. It is used with ground truth to calculate the loss, which then updates the model through back-propagation. This is repeated with the entire dataset multiple times to train the SS. The Figure 2.5 demonstrates the described process.

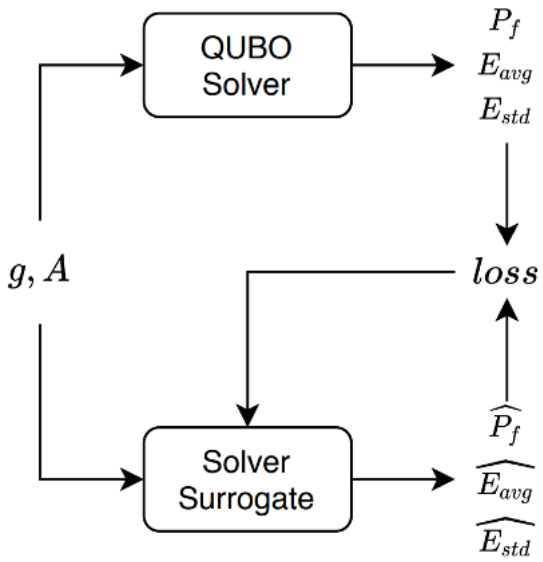


Figure 2.5: Solver Surrogate training (Huang et al. 2021).

After training, the surrogate is used to find penalty coefficient estimates. Problem features and an initial A are inputted into SS to predict P_f, E_{avg} and E_{std}. Using the predicted values, we can adjust A and rerun the SS. Huang et al. (2021) propose three strategies (*Parameter Selection Strategies*) for adjusting A. This process is repeated until a suitable estimate of the penalty coefficient ($\tilde{A}$) is found. The described steps are demonstrated in the Figure 2.6.

This method benefits from good penalty coefficient estimates found quicker than if calls to QUBO solver were made. It can be used to solve many variations of the same

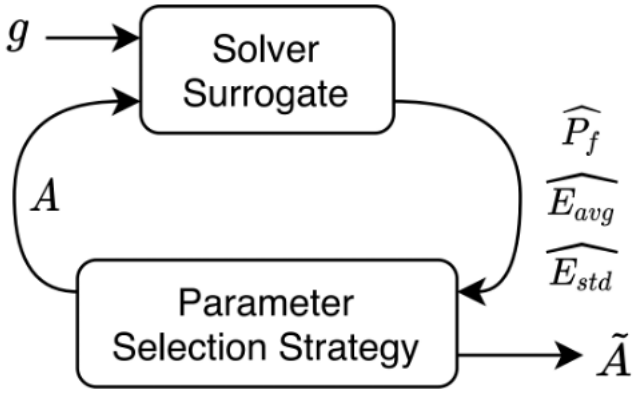


Figure 2.6: Estimating penalty coefficient with Solver Surrogate (Huang et al. 2021).

CO problem, which is common in industry (Fujitsu n.d., D-Wave Systems 2021). However, it will be impossible to train the model if we do not have a dataset with the problems solved in the past, their solutions and penalty coefficients used. In this case, we will need another strategy to estimate penalty coefficients while data collection takes place. The method from 2.5.3 could be used for this purpose.

2.6 Summary

In the Section 2.2, we studied annealing metaheuristics that solve QUBO on classical (2.2.2), quantum (2.2.4) and quantum-inspired classical devices (2.2.3). We also explained how they stem up from physical annealing (2.2.1). In the Section 2.3, we explained QUBO: what the model is (2.3.1), how it expresses constrained CO problems without constraints using quadratic penalties (2.3.2), how to formulate QUBOs in natural (2.3.3) and non-natural (2.3.4) ways. In the Section 2.4 we have collected some of the available algorithms that can be used for solving QUBOs, which include Simulated Annealing (2.4.1), Tabu Search (2.4.2), decomposing solver (2.4.3) and steepest descent algorithm (2.4.4). In the Section 2.5, we have described and critically evaluated existing techniques for penalty coefficient estimation. As analytical method (2.5.1) is slow, the *numerical 1* method does not always work (2.5.2), and the ML method (2.5.4) requires a dataset with problems already solved, our project will build upon the *numerical 2* method (2.5.3). It is fast, consistent and does not require additional data apart from the QUBO. Its primary disadvantage is that it does not consider quadratic penalties during the estimation. This drives the estimates to be larger than the actual lower bound. Our work will take a step towards solving this problem by modifying the method to take quadratic penalties into account in some form or another. We will then set an experiment to test if the produced estimates

lead to better QUBO solutions compared to the original approach. Our research question is: how can quadratic penalties be used to estimate lower penalty coefficients for QUBO models?

Chapter 3

Design & Implementation

3.1 Introduction

In this chapter, we present the design of our solution and details about its implementation.

Section 3.2 explains the setup and structure of our project, how to run it on a new device and the testing approach we have taken.

Section 3.3 introduces the datasets we use for the experiments.

Section 3.4 evaluates the Jupyter Notebooks that we have made along with the experimental design.

Section 3.5 considers the software implementation of our project, which is all contained in a Python package.

Section 3.6 presents the penalty estimators that we have implemented and how they work.

Section 3.7 discusses the experiments we are doing in further detail.

Section 3.8 evaluates what analysis we do, including the analysis of penalty coefficients produced, testing results for statistical significance, Run-Length Distribution and Run-Time Distribution.

3.2 Setup

3.2.1 Version Control

The entire project, including its implementation, is available on GitHub (Appendix D). The repository was used to control versions and track progress and changes. It also served as a backup of the project. The project is public, and anyone can download it. Unfortunately, GitHub does not support large files out of the box. To overcome this, we have used *Git Large File Storage*, which stores the large files on a different server but includes a pointer in the existing repository (GitHub n.d.). Since there is a pointer, the dataset will automatically be downloaded with the repository. Thanks to this, the limitation of GitHub will not affect people interested in the project.

3.2.2 Environment

Our programming of choice is Python for the reasons already described in Appendix B. Nevertheless, choosing its version is not such a trivial task. As in our project we install public packages, which can conflict with each other and the chosen Python version (and the dependency packages of our installed packages can conflict too), we have decided to use the Anaconda package manager (Anaconda n.d.). It will help to avoid all of the described headaches. Anaconda automatically deals with package conflicts right when they are installed. If there are any problems with the Python version, it can easily be changed at any time using the following command:

```
conda install python=<VERSION>
```

Therefore, it was decided to use the latest available version of Python, which was 3.10, and if any conflict arises, to downgrade it. However, no conflict has occurred, and the final project uses the same version. Using the latest version allowed us to utilise cutting edge features of the language to improve the eloquence of our code, such as structural pattern matching (Figure 3.1).

```python
def generate_penalties(self, obj_qubos, con_qubos=None, monotone_value=None):
    # Match input penalty algorithm name to the actual algorithm
    match self.algorithm:
        case 'Verma&Lewis':
            return self.__verma_and_lewis(obj_qubos)
        case 'Monotone':
            if monotone_value is None:  # Demand value argument
                raise TypeError(
                    "PenaltyAlgorithm.generate_penalties() with Monotone algorithm requires 'monotone_value' "
                    "argument")
            return self.__monotone(monotone_value)
        case 'Expected Constraint':
            return self.__expected_constraint(obj_qubos, con_qubos)
        case 'Minimum Lazy':
            return self.__minimum_lazy(obj_qubos, con_qubos)
        case 'Verma&Lewis check':
            return self.__verma_and_lewis_check(obj_qubos)
```

Figure 3.1: Example of structural pattern matching from *PenaltyAlgorithm.generate_penalties(...)*.

The Anaconda also has another useful feature: reproducability. All environment information can be easily exported with the following command (*env* is just an arbitrary name):

```
conda env export > environment.yml
```

The file created can be used to install and run the same environment with the same packages and python version on another computer using the following commands:

```
conda env create -f environment.yml
conda activate env
```

Environment.yml of our project is located in */Project and Deliverables/Implementation/* folder. Anyone who downloads the repository, navigates to this folder and installs our environment can run all of our code and reproduce our results, given they have Anaconda installed.

3.2.3 Software Implementation & Experimental Design

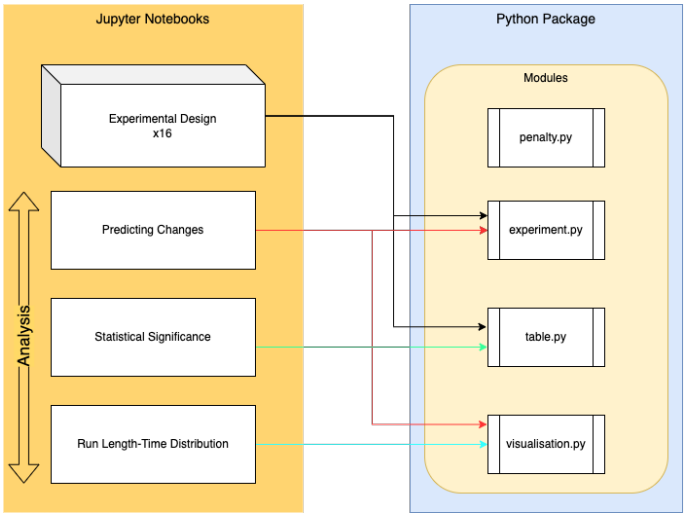


Figure 3.2: The Jupyter Notebooks and what modules they use.

Our project consists of two parts: software implementation and experimental design. We have tried to maintain a clear distinction between them throughout the project. To make the software easily accessible during the experiments and analysis, it was all included in a single Python package consisting of four modules (Python files). They are explained in greater detail in Section 3.5, but their brief overview is presented below:

penalty.py The module includes all the implemented penalty estimation algorithms.

experiment.py The module contains methods for experiments: data preparation and

running the QUBO solvers.

table.py The module collects all methods related to producing tables.

visualisation.py The module collects all methods related to producing visualisations.

However, our experiments and the analysis are shown in Jupyter Notebooks because they provide a natural way to present information. In total, we created nineteen notebooks. Sixteen of them are very similar and are used to produce experimental data with different datasets and penalty coefficients. The other three are for analysis. Again, they are explained in greater detail in Section 3.4, but their brief overview is presented below:

Experimental Design The group of sixteen notebooks where we prepare the data, run experiments, extract, explore and save the results.

Predicting Changes The notebook where we compare penalty coefficients estimated by all algorithms.

Statistical Significance The notebook where we test results for statistical significance.

Run Length-Time Distribution The notebook where we calculate and plot Run-Length Distribution (RLD) and Run-Time Distribution (RTD) to see what penalty coefficients produce feasible solutions faster.

A diagram illustrating the Jupyter Notebooks and what modules they use is displayed in Figure 3.2. As it can be seen, *penalty.py* module is never used by a notebook. This is because penalty coefficients are generated during data preparation in *experiment.py*. Therefore *penalty.py* is only used internally in software.

3.2.4 Testing

While this is an experimental project rather than user-oriented, we have attempted to maintain good software engineering practices. One of such practices is to have a defined *systems development life cycle* that is followed throughout application creation (FOLDOC 2013). The *systems development life cycle* is a set of clearly defined development stages that may differ depending on the developed system. We aimed to make our project well-designed, usable and consistent. With these goals in mind, we have developed and followed the following life cycle: *Design* → *Implementation* → *Testing* → *Documentation*. Every Python method that we have made results from one such cycle.

While design and implementation are the nuclei of this chapter and are thoroughly explained in the following sections, here we want to present the documentation style and then focus on testing. To make the software easily accessible for the potential

users, maintainable and modifiable, we have documented the developed package using a well-known NumPy standard (*Style guide — numpydoc* n.d.). An example of the documentation that we have written for a single method can be seen in Figure 3.3.

```
def generate_penalties(self, obj_qubos, con_qubos=None, monotone_value=None):
    """Generates penalty coefficients for the provided objective
    and optionally constraint QUBOs.

    Parameters
    ----------
    obj_qubos : list
        The objective function QUBOs that we want to generate M for.
    con_qubos : list, optional
        The constraint function QUBOs that we want to generate M for.
    monotone_value : int, optional
        A constant that Monotone algorithms will always return.

    Returns
    -------
    list
        The numpy.float64 penalty coefficients estimated for the provided QUBOs.

    Raises
    ------
    TypeError
        If monotone_value was not provided when using Monotone algorithms.

    """
```

Figure 3.3: *PenaltyAlgorithm.generate_penalties(...)* docstring.

Although conventional approaches like unit testing or iterative testing were not possible due to the nature of the project, the development cycle that we came up with allowed us to focus on the atomic parts of the project, and we ensured that every such aspect worked as intended before proceeding to the next one. We came across interesting implementation issues during the development that we had to deal with. One of them is the fact that we had to solve every QUBO problem multiple times with the same penalty coefficient if we wanted to achieve statistically significant results. However, the D-wave solvers did not allow us to get multiple results for the same problem. The obvious solution would be to run the solver in a for-loop multiple times. But the downside of this approach is that we will get different solutions every time we restart the notebook, which harms reproducibility. The D-wave solvers also allowed us to use random number generator seeds, which guarantee that the solver will produce the same solution for a QUBO as long as we use the same seed number. But we did not want to have the exact same solutions. So we have made a *seed-hack*: we run the solver multiple times in a for-loop, but every time we use a seed value that corresponds to the for-loop iteration (we call this number *repeat*). Therefore, we get the best of two worlds: we have many unique solutions for the same QUBO, but at the same time, if we rerun the notebook, the solutions produced will be the identical.

Another interesting implementation issue that arose is related to Run-Length Distribution, where we wanted to compare how many steps it takes to find feasible solutions with every penalty coefficient that we have estimated. This would allow us to make judgements on whether any particular penalty estimator is advantageous to another. But the D-wave solvers that we have used only provide the solution to the inputted QUBO

with no means to see what energies or states were explored by the algorithm at each step. But that is what Run-Length Distribution is supposed to show: how many runs were successful at every algorithm step. For our experiments, we used three solvers and all of them required a different approach to overcome this issue. With SA solver we could define how many steps we wanted to run it for, so instead we chose to run it with varying numbers steps and see how many solutions were feasible with each one. This solutions is not perfect, because setting a lower number of steps leads to higher annealing cooling temperatures. Nevertheless, it is still a good indication of the speed of finding feasible solutions. The TS solver, however, only allowed us to define the time in milliseconds that we wanted to run the solver for. Thus, we could use the same approach as with the SA, but we needed to plot the Run-Time Distribution instead. Using time is disadvantageous compared to using steps, because the progress made by a solver in certain amount of time depends on outside factors like the processor used for the experiment or how loaded it was. We tried to minimise those risks, which is explained in greater detail in Section 3.8.4. The Greedy Search solver did not allow to specify any parameters related to how long it will run for and this is logical, because the entire purpose of greedy algorithm is to make moves that minimise the energy until no such moves are left. Therefore, we did not plot a distribution for this solver.

Moreover, we have implemented a separate penalty estimator, Monotone, used solely for testing purposes. This estimator always returns the predefined coefficient and does not use any information about the QUBO. This is handy when we want to test newly implemented estimators. More particularly, we want to know whether the coefficients generated with an informed estimator would produce better QUBO solutions compared to the solutions produced using coefficients generated with an uninformed technique. For example, we set the Monotone to always return zero, making the solver ignore the constraint function. Then we observe how many constraints were broken. Suppose the designed estimator produces coefficients that lead to breaking a comparable amount of constraints to the described Monotone. If it does, it is a bad design, or the implementation is wrong, as the zero returning Monotone produces solutions of the worst feasibility. That is how we used Monotone during the design of our estimators.

At the beginning of the project, we have received methods for converting 1D QUBOs into 2D, converting 2D QUBOs into a dictionary that the D-wave solver can work with, and the code for Verma and Lewis estimator from Dr Ayodele. We have tested them because we wanted to be confident that they operate as intended. To achieve that, we studied the provided methods and made sure that we understood every operation. Then, we looked if the methods produced the expected outputs. This was easy to do with QUBO transformation methods, as we only needed to ensure that the transformations applied only

affect the shape of the QUBO and do not change the coefficients. Testing Verma and Lewis estimator was more challenging as we needed a perfect understanding of the estimation process and how it relates to the code that we were given. We noticed that in the code provided, we do not explicitly double the quadratic variables to take into account that in the QUBO they are halved due to its symmetric nature. While the solutions achieved by the estimator were sound, it caused some initial confusion, and to avoid that happening to the future users of the project and improve eloquence, we have modified the code and included the explicit multiplication, explaining why it is needed there. We have tested that the modified version produces coefficients equal to those produced by the original version to guarantee that it operates as intended.

While in this section, we have talked about everything related to testing, we have intentionally avoided discussing requirements testing, where we test if the requirements we have set at the beginning of the project were met. We can meaningfully talk about meeting the requirements only after presenting the results. Thus, the discussion happens later in Section 5.1.

3.3 Datasets

In our project, we use three datasets. They have been given to us by Dr Ayodele, the industry advisor of the project. The provided datasets are located within the *Project and Deliverables/Implementation/Data/* folder in our repository. Each dataset contains a distinct type of problem: Multiple Knapsack Problem (MKP), Quadratic Assignment Problem (QAP) and Travelling Salesman Problem (TSP). MKP and TSP problems are split into small and large.

All of the problems have been transformed into QUBO form by Dr Ayodele. Beasley (1998*b*), Burkard et al. (2012), Skorobohatyj (1995) have shared the original data files with even more problem instances, but not in the form of QUBO. Although, all the datasets provided to us by Dr Ayodele contain QUBOs, their state is slightly different. The MKP dataset contains QUBO sizes, objective and constraint QUBOs with all the constants inside, but in 1D form. On the other hand, the other datasets have QUBOs (in 2D form) and constants separately and lack QUBO sizes. That is why they required different data preparation methods, which are discussed in more detail in Section 3.5.2.

3.4 Jupyter Notebooks

The Jupyter Notebook is a web-based platform where code and its outputs can be viewed live in the form of a notebook that can also be saved. It is widely used amongst researchers (Perkel 2018) because it naturally showcases work clearly and concisely. Comments and

images can be inserted too to support the narrative. That is why we use Jupyter Notebooks to showcase our work. The project consists of nineteen notebooks, sixteen of which show the experiments. A diagram displaying the notebooks and what modules they use to operate was shown earlier in Figure 3.2.

3.4.1 Experimental Design

This is a group of sixteen notebooks that contains everything related to running the experiments: loading the data, data preparation (including M estimation), running QUBO solvers, extracting, exploring and saving the results. More detail on how experiments are designed is available in Sections 3.5.2 and 3.7. The notebooks have been named using the same pattern: <problems being solved><M estimation algorithm used>. One example of such filename is *Quadratic Assignment Problem Verma and Lewis*.

3.4.2 Predicting Changes

This notebook calculates penalty coefficients using all implemented algorithms and compares them. For the comparison, we plot the data distribution, with penalties on the x-axis and their density on the y-axis. The fact that we use density rather than frequency on the y-axis allows us to draw density curves. We use these lines to compare coefficients generated by all algorithms, see their magnitudes and perhaps see some patterns. There are three plots, one for each dataset. The results of this notebook are in Section 4.2.

3.4.3 Statistical Significance

This notebook checks whether the difference between solutions obtained using coefficients generated by the Expected Constraint, Minimum Lazy and Reality Check algorithms and the Verma and Lewis algorithm is statistically significant. To test this we use the saved files from the sixteen *Experimental Design* notebooks. Comparing the energies of produced solutions would be wrong because algorithms that produce small M and undervalue the constraint function will produce excellent energies while breaking many constraints. This way, the statistical significance test will not compare the quality of the produced solutions. Since our main concern is to lower penalty coefficients without harming the solution feasibility, we test if there is a statistical significance between the number of broken constraints of the solutions obtained using coefficients from Expected Constraint, Minimum Lazy and Reality Check algorithms and the Verma and Lewis algorithm. If there is not, it will mean that we have reduced the penalty coefficients without affecting the feasibility. Since the feasibility of our algorithms is never higher, which is logical since Verma and Lewis will either give a perfect lower bound or overestimate the M, we do not demonstrate on our plots which algorithm was better if the difference was significant. The results produced

by this notebook are present in Section 4.3.

3.4.4 Run Length-Time Distribution

Although the feasibility of solutions produced using penalty coefficients made Expected Constraint, Minimum Lazy and Reality Check algorithms is never higher compared to Verma and Lewis, maybe the lower coefficients contribute to finding feasible solutions faster, as described in Section 2.5. We plot the RLDs and RTDs to test whether the expectation holds. More information about RLD and RTD and why we need both of them can be found in Section 3.8.4. The results of this notebook are presented in Section 4.4.

3.5 Software Implementation

Since Jupyter Notebooks aid in showing action and results, we do not want a vast amount of code to distract the viewer from the narrative. Moreover, the same methods might be used in different notebooks, which will create unnecessary clutter. Therefore, we have created a Python package with the software implementation, and in the notebooks, we call the pre-coded methods. Thus, it is called *code* and is located in *Project and Deliverables/Implementation/* folder.

Our package consists of four modules, *.py* files that we describe in this section. Each file contains a single class. Since our classes are collections of methods that are used in experiments and analysis, rather than a type of data, all of them are either static or class methods (the only exception being the *PenaltyAlgorithm* class, which we explain in the following subsection). This means they can be called like functions without first initialising the object, making our notebooks more straightforward. Class methods differ from static only in that a static method is simply a procedure that knows its inputs, while a class method is also aware of which class it was called from. In some cases, this feature is useful, for example, when we want to call another method from the same class inside of a method.

Some of the implemented methods are private, meaning they can only be used within the class itself and cannot be called anywhere else (in the notebooks). They are mainly used as helpers in public methods and maintain the logical structure of the code.

3.5.1 penalty.py

This module contains the only class in the package that needs to be instantiated before its use - *PenaltyAlgorithm*. Because it is a data type that models a penalty algorithm: when we create an object, we specify which algorithm it is. An appropriate error will be thrown

if such an algorithm does not exist. Afterwards, we can call the same method, *generate_penalties*, to generate M for the inputted QUBO, regardless of the chosen estimation algorithm. Below, we list all the methods that are part of this class.

__init__ The constructor method. Accepts the name of the M algorithm that we want to use.

__new__ This method checks if the class supports the algorithm name that entered the constructor. If not, the method prevents an object from being created. A *ValueError* will be returned explaining what the problem was and what names are allowed.

generate_penalties The method that generates penalty coefficients. As mentioned in Section 3.2.2 and shown in Figure 3.1, it uses a new Python feature to connect the algorithm name stored in the object to the matching M estimation method. As an input, it always needs an objective function QUBO. However, it can also accept a constraint QUBO for constraint function aware algorithms and a monotone value if we use the Monotone.

__monotone A private static method that generates M using the Monotone (Section 3.6.1). That is, it returns the number that went into the *generate_penalty* as a monotone value.

__verma_and_lewis A private static method that generates M using the Verma and Lewis algorithm (Section 3.6.2). Initially implemented by Dr Ayodele, but was modified by us, which we explain in more detail in Section 3.2.4.

__verma_and_lewis_check A private static method that generates M using the Reality Check algorithm (Section 3.6.5).

__expected_constraint A private static method that generates M using the Expected Constraint algorithm (Section 3.6.3).

__minimum_lazy A private static method that generates M using the Minimum Lazy algorithm (Section 3.6.4).

3.5.2 experiment.py

This module contains a class of the same name with methods relating to running and preparing data for experiments.

__convert_1d_qubo_to_2d A private static method provided to us by Dr Ayodele. It transforms a 1D QUBO into a conventional 2D QUBO, which the rest of the program can then work with. The MKP dataset is in 1D format.

__convert_qubo_to_dwave_format Another private static method provided to us by Dr Ayodele. It, in turn, translates the 2D QUBO into a format that the D-Wave solvers that we use to solve QUBOs require. It is utilised in *data_prep_light* after penalty coefficients have been generated using the conventional 2D form.

data_prep_light A class method for the basic data preparation. It accepts objective QUBOs, constraint QUBOs, the name of the M algorithm, whether the problem is minimisation and ***kwargs* - optional arguments that are then passed to the PenaltyAlgorithm's *generate_penalties* method. One of such arguments is the monotone value described in Section 3.5.1. This method is used for the QAP and TSP data preparation. It is also used for the MKP inside *data_preparation* method, but only after the QUBOs are transformed into a 2D form.

The data preparation includes generating penalty coefficients, consolidating objective and constraint QUBOs into a full QUBO, and translating maximisation problems into minimisation by flipping the sign of the objective function. The latter is needed because solvers used for the experiments minimise the QUBO value to find the solution. The constraint function is always a minimisation as we want to reduce the imposed penalty, so we do not touch it.

data_prep A class method for advanced data preparation. It is used with the MKP dataset because its QUBOs are one dimensional. The method first converts them into a 2D layout and then applies the standard *data_prep_light*.

run_sampler A static method that accepts a full QUBO, objective and constraint QUBOs, their constants, initialised solver instance, with which the problems will be solved, number of times the experiment needs to be repeated with the same data, and ***kwargs* - additional arguments that will be passed to the solver as hyperparameters when running.

Since the solution process is pseudo-random and not all solvers can solve the one problem several times out of the box, we use *seed-hack*. We solve the same problem multiple times with a new seed and get multiple results for each QUBO. This approach is explained in more detail in Section 3.2.4. Thanks to the *seed-hack*, we will be able to say whether the feasibility of solutions achieved using Verma and Lewis coefficients is significantly different in each problem to the solutions achieved using Expected Constraint, Minimum Lazy and Reality Check coefficients (Section 3.8.3).

At the end, a dictionary containing objective and constraint energies of all solutions from all runs is returned. If we have previously converted the problem into minimisation, the objective energy will be converted back to maximisation.

3.5.3 table.py

This module contains a class of the same name with methods related to producing tables.

display_side_by_side A static method that allows displaying multiple tables next to one another. As in our notebooks sometimes we want to show several tables, and if we do this sequentially, it takes much space and the tables following each other are hard to compare, we use this method. To achieve this, we transform the DataFrames (tables) into HTML and horizontally merge them. We then use a feature of the Jupyter Notebook that allows displaying HTML. An example of such output is shown in Figure 3.4.

Greedy

	Size	Penalty	Objective Function	Broken Constraints	Energy (minimisation)
0	85	892	2831	0	-2831
1	85	892	1828	0	-1828
2	90	892	3009	1	-2117
3	85	892	1023	3	1653
4	90	892	2203	0	-2203
5	100	892	1103	1	-211
6	100	892	3050	0	-3050
7	100	892	2338	0	-2338
8	100	892	1185	2	599
9	110	892	3604	9	4424
10	110	892	3984	324	285024
11	110	892	3604	5	856
12	110	892	2768	1	-1876
13	120	892	3785	2	-2001

SA

	Size	Penalty	Objective Function	Broken Constraints	Energy (minimisation)
0	85	892	2618	14	9870
1	85	892	1505	1	-613
2	90	892	1777	5	2683
3	85	892	2441	20	15399
4	90	892	2326	34	28002
5	100	892	2757	1	-1865
6	100	892	2724	7	3520
7	100	892	3522	0	-3522
8	100	892	1910	5	2550
9	110	892	2536	17	12628
10	110	892	3061	117	101303
11	110	892	3505	16	10767
12	110	892	2776	1	-1884
13	120	892	4181	88	74315

Tabu

	Size	Penalty	Objective Function
0	85	892	1743
1	85	892	881
2	90	892	2242
3	85	892	557
4	90	892	2298
5	100	892	3696
6	100	892	515
7	100	892	852
8	100	892	2266
9	110	892	3166
10	110	892	2444
11	110	892	2548
12	110	892	1493
13	120	892	2962

Figure 3.4: Sample of the output produced by *Table.display_side_by_side(...)*.

record_results A static method that records the results of an experiment in a series of tables, where the number of tables is the number of times the experiment was repeated with the *seed-hack*. The method accepts the output returned by *Experiment.run_sampler(...)*, the sizes of QUBOs that were solved, the number of experiment repeats, and whether the problems are minimisation. From the first, it extracts the objective function values and the number of the broken constraints of solutions and displays them with all the other data that was provided. It also counts and displays the overall energy of the solution. For simplicity, the energy shown is for minimisation (this is why we specify the type of solved problems in the input). Three of such tables are displayed in Figure 3.4 (the final columns of the third table are not visible). These are the individual tables corresponding to particular experiment repeats (only a fraction of what this method returns).

columns_to_table A static method that accepts the output of *record_results* and the

name of the column that we are interested in. It then displays that column for all of the experiment repeats (for example, *Objective Function* or *Broken Constraints*). This method is important because *record_results* returns a list of tables, where each table corresponds to a repeat. Therefore, we cannot easily compare, for example, energies of solutions from all repeats without this method. The columns in this table correspond to the experiment repeat, and the rows correspond to the problem solved. Figure 3.5 shows what a produced table might look like.

	Broken Constraints 0	Broken Constraints 1	Broken Constraints 2	Broken Constraints 3	Broken Constraints 4	Broken Constraints 5
0	0	5	1	1	16	1
1	0	2	1	1	0	5
2	1	1	6	4	1	2
3	3	4	22	2	1	17
4	0	3	4	3	1	1
5	1	0	2	1	3	1
6	0	2	2	0	1	2
7	0	1	0	4	2	2
8	2	9	20	22	0	4
9	9	2	0	4	1	6
10	324	38	47	9	30	54
11	5	5	1	4	1	57
12	1	73	2	2	19	20

Figure 3.5: Sample of the output produced by *Table.columns_to_table(...)*.

feasibility_table A static method that is very similar to *columns_to_table*. It also accepts the output of *record_results*, but (calculates and) displays whether the solutions are feasible or not (Figure 3.6).

	Feasible 0	Feasible 1	Feasible 2	Feasible 3	Feasible 4	Feasible 5
0	False	False	True	True	False	False
1	True	True	False	False	False	True
2	True	False	False	False	True	False
3	True	False	False	True	True	True
4	False	False	True	False	True	False
5	False	True	True	True	False	False
6	False	False	False	False	False	False
7	False	False	False	True	False	False
8	True	True	True	True	False	True
9	True	False	True	True	False	True
10	True	False	True	False	True	False
11	False	False	True	True	False	False
12	True	True	True	True	True	False

Figure 3.6: Sample of the output produced by *Table.columns_feasibility_table(...)*.

feasibility_statistic A class method that also accepts the output of *record_results*. Displays how many instances of each problem were feasible throughout all runs, the feasibility rate (*feasible/all*), and the mean and Standard Deviation of the energy of the solutions. It also shows the total, mean and Standard Deviation of all the listed parameters at the bottom. As an auxiliary method, it uses *Table.feasibility_table(...)* inside. Figure 3.7 is an example of the output that this method can produce. Only the top rows are displayed as the entire table is too large.

	Feasible	Feasibility rate	Energy mean	Energy SD
0	9.000000	0.150000	-1.678333e+02	2195.203903
1	9.000000	0.150000	-9.236833e+02	1194.965051
2	3.000000	0.050000	3.902333e+02	2275.659244
3	2.000000	0.033333	4.185317e+03	5791.045477
4	6.000000	0.100000	1.634483e+03	4050.284868
5	11.000000	0.183333	-1.714650e+03	1229.468244
6	6.000000	0.100000	-1.095900e+03	1473.423728
7	10.000000	0.166667	-1.827783e+03	895.949077
8	8.000000	0.133333	3.124367e+03	8681.375662
9	3.000000	0.050000	3.081067e+03	9027.810514
10	0.000000	0.000000	5.644688e+04	66792.177739

Figure 3.7: Sample of the output produced by *Table.columns_feasibility_statistic(...)*.

significance_test A static method that uses the Student's t-test to check if the difference in the number of broken constraints of solutions obtained with penalty coefficients of Verma and Lewis and other algorithms is statistically significant across the entire dataset. The table displays the t-statistics and p-values. If the t-statistic is negative, the number of broken constraints with Verma and Lewis coefficients was smaller. An example of such a table with a comparison of Verma and Lewis to Expected Constraint coefficient solutions using the MKP dataset is shown in Figure 3.8.

	Greedy Algorithm	Simulated Annealing	Tabu Search
t-statistic	-2.706085e+01	-2.894882e+01	-7.839577e+00
p-value	6.176995e-132	4.311554e-147	8.449691e-15

Figure 3.8: Sample of the output produced by *Table.significance_test(...)*.

detailed_significance_test A static method that is a more detailed version of the *significance_test*. It tests if the difference was significant for every problem instance rather than the entire dataset. An example of such a table with a comparison of Verma and Lewis to Expected Constraint coefficient solutions using the MKP dataset is shown in Figure 3.9. Only the top rows are displayed as the entire table is too large.

Problem	Greedy Algorithm t-statistic	Greedy Algorithm p-value	Simulated Annealing t-statistic	Simulated Annealing p-value	Tabu Search t-statistic	Tabu Search p-value
0	-5.817758	1.012446e-06	-8.456664	2.897396e-10	-5.377673	4.059575e-06
1	-11.433647	7.257981e-14	-7.807491	2.043053e-09	-3.191075	2.841796e-03
2	-11.372881	8.498862e-14	-8.882317	8.264231e-11	-3.076830	3.868503e-03
3	-11.163522	1.469261e-13	-7.199482	1.323164e-08	-4.004885	2.783259e-04
4	-11.667644	3.969835e-14	-9.577043	1.119030e-11	-2.900908	6.156498e-03
5	-7.404661	7.017574e-09	-8.367622	3.777044e-10	-3.920765	3.569684e-04
6	-9.158312	3.707417e-11	-10.828362	3.571092e-13	-3.137988	3.281780e-03

Figure 3.9: Sample of the output produced by *Table.detailed_significance_test(...)*.

find_best_significance A static method that accepts *detailed_significance_test* output. It counts the number of problems where Verma and Lewis coefficients resulted in significantly less broken constraints, significantly more broken constraints and the number of problems where there was no significant difference compared to solutions that used coefficients generated by Expected Constraint, Minimum Lazy and Reality Check algorithms. An example of such a table with a comparison of Verma and Lewis to Expected Constraint coefficient solutions using the MKP dataset is shown in Figure 3.10.

	VL was better	No significant difference	VL was worse
Greedy	38	0	0
SA	37	1	0
TS	35	3	0

Figure 3.10: Sample of the output produced by *Table.find_best_significance(...)*.

3.5.4 visualisation.py

This module contains the *Figure* class with methods related to producing plots.

__get_distribution_data A private method that will make a RLD or RTD table, which is explained in more detail in Section 3.8.4. In the input, we can specify the maximum step or time that we want to reach, the magnitude at which we will increase it (if 100, the tested steps/times would be 1, 101, 201, ..., upper bound+1), and the number of times we want to repeat the distribution generation to get a more stable average.

run_distribution A static method that uses *__get_distribution_data* to generate RLD or RTD and then plots it. This is the method that is called in the *Run Length-Time Distribution* notebook described in Section 3.4.4. The results obtained with it can be seen in Section 4.4.

penalty_distribution This static method plots a distribution of penalty coefficients obtained using different M estimation algorithms, with coefficients on the x-axis and their density on the y-axis. The density curves are displayed, which allows us to compare the behaviour of the algorithms. It is used in *Predicting Changes* notebook described in Section 3.4.2.

__smooth_the_line A private helper method that we call whenever plotting a curve. It uses spline to interpolate the data provided, transforming the inputted line with many edges into a continuous curve.

3.6 Penalty Estimation Algorithms

In this section we describe all the penalty estimation algorithms that have been used in our project. Their implementation can be found in the *penalty.py* module.

3.6.1 Monotone

This algorithm has such a name because it is monotonic - it always returns the inputted coefficient. Therefore, we can use it to return an M that is equal to, for example, 30, for all QUBOs in an experiment. Although this is not present in the final notebooks, it was actively used for testing, especially with a return value of zero, forcing the solver to pay no attention to the constraint function and break any number of constraints. This allowed us to see if our coefficients affect the feasibility at all. We explore the idea of testing with Monotone in more detail in Section 3.2.4.

3.6.2 Verma and Lewis

The Verma and Lewis (VL) is the state-of-the-art algorithm by Verma & Lewis (2020) that was described in Section 2.5.3. It is good, but might overestimate the M.

3.6.3 Expected Constraint

The Expected Constraint (EC) is based on the VL algorithm but attempts to reduce it by calculating the expected constraint that the penalty function will impose. It divides the VL coefficient by that number. This approach is inspired by the Equation 2.24. We know that VL algorithm overestimates M because it assumes that the denominator of the equation is equal to one, where the denominator is the value of the quadratic constraint function: $(Ax_{to} - b)^2 = 1$. In EC, we avoid that assumption and use the expected value of the constraint function as a denominator. Consider a constraint QUBO 3.1, where $c_{i,j}$

is a coefficient that is associated with a product $x_i x_j$.

$$constraint\ QUBO = \begin{bmatrix} c_{1,1}(x_1x_1) & \cdots & c_{1,n}(x_1x_n) \\ \vdots & \ddots & \vdots \\ c_{1,n}(x_1x_n) & \cdots & c_{n,n}(x_nx_n) \end{bmatrix} \tag{3.1}$$

To calculate the expected value of the constraint function, we go through every decision variable in the constraint QUBO (every row) and calculate the penalty it would cause if every variable it is interacting with were one. We sum all the values calculated and divide the result by the number of decision variables. The first operation is equivalent to summing all the coefficients in the constraint QUBO, while the second is equivalent to dividing the result by the number of rows or columns of the QUBO matrix as the number of rows or columns in QUBO is the number of decision variables. This is proved in Equation 3.2.

$$\begin{aligned} expected\ constraint &= \frac{(c_{1,1} + \cdots + c_{1,n})}{n} + \cdots + \frac{(c_{n,1} + \cdots + c_{n,n})}{n} \\ expected\ constraint &= \frac{\sum_{i=1}^{n}\sum_{j=1}^{n} c_{i,j}}{n} \end{aligned} \tag{3.2}$$

Next, to get the penalty coefficient, we need to divide the VL by the expected constraint calculated in 3.2. The entire process can be seen in the Algorithm 3. We use ceiling division to divide VL coefficient by the expected constraint to avoid a scenario where the resultant M is zero. Also, we do not square the denominator but take its absolute value because the expected constraint is likely to be higher than the actual one. We do not want to increase the difference between them further.

Algorithm 3: Expected Constraint M estimation algorithm

```
1 qubo ← input
2 m_verma_lewis ← verma_lewis(qubo)
3 expected_constraint ← sum(qubo)/len(qubo[0])
4 m_expected_constraint ← m_verma_lewis//abs(expected_constraint)
```

We will show step by step how EC coefficient is calculated using the following QUBO:

$$y = -(x_1 + 5x_2) + M(10x_1x_2 + x_2) \tag{3.3}$$

According to Equation 2.27, the VL coefficient would be the maximum energy transition a single bit-flip could bring to the objective function. Therefore, the VL coefficient would be equal to five, as we get the maximum transition when we flip x_2. The expected constraint

would be the sum of all constraint coefficients divided by the number of decision variables:

$$expected\ constraint = \frac{10+1}{2} = 5.5 \tag{3.4}$$

Since the EC estimated penalty coefficient is VL prediction over the calculated expected constraint (ceiling division), the coefficient produced would be $5//5.5 = 1$. This example demonstrates how the estimator works and shows its main disadvantage: the underestimated penalty coefficient caused by the expected constraint in the calculation being too large. Consider the following state transition:

$$\{0,0\} \rightarrow \{0,1\} \tag{3.5}$$

The energy of the QUBO before the transition is zero. The transition with a penalty coefficient produced by EC would bring us to the state with the following QUBO energy:

$$\begin{aligned} y &= -(0+5*1)+1(10*0*1+1) \\ y &= -5+1 \\ y &= -4 \end{aligned} \tag{3.6}$$

The magnitude of the resulting objective function is larger than the magnitude of the constraint function due to the penalty coefficient being underestimated. Thus, the energy after the transition is more advantageous when minimising even though a constraint was broken, which should not happen with the optimal penalty coefficient. However, the example was made with the purpose of demonstrating the flaw of EC. The estimator could still produce good penalty coefficients in the majority of the cases. The following algorithm we present does not suffer from this problem.

3.6.4 Minimum Lazy

The Minimum Lazy (ML) is very similar to the EC algorithm. However, after counting the constraint that every decision variable would cause if everything it has interacted with were one, we find the minimum magnitude present in the resulting list rather than summing it up. Then, just like in EC, we divide (ceiling division) VL coefficient by that *minimum expected constraint* that we 'lazily' calculated.

The Algorithm 4 demonstrates how ML works. Every QUBO row is associated with a decision variable, and that is why we loop through them on lines 4-8. On line 5, we double all quadratic coefficients because the QUBO is symmetric, meaning that the quadratic coefficients appear twice: as ij and ji pairs. Thus, in every row, they are twice smaller than they should be, which is compensated by the fact that they will appear again in the QUBO. Nevertheless, since we do not take into account the rest of the QUBO

(unlike in the EC), but only a single row, they have to be doubled.

Algorithm 4: Minimum Lazy M estimation algorithm

```
qubo ← input
m_verma_lewis ← verma_lewis(qubo)
expected_constraints ← []
for qubo row do
    multiply all quadratic variables in the row by 2
    constraint ← sum(qubo row)
    expected_constraints.append(constraintmagnitude)
end
minimum_expected_constraint ← min(expected_constraints)
m_minimum_lazy ← m_verma_lewis//abs(minimum_expected_constraint)
```

3.6.5 Reality Check

The Reality Check (RC) always returns half of the lower bound estimated by the VL algorithm, so we can find the problems for which it definitely overestimates penalty coefficients. As both of the algorithms described above are based on dividing the VL prediction by some counting number, sometimes large, it is not always possible to see if we are running into another extreme: underestimating the penalty coefficients. That is why we need this *Reality Check*, where we divide (ceiling division) the prediction by the lowest possible integer after one.

Algorithm 5: Reality Check M estimation algorithm

```
m_reality_check ← verma_lewis(qubo)//2
```

3.6.6 Software Extensibility

This project is particularly extensible because new M estimation algorithms can be easily included and tested. To do that, we need to add the new algorithm to the *penalty.py* module (3.5.1), duplicate any of the *Experimental Design* notebooks (3.4.1) and change the name of the used penalty coefficient algorithm in the *Data Preparation* part. Then, by running that notebook, we will get solutions produced with new coefficients, which can be analysed in the *Statistical Significance* notebook (3.4.3) by changing the names of the input files. This will allow interested researchers to test new ideas quickly and potentially find even better algorithms.

3.7 Experiments

3.7.1 Goals

The goal of our experiments is to compare the solutions produced using Expected Constraint, Minimum Lazy and Reality Check penalty coefficients to solutions produced using VL coefficients, which is a current state-of-the-art. We had to decide what exactly we would compare in those solutions. Comparing the energies is meaningless as it depends on the penalty coefficient used. For example, suppose our M is zero. In that case, the energy of the solution obtained will be excellent as the constraint function will be inactive, and the solver will not pay any attention to the broken constraints. The same applies to solutions obtained with other small penalty coefficients - their energy may be good, but that does not mean that this solution is better than the others as it might be breaking many constraints.

The EC, VL and RC aim to reduce the VL coefficients. But by reducing them, we also impact the guarantees of solution feasibility. That is why our goal is to improve the search performance while preserving similar numbers of constraint violations.

3.7.2 Solvers

For our experiments, we will use three different QUBO solving algorithms from D-wave packages that we have described in Section 2.4. They are the Greedy Search (GS) also called Steepest Descent algorithm, Simulated Annealing (SA) and Tabu Search (TS). We will not compare results obtained by different solvers with each other as it is outside of this project's scope. Instead, we will compare the M estimation algorithms to see what effect different penalty coefficients have on the feasibility with different solvers.

3.7.3 Hyperparameters

All of the algorithms we are using have specific hyperparameters. We have studied them and will list the most important ones with the logic behind their optimisation.

num_reads The only hyperparameter that is present in all algorithms. This is the number of solutions that the algorithm produces for every problem. However, only the best solution is kept. The starting point is different on every run, allowing the algorithms to explore different regions of the fitness landscape.

num_sweeps The SA hyperparameter. Dictates how many steps will be made during the annealing process. The cooling rate is geometric and is automatically adjusted to work with the entered number of sweeps. The default value of 1000 was experimentally found to be good. Reducing it leads to lower numbers of feasible solutions.

Increasing it leads to slower runtimes at little to no feasibility rate increase. Increasing *num_reads* was found to give better results faster. Thus, only *num_reads* is changed for SA.

energy_threshold This optional TS hyperparameter can be used to terminate the algorithm when energy lower than *energy_threshold* is found. We do not use it as we do not have a clear 'successful energy', and we want to see how good of a result the algorithm can achieve in the given amount of time.

timeout The TS hyperparameter. Defines the running time per single solution in milliseconds. The default value is 20. It was found that larger timeout values result in more feasible solutions but slower run times. Moreover, increasing the timeout led to better solutions compared to increasing *num_reads* in the same amount of time. Therefore, the latter is set to the default value (1) and only the *timeout* is increased for the TS.

Therefore, the hyperparameters we will be changing are *num_reads* with GS and SA and *timeout* with TS. For every QUBO solver, we have chosen the hyperparameters that will solve the dataset (one repeat) in 30 seconds with the VL penalty coefficients. As for every dataset the optimal hyperparameters would be different, we have optimised them individually. Moreover, large TSP problems took significantly longer to run compared to small ones. That is why we divided TSP experiments into small and large, both with different hyperparameters. This way, we ensure that the experiment is fair and that every solver gets the same amount of time to find solutions with every dataset.

To achieve statistically significant results in our analysis, we run each algorithm on the same dataset 20 times except for experiments with the VL coefficients, where it is 60 times, which is explained in Section 3.8.3. Therefore, a single solver will take approximately 600 seconds or 1800 seconds to find QUBO solutions.

The biggest downside of this approach is that a different number of operations will happen in 30 seconds every time because the computer efficiency depends on many factors. To minimise their effects, we have optimised hyperparameters on the same computer, one experiment at a time and no other significant tasks were running on our CPU during this process.

3.8 Analysis

In this section, we describe the analysis carried out in our project.

3.8.1 Penalty Coefficients

We compare the penalty coefficients produced by different algorithms by plotting their distribution. This way, we can see the magnitudes of the generated penalties, what patterns they follow and how they compare, allowing us to analyze the tendencies of the algorithms. This has been described in greater detail in the *Predicting Changes* notebook discussion (Section 3.4.2). The results of this analysis can be read in Section 4.2.

3.8.2 Initial Results

In the *Experimental Design* Jupyter Notebooks (Section 3.4.1), after getting the results from solvers, we carry out some initial analysis, which we then save at the end. A lot of this analysis has already been described in the Section 3.5.3 as most of the *table.py* module is dedicated to it. In the beginning, we look at the sizes of the solved problems, M coefficients assigned to them, the objective function values, number of the broken constraints and the energies of the solutions (Figure 3.5).

However, since there are many such tables, one for each repeat, it is challenging to see a broader picture. Thus, we concentrate on separate columns and look at their values across all the repeats next. More specifically, we look at the solution energies, the number of broken constraints (Figure 3.5), and finally, whether the solutions were feasible (Figure 3.6).

In the end, we have the most interesting part - the feasibility statistic. We calculate how many instances of each problem were feasible throughout all runs, the feasibility rate, and the mean and Standard Deviation of the solution energies (Figure 3.7).

3.8.3 Statistical Significance

In Section 3.7.1, we have described our goal: test if the difference in the feasibility of the solutions produced using VL penalty coefficients and the penalty coefficients produced by other algorithms is statistically significant. As we have presented three more algorithms, we will be testing VL solutions three times. If we test multiple hypotheses with the same data, we run into *multiple comparisons problem* as the more conclusions we make from the same data, the more likely it is that they will be wrong (Turkey 1953). This is a *type I statistical error* and to avoid it we will produce three times more results for the algorithm tested multiple times. Before the significance test, we will randomly sample the solutions into three separate groups. Finally, we will test each independent group of VL solutions with solutions from other algorithms. The Student's t-test was chosen to test for statistical significance. Such a decision was made for four reasons: every time we compare two groups, we want to compare their means, but we do not know their Standard

Deviations, and the sample size is small (20). The results of these tests are shown in Section 4.3.

3.8.4 Run Time and Run Length Distribution

As discussed in the Section 2.5, lowering M might lead to QUBO problems being solved faster. Since our algorithms are focused on lowering VL coefficients, we want to see what effect this reduction will have on the speed of finding successful solutions. Since comparing the solution energies is meaningless when we change M (Section 3.7.1), and one of our aims is feasibility comparison, our success criterion will be whether the solution is feasible.

Unfortunately, the D-Wave QUBO solvers do not allow us to pull out the state energy (or feasibility) at each algorithm step. However, when we run SA, we specify the number of steps we want the solver to make, and with TS, we specify how long the solver will run for in milliseconds. Thus, we can produce solutions multiple times, increasing the steps or time and recording the number of feasible solutions. To see the dynamics, we will plot Run-Length Distribution (RLD) for SA and Run-Time Distribution (RTD) for TS, with percentage of successful solutions on y-axis and steps or time on the x-axis. It is important to mention that usually on y-axis the cumulative percentage of successful runs is displayed. But with our approach, cumulativity is lost as we are not running one continuous experiment, but rather multiple ones with varying steps/times. Therefore, sometimes the percentage can slightly drop, but the general increasing behaviour should hold.

There is a disadvantage to RTD that is not present in the RLD: the number of steps that can happen in the same amount of time depends on different factors. Some of them are the CPU we are using, how loaded it is at the moment of the run, and its temperature. Nevertheless, since we TS only works with time, we can only try to minimise the risks. That is why we use the same CPU and do not run anything else on the computer when calculations for RTD are taking place.

It is not possible to specify the number of steps the GS algorithm needs to make or the amount of time it should run for - it will move down the slope for as long as it can. That is why we do not use it here. For every plot made, we count the distribution ten times with different seeds and show their mean to ensure that the trend is valid and the curve we get is stable. The maximum time (*timeout*) and steps (*num_sweeps*) is the same as in the hyperparameters of the experiments (Section 3.7.3). If the success rate with coefficients from all algorithms reaches 100% significantly sooner, that number is decreased.

3.9 Summary

In Section 3.2, we have explained the setup of our implementation: the version control tool of our choice (3.2.1), the environment we have created and how to reproduce it (3.2.2) and the testing approach (3.2.4). Section 3.3 described all the datasets used, how we got them and their specifics. In Section 3.4, we described the Jupyter Notebooks implemented for experiments (3.4.1) and the analysis of M coefficients produced by different algorithms (3.4.2), the statistical significance of the experiments (3.4.3) and Run-Length Distribution and Run-Time Distribution (3.4.4). In Section 3.5, we went over every single module that the produced package contains: *penalty.py* (3.5.1), *experiment.py* (3.5.2), *table.py* (3.5.3) and *visualisation.py* (3.5.4). In Section 3.6, we presented all the penalty estimation algorithms that have been used in our project at some point or another, including *Monotone* (3.6.1), *Verma and Lewis* (3.6.2), *Expected Constraint* (3.6.3), *Minimum Lazy* (3.6.4) and *Reality Check* (3.6.5), and how easily new algorithms can be added and tested (3.6.6). Section 3.7 sets out the goals of our experiments (3.7.1) and lists solvers we have used (3.7.2) with their optimal hyperparameters (3.7.3). Finally, Section 3.8 explains the analysis carried out as part of this project, including the analysis of penalty coefficients (3.8.1), the initial analysis we apply to the results we get straight out of the solvers (3.8.2), and how we compare the M estimation algorithms with what statistical significance test we use (3.8.3).

Chapter 4

Results

4.1 Introduction

In this chapter, we present and discuss the results of the experiments described in Chapter 3.

Section 4.2 presents the distributions of penalty coefficients produced by different estimation algorithms for all datasets.

Section 4.3 shows how the feasibility differs with various penalty coefficients and solvers across all datasets

Section 4.4 evaluates the performance of QUBO solvers with different penalty coefficients.

Section 4.5 is a conclusion with the findings that we have made in this chapter.

4.2 Penalties

Estimator	min	Q1	median	Q3	max
VL	892.00	892.00	892.00	892.00	107200.00
EC	1.00	1.00	1.00	1.00	810.00
ML	1.00	1.00	1.00	5.00	49.00
RC	446.00	446.00	446.00	446.00	53600.00

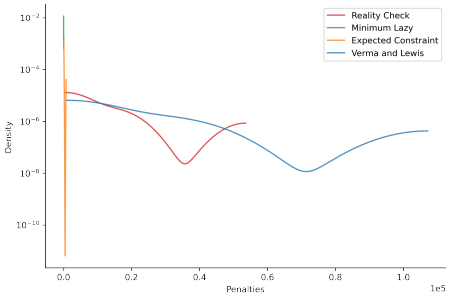
Table 4.1: Statistics of penalty coefficients estimated for the MKP dataset.

Estimator	min	Q1	median	Q3	max
VL	10920.00	26921.00	$8.958720e^5$	$4.602849e^6$	$9.048289e^9$
EC	546.00	926.00	44329.00	$1.358602e^5$	$1.190564e^8$
ML	5460.00	13460.50	$4.479360e^5$	$2.301424e^6$	$4.524144e^9$
RC	5460.00	13460.50	$4.479360e^5$	$2.301424e^6$	$4.524144e^9$

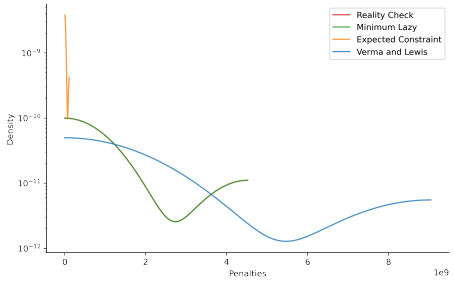
Table 4.2: Statistic of penalty coefficients estimated for the QAP dataset.

Estimator	min	Q1	median	Q3	max
VL	9446.00	10073.00	13841.00	20500.00	571102.00
EC	76.00	214.00	284.00	589.25	5192.00
ML	4723.00	5037.00	6921.00	10250.50	285551.00
RC	4723.00	5037.00	6921.00	10250.50	285551.00

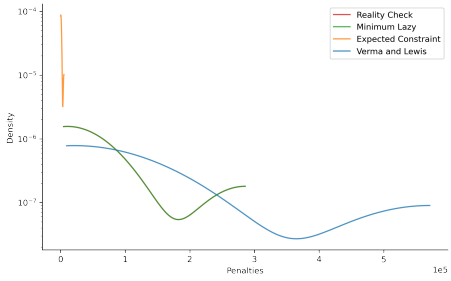
Table 4.3: Statistic of penalty coefficients estimated for the TSP dataset.



(a) Multiple Knapsack Problem dataset



(b) Quadratic Assignment Problem dataset



(c) Travelling Salesman Problem dataset

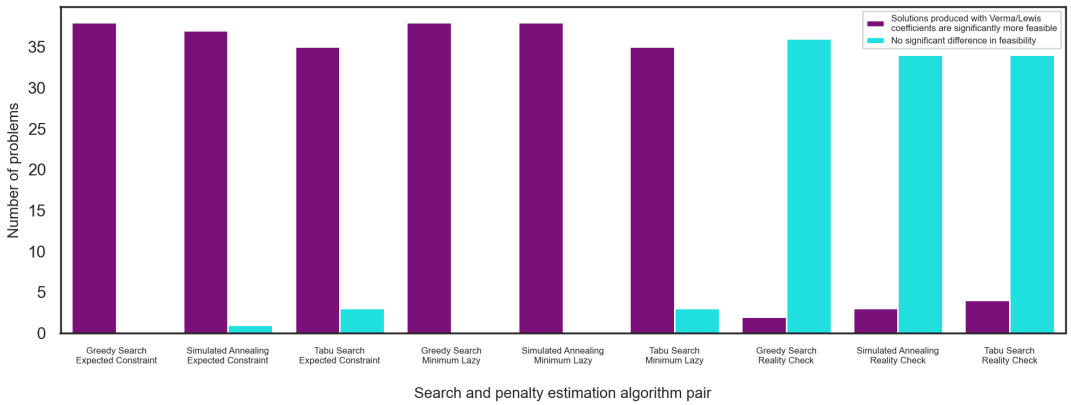
Figure 4.1: Distributions of penalty coefficients produced by all M estimation algorithms.

This section reviews the distributions of penalty coefficients produced by different estimation algorithms for all of the datasets. These are presented in 4.1. It is easy to spot that QAP and TSP dataset distributions have three lines even though we are looking at penalties produced by four algorithms. This is because the dark green line is a combination of ML (bright green) and RC (orange) lines, meaning that both algorithms return the same penalty coefficients. Since we know that RC always returns penalty coefficients that are twice smaller than VL, we can deduce that the *minimum expected constraint* that ML calculates is always equal to two (as it divides VL coefficient by the *minimum expected constraint* as explained in Section 3.6.4). Interestingly, the behaviour of ML is very different with MKP dataset, where it is returning the lowest penalty coefficients compared to the rest of the algorithms. Such a result means that the penalty estimations are problem-dependent. Moreover, since our penalty coefficient estimators use the algebraic structure of the QUBOs, QAP and TSP may share some structural property that causes ML to be the same as RC. Because the coefficients are lower in these cases, VL may be overestimating the penalties. Therefore, it is reasonable to believe that lower penalty coefficients for QAP and TSP is possible.

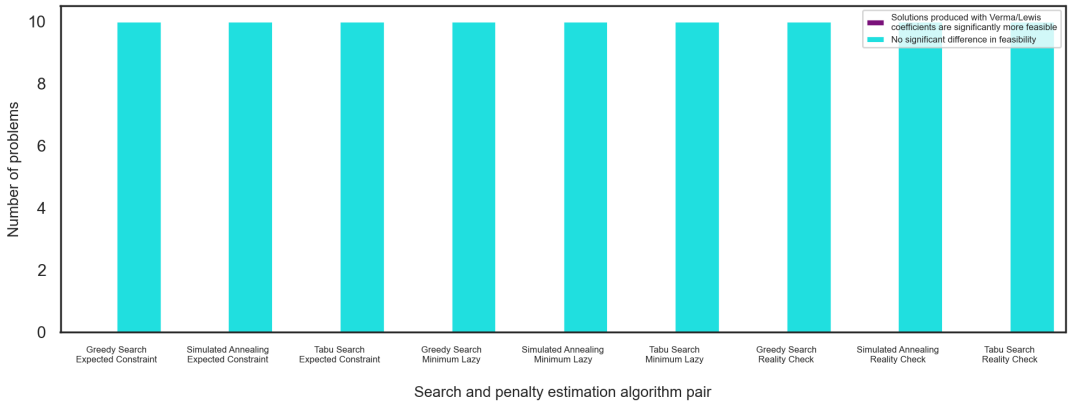
The RC curve always has the same shape with a dip in the middle that is similar to the VL, meaning that both distributions are bimodal. They both have higher quantities of smaller penalties compared to larger ones. But at the same time, RC penalties are smaller compared to the VL. This is expected because RC coefficients are halved VL coefficients. The density of the RC curve is a little higher because by concentrating more penalties in a smaller region of x, we increase their density. The distribution of ML in QAP and TSP datasets is equal to the distribution of RC. Therefore, all the observations we have made about RC apply to ML too in the context of QAP and TSP.

The EC estimator produces tiny coefficients with all the datasets. In Section 3.6.3, we have theorised that it might return undervalued penalty coefficients and were correct. But interestingly, the general *dipped* shape that all the other distributions follow (apart from ML with MKP dataset) is conserved with EC too. Therefore, the distribution shape is generally similar, but the penalties generated are much lower. The dip does not mean much by itself, but it indicates that the produced coefficients have two modes.

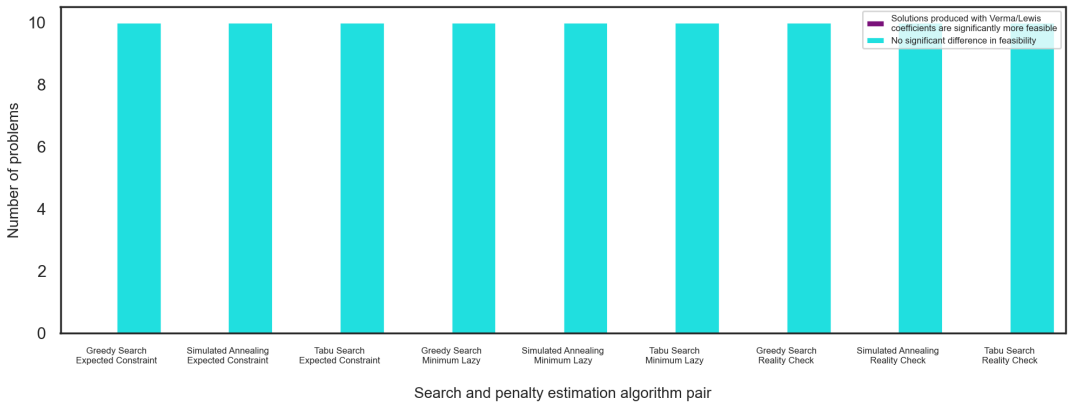
4.3 Feasibility



(a) Multiple Knapsack Problem dataset



(b) Quadratic Assignment Problem dataset



(c) Travelling Salesman Problem dataset

Figure 4.2: Feasibility comparison of solutions produced using Verma/Lewis and other M coefficients.

In this section we compare the feasibility of solutions produced using VL penalty coefficients with solutions produced using EC, ML and RC penalty coefficients. The feasibility comparison is presented in Figure 4.2. We can see that there was no statistically significant difference between the solutions produced with VL coefficients and the solutions produced with the other coefficients in QAP and TSP datasets. Therefore, the lower bounds predicted by VL algorithm were greatly overestimated. We know that the penalty coefficients generated by EC are a lot smaller compared to VL (Figure 4.1). However, the solutions produced by both groups of coefficients had no statistically significant difference in their feasibility.

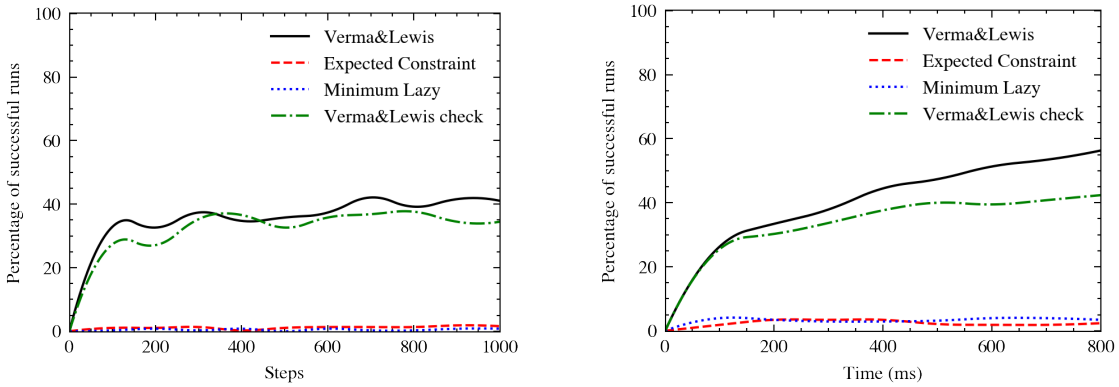
The situation is very different with MKP dataset, where RC coefficients sometimes produced solutions significantly worse compared to solutions achieved with VL, and EC and ML penalties almost always produced solutions that were significantly less feasible. Since solutions achieved using our coefficients were more feasible compared to VL only once with just one solver (Problem 3, MKP dataset, RC penalties, SA solver) and never again, every time there is a statistically significant difference in Figure 4.2, it is assumed that VL coefficients were better.

Nevertheless, even with MKP dataset, we could see that RC solutions in most cases had similar feasibility to VL. Therefore, we can conclude that VL likely overestimates penalty coefficients for most MKP problems by at least a factor of two. We can observe on Table 4.1 that the median penalty coefficient that RC estimates for MKP is 446.00, while the median coefficient that ML and EC estimate are both 1.00. We also see that penalties estimated by ML and EC are a lot smaller compared to RC penalties on Figure 4.1a. At the same time, the the solutions achieved by solvers using ML and EC coefficients are mostly infeasible (Figure 4.2a). Therefore, we conclude that VL overestimation is not as large as ML and EC predict with MKP dataset.

We also conclude that our penalty estimation algorithms could be effectively used to reduce the penalty coefficients in some problems. In others, they would produce solutions of lower feasibility. We have empirically found that they work well on QAPs and TSPs and not so well on MKPs. This might be due to the difference in the QUBO structure that different problems have, but it has to be investigated in future work. If it is possible to tell the difference between problems where VL will overestimate using the QUBO structure, a one for all algorithm could be made to reduce the lower bound of the penalty coefficients without affecting the feasibility of produced solutions.

4.4 Run Length and Time Distributions

In this section, we will present the effect that different penalty coefficients have on the performance of the solving algorithms. In particular, on the Run-Length Distribution (RLD) we look at how many steps it took the Simulated Annealing (SA) solver to reach feasible solutions, and on the Run-Time Distribution (RTD) we look at how many milliseconds it took the Tabu Search (TS) solver to, again, reach feasible solutions. It is important to mention that the green line in Figures 4.3, 4.4 and 4.5 that is called 'Verma&Lewis check' is referring to the RC estimator.



(a) Simulated Annealing Run-Length Distribution

(b) Tabu Search Run-Time Distribution

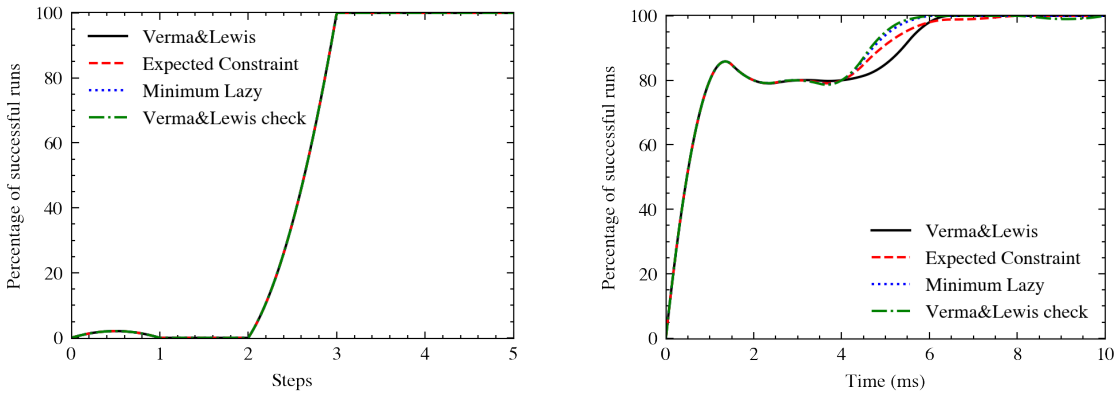
Figure 4.3: Run Distributions obtained using M coefficients of all algorithms with Multiple Knapsack Problem dataset.

The RLDs and RTDs for MKP dataset is presented in Figure 4.3. We can see that both EC and ML coefficients produced mostly infeasible solutions with both SA and TS. Moreover, their feasibility does not increase with more steps or time. This is likely because the penalty coefficients are underestimated. Hence the solutions break the constraints no matter how long they took to achieve.

On the other hand, VL and RC coefficients produced solutions that were in 35%-40% cases feasible for the majority of RLD (Figure 4.3a). The visible feasibility rise stopped after 200 steps. Although both distributions were similar, most of the time RC lagged. In RTD, however, both VL and RC coefficients produced more feasible solutions (Figure 4.3b), and more importantly, the feasibility of both curves was steadily increasing throughout the entire distribution. Which cannot be said about their feasibility in RLD, where it was not improving after 200 steps.

For the first 100 ms, the feasibility of VL and RC solutions is identical, but after that VL finds successful solutions faster. The more time increases, the larger the

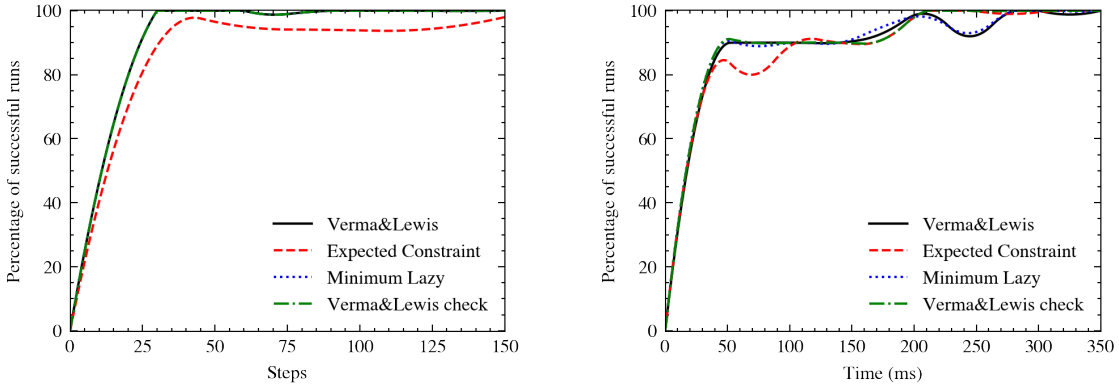
gap between them becomes. This might be because while RC estimates good penalty coefficients for most problems, it does underestimate coefficients for some of them. Thus, finding feasible solutions for them is either harder or not possible whatsoever.



(a) Simulated Annealing Run-Length Distribution (b) Tabu Search Run-Time Distribution

Figure 4.4: Run Distributions obtained using M coefficients of all algorithms with Quadratic Assignment Problem dataset.

The RLDs and RTDs for QAP dataset is presented in Figure 4.4. There (4.4a) we can see that the RLD of all coefficients is exactly the same. In RTD (4.4b), there is a minor difference in millisecond 5, but otherwise, the distribution is identical. That is interesting considering that in Figure 4.1b we can see that coefficients produced by EC algorithm are many times smaller than the rest. This, in turn, means that even though the penalty coefficients have decreased by a large factor, the speed of finding feasible solutions did not change.



(a) Simulated Annealing Run-Length Distribution (b) Tabu Search Run-Time Distribution

Figure 4.5: Run Distributions obtained using M coefficients of all algorithms with Travelling Salesman Problem dataset.

The RLDs and RTDs for TSP dataset is presented in Figure 4.5. With that dataset the VL, ML and RC have the same RLD (4.5a). It is logical that the distribution of ML and RC is equivalent with both QAP and TSP datasets because, as we have noticed in Section 4.2, the penalty coefficients generated by both of the algorithms are the same. It also makes sense why there could be little differences in RTD (4.5b): because even with the same coefficients, the progress made by the solver in given time depends on extraneous factors. Nevertheless, as we can see in both Figures 4.4 and 4.5, the difference, although it exists, is minimal. Therefore, we have done a good job of minimising their effects, something we have explained in more detail in Section 3.8.4.

The SA algorithm operating with penalty coefficients estimated by the EC consistently produces a smaller number of feasible solutions compared to the solutions produced using other coefficients (4.5a). Furthermore, even when the curve converges, the feasibility does not reach 100%, like others. We have also produced a RLD for 1000 steps, which can be found in the *Run Time-Length Distribution* notebook, but sub-perfect feasibility continued. We could conclude that this is because EC underestimates the penalty coefficients, as we have already seen in Figure 4.1c that it produces coefficients that are much lower than the coefficients by the other algorithms. However, if we look at the RTD (4.5b), we see that EC does reach perfect feasibility. Moreover, the RTD of all algorithms is comparable. Although there are some differences (EC is a little slower on step 50 but reaches 100% feasibility a little faster than VL and ML), they are minor, and the distribution is very similar in all algorithms. We conclude that the TS algorithm tolerates lower penalty coefficients better than does SA.

4.5 Conclusions

In Section 4.2, we have looked at the distribution of the penalty coefficients produced by different algorithms. We observed that ML and RC return the same penalties for the QAP and TSP datasets meaning that the *minimum expected constraint* that the ML calculates for those problem is always two, making the final result equal to the one produced by RC, where we just halve the VL coefficients. The ML, however, showed very different behaviour with MKP dataset, where it returned the smallest coefficients compared to other algorithms. Even smaller than the EC, which produced the smallest coefficients on all other datasets. Thus, it was empirically found that the penalty estimation algorithm behaviour differs greatly depending on the problem. At the same time, we saw that all the algorithms produce penalties that follow the same distribution pattern, with a dip in the middle of the curves meaning that the penalties that are neither small nor large are the rarest among all algorithms.

In Section 4.3, we have looked at how the feasibility differs with various penalty

coefficients and solvers across all datasets. We have found that VL never produces solutions that are significantly worse than the other penalties. In most cases, there was no statistically significant difference, but sometimes VL solutions were significantly more feasible. This is logical because even if the penalty coefficients are overestimated, given enough time, it should find solutions of comparable feasibility to the other algorithms. The VL solutions were almost always significantly better compared to the solutions of the other algorithms only on MKP dataset. The only exception is RC, which for most MKPs produced solutions of the same feasibility as with VL, meaning that it is likely that for most of the MKP problems VL algorithm overestimated the M by at least a factor of two.

A very different behaviour was observed with the QAP and TSP datasets, where penalty coefficients estimated by our algorithms always produced solutions with no significant difference in feasibility compared to the VL solutions. Considering that in Section 4.3 we have observed that the EC estimates coefficients a lot smaller compared to the other estimators, we can conclude that VL largely overestimates the $M$for QAP and TSP problems.

Then, in Section 4.4, we looked at RLD and RTD. Here we wanted to see if reducing the M will lead to feasible solutions being found faster. For MKP dataset, we have found that EC and ML are barely finding feasible results and RC is slower than the VL. This matches with the results we have seen in Section 4.3, where EC and ML solutions were significantly less feasible compared to the solutions produced using VL coefficients, while RC achieved solutions with no statistically significant difference to VL in majority of the problems. Therefore, we conclude that penalty coefficients returned by ML and EC for MKP dataset are undervalued. On the other hand, with the QAP dataset, all the penalties resulted in the same performance when used with SA algorithm and almost identical performance with TS.

With the TSP dataset, we saw that VL, RC and ML coefficients led to the same speed in SA algorithm, but EC coefficients resulted in feasible solutions being found a little slower and the feasibility never achieved 100%. In Section 4.3, we have, however, found that this difference in feasibility was not statistically significant with the maximum number of steps (1000). Using TS, all penalty coefficients produced feasible solutions at similar speeds.

Although it is outside of the scope of this project, it is interesting that the solutions found with TS were more likely to be feasible compared to SA algorithm, especially considering that they were given approximately the same amount of time to run. For example, in MKP, all algorithms achieved higher feasibility with TS. And in TSP, solutions achieved with SA using EC coefficients were not all feasible, but with TS, a perfect

feasibility was achieved by all coefficients.

Considering the reached results, we conclude that while the VL algorithm overestimates the M in most cases, it does not affect the speed of finding feasible solutions. While we have estimated penalty coefficients that were much smaller than the ones generated by VL, the feasibility of QAP and TSP solutions was not negatively affected. However, some of the MKP solutions became less feasible with smaller M. The reduced penalties, when resulting in no feasibility loss in produced solutions, appear to confer no performance advantages on the QUBO solving algorithms tested. We also suppose that the results we have achieved might be specific to the datasets, problems, or solver algorithms we have used.

Chapter 5

Conclusion

In this chapter, we analyse whether the requirements set at the beginning of the project were met (Appendix B). We also summarise the conclusions reached, propose future work and present a personal reflection on the project.

5.1 Requirements Testing

5.1.1 Software Functional Requirements

Must

- The solution must generate coefficients M to already formulated QUBO problems.
  The requirement was fully met.
  - The inputted QUBO should be split into the original objective function and the constraint function.
    The requirement was fully met.
  - It could generate the M to the equivalent Ising model.
    The requirement was refined as we decided to focus the project solely on QUBOs.
- At least one novel algorithm must be developed and implemented.
  The requirement was exceeded. We have developed and implemented two new algorithms: EC and ML.
  - The algorithm must generate M given the original objective function with the constraint function.
    The requirement was fully met.

  - It must consider the constraint function as this is the gap in the research identified in the literature review (2.6).
    The requirement was fully met.

- At least one existing algorithm must be implemented.
  The requirement was fully met.

  - This algorithm could be the *numerical 2* approach described in the literature review (2.5.3).
    The requirement was fully met.

Should

- It should be easy to specify the problem that needs M to be estimated.
  The requirement was fully met.

  - Two dictionaries that map quadratic variables to coefficients could represent the objective and constraint functions of QUBO.
    The requirement was refined. We used a 2D array to represent both objective and constraint QUBOs instead, as making dictionaries was unnecessary.

  - Two matrices could represent coefficients of objective and constraint functions of QUBO.
    The requirement was fully met.

- The solution should allow the user to choose the M estimation algorithm.
  The requirement was fully met.

- The software should work with a single constraint function.
  The requirement was fully met.

  - Although the developed algorithm could potentially work with multiple constraint functions, this is beyond this project's scope.
    The developed algorithms should theoretically work with multiple constraint functions, but we did not test this because it is beyond this project's scope.

    * The solution developed will only return a single M as only one constraint function is expected.
      The requirement was fully met.

  - Multiple constraint functions could be united into one. It could be inputted into the solution to generate an M.

This should theoretically work, but no tests were made as it is beyond this project's scope.

Will not

- The solution will not solve the QUBO.
  The requirement was fully met. We solve QUBOs as part of the project, but we use solving algorithms from external packages.
  - It only estimates the M.
    The requirement was fully met.
  - The M generated can then be used to solve the problem with QUBO solvers, including QA, DA, or other classical algorithms.
    The requirement was fully met.

5.1.2 Software Non-Functional Requirements

Must

- The software must be documented.
  The requirement was fully met.
  - The documentation must explain the purpose and functionality of implemented classes and methods.
    The requirement was fully met.
  - The documentation should follow NumPy docstring standard.
    The requirement was fully met.

Should

- The solution should be programmed in Python 3.
  The requirement was fully met. We used Python 3.10.
- The basic working version of the solution should be completed by Sunday, 16^{th} of January 2022.
  The requirement was fully met.

Could

- The software to be made could be a Python package.
  The requirement was fully met.

5.1.3 Experiment Functional Requirements

Must

- All implemented algorithms must be used for experiments.
  The requirement was fully met.
- The coefficients M generated by the algorithms must be used to solve QUBO problems.
  The requirement was fully met.
  - The solutions and the time required to reach them must be recorded for further analysis.
    The requirement was refined. Instead, we used more sophisticated methods for the speed analysis like the RLD and RTD.
  - For the same reason, the feasibility of the proposed solutions must be determined and recorded.
    The requirement was fully met.
- The QUBO problems must belong to two different CO problems and come from two distinct datasets.
  The requirement was exceeded. We have used three CO problems coming from three datasets.
  - Datasets must be publicly available.
    The requirement was fully met.
  - One of the datasets could include the Travelling Salesman Problems.
    The requirement was fully met.
    * Then the formulated QUBOs for problem instances could be provided by industry advisor, Dr Ayodele.
      The requirement was fully met.
  - Another dataset could include Quadratic Assignment Problems.
    The requirement was fully met.
    * Then the formulated QUBOs for problem instances could be provided by industry advisor, Dr Ayodele.
      The requirement was fully met.
  - Another dataset could include Multidimensional Knapsack problems.
    The requirement was fully met.

    * Then the formulated QUBOs for problem instances could be provided by industry advisor, Dr Ayodele.
      The requirement was fully met.

  – Another dataset could include generalised assignment problems.
    The requirement was refined as we have exceeded the number of required datasets.

  – Another dataset could include bin packing problems.
    The requirement was refined as we have exceeded the number of required datasets.

Should

- The QUBOs with generated coefficients M should be solved on a classical computer using QUBO solvers defined in the literature review (2.4).
  The requirement was partly met. We have used three out of four described solvers.

Could

- An cluster provided by National Subsea Centre could be used to run the classical computer experiments faster.
  The requirement was refined as our machine was powerful enough.

- A Fujitsu DA provided by industry advisor, Dr Ayodele, could be used to run the experiments.
  The requirement was refined as it was expected that DA would produce results similar to SA, which we have used, because it is based on it.

5.1.4 Experiment Non-Functional Requirements

Must

- If QUBOs for the chosen datasets are not available, they must be formulated manually.
  The requirement was not relevant because the datasets provided by Dr Ayodele were in QUBO formulation already.

  – The QUBOs produced should have a single constraint function as the implemented software will only generate a single M. Multiple constraint functions, however, could be joined into one by summation.
    The requirement was not relevant for the same reason.

- Rigorous statistical testing and analysis of results must be performed to compare the implemented M estimation algorithms.
  The requirement was fully met.
  - The results should be tested for significance.
    The requirement was fully met. We have used Student's t-test.
- Visualisations demonstrating the results of experiments should be produced.
  The requirement was fully met.
  - The plots could be made for every combination of datasets, QUBO solvers and M algorithms.
    The requirement was fully met.
  - A series of bar plots could be produced that compare the proportion of feasible and infeasible QUBO solutions obtained using different M estimation algorithms.
    We have produced bar plots, but we instead compare whether there was a statistically significant difference in the feasibility of our algorithms compared to VL.
  - The RTD of QUBO solving could be visualised to see how coefficients M generated by different algorithms affect the run time.
    The requirement was exceeded. We visualised RTD for TS and RLD for SA.

Should

- All experimental hypotheses should be tested until the mid-week of February.
  The requirement was not met. The set deadline was too optimistic.
- The experiment analysis should be completed by March, including publication-quality visualisations.
  The requirement was not met. The set deadline was too optimistic.

Could

- The experiments could be performed using Jupyter Notebooks.
  The requirement was fully met.

5.2 Conclusions & Future Work

At the beginning of our project, we have studied what QUBO is, how to solve problems in such formulation, how M arises and why it is essential to find optimal M. We reviewed

state-of-the art M estimation algorithms, including VL. It is a robust algorithm that provides a guarantee that the generated penalty coefficient is not undervalued. However, it may overestimate the M because it does not consider the constraint function for the calculation. Overestimating the penalty coefficient impedes the problem-solving process. Thus, we explored ways to utilise the knowledge of the constraint function to reduce the penalty coefficient without harming the feasibility of the solutions. We designed two new M estimation algorithms based on VL. Also, we implemented an algorithm that would always return half of the VL prediction to check if the penalties VL produces are overestimated by at least a factor of two. We then used the implemented algorithms to generate penalty coefficients for problems from three different datasets. We solved the QUBOs with all of the generated coefficients with three QUBO solving algorithms. The solutions obtained were used to test whether the different penalty coefficients resulted in solutions of varying feasibility. We have also plotted RLD and RTD for all datasets to see if the speed of finding feasible solutions changed with different coefficients. There was no statistically significant difference in the feasibility of all the algorithms compared to VL across all solvers used on QAP and TSP datasets, meaning that the VL algorithm greatly overestimated the coefficients. However, with the MKP dataset, only the algorithm that halved the VL coefficient achieved solutions of comparable feasibility to the VL in the majority of the problems. This means that the EC and ML algorithms underestimated the M, which led to worse feasibility, but the VL still overestimated the penalty coefficient by a factor of two in the majority of the MKP problems. We then looked at how the reduction of the penalty coefficients impacted the speed of finding feasible solutions. We found that even in the cases where VL greatly overestimated compared to our algorithms, there was no visible performance advantage of using lower coefficients. Therefore, we have concluded that even though the VL returns penalty coefficients that are too large, there is no advantage to finding a more precise M, at least when it comes to finding feasible solutions faster.

Future work could include testing if the conclusion holds for other solvers, especially QA. It is easy to do if such device is available, as we could replace the solver in the Jupyter Notebooks and run them. The results we achieved are specific to problems and datasets that we have used in our project. It is worth testing what effect reducing the penalty coefficients for QUBOs of other CO problems or datasets will have on both the feasibility of solutions and the performance of the solvers. It is particularly interesting if reducing the penalties for other problems will improve the performance of the QUBO solving algorithms or if the performance achieved with VL coefficients will always be optimal. Thus, testing other problems and datasets in future would be helpful. Moreover, even though reducing the penalty coefficients did not make finding feasible solutions faster, it would be interesting to know if the feasible solutions found with lower coefficients are

better in terms of energy. One could compare the objective function values of feasible solutions achieved using different penalty coefficients to investigate this.

5.3 Reflection

This project was a great chance to learn more about QUBOs, optimisation algorithms, adiabatic computing and solving niche problems in general. Moreover, the research experience we have obtained throughout this work is invaluable. It was beneficial that we carried out and documented an extensive Literature Review early on, allowing us to be more creative later on. If we were to do the project again, a posteriori, we would not change the methods and the approaches we have used. Surprisingly, the project went according to the plan. This is primarily due to the wise and professional supervision, which prevented us from falling into pitfalls. We would instead change some of the deadlines we have set for ourselves in the *Requirements Engineering* stage as they were too optimistic. The project involved a lot of research and thinking out of the box. For example, the D-Wave solvers did not provide enough information to produce RLD and RTD. Thus, we had to design a non-conventional way to get the required data. This was an exploration rather than a set of tasks, which we did not foresee at the project beginning. Thus, more realistic deadlines should have been set. At the end of the project, we have reached an exciting and insightful conclusion (unexpected too), which is the best possible culmination for this dissertation.

Chapter 6

Summary

In Chapter 1, we explained the background (1.1) and motivation of the project (1.2). We set out objectives (1.3), described the key techniques (1.4) and looked at possible legal, social, ethical, professional and security issues that could arise (1.5).

In Chapter 2, we presented an overview of annealing algorithms (2.2), discussed QUBO, where the penalty coefficient comes from and why it is important to have it's optimal value (2.3). We reviewed algorithms for solving QUBOs (2.4) and estimating penalty coefficients (2.5).

In Chapter 3, we discussed the solution design and implementation. In particular, we presented our setup (3.2), the datasets we have used (3.3), the experiment and analysis that we carried out in Jupyter Notebooks (3.4), the Python package containing our software implementation (3.5), the implemented penalty estimation algorithms (3.6), discussed more thoroughly the experiments (3.7) and the analysis (3.8).

In Chapter 4, we presented and discussed the results: how penalty coefficients estimated by different algorithms compare (4.2), how the feasibility of solutions produced using VL penalty coefficients with solutions produced using EC, ML and RC penalty coefficients compare (4.3), and what effect different penalty coefficients have on the performance of the solving algorithms (4.4).

In Chapter 5, we have tested if we have met the requirements that we set out at the beginning of the project (5.1), formulated conclusions and future work (5.2), and reflected on the project (5.3).

References

Alkhamis, T. M., Hasan, M. & Ahmed, M. A. (1998), 'Simulated annealing for the unconstrained quadratic pseudo-Boolean function', *European Journal of Operational Research* **108**(3), 641–652.

Anaconda (n.d.), 'Anaconda'.
URL: *https://www.anaconda.com/*

Aramon, M., Rosenberg, G., Valiante, E., Miyazawa, T., Tamura, H. & Katzgraber, H. G. (2019), 'Physics-Inspired Optimization for Quadratic Unconstrained Problems Using a Digital Annealer', *Frontiers in Physics* **0**(APR), 48.

Beasley, J. (1998*a*), 'Heuristic algorithms for the unconstrained binary quadratic programming problem', *undefined* .

Beasley, J. (1998*b*), 'Multiple Knapsack Problem Dataset'.
URL: *http://people.brunel.ac.uk/∼mastjjb/jeb/orlib/mknapinfo.html*

Bertsimas, D. & Tsitsiklis, J. (1993), 'Simulated Annealing', *Statistical Science* **8**(1), 10–15.

Booth, M., Reinhardt, S. P. & Roy, A. (2017), 'Partitioning Optimization Problems for Hybrid Classical / Quantum Execution', *D-Wave Technical Report Series* **14-1006A-A**.
URL: *www.dwavesys.com*

Boyd, J. (2018), 'Fujitsu's CMOS Digital Annealer Produces Quantum Computer Speeds', *IEEE Spectrum* .
URL: *https://spectrum.ieee.org/ieee-courses-on-digital-transformation*

Brownlee, A. E. I. (2009), 'Multivariate Markov networks for fitness modelling in an estimation of distribution algorithm.'.
URL: *https://rgu-repository.worktribe.com/output/247858/multivariate-markov-networks-for-fitness-modelling-in-an-estimation-of-distribution-algorithm https://rgu-repository.worktribe.com/output/247858/multivariate-markov-networks-for-fitness-modelling-in-an-estimation-of-distribution-algorithm.abstract*

Burkard, R., Cela, E., Karisch, S. & Rendl, F. (2012), 'QAPLIB - Problem Instances and Solutions'.
URL: *https://coral.ise.lehigh.edu/data-sets/qaplib/qaplib-problem-instances-and-solutions/*

Collins, E. F. (1921), 'ELECTRICALLY HEATED GLASS ANNEALING LEHR1', *Journal of the American Ceramic Society* **4**(5), 335–349.
URL: *https://onlinelibrary.wiley.com/doi/full/10.1111/j.1151-2916.1921.tb18664.x*

Crosson, E. & Harrow, A. W. (2016), 'Simulated Quantum Annealing Can Be Exponentially Faster Than Classical Simulated Annealing', *Proceedings - Annual IEEE Symposium on Foundations of Computer Science, FOCS* **2016-Decem**, 714–723.

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 2019 9:1* **9**(1), 1–12.
URL: *https://www.nature.com/articles/s41598-019-53585-5*

D-Wave Systems (2021), 'Volkswagen: Navigating Tough Automotive Tasks with Quantum Computing'.
URL: *www.dwavesys.com/d-wave-launch*

Dattani, N., Szalay, S. & Chancellor, N. (2019), 'Pegasus: The second connectivity graph for large-scale quantum annealing hardware'.

dwave-greedy (2019).
URL: *https://github.com/dwavesystems/dwave-greedy*

dwave-neal (2015).
URL: *https://github.com/dwavesystems/dwave-neal*

dwave-tabu (2018).
URL: *https://github.com/dwavesystems/dwave-tabu*

Farhi, E., Goldstone, J., Gutmann, S., Lapan, J., Lundgren, A. & Preda, D. (2001), 'A Quantum Adiabatic Evolution Algorithm Applied to Random Instances of an NP-Complete Problem', *Science* **292**(5516), 472–476.
URL: *https://www.science.org/doi/abs/10.1126/science.1057726*

FOLDOC (2013), 'Systems Development Life Cycle'.
URL: *http://foldoc.org/Systems+Development+Life+Cycle*

Fujitsu (n.d.), 'Reducing traveling distance for parts picking operations by up to 45%'.
URL: *https://www.fujitsu.com/global/services/business-services/digital-annealer/case-studies/201804-fjit.html*

GitHub (n.d.), 'Git Large File Storage'.
URL: *https://git-lfs.github.com/*

Glover, F., Kochenberger, G. & Du, Y. (2019), 'Quantum Bridge Analytics I: a tutorial on formulating and using QUBO models', *4OR* **17**(4), 335–371.

Higham, C. F. & Bedford, A. (2021), 'Quantum Deep Learning: Sampling Neural Nets with a Quantum Annealer'.
URL: *https://arxiv.org/abs/2107.08710v1*

Huang, T., Goh, S. T., Gopalakrishnan, S., Luo, T., Li, Q. & Lau, H. C. (2021), 'QROSS: QUBO Relaxation Parameter optimisation via Learning Solver Surrogates', *2021 IEEE 41st International Conference on Distributed Computing Systems Workshops (ICDCSW)* pp. 35–40.
URL: *https://ieeexplore.ieee.org/document/9545928/*

Johnson, D. S., Aragon, C. R., Mcgeoch, L. A. & Schevon, C. (1989), 'Optimization by Simulated Annealing: An Experimental Evaluation; Part I, Graph Partitioning', **37**(6).

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. & Rose, G. (2011), 'Quantum annealing with manufactured spins', *Nature 2011 473:7346* **473**(7346), 194–198.
URL: *https://www.nature.com/articles/nature10012*

Kadowaki, T. & Nishimori, H. (1998), 'Quantum annealing in the transverse Ising model', *Physical Review E - Statistical Physics, Plasmas, Fluids, and Related Interdisciplinary Topics* **58**(5), 5355–5363.

Katzgraber, H. G. & Novotny, M. A. (2018), 'How Small-World Interactions Can Lead to Improved Quantum Annealer Designs', *Physical Review Applied* **10**(5), 054004.
URL: *https://journals.aps.org/prapplied/abstract/10.1103/PhysRevApplied.10.054004*

Kirkpatrick, S., Gelatt, C. D. & Vecchi, M. P. (1983), 'Optimization by Simulated Annealing', *New Series* **220**(4598), 671–680.

Kochenberger, G. A. & Glover, F. (2006), 'A Unified Framework for Modeling and Solving Combinatorial Optimization Problems: A Tutorial', *Multiscale Optimization Methods and Applications* pp. 101–124.
URL: *https://link.springer.com/chapter/10.1007/0-387-29550-X_4*

Kochenberger, G., Hao, J. K., Glover, F., Lewis, M., Lü, Z., Wang, H. & Wang, Y. (2014), 'The unconstrained binary quadratic programming problem: A survey', *Journal of Combinatorial Optimization* **28**(1), 58–81.
URL: *https://link.springer.com/article/10.1007/s10878-014-9734-0*

Lewis, M. & Glover, F. (2017), 'Quadratic Unconstrained Binary Optimization Problem Preprocessing: Theory and Empirical Analysis', *Networks* **70**(2), 79–97.
URL: *https://arxiv.org/abs/1705.09844v1*

Liang, F., Cheng, Y. & Lin, G. (2014), 'Simulated Stochastic Approximation Annealing for Global Optimization With a Square-Root Cooling Schedule', *https://doi.org/10.1080/01621459.2013.872993* **109**(506), 847–863.
URL: *https://www.tandfonline.com/doi/abs/10.1080/01621459.2013.872993*

Lucas, A. (2014), 'Ising formulations of many NP problems', *Frontiers in Physics* **0**, 5.

Ohzeki, M., Miki, A., Miyama, M. J. & Terabe, M. (2019), 'Control of Automated Guided Vehicles Without Collision by Quantum Annealer and Digital Devices', *Frontiers in Computer Science* **1**, 9.

Ohzeki, M. & Nishimori, H. (n.d.), 'Quantum annealing: An introduction and new developments', *Journal of Computational and Theoretical Nanoscience* **8**(6), 963–971.
URL: *https://tohoku.pure.elsevier.com/en/publications/quantum-annealing-an-introduction-and-new-developments*

Palubeckis, G. (2004), 'Multistart Tabu Search Strategies for the Unconstrained Binary Quadratic Optimization Problem', *Annals of Operations Research 2004 131:1* **131**(1), 259–282.
URL: *https://link.springer.com/article/10.1023/B:ANOR.0000039522.58036.68*

Perkel, J. M. (2018), 'Why Jupyter is data scientists' computational notebook of choice', *Nature* **563**(7729), 145–146.

Poole, D. L. & Mackworth, A. K. (2017), 'Artificial Intelligence: Foundations of Computational Agents', pp. 136–137.
URL: *https://www.cambridge.org/core/product/identifier/9781108164085/type/book*

qbsolv (2017).
URL: *https://github.com/dwavesystems/qbsolv*

Seker, O., Tanoumand, N. & Bodur, M. (2020), 'Digital Annealer for quadratic unconstrained binary optimization: a comparative performance analysis'.
URL: *http://arxiv.org/abs/2012.12264*

Skorobohatyj, G. (1995), 'The TSPLIB Symmetric Traveling Salesman Problem Instances'.
URL: *http://elib.zib.de/pub/mp-testdata/tsp/tsplib/tsp/index.html*

Snelling, D., Shahane, G., Shipman, W., Balaeff, A., Pearce, M. & Keinan, S. (2020), 'A Quantum-Inspired Approach to De-Novo Drug Design'.
URL: *https://chemrxiv.org/engage/chemrxiv/article-details/60c74b47bb8c1a1ac53db18d*

Style guide — numpydoc (n.d.).
URL: *https://numpydoc.readthedocs.io/en/latest/format.html*

Tanaka, S. & Tamura, R. (2012), 'QUANTUM ANNEALING AND QUANTUM FLUCTUATION EFFECT IN FRUSTRATED ISING SYSTEMS'.

Turkey, J. (1953), 'The problem of multiple comparisons'.

Verma, A. & Lewis, M. (2020), 'Penalty and partitioning techniques to improve performance of QUBO solvers', *Discrete Optimization* p. 100594.

Vyskočil, T., Pakin, S. & Djidjev, H. N. (2019), 'Embedding Inequality Constraints for Quantum Annealing Optimization', *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* **11413 LNCS**, 11–22.
URL: *https://link.springer.com/chapter/10.1007/978-3-030-14082-3_2*

Wu, H. & Fan, G. (2020), 'An overview of tailoring strain delocalization for strength-ductility synergy', *Progress in Materials Science* **113**, 100675.

Zaman, M. (2021), 'PyQUBO: Python Library for Mapping Combinatorial Optimization Problems to QUBO Form.', *CoRR* **abs/2103.01708**.
URL: *https://arxiv.org/abs/2103.01708*

Appendix A

Project Plan

Objective 1 Create a new method of estimating penalty coefficients for QUBO problems and compare it to an already existing method.

- Study QUBO model and its applications.
- Study algorithms used for solving QUBOs.
- Study techniques used for formulating QUBOs. Study relevant programming libraries.

Objective 2 Study the existing methods of estimating M and critically evaluate them.

- Study the existing techniques for estimating M.
- Evaluate them.

Objective 3 Implement one of the studied methods.

- Choose one M estimation technique.
- Implement it.

Objective 4 Propose and implement a new method for estimating M.

- Propose a new method for estimating M.
- Explain the advantages of the proposed method over the existing methods.
- Implement it.

Objective 5 Formulate QUBOs for instances of a single CO problem and run optimisation algorithms with M coefficients generated by an existing method (implemented

in *Objective 3*) and by a new method (implemented in *Objective 4*).

- Choose a CO problem.
- Find a dataset containing instances of the chosen problem and solutions for them.
- Formulate QUBOs for those instances.
- Estimate M coefficients for those QUBOs using the existing method implemented in *Objective 3*.
- Estimate M coefficients for the same QUBOs using the proposed method.
- Solve QUBOs with the DA's algorithm on a standard computer with all the generated M coefficients.
- Solve QUBOs with the Digital Annealing algorithm on Fujitsu DA with all the generated M coefficients.
- Choose another classical algorithm and solve QUBOs using it on a standard computer with all the generated M coefficients.

Objective 6 Compare the quality of the produced solutions. Disseminate.

- Make graphs to visualise the difference in the quality of the solutions produced by different algorithms and different M estimation methods.
- Make comparisons and discuss the observed differences or similarities.
- Disseminate.

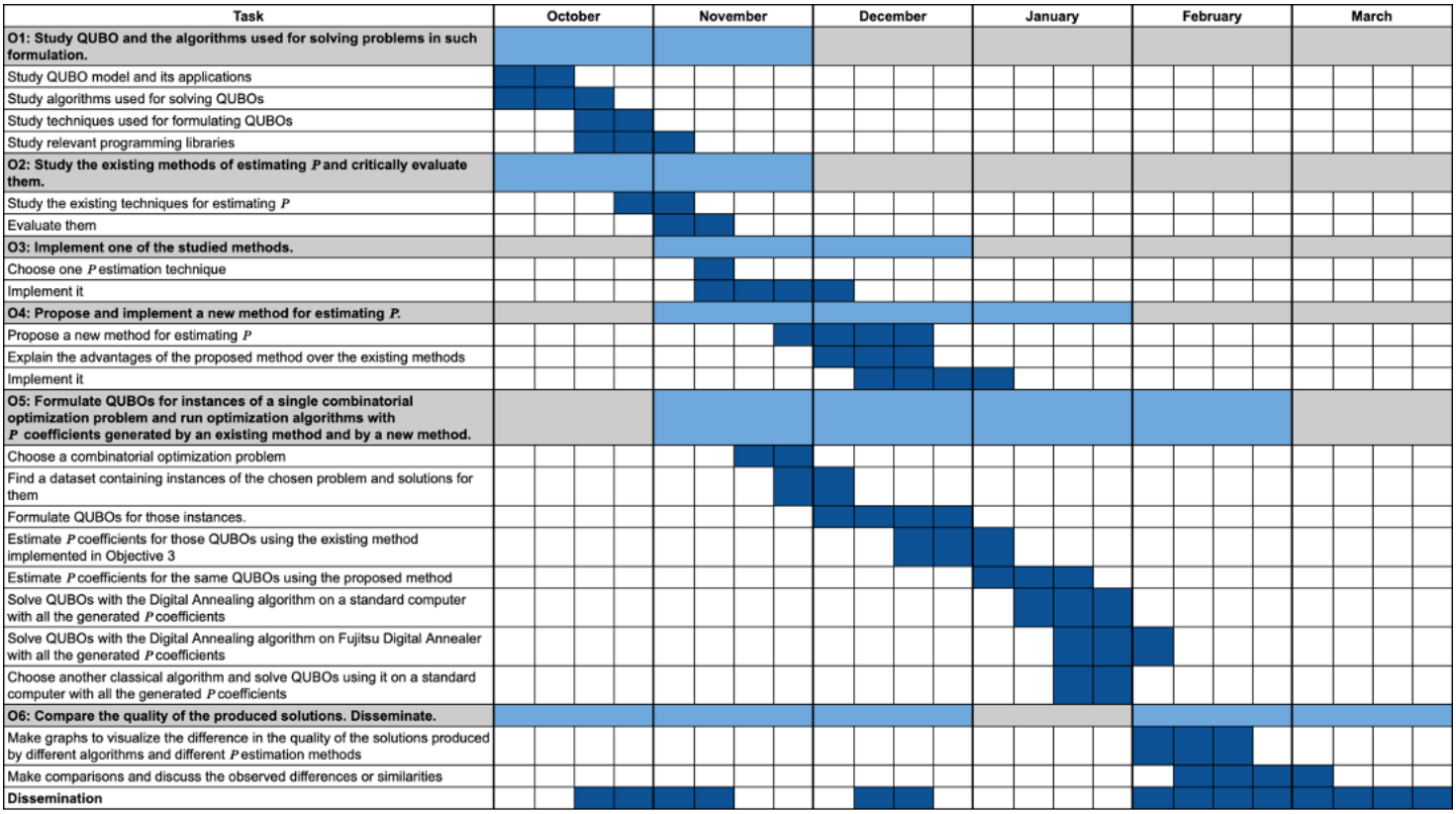


Figure A.1: Initial project schedule.

Appendix B

Project Specification

To fulfil the project's aim, a software solution that will estimate the coefficient M of QUBO problems using both novel and existing algorithms must be made. This solution should then be used to conduct experiments to find out how the algorithms compare. As the project is split into two parts, software development and experiments, the requirements will consist of two sections. Each of the sections will be further split into functional and non-functional requirements. The requirements are defined using the MoSCoW methodology. This methodology divides the requirements into four groups of priorities (high to low): *Must Have*, *Should Have*, *Could Have* and *Will Not Have*. Moreover, every requirement could have multiple minor requirements with varying priorities that need to be completed to fulfil the significant requirement. While additional subheadings are not used for them, they are also ordered according to the MoSCoW methodology.

B.1 Software

B.1.1 Functional Requirements

Must

- The solution must generate coefficients M to already formulated QUBO problems.
  - The inputted QUBO should be split into the original objective function and the constraint function.
  - It could generate the M to the equivalent Ising model.
- At least one novel algorithm must be developed and implemented.
  - The algorithm must generate M given the original objective function with the

constraint function.

  - It must consider the constraint function as this is the gap in the research identified in the literature review (2.6).

- At least one existing algorithm must be implemented.

  - This algorithm could be the *numerical 2* approach described in the literature review (2.5.3).

Should

- It should be easy to specify the problem that needs M to be estimated.

  - Two dictionaries that map quadratic variables to coefficients could represent the objective and constraint functions of QUBO.

  - Two matrices could represent coefficients of objective and constraint functions of QUBO.

- The solution should allow the user to choose the M estimation algorithm.

- The software should work with a single constraint function.

  - Although the developed algorithm could potentially work with multiple constraint functions, this is beyond this project's scope.

    * The solution developed will only return a single M as only one constraint function is expected.

  - Multiple constraint functions could be united into one. It could be inputted into the solution to generate an M.

    * Then the generated M would apply to all the original constraint functions.

Will not

- The solution will not solve the QUBO.

  - It only estimates the M.

  - The M generated can then be used to solve the problem with QUBO solvers, including QA, DA, or other classical algorithms.

B.1.2 Non-Functional Requirements

Must

- The software must be documented.
  - The documentation must explain the purpose and functionality of implemented classes and methods.
  - The documentation should follow NumPy docstring standard.

Should

- The solution should be programmed in Python 3.
  - Because many QUBO-related libraries listed in the literature review are available in Python 3.
- The basic working version of the solution should be completed by Sunday, 16^{th} of January 2022.
  - As the proof-of-concept demonstration will happen in the following week.
  - The basic version of the software must be completed before the experiments can take place. Having the basic version by the set deadline will leave enough time for thorough experimentation and improvements.

Could

- The software to be made could be a Python package.
  - As QUBO solving libraries are available in Python, generating the M in the same language is convenient. Then a single loop can be used to define QUBO, generate M, redefine QUBO with new coefficients and solve it. This will make experiments automatable, and the M estimation algorithms could be tested on entire datasets efficiently.
  - The produced module can easily be imported into new and existing projects using the import keyword, given that the module folder is copied into the project.

B.2 Experiments

B.2.1 Functional Requirements

Must

- All implemented algorithms must be used for experiments.

  - Including the algorithms that were implemented from the literature and self-proposed algorithms.
  - As the aim of the project is to compare the algorithms.
- The coefficients M generated by the algorithms must be used to solve QUBO problems.
  - The solutions and the time required to reach them must be recorded for further analysis.
  - For the same reason, the feasibility of the proposed solutions must be determined and recorded.
- The QUBO problems must belong to two different CO problems and come from two distinct datasets.
  - Datasets must be publicly available.
    * To make the findings reproducible.
    * And to avoid ethical problems when running experiments externally.
  - One of the datasets could include the Travelling Salesman Problems.
    * Then the formulated QUBOs for problem instances could be provided by industry advisor, Dr Ayodele.
  - Another dataset could include Quadratic Assignment Problems.
    * Then the formulated QUBOs for problem instances could be provided by industry advisor, Dr Ayodele.
  - Another dataset could include Multidimensional Knapsack problems.
    * Then the formulated QUBOs for problem instances could be provided by industry advisor, Dr Ayodele.
  - Another dataset could include generalised assignment problems.
  - Another dataset could include bin packing problems.

Should

- The QUBOs with generated coefficients M should be solved on a classical computer using QUBO solvers defined in the literature review (2.4).

Could

- An cluster provided by National Subsea Centre could be used to run the classical computer experiments faster.
- A Fujitsu DA provided by industry advisor, Dr Ayodele, could be used to run the experiments.

B.2.2 Non-Functional Requirements

Must

- If QUBOs for the chosen datasets are not available, they must be formulated manually.
  - The QUBOs produced should have a single constraint function as the implemented software will only generate a single M. Multiple constraint functions, however, could be joined into one by summation.
- Rigorous statistical testing and analysis of results must be performed to compare the implemented M estimation algorithms.
  - The results should be tested for significance.
- Visualisations demonstrating the results of experiments should be produced.
  - The plots could be made for every combination of datasets, QUBO solvers and M algorithms.
  - A series of bar plots could be produced that compare the proportion of feasible and infeasible QUBO solutions obtained using different M estimation algorithms.
  - The run time distributions of QUBO solving could be visualised to see how coefficients M generated by different algorithms affect the run time.

Should

- All experimental hypotheses should be tested until the mid-week of February.
- The experiment analysis should be completed by March, including publication-quality visualisations.

Could

- The experiments could be performed using Jupyter Notebooks.
  - As Jupyter Notebooks are easy to follow and eloquent.

– As the experiments could be reproduced by rerunning the notebooks.

Appendix C

Project Log

C.1 Week 1

C.1.1 Objectives

- Learn the basics of Simulated Annealing.

C.1.2 Tasks

- ~~Understand Simulated Annealing Explained By Solving Sudoku - Machine Learning.~~
- ~~Understand Simulated Annealing.~~
- ~~Understand QUBO on D-Wave Chimera Graph: 1, 2, 3.~~

C.1.3 Meeting (02/09/2021)

- NSC tour.
- Discussed the possible project.
- Discussed Simulated Annealing.
- Need to read The Metropolis–Hastings algorithm and Optimization by Simulated Annealing.
- Will have to explain Simulated Annealing at the next meeting.

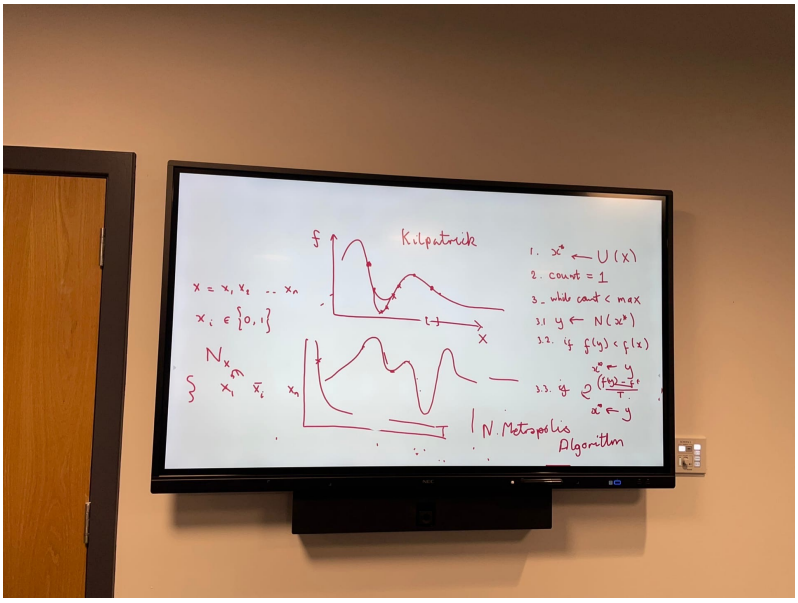


Figure C.1: Week 1 meeting notes.

C.2 Week 2

C.2.1 Objectives

- Become a master of Simulated Annealing.
- Make a lecture on Simulated Annealing to present during the meeting.

C.2.2 Tasks

- ~~Watch Optimization - I (Simulated Annealing) lecture.~~
- ~~Watch Markov chain Monte Carlo (MCMC) introduction.~~
- ~~Read A Short History of Markov Chain Monte Carlo; Subjective Recollections from Incomplete Data.~~
- ~~Watch Markov Chains Clearly Explained!~~
- ~~Watch An introduction to the Random Walk Metropolis algorithm.~~
- ~~Read Optimization by Simulated Annealing.~~
- ~~Read The Metropolis–Hastings algorithm.~~
- ~~Understand What is the difference between Simulated Annealing and Monte-Carlo Simulations?~~

- ~~Read Markov Chain Monte Carlo & Simulated Annealing.~~
- ~~Make lecture notes explaining Simulated Annealing in details.~~
- ~~Make plots or the presentation with matplotlib.~~
- ~~Research how to make plots look scientific (SciencePlots).~~
- ~~Make a PowerPoint that would align with lecture notes.~~

C.2.3 Meeting (09/09/2021)

- Should I learn about Quantum Annealing before Digital Annealer as it is inspired by it?
  - Yes, if I struggle with Digital Annealer, but it is not urgent.
- Present the Simulated Annealing lecture.

C.3 Week 3

C.3.1 Objectives

- Get more familiar with the Digital Annealer.

C.3.2 Tasks

- ~~Watch Digital Annealer technology.~~
- ~~Watch Fujitsu Forum 2019 Keynote - From mathematical to industrial optimization.~~
- ~~Read IEEE SPECTRUM: Digital Annealer.~~
- ~~Skim through Quantum Deep Learning; Sampling Neural Nets with a Quantum Annealer.~~
- ~~Watch Quantum Computing by a Quantum Annealer.~~
- ~~Go through this series of articles again: 1, 2, 3.~~

C.3.3 Meeting (16/09/2021)

- Ask about the benefits of using the sigmoid function for Simulated Annealing.
- How do superconductors work with QA?
- What is the architecture of DA?

  - Hardware of DA is not very important as it is outside of the project scope.

- Is there a difference between DA and QA with respect to this project? We use QUBO and penalty values are universal. As far as I understand, it is just a different technology to do the same thing and optimising penalties benefits both in the same way.
- Is there a difference between DA/QA and SA? Is the penalty optimisation somehow unique to the quantum tunnelling approach?
  - Penalty optimisation will most likely increase the effectiveness of both DA and QA, but we only have access to DA.
- D-Wave Ocean Software Documentation - is there something similar to this for DA?
- Read the EDA lecture sent.
- Read the papers sent.

C.4 Week 4

C.4.1 Objectives

- Prepare for a meeting with Dr Ayodele.
- Get as familiar as possible with pre-project ideas.

C.4.2 Tasks

- ~~Go through Pre-Project Meeting Notes notes and write out everything I do not understand.~~
- ~~Find papers that are related to QUBO penalty optimisation.~~
- ~~Read sections related to penalty values in A Tutorial on Formulating and Using QUBO Models.~~
- ~~Read the *EDA's and Markov Networks* lecture sent on Teams.~~
- ~~Understand PyQUBO.~~
- ~~Understand qbsolv.~~
- ~~Read up on maximum satisfiability problem (MAX-SAT).~~
- ~~Go through the ethics form and find controversial points.~~

- ~~Read up on knapsack problem and it's QUBO representation from A Tutorial on Formulating and Using QUBO Models.~~
- ~~Read up on general assignment problem.~~
- ~~Make a list of questions for Dr Ayodele and Professor McCall on things I still do not understand from their meeting notes.~~

C.4.3 Meeting (23/09/2021)

Today's meeting is with Dr Ayodele, industry advisor.

- Can a process of evaluating sub-patterns be used to set tighter penalty bounds to improve the QUBO solution process?
  - Are we going to have penalty bounds, not a single penalty value? So the minimum value will be increasing to the maximum value with a certain... reduction rate? during the annealing to find the best answer.
    - Penalties cannot be changed during the annealing. They must be fixed.
  - In Glover's paper it says that you can have multiple penalties. I assume the scope of this project is just a single global penalty.
    - We might have multiple penalties if there is more than one constraint function, but this is unlikely.
- Use problem structure / probing techniques to extract variable interaction structure.
  - Problem structure?
  - Probing techniques?
  - Variable interaction structure?
  - Has this been done before?
- Use partial evaluation techniques to find major variable interaction components in untransformed problems and estimate the effect on overall fitness. Use this to estimate penalty ranges to achieve a tighter bound.
  - Partial evaluation techniques?
  - Variable interaction components?
  - Untransformed problem: not in form of QUBO?
    - We can try to optimise for P not in the form of QUBO.

  - Do we do that in a simulation using random walk?

- Use PyQUBO, qbsolve, for transformation but amend penalty using estimation process.
  - Is there something similar to D-Wave Ocean for DA?
    * There is no API for Digital Annealer, but Dr Ayodele can manually run it.
    * It is very easy to code the Digital Annealer algorithm or an ordinary computer as it has minor differences from Simulated Annealing in terms of implementation.

- Create own solver: GA, DEUM...
  - Our solver will have the same QUBO format and get the same input, but it will not have any penalties?
  - qbsolve allows Tabu Search as well. Maybe try that too?

- Run experiments to compare solution time for own solvers vs transformation + DA.
  - Is it also meaningful to compare generated penalties vs standard penalties, which is 10?
    * We may test our solution against some default P-value or try to optimise P with methods done by other people to compare the performance. It is not meaningful to compare Digital Annealer with Genetic Algorithm as one is in the form of QUBO and the second is not.

- Ask about ethics form: 1c, 2, 3d, 5c, 10.
  - 1c - yes, 2 - no, 3d - yes, 5c - yes, 10 - maybe (no).

- Ask about relevant papers.

- For penalty coefficient optimisation implementation, look at the harmonic analysis and Walsh function.

- Make a draft of the project proposal.

C.5 Week 5

C.5.1 Objectives

- Make a project log.

- Make a project proposal draft.
- Have a look at the harmonic analysis and Walsh function to get ideas for penalty optimisation.

C.5.2 Tasks

- ~~Research what project logs can be like.~~
- ~~Make a template of my project log.~~
- ~~Make a full project log and fill it in.~~
- ~~Skim through Towards Prediction of Financial Crashes with a D-Wave Quantum Computer.~~
- ~~Look at harmonic analysis and Walsh function.~~
- ~~Understand the difference between the Ising and QUBO models. Difference between BQM, Ising, and QUBO problems?~~
- ~~Read Penalty and partitioning techniques to improve performance of QUBO solvers.~~
- ~~Research variable interaction components.~~
- ~~Read QUBO RELAXATION PARAMETER OPTIMISATION VIA LEARNING SOLVER SURROGATES.~~
- ~~Read about SA/DA algorithm difference from Physics-Inspired Optimization for Quadratic Unconstrained Problems Using a Digital Annealer.~~
- ~~Found a TSP dataset: TSPLIB – A Traveling Salesman Problem Library.~~
- ~~Make an initial project proposal draft.~~
- Read Embedding Inequality Constraints for Quantum Annealing Optimization.
- Create a repository for dissertation materials.

C.5.3 Meeting (30/09/2021)

- Discussed progress.
- Discussed the proposal draft. Feedback:
  - Write references.
  - Make it look professional.

  - Add Dr Ayodele as an industry advisor.
  - Elaborate on Objective 6 and add dissemination.
- Discussed possible penalty coefficient estimation algorithms.
- Read Partial structure learning by subset Walsh transform.

C.6 Week 6

C.6.1 Objectives

- Finish the detailed project proposal.

C.6.2 Tasks

- ~~Make the second draft of project proposal.~~
- ~~Complete the ethics form.~~
- ~~Add a Gantt chart to the project proposal.~~
- Read Partial structure learning by subset Walsh transform.

C.6.3 Meeting (7/10/2021)

- Discussed new proposal.
- Discussed the ethics form.

C.7 Week 7

C.7.1 Objectives

- Read more literature.
- Make a git repository.

C.7.2 Tasks

- ~~Make a git repository.~~
- ~~Read the entire honours guidelines document.~~
- ~~Read Introduction to Walsh Analysis.~~
- ~~Read Embedding Inequality Constraints for Quantum Annealing Optimization.~~

- Read Partial structure learning by subset Walsh transform.

C.7.3 Meeting (14/10/2021)

- One of the lit review grading aspects: analysis of context and consideration of alternatives. What does it mean?

C.8 Week 8

C.8.1 Objectives

- Understand Walsh transformation and try to link it to penalty coefficients.

C.8.2 Tasks

- ~~Read the lecture sent by Professor McCall.~~
- ~~Read the relevant part of The role of Walsh structure and ordinal linkage in the optimisation of pseudo-Boolean functions under monotonicity invariance.~~
- ~~Go through Genetic Algorithms and Walsh Functions; Part I, A Gentle Introduction.~~
- Watch Surrogate-assisted Multi-objective Combinatorial Optimization.

C.8.3 Meeting (21/10/2021)

- Discuss Walsh interactions.
  - Are we talking about i-i pairs?
  - Complexity.
  - Ways to calculate the penalty.
- Possible benchmarks.
- Numerical approach.

C.9 Week 9

C.9.1 Objectives

- Start writing the Literature Review.

C.9.2 Tasks

- ~~Make a template of Literature Review.~~
- ~~Make a draft of the Annealing section.~~
- ~~Make a draft of the Simulated Annealing section.~~
- Read chapter three of Multivariate Markov networks for fitness modelling in the estimation of distribution algorithm.

C.9.3 Meeting

- Discuss Literature Review.
- We are better at saying things rather than writing things. Maybe it is better to first explain verbally what I am about to write?

C.10 Week 10

C.10.1 Objectives

- Research Walsh interactions.

C.10.2 Tasks

- ~~Read Walsh Functions A Gentle Introduction sources Complex Systems.~~
- ~~Read chapter three of Multivariate Markov networks for fitness modelling in an estimation of distribution algorithm.~~
- ~~Read Simulated Stochastic Approximation Annealing for Global Optimization With a Square-Root Cooling Schedule.~~

C.10.3 Meeting (4/11/2021)

- We convert the penalty function into an energy distribution function defined as a sum of Walsh coefficients of a clique, α, times the Walsh functions of a clique. See the image. Where α are the interactions of the clique we are looking at. Am I correct?
- So are we trying to find α coefficients?
- We will have a perfect model structure as all the interactions are explicitly defined.
- We do not care about the fitness of individuals.

- Research least squares estimation.
- Research estimation of Markov Random Field.

C.11 Week 11

C.11.1 Objectives

- Literature Review.

C.11.2 Tasks

- ~~Write a second draft of Simulated Annealing section.~~
- ~~Write a draft of Quantum Annealing section.~~
- ~~Read Quantum annealing.~~
- ~~Read Quantum Deep Learning: Sampling Neural Nets with a Quantum Annealer.~~
- ~~Read QUANTUM ANNEALING AND QUANTUM FLUCTUATION EFFECT IN FRUSTRATED ISING SYSTEMS.~~
- ~~Read Quantum annealing: An introduction and new developments.~~
- ~~Read OPTIMIZATION BY SIMULATED ANNEALING: AN EXPERIMENTAL EVALUATION; PART 1, GRAPH PARTITIONING.~~
- ~~Read Digital Annealer for quadratic unconstrained binary optimization.~~
- ~~Read Ising formulations of many NP problems.~~
- ~~Write a draft section about QUBO.~~
- ~~Write a draft section about Digital Annealing device.~~

C.11.3 Meeting (11/11/2021)

- Discuss Quantum Annealing.
- Discuss deadlines.
- Can a video lecture be a reference?

C.12 Week 12-13

C.12.1 Objectives

- Literature Review.

C.12.2 Tasks

- ~~Write QUBO model section.~~
- ~~Write Natural QUBO Formulation section.~~
- ~~Write Non-Natural QUBO Formulation section.~~
- ~~Write Constraints and Penalties section.~~
- ~~Write Analytical penalty optimisation section.~~
- ~~Write Numerical 1 penalty optimisation section.~~
- ~~Write Numerical 2 penalty optimisation section.~~
- ~~Write ML penalty optimisation section.~~
- ~~Fix Simulated Annealing section (negative when maximizing).~~
- ~~Write and collapse Libraries and Algorithms sections into a single one.~~
- ~~Add Walsh interactions to Numerical 2.~~
- ~~Fix Q matrix with arbitrary value of 8.~~
- ~~Figures and tables.~~
- ~~Number the formulas.~~
- ~~Pseudocode with labels, lines and table of algorithms.~~
- ~~Write Introduction.~~
- ~~Write Summary.~~

C.12.3 Meeting (17/11/2021)

- Need a table of algorithms.
- Need a table of figures.
- Need to add line numbers to algorithms.

- Need to add an introduction with the motivation (including what CO problems are and why it is important to be able to solve them quickly), what the following sections will describe and why it is important for our motivation.
- Explain what Combinatorial Optimisation is and why solving such problems quickly is important.
- *One promising approach is Walsh interaction, which attaches explicit energies to the variables, which is naturally...*
- In summary, summarize all we have talked about. Identify gaps in research and propose a direction to move into (reduce M from Numerical 2 by taking into account the quadratic penalties).
- Explain objective function.
- Number equations.
- Word limit is 5000 + 10% with references.
- Do not add the arbitrary value of $M = 8$ to the Q matrix, but rather see how M affects the matrix.
- We do not flip the negative in SA when maximizing.

C.12.4 Meeting (24/11/2021)

- Discuss the Literature Review draft.

C.13 Week 14

C.13.1 Objectives

- Understand what Requirements Engineering requires.

C.13.2 Tasks

- ~~Read requirements lecture one.~~
- ~~Read requirements lecture two.~~
- ~~Make a draft of requirements.~~
- ~~Structure the draft.~~
- ~~D-Wave Guide.~~

- ~~Add little details to the draft.~~

C.13.3 Meeting (7/12/2021)

- Discuss the Requirements Engineering draft.
- How many pages does it have to be?
- Scientific requirements. What are they like?
- Specify that it is only for a single constraint function.
- Might be used for more, but it is not a necessity.
- Must be tested on a classical machine.
- Could be tested don DA.
- Should work with a QUBO.
- Should be tested using at least two datasets.
- Should be tested with other penalty estimation algorithms.
- Any stakeholders?
- How to measure differences?
- Rigorous statistical testing and analysis of results.
- Requirements related to the industry advisor company.
- Time constraints.
- Visualisation requirements?
- Single solution time requirements.
- What sections? Functional, non-functional, experiments, software...
- Easy to swap problems in and out.
- Methods of settings penalties.
- Have a way of evaluating the effect they have on performance.

C.14 Week 15

C.14.1 Objectives

- Finish Requirements Engineering.

C.14.2 Tasks

- ~~Finish Requirements Engineering.~~
- ~~Make post-meeting changes.~~

C.14.3 Meeting

- Discuss the deliverable.
- Fix the following:
  - Find → generate penalty coefficients.
  - Calculate → generate.
  - Always say M (consistency).
  - Completed → basic working version.
  - Deadline of experiments → All experimental hypotheses must be tested until...

C.15 Week 16

C.15.1 Meeting

- When using D-Wave libraries, do we manually do the multiplication of penalty coefficient with the constraint function and then addition with the original objective function?

C.16 Winter Break

C.16.1 Objectives

- Proof of concept.

C.16.2 Meeting

- Show the progress.

- Set goals.

C.17 Week 17

C.17.1 Objectives

- The Run Length Distribution implementation.

C.17.2 Tasks

- ~~Read On the Use of Run Time Distributions to Evaluate and Compare Stochastic Local Search Algorithms.~~
- ~~Find a way to implement Run Length Distribution.~~
- ~~Read Incorporating a Metropolis method in a distribution estimation using Markov random field algorithm.~~
- ~~Read Towards a Characterisation of the Behaviour of Stochastic Local Search Algorithms for SAT.~~
- ~~Implement RLD for greedy algorithm.~~
- ~~Implement RTD for tabu search.~~
- Implement new penalty estimation algorithm.

C.17.3 Meeting

- RLD notes:
  - We need to test it on a single problem rather than a cluster of different ones. I will just pick a problem just like in the Incorporating a Metropolis method in a distribution estimation using Markov random field algorithm paper.
  - *The RLD shows, for each algorithm, the cumulative percentage of successful runs that terminated within a certain number of function evaluations.*
  - All the papers I have came across had a clear indication of success for the algorithm (the solution). But QUBO has no clear solution. I think it is fair to use the mean solution cost of Verma and Lewis as a target.
  - The RLD cannot be implemented for Simulated Annealing in the form it appears in the dwave-neal. It works on a predefined number of steps.

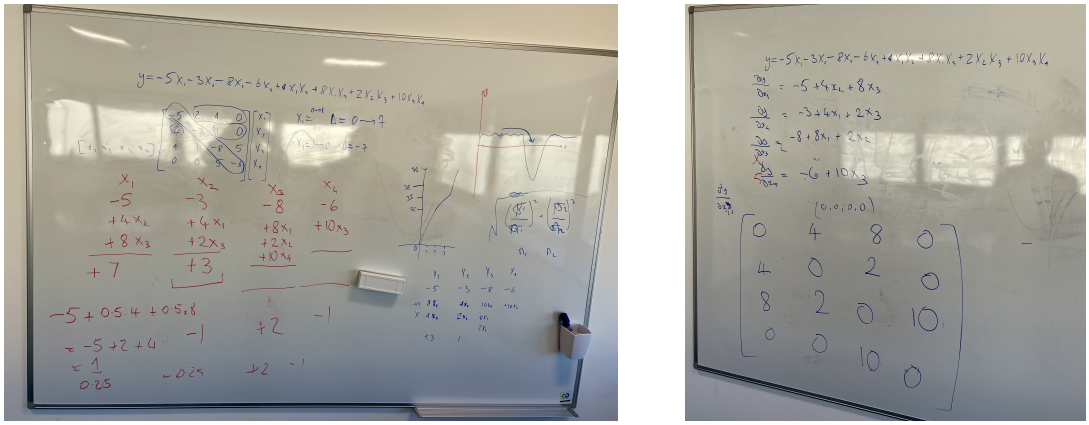
C.18 Week 18

C.18.1 Objectives

- Coding.
- Explaining what I am doing.

C.18.2 Tasks

- ~~Fix the weird energy levels.~~
- ~~Add another dataset.~~
- ~~Code new penalty estimation method.~~
- ~~Write an explanation on what is going on.~~
- ~~Make a notebook that shows how the penalty value will change and why.~~



(a) (b)

Figure C.2: Week 18 meeting notes.

C.18.3 Potential future work

- Make a python module, refactor code.
- Run experiments.
- Add statistical testing.
- Tune hyperparameters.
- Implement new penalty estimation method.

- Add new problems (datasets).
- Tables and plots.

C.19 Week 19

C.19.1 Objectives

- Statistical significance.

C.19.2 Tasks

- ~~Change some formulas in Literature Review.~~
- ~~Research significance tests.~~
- Code t-test.

C.20 Week 20-21

C.20.1 Objectives

- Statistical significance.
- Code improvement.

C.20.2 Tasks

- ~~Change some formulas in Literature Review.~~
- ~~Methodify the main pipeline.~~
- ~~Make nice RLD and RTD plots.~~
- ~~Find statistical significance test.~~
  - T-test.
  - Null hypothesis: new penalty estimation technique will not produce penalty coefficients that will significantly affect the ground energies achieved in any of the problems across every algorithm.
    - Run an algorithm on all problems many times with both sets of penalty coefficients.

        * Check if the difference in means was significant in the problems (compare the problems independently).
        * See if the difference was positive or negative, where it was spotted.
        * Do it with another algorithm and repeat the process.
        * Do it with another dataset and repeat the process.
    - Null hypothesis: new penalty estimation technique will not produce penalty coefficients that will significantly affect the ground energies achieved in all problems and across every algorithm.
        * Take all runs of all problems as equals.
        * Calculate general difference.
    - Using energy as a statistical significance test is a bad idea as different penalty values will most certainly affect the significance. And lower penalty values, for example, zero, will give the best energies. But they will also break the most constraints. In statistical significance testing, we have to capture the number of constraints that were broken rather than the energy.

- ~~Code data saving and loading.~~
- ~~Make a table with the number of constraints broken.~~
- ~~Code statistical test 1.~~
- ~~Code statistical test 2.~~

C.20.3 Penalty ideas

- Do a very slow binary tree search (all possibilities).
    - Exhaustive search kills the entire purpose of optimisation.
- Double search.

C.21 Week 22

C.21.1 Objectives

- Minor code improvements.
- Starting dissertation.

C.21.2 Tasks

- ~~Make draft sections.~~
- ~~Acknowledgments.~~
- ~~Choose best answer, not first during minimisation.~~
- Cover page.

C.21.3 Meeting

- Make a single sentence and make the whole dissertation a document meant for the reader to understand that sentence.
- *To optimise better with QA, we need to choose a penalty coefficient that will punish the algorithm for ..., and to choose a better penalty coefficient, I have made a more informed algorithm, but sacrificed guarantees of feasibility.*
- The dissertation can have the following sections:
  - Project Overview
  - Literature Review
  - Design & Implementation
    - Talk about choices: Jupyter Notebooks etc...
  - Results
    - Show tables, graphs etc... Talk about the results.
  - Conclusion

C.22 Week 23-24

C.22.1 Objectives

- Penalty functions.
- Implementation section.

C.22.2 Tasks

- ~~Make Penalty class with different penalty generation algorithms.~~
- ~~Split all supplementary methods into classes.~~

- ~~Methodify Run Time/Length Distribution code.~~
- ~~Make a python module with all the code used in experiments.~~
- ~~Make visualisation module.~~
- Tune hyperparameters.
  - ~~Make a list of possible hyperparameters to tune.~~

C.22.3 Meeting

- Show the python package.
- Show new notebooks.
  - How clean it is and how to change the penalty/data?
- Discuss hyperparameters.
- Word → LaTeX migration.

C.23 Week 25

C.23.1 Objectives

- Tune hyperparameters.
- Load other datasets.
- Transfer word submissions to LaTeX.

C.23.2 Tasks

- ~~LaTeX styles.~~
- ~~LaTeX title page.~~
- ~~LaTeX Project Overview.~~
- ~~Test hyperparameters.~~
- LaTeX Literature Review.
- Jupyter Notebook for another dataset.

C.23.3 Meeting

- Is RTD for tabu search reasonable? As we are defining how long each run will take before we run the algorithm. Thus we know that the run time of all runs, both successful and unsuccessful, will be roughly the same.
- What target energy to use with RLD? Currently, I am using mean energy. But then if I repeat this experiment with another penalty coefficient algorithm, the mean will change and RLD will change. In a way that will prohibit us from comparing them. At the same time, it is impossible to deduce energy that corresponds to a feasible solution.
- Possible hyperparameter tactics: choose Tabu Search hyperparameters that give a solution rate of 0.5.
- Run tabu many times with different times for RTD.
- For RLD check all solutions for feasibility and base the graph on it.
- Move project plan to Appendix.
- 1.6 Content Summary (explain the overall structure).
- Add line numbers to algorithms.

C.24 Week 26

C.24.1 Objectives

- LaTeX.
- Make new notebooks for other datasets.

C.24.2 Tasks

- ~~LaTeX Literature Review.~~
- ~~Move project proposal to Appendix.~~
- ~~Add line numbers to algorithms.~~
- ~~Add Summary section at the end.~~
- ~~Make a notebook for the Quadratic Assignment Problem.~~
- ~~Make a data preparation method for other datasets.~~

- ~~Make a notebook for the Travelling Salesman Problem.~~
  - We run TSP large and small separately because of large run for a significantly longer time.

C.24.3 Meeting

- Report presentation.
- Ask about testing (in mark scheme).
- Starting to see interesting things: MKP was super fast!
- TSP split into two parts: large and small.
- Decided to tune hyperparameters more thoroughly.
- Give algorithms X time per dataset rather than the problem.
  - Datasets can have different number problems.
  - Problems are of different difficulty.
- Testing subsections.
  - Write something general about an approach to software engineering.
  - Unit tests?
  - Major decision issues.
  - Efficiency.
  - Issues that I have faced when working.

C.25 Week 27

C.25.1 Objectives

- LaTeX.

C.25.2 Tasks

- ~~Move Requirements Engineering to LaTeX.~~
- ~~Add a section explaining Multiple Knapsack Problem to the dissertation.~~
- ~~Add a section explaining Travelling Salesman Problem to the dissertation.~~

- ~~Add a section explaining Quadratic Assignment Problem to the dissertation.~~
- ~~Code the discussed penalty estimation algorithm.~~
- ~~Research Gaussian Elimination.~~
- ~~Think about turning point penalty estimation algorithm.~~

C.26 Week 28

C.26.1 Objectives

- Poster.

C.26.2 Tasks

- ~~Make a poster draft.~~
- ~~Make visualisations for poster.~~
- ~~Finish the poster.~~

C.27 Week 29

C.27.1 Objectives

- Finish code.
- Prepare for Degree Show

C.27.2 Tasks

- ~~Penalty Coefficient Distribution.~~
- ~~Run Time and Run Length Distributions: update the code.~~

C.28 Week 30-32

C.28.1 Objectives

- Write the dissertation.

Appendix D

Source Code

https://github.com/rienath/Estimating-Penalty-Coefficients-for-QUBO-Problems

STUDENT PROJECT ETHICAL REVIEW (SPER) FORM

The aim of the University's *Research Ethics Policy* is to establish and promote good ethical practice in the conduct of academic research. The questionnaire is intended to enable researchers to undertake an initial self-assessment of ethical issues in their research. Ethical conduct is not primarily a matter of following fixed rules; it depends on researchers developing a considered, flexible and thoughtful practice.

The questionnaire aims to engage researchers discursively with the ethical dimensions of their work and potential ethical issues, and the main focus of any subsequent review is not to 'approve' or 'disapprove' of a project but to make sure that this process has taken place.

The *Research Ethics Policy*: www.intranet.rgu.ac.uk/credo/staff/page.cfm?pge=7060

Student Name	Raufs Dunamalijevs
Supervisor	Professor John McCall
Project Title	Estimating Penalty Coefficients for QUBO Problems
Course of Study	Computer Science
School/Department	School of Computing

Part 1 : Descriptive Questions			
1	Does the research involve, or does information in the research relate to:	Yes	No
	(a) individual human subjects		✓
	(b) groups (e.g. families, communities, crowds)		✓
	(c) organisations	✓	
	(d) animals?		✓
	Please provide further details:		
	The research involves Fujitsu Laboratories as their Digital Annealer hardware would be used to run an optimization algorithm.		
2	Will the research deal with information which is private or confidential?	Yes	No
			✓

Part 2: The Impact of the Research			
3	In the process of doing the research, is there any potential for harm to be done to, or costs to be imposed on	Yes	No
	(a) research participants?		✓
	(b) research subjects?		✓
	(c) you, as the researcher?		✓
	(d) third parties?	✓	
	Please state what you believe are the implications of the research:		
	There are costs imposed on Fujitsu Laboratories as their hardware might be used to run some of the experiments.		
4	When the research is complete, could negative consequences follow:	Yes	No
	(a) for research subjects		✓
	(b) or elsewhere?		✓

Part 3: Ethical Procedures			
5	Does the research require informed consent or approval from:	Yes	No
	(a) research participants?		✓
	(b) research subjects		✓
	(c) external bodies		✓
6	Are there reasons why research subjects may need safeguards or protection?	Yes	No
			✓
7	Has PVG membership status been considered?		✓
	(a) PVG membership is not required.	✓	
	(b) PVG membership is required for working with children.		✓
	(c) PVG membership is required for working with protected adults.		✓
	(d) PVG membership is required for working with both children and protected		✓
8	Are specified procedures or safeguards required for recording, management, or storage of data?	Yes	No
			✓

Part 4: The Research Relationship			
9	Does the research require you to give or make undertakings to research participants or subjects about the use of data?	Yes	No
			✓
	If you answered yes to the above, please outline the likely undertakings:		
10	Is the research likely to be affected by the relationship with a sponsor, funder or employer?	Yes	No
			✓

Part 5: Other Issues			
11	Are there any other ethical issues not covered by this form which you believe you should raise?	Yes	No
			✓

Statement by Student I believe that the information I have given in this form is correct, and that I have addressed the ethical issues as fully as possible at this stage.			
Signature		Date	07.10.2021

If any ethical issues arise during the course of the research, students should complete a further Student Project Ethical Review (SPER) form.

The *Research Ethics Policy*: www.intranet.rgu.ac.uk/credo/staff/page.cfm?pge=7060

Part 6: To be completed by the supervisor			
12	Does the research have potentially negative implications for the University?	Yes	No ✓
13	Are any potential conflicts of interest likely to arise in the course of the research?	Yes	No ✓
14	Are you satisfied that the student has engaged adequately with the ethical implications of the work? [In signifying agreement, supervisors are accepting part of the ethical responsibility for the project]	Yes ✓	No
15	**Appraisal:** Please select one of the following		
	The research project should proceed in its present form – no further action is required	✓	
	The research project requires ethical approval by the School Ethics Review Panel		
	The research project needs to be returned to the student for modification prior to further action		
	The research project requires ethical review by an external body. If this applies please give details		
	Title of External Body providing ethical review		
	Address of External Body		
	Anticipated date when External Body may consider project		

Affirmation by Supervisor			
I have read the student's responses and have discussed ethical issues arising with the student. I can confirm that, to the best of my understanding, the information presented by the student is correct and appropriate to allow an informed judgement on whether further ethical approval is required.			
Signature		**Date**	07-OCT-2021

Estimating Penalty Coefficients For Quadratic Unconstrained Binary Optimisation Problems

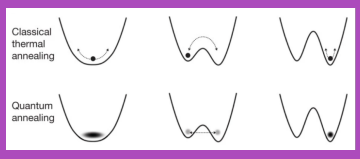
THE ROBERT GORDON UNIVERSITY ABERDEEN

National Subsea Centre

Raufs Dunamalijevs
Supervised by Professor John McCall

MOTIVATION

Quadratic Unconstrainted Binary Optimisation (QUBO) is a way to formulate combinatorial optimisation problems. A QUBO problem can be mapped on a network of qubits and solved on a Quantum Annealing device, which can take advantage of quantum tunnelling when exploring the fitness landscape.



In QUBO, constraints of the problem transform into a penalty function, which worsens the objective value if it is infeasible. The penalty coefficient *(M)* controls the degree to which the penalty function will influence the overall fitness. If *M* is too low, the broken constraints will be undervalued, and the solution produced will be infeasible. If the *M* is too large, the penalties will overwhelm the objective function making it harder to differentiate between good and bad solutions. The current state of the art gives theoretical feasibility guarantees at the expense of overestimating *M*.

PROJECT AIMS

- Study the existing methods of estimating *M* and critically evaluate them.
- Implement one of them.
- Propose and implement new algorithms for estimating *M*.
- Generate *M* for a variety of QUBO problems using implemented algorithms.
- Solve the QUBOs using generated *M*'s.
- Compare the quality of the produced solutions.

METHODS

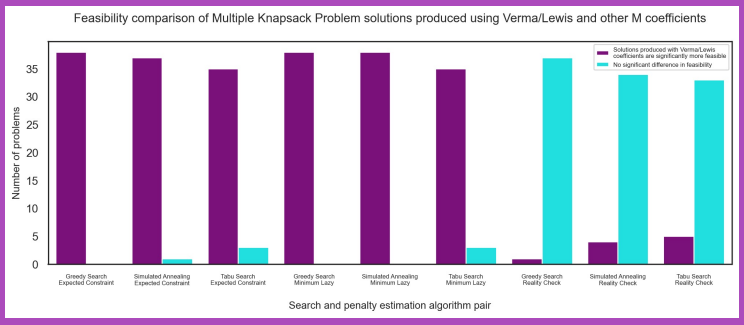
We have implemented two new penalty estimation algorithms. Both are based on the Verma and Lewis (VL) approach that produces the estimated lower bound of the penalty coefficient. We have tried to improve its weak point and have taken into account the penalty function structure.

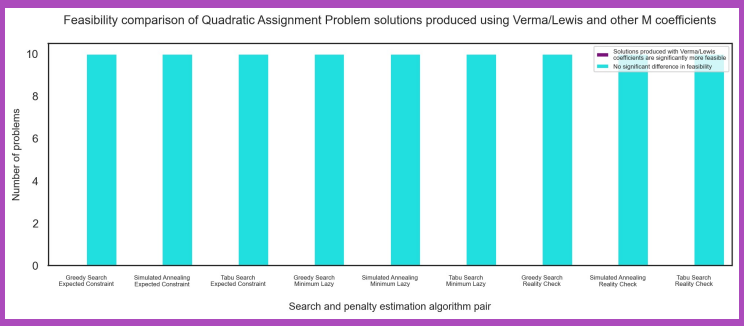
Our first algorithm *(Expected Constraint)* calculates the expected penalty if it were imposed by counting the amount of penalty each decision variable is associated with and then dividing it by the total number of the variables. We then divided the VL prediction by the expected penalty to return a reduced lower bound.

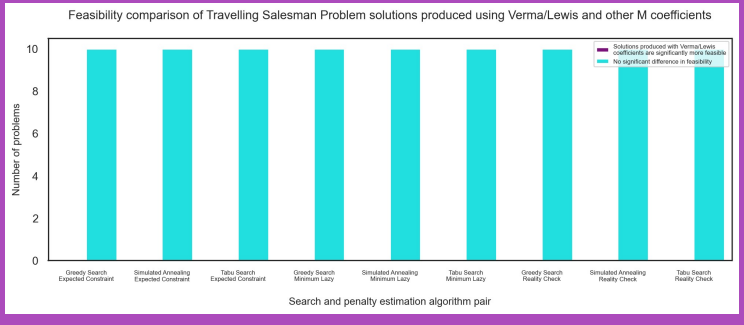
The second algorithm *(Minimum Lazy)* is similar to the first one, but instead of finding the total expected penalty, we find the minimum penalty associated with any of the variables.

Finally, we made an algorithm that always returns half of the predicted VL lower bound, so we can find the problems for which VL definitely returns an overestimated coefficient. As most of the time, other algorithms produce numbers that are much smaller than the VL prediction, we do not always see if we are running into the other extreme: underestimating the penalty coefficients. That is why we need this *Reality Check*.

RESULTS







CONCLUSION

While VL lower bounds can be used effectively in Multiple Knapsack Problems, it has been shown that Quadratic Assignment and Travelling Salesman Problems produce feasible solutions with *M*'s that are much lower. Moreover, even in most Multiple Knapsack Problems, halving VL coefficients will still produce solutions of similar feasibility in most cases. This result means that the predicted bound in many cases is not the actual lower bound as was theorised. Empirical demonstrations suggest that other methods could be more effective in finding penalty coefficients. We have produced a list of problem instances from three different datasets that are likely to have smaller lower bounds than the ones predicted by VL. Potential future work can include identifying whether the algebraic forms of these problems have something in common. If it is possible to identify such QUBOs quickly, a one-for-all method could be made to produce better penalty coefficients.

KEY PAPERS

Glover, F., Kochenberger, G. & Du, Y. (2019), 'Quantum Bridge Analytics I: a tutorial on formulating and using QUBO models', *4OR* **17**(4), 335–371.

Verma, A. & Lewis, M. (2020), 'Penalty and partitioning techniques to improve performance of QUBO solvers', *Discrete Optimization* p. 100594.

Kochenberger, G., Hao, J. K., Glover, F., Lewis, M., L¨u, Z., Wang, H. & Wang, Y. (2014), 'The unconstrained binary quadratic programming problem: A survey', *Journal of Combinatorial Optimization* **28**(1), 58–81