Acknowledgements

I would like to thank my supervisors, Prof. João Paulo Fernandes, Prof. Luís Paquete, and Prof. Rui Maranhão, whose support and knowledge helped me reach a level of quality and rigor for this thesis that I would not be able to reach by myself. I'm grateful for all that I have learned with all of you and, above all, thank you for bearing with me and my forgetful nature.

I'm also grateful for my family for supporting me every day during quarantine. And to my friends, thank you for support and for staying with me even when I seemingly disappeared from the grid at times. Here's to many more years of great friendship.

Pedro Miguel Dias da Silva,
Coimbra, October 2021

Pedro Miguel Dias da Silva

QUANTUM COMPUTING FOR OPTIMIZING POWER FLOW IN ENERGY GRIDS

1 2 9 0

UNIVERSIDADE Ð COIMBRA

1 2 9 0

UNIVERSIDADE Ð
COIMBRA

Pedro Miguel Dias da Silva

QUANTUM COMPUTING FOR OPTIMIZING POWER FLOW IN ENERGY GRIDS

Dissertation in the context of the Master in Informatics Engineering, Specialization in Software Engineering, advised by Professor João Paulo Fernandes, Professor Luís Paquete, and Professor Rui Maranhão, and presented to Faculty of Sciences and Technology / Department of Informatics Engineering

October 2021

This page is intentionally left blank.

Faculty of Sciences and Technology

Department of Informatics Engineering

Quantum Computing for Optimizing Power Flow in Energy Grids

Pedro Miguel Dias da Silva

Dissertation in the context of the Master in Informatics Engineering, Specialization in Software Engineering, advised by Prof. João P. Fernandes, Prof. Luís Paquete, and Prof. Rui Maranhão and presented to the
Faculty of Sciences and Technology / Department of Informatics Engineering.

October 2021

1 2 9 0

UNIVERSIDADE Ð COIMBRA

This page is intentionally left blank.

Abstract

Quantum Computing is beginning to gather even more attention at a time where efforts are being made into familiarizing younger audiences into not only learning programming on a classical computer, but also on a quantum one.

This new paradigm of computation is set to revolutionize several industries as the hardware keeps developing, with the potential to solve problems that a classical computer would consider intangible, as well as giving some specific problems a so sought after speed-up. This is done by applying the properties of quantum physics, like superposition and entanglement, for computation. These properties not only allow to process a larger amount of data simultaneously, but also allows to tackle problems in a completely different way that would not be possible in a classical computer.

This thesis focuses on solving a known and relevant problem in the electrical industry and studying its application on a quantum environment. The Unit Commitment Problem, the problem in question, consists in minimizing the cost of power production, for a certain time horizon, by scheduling different generating units in order to meet a certain demand given by a valid forecast. Given that this is an NP-hard problem, it quickly becomes intractable on classical computers when considering real world scenarios on a large scale.

A test scenario was also designed to study, by conducting an experimental analysis, the influences that each of the parameters have on the solution quality. To that end, the formulation of the Unit Commitment Problem was also translated to a suitable QUBO form which is then solved through a quantum annealer from D-Wave. For that test scenario, both the parameters from the problem formulation as well as the parameters related to the quantum computer were considered.

The results from the experimental analysis suggest that most parameters do have an impact on the solution quality. With some having a greater impact overall such as Grids, that are representing how accurate the linearization of the problem is, as well the delta value associated with the first constraint, a value that is tied to how much of a weight the first constraint, that restricts each unit to a single production level, has. While the parameters with the overall greater impact are tied to the formulation of the problem, parameters like chain strength that affects the strength of coupling between qubits representing a single variable also have a significant impact on the solution quality. While most parameters have a statistical impact on the solution quality, the delta associated with the second constraint, that restricts power generation to equal the demand, fails to have an impact.

Keywords

Unit Commitment Problem, Mixed-Integer Linear Programming, Quantum Computing, Quantum Annealing, Quadratic Unconstrained Binary Optimization, Optimization

This page is intentionally left blank.

Contents

This page is intentionally left blank.

Acronyms

D-WAVE	D-Wave Systems, Inc.. v, ix, xi, 3, 5, 11, 15–19, 25, 26, 28, 29, 31–37, 39, 45
GLPK	GNU Linear Programming Kit. v, 19, 27, 29
IBM	International Business Machines Corporation. v, xi, 3, 4, 11–13, 15
JSON	JavaScript Object Notation. v, 26
MILP	Mixed-Integer Linear Programming. v, 7, 19
MINLP	Mixed-Integer Non-Linear Programming. v, 7
QPU	Quantum Processing Unit. v, 11, 13, 26, 45, 46, 49
QUBO	Quadratic Unconstrained Binary Optimization. v, 4, 5, 19–26, 29, 32, 34, 36, 37

This page is intentionally left blank.

List of Figures

This page is intentionally left blank.

List of Tables

This page is intentionally left blank.

Listings

This page is intentionally left blank.

Chapter 1

Introduction

In recent years, there has been an increasing amount of attention around applications of quantum solutions due to the potential speed-up that it can bring to some fields. The fields researching quantum applications vary, as they range from typical optimization problems like scheduling problems, portfolio selection and pathfinding, to simulating and analyzing chemicals, molecules and quantum effects, and even some applications in machine learning. [3]

The study of possible quantum computing applications is nothing new, as it has been the topic of research spanning back several decades even before any substantial hardware was available [4, 5, 6, 7, 8]. It was not until the late 1990s that experimental implementations of a quantum computer took off from the simple 2-Qubit systems to the thousand-Qubit systems that D-Wave Systems, Inc. currently offers. In recent years with the increasing research on part manufacturing along with other advancements in hardware, the capabilities that the current hardware offers already allows for a speedup when compared to classical computers in some specific problems. Methods such as Boson sampling [9], and Monte Carlo methods [10] already have a considerable speed-up over classical alternatives and are usually the methods used when trying to prove quantum supremacy. Another algorithm to note is Shor's factoring algorithm that, in theory, offers a speedup when compared to the best classical method [11, 12]. Even so, the quantum hardware is still not developed enough to allow Shor's algorithm process large numbers, as some of the recent attempts managed to factor the number 21 and failed to correctly factor the number 35. [13]

Aside from the improved hardware capabilities, efforts are also being made to improve the availability of quantum hardware to the general public. Through the use of cloud services, anyone could potentially learn and try to solve a problem in an actual quantum machine. To list a few: IBM's Qiskit [14], D-WAVE's Leap [15], Amazon's AWS Braket [16], Qutech's Quantum Inspire [17], Microsoft's Azure Quantum [18], and even some Universities that allow access to their quantum computers through the cloud such as the University of Bristol. [19]

One of the biggest goals for quantum computing is to achieve quantum supremacy. That is, to solve a problem on a quantum computer that would be considered intractable on a classical computer. Some teams around the world have already claimed to have achieved quantum supremacy like a team from Google [20] and a team in China [9]. This experiment by Google consisted in using a 53 qubit quantum processor to make a true random number generator. Although this experiment claims that what they achieved in 200 seconds would take a classical computer 10000 years to solve, IBM reported that by using a different classical technique could lower the estimate of the classical method to 2.5 days. This response by IBM seems to invalidate Google's claim for quantum supremacy [21] but nonetheless, this experiment is still shows a significant improvement and even though no supremacy was achieved, a sizeable speed-up is still within reach with current methods and hardware.

The remarkable doubly exponential growth of quantum computational power [22] as well as the improvement of the fabrication process [23] will help to improve the reliability as well as tackle some large scale problems that the quantum hardware currently cannot.

One of the prominent fields that has turned their attention to quantum is optimization problems. While more and more research is put into improving performance of these algorithms using different kinds of heuristics that tend to mimic biological behaviors to the make algorithms slightly ever faster. [24, 25, 26, 27, 28]

There are many different applications for optimization problems but a specific field has had its importance increase over the years with current efforts to diminish environmental impact and turn to more green friendly options. The area of power production has some major problems with varying degrees of complexity, with some examples being facility allocation that focuses on optimizing facility location for maximum power production, heat exchanger network that aims to minimize heating and cooling costs for industrial plants, and unit commitment problem that revolves around minimizing the costs of energy production units. [29]

Focusing on the Unit Commitment Problem that, while having had a lot of research [30, 31, 32, 33, 34] focused around developing different kinds of algorithms to push ahead of the current state of the art due to its NP-hard nature [29], has not had much attention into its application in a quantum environment. Having a paper covering it [29], with the goal of making a broad review of the quantum environment for problems and with a bigger emphasis on the results and how it compares to the classical solutions while the process to implement those problems are not explained in detail, increasing the entry barrier for those who want to replicate, or expand on, any of the experiments.

The goals for this thesis are to create a detailed approach starting from the base formulation of the Unit Commitment Problem and transforming it into a QUBO form that a quantum annealer accepts. Afterwards, a test scenario is created that allows for an experimental analysis of the various effects of several parameters and how much they impact the solutions returned by the computer. This leads to the hypothesis raised by this thesis: *"Varying the parameters from both the unit commitment problem and from the quantum annealer influences the quality of the solutions returned by the quantum annealer"*. If the hypothesis is confirmed, then the parameters that have the biggest impact are compiled and a discussion on the optimal configuration of the parameters will follow.

The results from testing show that most of the parameters have an impact on the results, with some having a greater impact than others. Some of the parameters with a bigger influence are the constraint type considered and the delta value, or the weight, of the first constraint. These are both part of the formulation of the problem but a parameter for the quantum annealer, the embedding type used, also has a considerable impact, specially on chain break occurrence. All of these findings, as well as the parameters, will be explained in more detail in section 4.

The various contributions made by this thesis can be listed as follows

- A review of the state of the art for quantum computing, with a bigger focus on the Unit Commitment Problem.

- Implementation of a quantum algorithm using a D-WAVE Quantum Annealer, including the process of translating the base problem formulation into a QUBO form that D-WAVE accepts.
- An experimental analysis to study the influences of various parameters, from both the formulation and the quantum annealer, on the quality of the solutions returned by the quantum annealer.

This thesis will be structured according to the following outline

- **Chapter 2 - Background** will expand on the concepts required to understand the thesis like quantum concepts and the optimization problem being studied.
- **Chapter 3 - Development and Solution** follows the process of translating the base formulation for the problem and finally solving it in a quantum environment.
- **Chapter 4 - Experimental Analysis** contains the experimental analysis of a formulated test case to study the impact of the different parameters to both the formulated problem as well as the quantum hardware.
- **Chapter 5 - Conclusion** will cover the conclusions that resulted from this thesis as well the outlook for the future concerning applications of these kind in quantum.

This page is intentionally left blank.

Chapter 2

Background

This section will focus on introducing the concepts discussed in this thesis like the formulation for the Unit Commitment problem and the workings behind some quantum mechanics that are being implemented when solving the problem.

2.1 The Unit Commitment Problem and its Formulation

The Unit Commitment problem is an optimization problem which consists of minimizing the cost of power generation, for a certain time horizon, by properly scheduling the generation units, while having production values that meet a demand given by a valid forecast. One could also consider which units are operational at a given time, as well as what type of unit they are.

Usually, there are three different types of power generating units in the electrical power industry [35] [36]:

- Thermal Units(including nuclear ones)
- Hydro Units
- Renewable Energy Units

Note that the problem of the intermittence of renewable sources have additional constraints such as being able to produce power during the day for solar farms, with the weather being yet another a factor, or for the wind farms that are only able to produce power when there is sufficient wind [32]

The Unit Commitment problem is typically formulated as Mixed-Integer Linear Programming (MILP) problem, and sometimes as a Mixed-Integer Non-Linear Programming (MINLP) problem, due to the natural ease to formulate the problem with nonlinear functions that describe complex properties and integer variables that are used to capture the discrete decisions of the model [37]. It has also been proven that this problem is NP-hard, which means that it should not be expected to solve this problem in a polynomial amount of time with respect to size [31].

It is at this stage that Quantum Computing comes into play, and while in theory solving the problem in polynomial time is still unlikely, Quantum Computing can be used to reach a speedup when compared to classical computing solutions by taking advantage of the properties of Quantum Mechanics.

In the following, an overview of the formulation that is typically used for the Unit Commitment problem based on the formulation in this thesis [38]

The Objective Function

$$\sum_{k\in K}\sum_{j\in J} c_j^p(k) + c_j^u(k) + c_j^d(k) \tag{2.1}$$

The objective function describes the problem to solve, more specifically the quantities that must be minimized. In this formulation, $c_j^p(k)$, $c_j^u(k)$ and $c_j^d(k)$ refer, respectively, to the production cost, startup cost, and shutdown cost of a unit j in a certain time period k.

Equation 2.2 refers to the quadratic part of the objective function.

$$c_j^p(k) = a_j v_j(k) + b_j p_j(k) + c_j p_j^2(k), \quad \forall_j \in_J, \forall_k \in_K \tag{2.2}$$

This quadratic function represents the production cost of a unit in which a,b and c are the quadratic coefficients and $v_j(k)$ is a binary variable that is 1 if unit j is online in period k and 0 otherwise. The term $p_j(k)$ represents the power output of a unit j in a certain time period k.

Regarding the startup cost, it can be considered a constant value but, in some cases it is represented by a piecewise linear function to better represent the exponential start-up time. This is due to thermal units being more easily lit up when it is been offline for less time. What is typically referred to as "cold" startup is a much more expensive way to activate the unit than one that was just offline for an hour, since there is already some scattered heat that helps bringing it back up online. Shutdown cost is simpler to represent since it is usually a constant value but can also be modeled by a piecewise linear function in a similar manner to startup cost.

The Constraints

The problem has the following constraints:

$$\sum_{j\in J} p_j(k) = D(k), \forall k \in K \tag{2.3}$$

This constraint aims to balance the power production of all units in a certain time period k, so that it can be equal to the demand $D(k)$.

$$\sum_{j\in J} \overline{p}_j(k) \geq D(k) + R(k), \forall k \in K \tag{2.4}$$

Equation 2.4 represents the maximum available power output, $\overline{p}(k)$, that must be greater or equal than the sum of the demand $D(k)$ and the spinning reserve $R(k)$. This constraint aims to provide a margin for the spinning reserve, also known as operating reserve, with the goal of having a power reserve ready for unexpected spikes in demand, a generator breaking down or any other kinds of disruptions.

There are other typically used constraints like [38]:

- Ramp-up and ramp-down constraints, where it more accurately models the evolving power production instead of being a constant.

- Minimum and maximum production limit. Unit power production will not go below or above a set interval.
- Minimum and up and downtime for the units. Once a unit is online it must be up for 3 hours or once it goes offline it must be resting for 4 hours, for example.
- Mandatory online and offline units. Essentially restricting the problem that unit j must be online or offline. This could be desired due to some units being more efficient than others.

This thesis covers a simplified version where the time factor will not be considered, as well as shutdown and startup costs, as well as spinning reserve. This simplified form, as well as its implementation in a quantum machine will be explained in more detail in Section 3.1 and Section 3.1.1 respectively.

2.2 Quantum

Using the effects of quantum mechanics like superposition and entanglement, quantum algorithms are able to do things that were not feasible in classical computing, like using destructive quantum interference to naturally remove unwanted solutions in the course of the algorithm as used for example in Shor's factoring algorithm [11] by cleverly making use of quantum Fourier transforms.

Instead of using a classical bit, representing either a state of 0 or 1, quantum computing makes use of a qubit, where a special state called superposition allows both 0 and 1 states to exist at the same time. The standard notation for a qubit uses the dirac notation, also known as the braket notation (from the bra $\langle a|$ and ket $|b\rangle$ it represents).

Although it is possible for a qubit to be in a superposition of both "0" and "1", after measurement it collapses into a single state, meaning that the two states are mutually exclusive. What follows is a standard definition of these two states without superposition.

$$|0\rangle = \begin{bmatrix}1\\0\end{bmatrix} \; |1\rangle = \begin{bmatrix}0\\1\end{bmatrix} \tag{2.5}$$

With this in mind, an arbitrary state vector can be defined as:

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle \tag{2.6}$$

With the coefficients α and β representing the probability amplitudes of their respective states, the reason why it is an amplitude will be explained shortly. $|\alpha|^2$ is then, the probability of the qubit being in the state "0" and $|\beta|^2$ the probability of being in the state "1". [39]

Since $|\psi\rangle$ must also be a unit vector then the normalization is described as:

$\sqrt{|\alpha|^2 + |\beta|^2} = 1$.

With this, it becomes possible to represent a nearly infinite combination of states with the coefficients α and β. This opens up a lot of computing possibilities for storing temporary information than just the classical "0" or "1" bits.

Yet, this formulation has some problems, like not being able to differentiate between some states through just the use of a global phase.

To circumvent this, the difference in phase of α and β is measured instead.

Trigonometry can be used to obtain: $\sqrt{\sin^2 x + \cos^2 x} = 1$.

With this in mind, α and β can be described as cos and sin, respectively. $\alpha = \cos\frac{\theta}{2}, \quad \beta = \sin\frac{\theta}{2}$

With this any state of the qubit can be described using ϕ and θ:

$$|\Psi\rangle = \cos\tfrac{\theta}{2}|0\rangle + e^{i\phi}\sin\tfrac{\theta}{2}|1\rangle \quad \theta, \phi \in \mathbb{R} \tag{2.7}$$

Using θ and ϕ as spherical coordinates (with radius=1, since the magnitude of a qubit state is 1), any qubit state can be plotted on the surface of a sphere, also know as the Bloch Sphere [40] as shown in Figure 2.1.

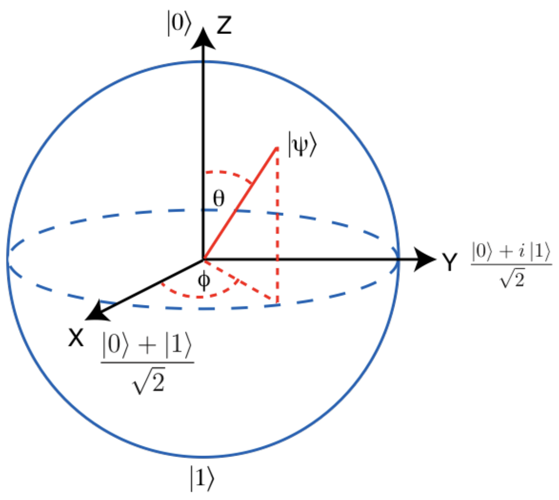


Figure 2.1: Bloch Sphere (Qiskit) [1]

This is just a basic introduction to the fundamentals of the quantum field but, what can truly be achieved with these mechanics depends on the use of entanglement.

When two, or more, qubits from different systems come together they form a composite system that correlate qubits in such a way that it becomes impossible to describe each qubit individually, not to mention that it also becomes impossible to measure on qubit without influencing the result of the other.

This correlation is what is called entanglement and one such example of a system with a maximum level of entanglement would be a Bell state:

$$|\Phi^+\rangle = \tfrac{1}{\sqrt{2}}(|00\rangle + |11\rangle) \quad |\Phi^-\rangle = \tfrac{1}{\sqrt{2}}(|00\rangle - |11\rangle)$$
$$|\Psi^+\rangle = \tfrac{1}{\sqrt{2}}(|01\rangle + |10\rangle) \quad |\Psi^-\rangle = \tfrac{1}{\sqrt{2}}(|01\rangle - |10\rangle)$$

Focusing on one of the four Bell states, for example

$$|\Phi^+\rangle = \tfrac{1}{\sqrt{2}}(|0_A 0_B\rangle + |1_A 1_B\rangle)$$

In this composite of qubits A and B has a 50% probability of being measured in both the state $|00\rangle$ and $|11\rangle$, and these states are in superposition until it collapses onto a single state on measurement. This means that when measuring a qubit, in this example, if the measurement reads the state $|0\rangle$ then the state of the other entangled qubit must also be $|0\rangle$, with the final state of the qubits being $|00\rangle$.

This means that the final state of a qubit in a maximally entangled system is random, but it is also known that once the result of a qubit is known, then the state of the other qubit is also known since they are entangled.

These Bell states ere essential in some specific applications like Quantum Teleportation that is being researched towards improving how the world communicates securely. This protocol transfers the state of a qubit to another over a distance trough the use of entanglement.

This allows quantum states to be shared across distances and for communications to rely on quantum cryptography that creates an unbreakable security, since any attempts to read the key that encrypts the message will either change it or destroy it.

And yet, since it relies on entanglement that acts across distances it may seem that it is possible to achieve an instantaneous communication over seemingly infinite distances by measuring the state of an entangled qubit and instantly know the state of another qubit far far away, the reality is that it is not.

This effect is known as the No-communication theorem [41]. Due to this theorem it is not possible to transmit information through the use of entanglement, for example, entangling two qubits and then sending one of the qubits to Mars, then trying to send the state $|1\rangle$ to Mars by influencing the qubit on this side. This is where the theorem comes in, stating that it is not possible to create a desired outcome since the measurement collapses the qubit into a random state and can't be forced to collapse into a desired state.

One of the biggest problems left to solve in the quantum computing field are the different kinds of errors that qubit states are susceptible to. Even though efforts have seen an advance in error correction, some kinds of errors still elude these efforts.

These kind of errors tend to be the uncorrelated and random errors that can be fixed with simple error correction algorithms also used in classical computing, with just a slight quantum twist. But correlated errors that can affect the whole system, a large array of entangled qubits, are a major setback for quantum computing, since these errors can completely invalidate the problem being solved. A recent paper has also found that quantum computers with a superconducting system are susceptible to correlated errors caused by cosmic radiation, with the implication that shielding from such rays is necessary to increase system accuracy and reduce errors of this kind. [42]

2.2.1 Available Frameworks

When considering how to build a quantum system, there are several possible architectures that can be used. The ones being discussed in this thesis are the Quantum circuits and Adiabatic Quantum Computers models, are the ones with the most active development that offer global publicly accessible solutions through the cloud that any person with access to the internet can make use of. Two of the biggest companies working on these models are International Business Machines Corporation (IBM), using the Quantum Circuits model, and D-Wave Systems, Inc. (D-WAVE), using the Adiabatic model. IBM and D-WAVE both have their QPU available for use through the cloud while also giving access to several tools for the making of quantum algorithms as well as for miscellaneous support. While their pricing strategies differ, anyone can still create a quantum algorithm and run it on their machines for free.

Gate-based Quantum Computers

The gate-based, or circuit-based, quantum computers are the ones most similar to their classical counterparts, but that's where the similarities start and end, applying transformations to qubits instead of pushing electric signals through various gates to check the resulting output. Following a gate-based architecture, this model uses gates to apply a certain operation to the state of a qubit, which in the end, after the evolution of the system through various gate operations, leads to the solution of a desired problem or goal but, unlike classical gates, the gates in this quantum model are reversible.

Consider IBM's Qiskit for a moment, this gate-based quantum framework can not only solve a variety of problems, but it also shows significant speedups in problems related to integer factorization and solving linear system equations when compared to classical approaches. And this while using a few qubits to do so.

One of the problems with this model is related to qubit decoherence, this means that the states of qubits are destroyed by interactions with the environment, resulting in the state collapsing and altering the final result as a consequence. After overcoming this setback, a universal quantum computer is one step closer to be built and to use both classical and quantum models and achieve a synergetic performance [29].

IBM Q also has the short term goal to develop scalable quantum systems and the long term goal to create a universal quantum computer, a system that would be able to solve any kind of computational problem in a quantum environment, that is so sought after in the field.

Available through an open-source framework, anyone can use the various tools and components of Qiskit to create a quantum algorithm. All that is needed is to create an account in the IBM Quantum Experience page or link various other services like a Github or a Google account to automatically create an account and sign in. With this, several options to create a program exist: either use the direct circuit composer to directly embed a circuit through a visual aid that represents the gates in the overall circuit, or use the quantum labs section to create a Jupyter notebook and program the circuit in python, or even make use of the API to program it through external means.

With both a free plan and a premium plan, Qiskit allows users to run code on an actual quantum computer, although there tends to be a waiting queue to run the programs on a given system. For simple enough programs, or even just for learning purposes, Qiskit also provides with a a simulator to run quantum code on a simulated environment of a classical computer.

The available systems for a free user can be seen in Table 2.2, where quantum volume is a metric that measures the overall capabilities of a quantum computer and its error rates, while the ones in Table 2.1 refer to the ones exclusively available to the IBM Quantum Network. This Network is a premier access to Fortune 500 companies, academic institutions, research labs and even some startups that work together with IBM to help develop quantum computing even further.

A table referencing the available simulators in Qiskit are also listed in Table 2.3.

The public IBM computers tend to stay around the 5 qubit order, opting instead to improve

System	Qubits	Quantum Volume	QPU type
ibmq_montreal	27	128	Falcon r4
ibmq_kolkata	27	128	Falcon r5.11
ibmq_mumbai	27	128	Falcon r5.1
ibmq_dublin	27	64	Falcon r4
ibmq_hanoi	27	64	Falcon r5.11
ibmq_cairo	27	64	Falcon r5.11
ibmq_manhattan	65	32	Hummingbird r2
ibmq_brooklyn	65	32	Hummingbird r2
ibmq_toronto	27	32	Falcon r4
ibmq_sydney	27	32	Falcon r4
ibmq_guadalupe	16	32	Falcon r4P
ibmq_casablanca	7	32	Falcon r4H
ibmq_lagos	7	32	Falcon r5.11H
ibmq_nairobi	7	32	Falcon r5.11H
ibmq_jakarta	7	16	Falcon r5.11H

Table 2.1: Available systems on Qiskit for the IBM Quantum Network as of August 2021

System	Qubits	Quantum Volume	QPU type
ibmq_santiago	5	32	Falcon r4L
ibmq_manila	5	32	Falcon r5.11L
ibmq_bogota	5	32	Falcon r4L
ibmq_quito	5	16	Falcon r4T
ibmq_belem	5	16	Falcon r4T
ibmq_lima	5	8	Falcon r4T
ibmq_armonk	1	1	Canary r1.2

Table 2.2: Available systems on Qiskit for free users as of August 2021

System	Qubits	Simulator Type
ibmq_simulator_stabiliz	5000	Clifford simulator
ibmq_simulator_mps	100	Matrix Product State
ibmq_simulator_extende	63	Extended Clifford (eg. Clifford+T)
ibmq_simulator_stateve	32	Schrödinger wavefunction
ibmq_qasm_simulator	32	General, context-aware

Table 2.3: Available simulators on Qiskit for all users as of August 2021

the overall performance and error rate as seen in the Quantum Volume of some of the listed at the top. The simulators in table 2.3 is just a classical computer running simulated quantum code to solve those more simple problems, or usually used during tutorials to help speedup the process without having to wait in a queue for the results.

To start programming on Qiskit first one must learn about the gates that interact with a qubit. These gates can also be represented by a Pauli matrix since the operations are based on the algebra that defines the quantum field. The details that follows is inspired by the Qiskit Documentation [40]

The most elementary gates are:

An X Gate $X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$

An Y Gate $Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}$

An Z Gate $Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$

How a gate operation affects the state of a qubit: $X|0\rangle = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} = |1\rangle$

But with just these Pauli-gates it is impossible to reach a state other than $|0\rangle$ or $|1\rangle$. This essentially allows for the same operations as a classical bit, but there's a bit more to it than just that. Now this is where one of the most fundamental Quantum gate comes in.

An Hadamard Gate $H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$

With this the qubit can escape the poles of the Bloch sphere and takes a state of superposition of $|0\rangle$ and $|1\rangle$. Two states can be derived this way:
$H|0\rangle = |+\rangle$
$H|1\rangle = |-\rangle$

This way it is possible to reach the state of what is called an eigenstate of the X-gate:
$|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 1 \end{bmatrix}$
$|-\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ -1 \end{bmatrix}$

An R_ϕ-Gate $R_\phi = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\phi} \end{bmatrix}$ with ϕ as a real number.

This gate is a bit special, in the way that it is parametrized. This gate performs a rotation of ϕ around the X/Y/Z-axis, depending on what is needed.

Aside from these gates there's also S and T gates which are just an R_ϕ-Gate on the Z-axis with ϕ values of $\phi = \pi/2$ and $\phi = \pi/4$, respectively.

Still, these individual operations hold little value when compared to a classical computer. it is when qubits interact with each other that Quantum Computing truly shines.

A CNOT-Gate essentially applies an X-gate to a qubit if the control Qubit is $|1\rangle$. A representation of this gate can be viewed in Figure 2.2.

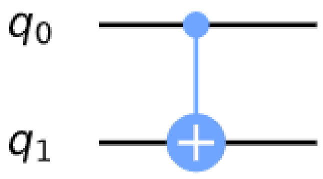


Figure 2.2: CNOT-Gate (Qiskit)

This CNOT gate is important, because when it is used together with an Hadamard Gate, an entanglement is created between two qubits. A representation of what an entanglement process would look like on a quantum circuit can be viewed in Figure 2.3.

Aside from these there is also a swap gate that allows two qubits to be swapped. It's a simple gate that allows some specific algorithms to work.

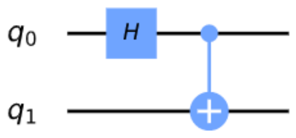


Figure 2.3: Entanglement (Qiskit)

With these, now a myriad of problems can be represented in a quantum circuit and solved. The size, and complexity, of the problems depend on the number of qubits available.

Adiabatic Quantum Computers

This thesis will be focusing on the adiabatic systems, and the quantum annealing procedure of these systems, provided by D-WAVE since they are known for being commercially available quantum computers. Instead of going for the familiarity of a gate-based architecture like IBM's Qiskit, D-WAVE opts instead to follow the route of quantum annealing. This quantum counterpart of simulated annealing uses quantum fluctuations instead of the thermal ones the simulated kind uses. Unlike the versatile gate circuit model, these quantum annealing machines are built specifically for a purpose, one that is usually optimization, machine learning, among others [29].

When comparing to the gate model Qiskit uses, there is the advantage that the quantum annealing model is much more resistant when it comes to state interference due to external noise, and therefore has less of a need to divert efforts towards error correction and therefore losing out on potential processing power, and also has the ease to have much higher numbers of qubits in the hardware. The drawback of this model is that there are very few problems, since it's focusing more on the annealing process and can't compete with the flexibility of a universal quantum computer, which can be solved exponentially faster when comparing to classical algorithms [29].

The D-WAVE systems, commercially available, have been steadily evolving over time as can be seen in Table 2.4

System	Release date	Qubits	Couplers	Josephson junctions
D-Wave One	May 2011	128	352	24,000
D-Wave Two	May 2013	512	1,472	-
D-Wave 2X	August 2015	1,152	3,360	128,000
D-Wave 2000Q	January 2017	2,048	6,016	128,472
Advantage	2020	5,640	40,484	1,030,000

Table 2.4: Available systems on D-WAVE [2]

The biggest evolution would have to be the Advantage system. Compared to the previous D-WAVE 2000Q system, not only has the available qubits essentially doubled, but the couplers and Josephson junctions have increased explosively. These couplers are connectors responsible for connecting a qubit to its neighbors inside and outside their own units, units that consist of a group of intertwined qubits, and the Josephson junctions are responsible for ensuring the superconductive properties of the circuit so that electrons can travel with minimal resistance.

A brief explanation on how quantum annealing works, based on the D-WAVE documentation [43], follows.

D-WAVE's systems implement a qubit as a circulating current and a corresponding magnetic field.

As seen in Figure 2.4 the process begins with a single well where the qubit is in superposition and only one minimum exists (a). As the annealing process runs and the barrier is raised, creating what is know as a double-well potential (b) where the chances of the qubit ending in one state is the same as the other.

But that's not everything, the process is not just random, the probabilities of individual states can be modified by applying an external magnetic field to the qubit (c). This destroys the equilibrium and raises the odds of the qubit ending up in a certain state. The programmable quantity that controls this external magnetic field is referred to as a bias, this bias pushes the qubit into collapsing into the most probable state, and therefore lowering the energy of the system, and falling into the lower well.

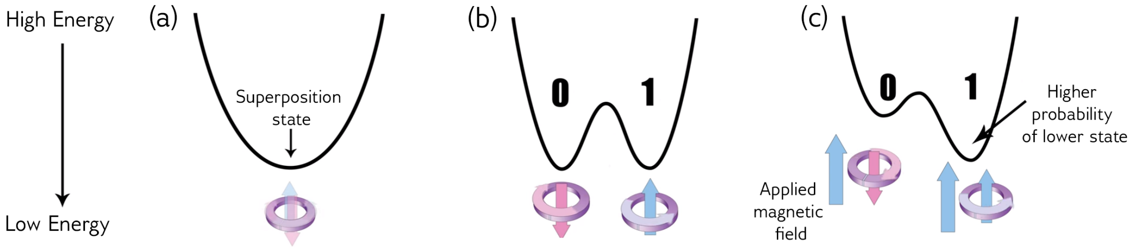


Figure 2.4: Quantum Annealing process and bias (D-WAVE)

The bias alone is not the only variable able to influence these qubits though. As in the gate model, the advantage of this system reveals itself when you link qubits together.

By entangling qubits together through the use of the couplers linking each qubit together, it is possible to influence more than one qubit at once, as if they were a single entity. This coupler strength is a correlation that is also a programmable setting that is defined beforehand.

This process can be seen better in Figure 2.5 where the energy of each state, influenced by the bias on the qubits and the coupler strength between them, leads to this landscape where the qubit states are eventually settling on the minimum energy state of (1,1).

The values chosen for the bias and coupler strength are just as important if not more than the problem formulation. These properly define an energy landscape so that the annealer can find the minimum energy state of the problem. This landscape becomes exponentially more complex as more qubits are added, doubling the possible states, of entangled qubit, for every qubit added. If the Advantage system in Table 2.2 is considered and all qubits used, this leads to an astronomical number, 2^{5640}, of possible states.

This is the advantage of Quantum Annealing. Starting in a state of superposition, after the biases and couplers, couplers that are entangling qubits together, are introduced and the quantum annealing process begins. By the end every state is in a classical state that represents the minimal energy of the problem , or one that is very close to it. This process takes a mere matter of microseconds from start to finish.

In D'Wave's quantum processors qubits are ordered in what is called a Unit Cell, where each cell has eight qubits and each qubit is connected to twelve other ones through internal couplers, and then the qubits of each unit cell is connected to other adjacent ones through

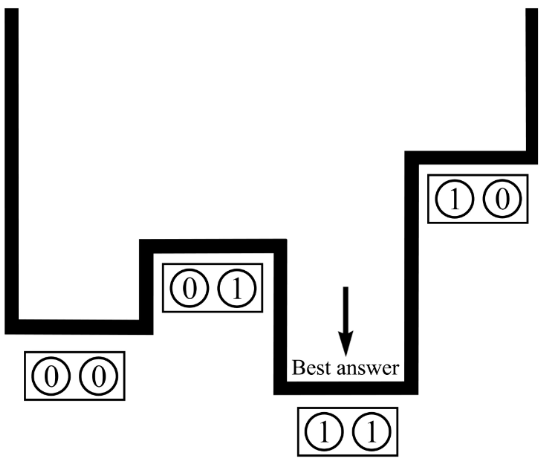


Figure 2.5: Energy well of 2 entangled qubits (D-WAVE)

external couplers.

Since a single qubit is representing a given variable, for problems where there is a need to represent a variable in more than 12 places at once a problem arises. The workaround for this is what is called a chain, a chain is a link of several correlated qubits that exist to represent the same variable throughout the several connections it has. But, the problem's not fixed just yet.

A chain break can also occur, that is, when the link of qubits that represents a single variable, has conflicting states when they are measured after the annealing. This conflict is usually resolved by the default method of majority vote, where the most common state is the deciding one, but there are still situations where wrong values can happen.

There is also a parameter called chain strength, that determines how strongly correlated the linkage between the qubits in a chain should be, and a higher value will lower the odds of a chain break happening. To note that the default value that D-WAVE uses for this parameter is completely inadequate for most problems that are of a higher complexity.

This was a quick overview of how the workings behind the quantum annealing systems of D-WAVE but one detail still remains, the quantum process making it work.

The Hamiltonian is a function, or more precisely an operator, that describes the system's energy, both kinetic and potential energy. The process of exploring the concepts of Hamiltonians is what makes the annealing process in these quantum annealers work.

Figure 2.6 shows how exactly this process works. Given an initial Hamiltonian, prepared in the lowest-energy eigenstate and serves as the starting point, on the left and an arbitrary number of other eigenstates, the Hamiltonian evolves into the desired final Hamiltonian that describes the problem to solve, using the biases and couplers to alter the initial Hamiltonian.

At the end of the annealing process, the ground state represents the solution for the problem. This is, of course, if the adiabatic process is done slowly, if there is no external influence compromising this process and if what is represented in the figure as the minimal gap is large enough. This gap matters because if the gap between eigenstates is sufficiently small enough, the ground state can jump into a higher energy state and the problem would then return a near optimal solution. Although this circumstance is to be avoided if possible,

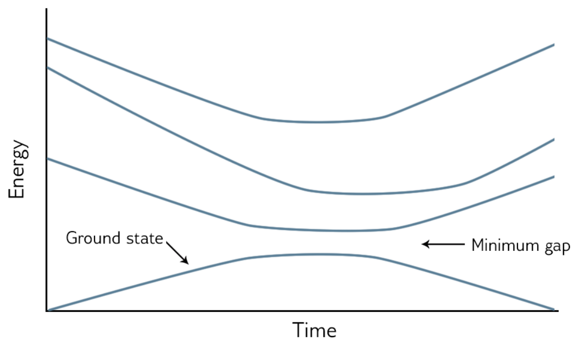


Figure 2.6: Eigenspectrum and the evolution of an Hamiltonian (D-WAVE)

the near optimal solutions can still be very useful.

Chapter 3

Development and Solution

This section will explain the steps required to implement the Unit Commitment Problem onto a quantum annealer and obtain a solution.

Firstly, since the D-WAVE API receives a QUBO formulation as input, the base form for the problem must be translated into QUBO. However, the problem must be first discretized due to its non-linear nature not being compatible with the QUBO formulation.

With the QUBO form ready what follows is the actual implementation. Through the use of a simple API call through a python program it is possible to obtain a solution to the formulated problem.

Then, to compare the solution obtained against a proven optimal solution, an implementation in a classical solver was also done. GLPK was chosen as the solver to obtain the optimal solutions to our test cases.

3.1 Problem Formulation

This section details the approach of the transition from the MILP formulation of the Unit Commitment Problem to the QUBO formulation. Then, the latter is transformed in a matrix form in order to be read by D-WAVE.

Problem 3.1 gives the formulation for the Unit Commitment Problem, where U is the index set for the units, constants A_i, B_i and C_i, $i \in U$, correspond to the coefficients that are related to the quadratic cost function for a unit i that is represented as the objective function pertaining to the production cost values, and L is the power demand that must be reached. The last constants represent the minimum and the maximum value of the range of the possible production values of each unit i, p_i, and the respective range where it can fluctuate.

$$\begin{aligned}
&\min \sum_{i \in U} A_i y_i + B_i p_i + C_i p_i^2 \\
&\text{s.t.} \\
&\quad \sum_{i \in U} p_i = L \\
&\quad P_{min,i} y_i \leq p_i \leq P_{max,i} y_i \quad \forall i \in U \\
&\quad y_i \in \{0, 1\}
\end{aligned} \tag{3.1}$$

The variables of this problem are y_i, which is the binary variable that corresponds to switching on a unit i ($y_i = 1$) or switching it off ($y_i = 0$) and p_i that denotes the production value of a unit i. The first constraint limits the global production values to be equal to the demand (L).

Aside from this constraint there is also the domain space for the variable p_i. Production values start at $P_{min,i}$ and, as the thermal unit heats up, the production increases until it reaches the maximum allowed production, $P_{max,i}$ for the unit. Note that p_i takes value 0 if unit i is switched off.

In order to transform the problem above in a QUBO formulation p_i, it must be first discretized as suggested in [29]. Let N be the set of limited values the continuous p_i will be split into, here forth referred to as grids. We define $D_{ik} \in U, k \in \{1, ..., N+1\}$ as follows.

$$D_{ik} = P_{min,i} + (k-1) \times \left(\frac{P_{max,i} - P_{min,i}}{N}\right) \tag{3.2}$$

The value D_{ik} is the discretization of p_i at a certain production level k. More precisely it is the kth value out of the N different values that p_i was split into. The domain of k is expanded to $N+1$ to be able to reach the max value of production, since formulation uses $(k-1)$ as a multiplier to pick the appropriate production value.

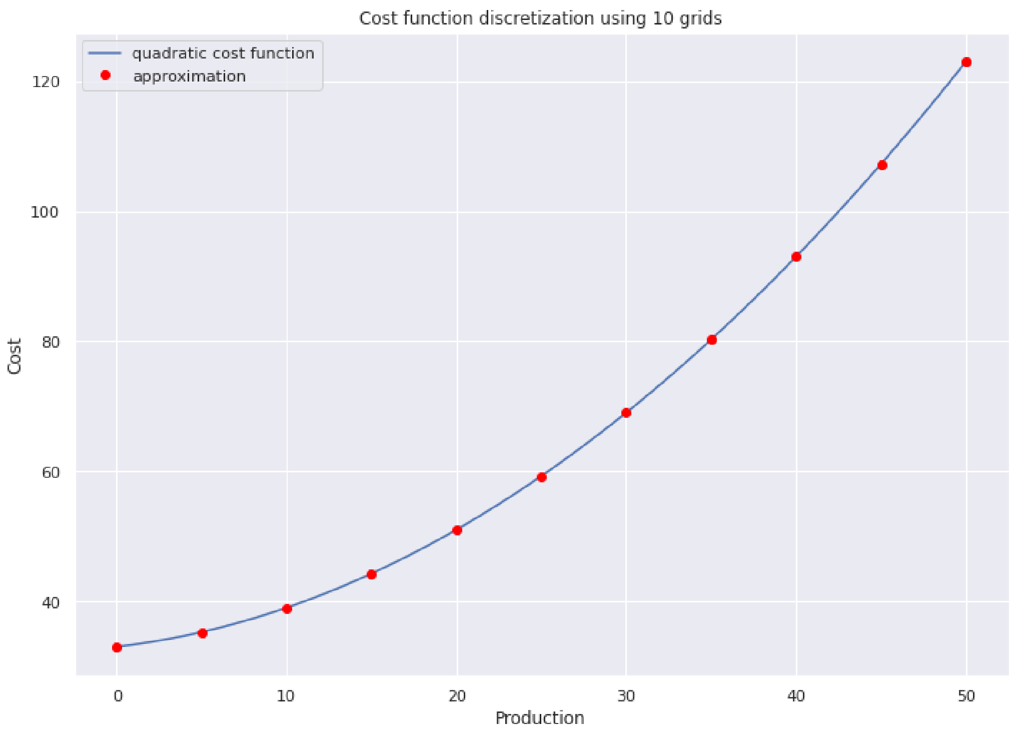


Figure 3.1: Discretization example using 10 grids

Figure 3.1 shows the effect of this discretization process for 10 grids. In this case, the approximation error obtained is around 14.28% and if the grids were to be doubled then the error would lower to 6.6%.

Problem 3.3 show the new formulation based on the discretization done previously in

Equation 3.2.

$$
\begin{aligned}
&min \sum_{i\in U} A_i\left(1-v_i\right) + B_i\left(\sum_{k=1}^{N+1} D_{ik}z_{ik}\right) + C_i\left(\sum_{k=1}^{N+1} D_{ik}z_{ik}\right)^2 \\
&\text{s.t.} \\
&\quad v_i + \sum_{k=1}^{N+1} z_{ik} = 1 \quad \forall i \in U \\
&\quad \sum_{i\in U}\sum_{k=1}^{N+1} D_{ik}z_{ik} = L \\
&\quad v_i, z_{ik} \in \{0,1\}
\end{aligned}
\tag{3.3}
$$

The process of discretization leads to some amount of error in the approximation which means that the problems 3.1 and 3.3 are not outright equivalent. At a sufficiently high amount of grids in N the approximation becomes so close that the error becomes negligible.

Another noticeable change is the new variables v_i and z_{ik} acting as a replacement for the previous y_i variable in Problem 3.1 and v_i takes the role of a nullifier for the coefficient A_i when unit i is offline. This means that when $v_i = 1$ then that unit cannot have any active production levels, with $z_{ik} = 0$ for all k. Similarly, if $z_{ik} = 1$ for any k then $v_i = 0$.

3.1.1 Translation to QUBO

In the following, the transformation of Problem 3.3 into a QUBO is described. Before proceeding any further, the squared summation on the third term of the objective function needs to be expanded. The expansion of the quadratic term of the objective function in 3.3 is as follows:

$$
\sum_{i\in U} C_i\left(\sum_{k=1}^{N+1} D_{ik}z_{ik}\right)^2 = \sum_{i\in U}\sum_{k=1}^{N+1} C_i D_{ik}^2 z_{ik} + \sum_{i\in U}\sum_{k=1}^{N+1}\sum_{m=k+1}^{N+1} 2C_i D_{ik} D_{im} z_{ik} z_{im} \tag{3.4}
$$

Problem 3.3 can be directly translated to QUBO form. Since they are both constraints based on equality, the QUBO equivalent for an equality constraint is used, where Δ_A and Δ_B are the delta values that determine the weight that each of the two constraints have in the objective function.

The objective function of QUBO is as follows

$$\begin{aligned} \min \sum_{i\in U}\sum_{k=1}^{N+1} & \left(A_i + B_i D_{ik} + C_i D_{ik}^2\right) z_{ik} \\ & + \sum_{i\in U}\sum_{k=1}^{N+1}\sum_{m=k+1}^{N+1} 2C_i D_{ik} D_{i,m} z_{ik} z_{im} \\ & + \Delta_A \sum_{i\in U}\left(\sum_{k=0}^{N+1} z_{ik} - 1\right)^2 \\ & + \Delta_B \left(\left(\sum_{i\in U}\sum_{k=1}^{N+1} D_{ik} z_{ik}\right) - L\right)^2 \end{aligned} \tag{3.5}$$

Note that the first constraint in Problem 3.3 is now in the objective function of Problem 3.5, which has a summation outside the actual constraint. That is because the goal is to constrain each unit to only a single production level, so this can be viewed as several constraints joined in a single term rather that a single constraint applying to all the intervening variables. Aside from this, variable v_i was also replaced and is now considered as $z_{ik} = 0$, causing the first constraint to now start at $k = 0$ to accommodate for this change.

There is no need to modify the domain of the other summations aside from the first constraint since any value resulting from the interaction between two variables would amount to $D_{ik} \times D_{i(0)} = 0$, since any production level of a unit where $k = 0$ is also 0 and would not change the final matrix form. This means that only that first constraint would have in the summation with a starting value of $k = 0$. This will become more clear with the expansion in problem 3.7.

Before proceeding any further, the quadratic terms must still be expanded so that a matrix notation is reached. The quadric term corresponding to the first constraint can be expanded as follows (see [44] for more details):

$$\Delta_A \sum_{i\in U}\left(\sum_{k=0}^{N+1} z_{ik} - 1\right)^2 = \Delta_A \sum_{i\in U}\left(\sum_{k=0}^{N+1} z_{ik}^2 + 2\sum_{k=0}^{N+1}\sum_{j=k+1}^{N+1} z_{ik} z_{ij} - 2\sum_{k=0}^{N+1} z_{ik} + 1\right) \tag{3.6}$$

For the sake of simplicity consider $x_{ik} = D_{ik} z_{ik}$

$$\begin{aligned} \Delta_B \left(\sum_{i\in U}\sum_{k=1}^{N+1} x_{ik} - L\right)^2 = & \Delta_B \left(\sum_{i\in U}\sum_{k=1}^{N+1} x_{ik}^2\right) \\ & + \Delta_B 2 \left(\sum_{i\in U}\sum_{k=1}^{N+1}\left(\sum_{m=k+1}^{N+1} x_{ik} x_{im} + \sum_{j=i+1}^{U}\sum_{m=1}^{N+1} x_{ik} x_{jm}\right)\right) \\ & - \Delta_B 2L \left(\sum_{i\in U}\sum_{k=1}^{N+1} x_{ik}\right) + L^2 \end{aligned} \tag{3.7}$$

3.1.2 Relaxed Qubo Formulation

Due to the results of the initial tests, a relaxation of Problem 3.7 was considered by changing the constraint type of the second constraint, related to the production meeting Demand, from an equality type constraint into an inequality. This allowed the problem to consider solutions where some production, to an extent, that goes beyond the Demand to be considered feasible. What follows is original part in equation 3.8 and then the reformulated part in equation 3.9:

$$\sum_{i \in U} \sum_{k=1}^{N+1} D_{ik} z_{ik} = L \tag{3.8}$$

$$\sum_{i \in U} \sum_{k=1}^{N+1} D_{ik} z_{ik} >= L \tag{3.9}$$

This change in QUBO can be observed with the original formulation in Equation 3.10 and the relaxed Equation 3.11

$$\Delta_B \left(\sum_{i \in U} \sum_{k=1}^{N+1} D_{ik} z_{ik} - L \right)^2 \tag{3.10}$$

$$\Delta_B \left(\left(\sum_{i \in U} \sum_{k=1}^{N+1} (D_{ik} z_{ik}) + \sum_{s=1}^{S+1} \left(-2^{s-1}\right) \ell_s \right) - L \right)^2 \tag{3.11}$$

In order to deal with the inequality, slack variables were added to the second constraint to allow for some relaxation towards the required production to meet demand. S is the amount of slack variables considered and ℓ_s is a decision variable that activates certain slack level s. For instance, if 5 slack variables are to be considered, that would allow the problem to consider feasible solutions that have up to a surplus of $2^0 + 2^1 + 2^2 + 2^3 + 2^4 = 31$. Adding these slack variables may seem to complicate the expansion but if these slack variables are considered to be a counterpart of z_{ik} that has a negative production value instead, then the expansion is done in the same way as shown previously in equation 3.7.

3.1.3 QUBO Matrix formulation

In the following, the matrix form of Problem 3.5 is presented. For the sake of visual clarity $M = N + 1$ was considered.

Some abbreviations were also made to save space:

- $prod(i,k) = D_{ik} = \left(P_{min,i} + (k-1)\left(\frac{P_{max,i} - P_{min,i}}{N}\right)\right)$
- $cost(i,k) = A_i + B_i \times prod(i,k) + C_i \times prod(i,k)^2$
- $A(i,k,\ell) = 2C_i \times prod(i,k) \times prod(i,\ell)$
- $B(i,k,j,\ell) = 2 \times prod(i,k) \times prod(j,\ell)$

Also note that when $k = 0$ both $prod(i, 0)$ and $cost(i, 0)$ return the value of 0 since $k = 0$ is representing a unit in an off state.

In the following, the QUBO matrix formulation (Equation 3.13) of the first terms of the objective function in Problem 3.5 (Equation 3.12) is shown.

$$\begin{aligned}\min \sum_{i\in U}\sum_{k=1}^{N+1}\left(A_i + B_i D_{ik} + C_i D_{ik}^2\right) z_{ik} \\ + \sum_{i\in U}\sum_{k=1}^{N+1}\sum_{m=k+1}^{N+1} 2C_i D_{ik} D_{i,m} z_{ik} z_{im}\end{aligned} \tag{3.12}$$

$$\text{Let } W_i = \begin{bmatrix} 0 & 0 & \dots & \dots & \dots & 0 \\ 0 & cost_{(i,1)} & A_{(i,1,2)} & \dots & A_{(i,1,M-1)} & A_{(i,1,M)} \\ \vdots & 0 & cost_{(i,2)} & \dots & A_{(i,2,M-1)} & A_{(i,2,M)} \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & cost_{(i,M-1)} & A_{i,M-1,M} \\ 0 & \dots & \dots & \dots & 0 & cost_{(i,M)} \end{bmatrix} \text{ and let W} = \begin{bmatrix} W_1 & 0 & \dots & 0 \\ 0 & W_2 & \dots & 0 \\ \vdots & \ddots & \ddots & 0 \\ 0 & 0 & \dots & W_U \end{bmatrix} \tag{3.13}$$

W allows to represent the first two terms of Problem 3.5 with a matrix of size $(M + 1) \times (M + 1)$.

In the following is the QUBO matrix form for the first constraint (Equation 3.15), with the original QUBO form corresponding to the third term in Problem 3.5 (Equation 3.14)

$$\Delta_A \sum_{i\in U}\left(\sum_{k=0}^{N+1} z_{ik} - 1\right)^2 = \Delta_A \sum_{i\in U}\left(\sum_{k=0}^{N+1} z_{ik}^2 + 2\sum_{k=0}^{N+1}\sum_{j=k+1}^{N+1} z_{ik} z_{ij} - 2\sum_{k=0}^{N+1} z_{ik} + 1\right) \tag{3.14}$$

$$\text{Let } T_i = \begin{bmatrix} 0 & 1 & 1 & 1 & \dots & 1 \\ \vdots & 0 & 1 & 1 & \dots & 1 \\ \vdots & \ddots & 0 & 1 & \dots & 1 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 & 1 \\ 0 & \dots & \dots & \dots & \dots & 0 \end{bmatrix} \text{ and let T} = \begin{bmatrix} T_1 & 0 & \dots & 0 \\ 0 & T_2 & \dots & 0 \\ \vdots & \ddots & \ddots & 0 \\ 0 & 0 & \dots & T_U \end{bmatrix} \tag{3.15}$$

Where T_i is a matrix of size $(M + 1) \times (M + 1)$

$$Constraint1 = \Delta_A \times I_{(U\times(M+1))\times(U\times(M+1))} + 2\Delta_A \times T - 2\Delta_A \times I_{(U\times(M+1))\times(U\times(M+1))} + U\Delta_A \tag{3.16}$$

Now follows the second constraint in QUBO matrix form, followed first by the original

QUBO form:

$$\begin{aligned}\Delta_B\left(\sum_{i\in U}\sum_{k=1}^{N+1}x_{ik}-L\right)^2 =&\Delta_B\left(\sum_{i\in U}\sum_{k=1}^{N+1}x_{ik}^2\right)\\ &+\Delta_B 2\left(\sum_{i\in U}\sum_{k=1}^{N+1}\left(\sum_{m=k+1}^{N+1}x_{ik}x_{im}+\sum_{j=i+1}^{U}\sum_{m=1}^{N+1}x_{ik}x_{jm}\right)\right)\\ &-\Delta_B 2L\left(\sum_{i\in U}\sum_{k=1}^{N+1}x_{ik}\right)+L^2\end{aligned} \tag{3.17}$$

$$\begin{aligned}&Constraint2 = \Delta_B \times I_{(U\times(M+1))\times(U\times(M+1))}\\ &+\Delta_B\begin{bmatrix} 0 & B_{(1,0,1,1)} & B_{(1,0,1,2)} & \cdots & B_{(1,0,1,M)} & B_{(1,0,2,0)} & \cdots & B_{(1,0,U,M)}\\ & 0 & B_{(1,1,1,2)} & \ddots & \ddots & \ddots & \ddots & B_{(1,1,U,M)}\\ & & 0 & \ddots & \ddots & \ddots & \ddots & \vdots\\ & & & 0 & B_{(1,M,2,M)} & \ddots & \ddots & B_{(1,M,U,M)}\\ & & & & 0 & B_{(2,0,2,1)} & \ddots & B_{(2,0,U,M)}\\ & & & & & 0 & \ddots & \vdots\\ & & & & & & 0 & B_{(U,M-1,U,M)}\\ & & & & & & & 0\end{bmatrix}\\ &-2\times L\times\Delta_B\times I_{(U\times(M+1))\times(U\times(M+1))}\end{aligned} \tag{3.18}$$

These last two matrix representations are simple to understand. The more straightforward terms that do require a matrix to represent are instead multiplying with an identity matrix because those two terms only affect the diagonal portion of the matrix and can be split off to simplify the representation of the more complex matrix.

In the second constraint, unlike the one in equation 3.13 and 3.15 all the production levels of each unit are supposed to be connected to each and every single level of production, regardless of which unit it belongs to. This is because the goal is to equal all active units in U with a certain level k of production to the given Demand.

3.2 D-Wave

This section showcases a code snippet from the python program that builds the matrix that will be sent to D-WAVE for solving as well as some insight on some important functions and procedures in the D-WAVE API.

There are two routines that have great importance in the program:

```
        import networkx as nx

        sampler = DWaveSampler()
        embedding = find_embedding(nx.complete_graph(scope).edges(),
    sampler.edgelist)
```

```
        embedding = find_clique_embedding(scope, target_graph=sampler.to_networkx_graph())

```

Listing 3.1: Find Embedding

```
1       composite = FixedEmbeddingComposite(sampler, embedding=embedding)
2       response = composite.sample_qubo(Qubo, chain_strength, num_reads, label)
3
```

Listing 3.2: API call that solves the QUBO

The first one is mainly responsible for finding an embedding for the problem, or more specifically to the number of variables used, in the quantum processor. This is done to allow the mapping of the QUBO onto the QPU and is highly recommended to save the embedding since it takes a sizable amount of time to find an embedding when a large number of variables is used. This can be done by dumping the embedding onto a JSON file, for example, and then reading it once that specific embedding is required again.

The second one is the actual routine call that sends the QUBO problem to D-WAVE for solving. The function in line 2 has many more parameters aside from the ones listed, but these could be considered the essential ones with the label being the name shown in the D-WAVE website when browsing the problems solved.

The code related to the implementation of the QUBO matrix form in python can be found in appendix 5, while any other code and programs can also be accessed in the public github at https://github.com/Pmsilva1/UC_Solver. With the program related to the D-WAVE implementation found inside the QUBO folder with the name *uc_ dwave.py*.

3.2.1 Verification

For the purpose of verifying whether the implementation of the formulation in python was free of errors, a simple method was used: the QUBO matrix being created in the code to send to D-WAVE was split and saved onto different matrixes where each one represents a different section of the QUBO formulation in matrix form. These different sections are split into the matrix form of objective function, the matrix form for the first constraint, the matrix form of the second constraint, and the matrix form of the second constraint with the modified constraint type used.

Then, with the solution that is obtained from the D-WAVE system the final value is calculated through the QUBO formula using the matrix notation, with x being the solutions to the problem and Q the QUBO formulation in a matrix form:

$f_Q(x) = x^TQx$

The QUBO implementation code can be found previously in section 3.2 on page 25

To check against this value the different sections of the QUBO formulation was iterated in python through loops while using the summations from problem 3.5 as a reference and calculating the final value of the problem using a set of solutions obtained previously through the D-WAVE solver.

This verification did not find any errors in the code implementation for the problem formulation but, it does not guarantee that the formulation that is based on is correct or that the implementation is truly error free.

3.3 GLPK

This section will follow some snippets of the code developed in GLPK to explain the process behind implementing the original problem formulation in problem 3.3 in this environment and reintroduced in problem 3.19.

$$
\begin{aligned}
& min \sum_{i \in U} \left(\sum_{k=1}^{N+1} \left(A_i + B_i D_{ik} + C_i D_{ik}^2 \right) z_{ik} + \sum_{m=k+1}^{N+1} 2 C_i D_{ik} D_{im} w_{ikm} \right) \\
& \text{s.t.} \\
& \quad \sum_{k=1}^{N+1} z_{ik} = 1 \quad \forall i \in U \\
& \quad \sum_{i \in U} \sum_{k=1}^{N+1} D_{ik} z_{ik} = L \\
& \quad \sum_{m=1}^{N+1} w_{ikm} \leq z_{ik} \quad \forall i \in U, \forall k \in \{1, ..., N+1\} \\
& \quad \sum_{k=1}^{N+1} w_{ikm} \leq z_{im} \quad \forall i \in U, \forall m \in \{1, ..., N+1\} \\
& \quad w_{ikm} \geq z_{ik} + z_{im} - 1 \quad \forall i \in U, \forall k \in \{1, ..., N+1\}, \forall m \in \{1, ..., N+1\} \\
& \quad z_{ik}, w_{ikm} \in \{0, 1\}
\end{aligned}
\tag{3.19}
$$

Note that when $w_{ikm} = 1$ that means that $z_{ik} = z_{im} = 1$ and the opposite applies when $w_{ikm} = 0$ with $z_{ik} = z_{im} = 0$.

Creating the input files, a snippet of which can be seen in listing 3.3, is a simple and straightforward process. Line 1 and 2 show the value of U and *demand*, respectively, while lines 4 through 9 define the various values that the vector A has. To note that the *end;* in line 10 must be present at the end of the input files as well as the main GLPK file.

```
 1 param U := 5;
 2 param demand := 30;
 3 
 4 param : A :=
 5   1 49.1
 6   2 83.1
 7   3 61.4
 8   4 36.4
 9   5 79.1;
10 end;
```

Listing 3.3: Snippet of an input file

Listing 3.4 shows the definition of parameters that can either be defined directly, like N and prod, or imported from an input file like the rest of parameters. There is also the definition of the binary decision variables, in this case z and zw.

```
# Size of units and grids
param U;
param N := 5;

# Params
param demand;    # Power demand
param Pmin{1..U}; # min prod of unit
param Pmax{1..U}; # max prod of unit

# Quadratic Coefficients
param A{1..U};
param B{1..U};
param C{1..U};
param prod{i in 1..U, k in 1..N+1} := (Pmin[i] + (k-1)*((Pmax[i] - Pmin[i])
     / N ));

# Variables
var z{1..U, 1..N+1} binary;
var zw{1..U, 1..N+1, 2..N+1} binary;
```

Listing 3.4: Params and Vars

Lastly the formulation of the problem, shown in listing 3.5, is implemented almost directly onto the program through two major sections: objective function and constraints.

```
# Objective Function
minimize obj:
    sum{i in 1..U, k in 1..N+1} (z[i,k] * (A[i] + B[i]* prod[i,k] + C[i]*
    prod[i,k]**2)
    + sum{m in k+1..N+1} 2 * C[i] * prod[i,k] * prod[i,m] * zw[i,k,m] );

# Restrictions
s.t. zw1{i in 1..U, k in 1..N+1}:
    sum{m in k+1..N+1} zw[i,k,m] <= z[i,k];

s.t. zw2{i in 1..U, m in 2..N+1}:
    sum{k in 1..N+1} zw[i,k,m] <= z[i,m];

s.t. zw3{i in 1..U, k in 1..N+1, m in k+1..N+1}:
    zw[i,k,m] >= z[i,k] + z[i,m] - 1;

s.t. cdemand:
    sum{i in 1..U, k in 1..N+1} prod[i,k]*z[i,k] >= demand;

s.t. binary_z{i in 1..U}:
    sum{k in 1..N+1} z[i,k] <= 1;
```

Listing 3.5: Objective Function and Constraints

While similar to the formulation, some differences still exist. Variables z and zw are representing the same z_{ik} variable from the original problem, with the slight difference that zw is modelling the multiplication of $z_{ik}z_{im}$, the new constraints zw1,zw2 and zw3 are constricting this new zw so that it stays within the bounds of the original $z_{ik}z_{im}$ form. The variable v_i is also removed, since it's possible to keep the constraint inequality in this model.

Important Parameters

There are five important parameters in the D-WAVE program:

- $deltaP_A$
- $delta_B$
- $chainstrength$
- $constrainttype$

- *grids*

The first three parameters are essential to fine tuning any kind of program sent to D-WAVE, whereas constraint type and grids is more of a formulation change.

The first two, *deltaP*$_A$ and *delta*$_B$, are the familiar deltas that have been introduced in the formulation before, and are responsible for giving a proper weight to each restriction. In this case, since the first restriction deals with only values in the single digits, while the second one can scale much higher depending on production values and Demand, the once *delta*$_A$ was changed into a percentage form, where it will take a percentage of the highest value, or bias in quantum terms, of the created matrix for the second restriction, and only then apply it to the first restriction. This was made in mind to allow some sort of automation since without it, fine-tuning would be much more complex.

The *constrainttype* and *grids* parameters are related to the formulation of the problem in some way, where *constrainttype* changes whether the second constraint is an equality or inequality and *grids* is related to how accurate, the more grids the less error, the discretization is to the original continuous formulation.

And finally, *chainstrength*. This parameter is related to program behavior in D-WAVE and will be explained in more detailed in the next chapter.

3.4 Problems and Difficulties

- D-Wave documentation
- GLPK limitations
- Implementing the formulation

One of the biggest difficulties for implementing this formulation for a D-WAVE system was how structured the available documentation for the D-WAVE systems are. It followed along the lines of a typical programmer's type of documentation which is usually tailored for people already schooled in that language or program structure. But for someone who is considered entry-level on quantum computing, a lot of terms and parameters will be confusing and hard to understand, specially with some cases that barely have any useful description on what it does.

Aside from this there is also the lack of clear examples that help the reader learn and try as they explore the different terms and capabilities for the hardware, as done for example in Qiskit's case.

GLPK also has some limitations that did not allow to map the original Unit Commitment Problem since it has a non-linear nature and GLPK only solves linear problems. This was not much of a problem since it allowed both GLPK and D-WAVE implementations to solve a similar environment to compare solutions that could be attainable by both.

Another difficulty was in understanding the process of transforming the original Unit Commitment Problem to a QUBO form since the paper used as a support [29] had some mistakes and in some transformations had no mentions of the steps or formulas used to accomplish said transformation. This lead to lost time trying to correct the misunderstandings and searching for the appropriate steps for the conversion to fully understand the process.

This page is intentionally left blank.

Chapter 4

Experimental Analysis

This chapter describes an experimental analysis that allow us to understand the ability of D-WAVE to solve a typical formulation of the Unit Commitment problem and explore what optimal settings should be used as well as how the results compare to a simple classical approach of the same formulation that will be used as a basis for the most optimal solution for the problem, that is, the cheapest approach to produce enough power to meet demand. This chapter will begin with some informal tests, developed while writing the code, and some initial observations taken from the results. Afterwards, an in-depth experimental analysis with statistical testing is described to test the hypothesis raised.

4.1 Preliminary Experiments

Some preliminary experiments were conducted in D-WAVE in order to collect some first insights on its performance. In these first set of experiments, 120 variables were considered and the results observed were sub-optimal at best. Rarely were any of the constraints met and there seemed to be some randomness to the results, as solutions were only returned once and seemed to have no consistency as to be focused around the region of the most optimal solution for example.

After this, the number of variables used was reduced to 30 and the results were much more favorable. The results on this set of solutions actually contained the most optimal solution which was also returned twice in a run with 2000 shots.

This result is important because when solving smaller quantum problems, the most commonly returned solution is likely to be the most optimal solution to the problem, since the annealer reached this lower energy state several times, hinting that it might be one of the best solutions attainable.

But, with problems on this scale identical solutions that are returned more than once are a rare occurrence, and not always the most optimal solution. Due to this, processing the solutions is required to find the cheapest solution among them that also meet the constraints. A more detailed breakdown of the observations follows.

With the input values created initially with 120 variables, the results were a little underwhelming due to several units breaking as well as the production of several solutions not reaching the required demand.

It was also observed that by reducing the number of variables the severity of the solutions that failed to meet the constraints seemed to lower as well, while also having the rare solution that met both constraints. It is an expected result, since the number of variables is related to how hard the problem is to solve.

Aside from the impact of number of variables used, there is also the impact from the two constraints that must be factored in, as well as the second term of the objective function in equation 3.4. The correlations that these terms create between the variables of the problem increase the difficulty to directly embed the problem onto the D-WAVE quantum processors due to the limited number of connections between qubits, requiring use of chains to better represent the big net of connections that is created between the variables. But problems arise when these chains break, causing conflicting results for a single variable and further complicating the concept in quantum computing of "the most returned solution is often the best".

One important parameter to take note is chain strength, related to how strongly connected the qubits inside the chain are. At the time of writing the default value is 1, a value that is extremely overshadowed by the biases from the deltas for each constraint that can reach numbers in the four or five digits. These biases can be considered the strength or influence of a term in the QUBO form that ultimately leads to how the annealer decides which factors to consider to lower the system energy.

Considering that these biases are then normalized to values between 0 and 1, that chain bias is further made obsolete and the chain breaks continue. This might not be of great consequence for problems with few variables and with little to no correlations between the variables that do not need chains at all, since these correlations can be represented with the 15 couplers per qubit in the Pegasus system [45], but as the problem size grows the effects that default value for chain strength has is immediately noticeable, as the percentage of chain breaks grows considerably.

This leads to how the problems are actually embedded on the annealer. In this thesis these problems are sent to the Advantage D-WAVE system, also mentioned as Pegasus, designed for a total maximum of 5640 qubits. The actual qubits available vary from chip to chip but there are always at least 5000 working qubits in every chip [46].

It's possible to see how the embedding process influences the problem by using the D-WAVE inspector to see various details in the returned solutions. Figure 4.1 shows that the embedding for a problem with 35 variables is only taking up a minor portion of the processor while the embedding shown in Figure 4.2, with 125 variables, is taking about half of it. The amount of variables used between the figures go up by around three and a half and yet the amount of qubits used increases by more than ten times. This is most likely due to the amount of scaling that has to be done to represent the ever growing connections between variables and to that end, more and more chains need to be created, to the point that there are several chains to represent the connections of a single variable instead of a one chain per variable.

When observing all the solutions returned by these unformulated tests on a graph, a representative graph can be seen in Figure 4.4, the solutions seemed to be akin to randomly picked for a problem with 120 variables. To compare the performance of actual random solutions, a simple program that outputs a random solution to an identical problem sent to D-WAVE was created to check if a similar observation could be seen.

This program, developed in python, made use of the Random library and the randrange() function that uses a uniform distribution. The program ensures that the first constraint is always maintained by going through every unit available and picking a random level of

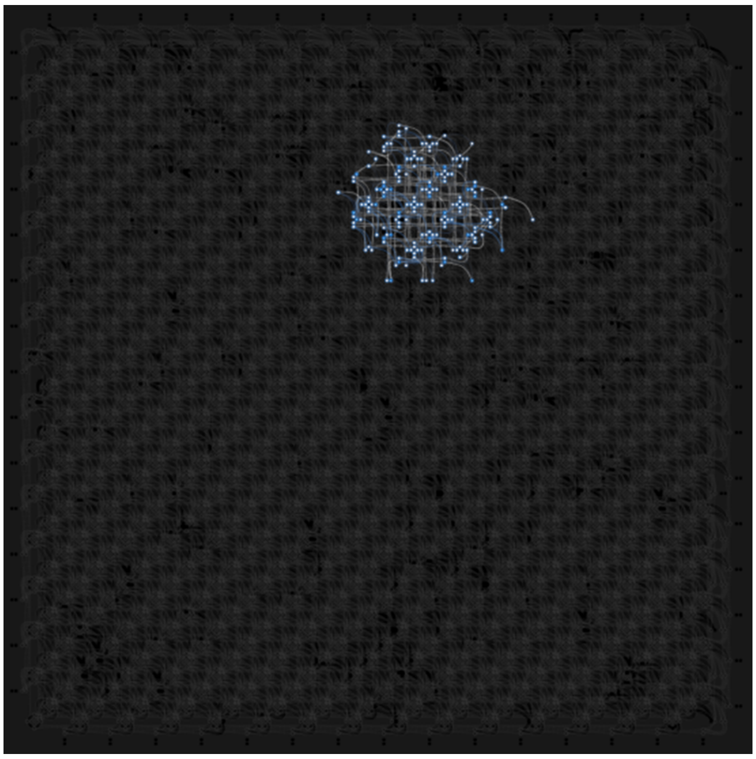


Figure 4.1: Embedding on D-WAVE with for a problem with 35 variables

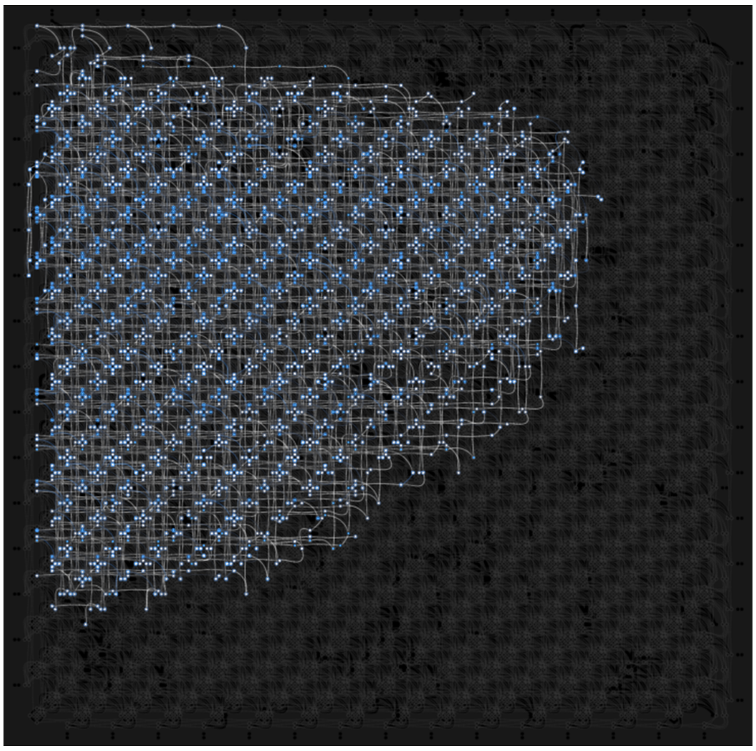


Figure 4.2: Embedding on D-WAVE with for a problem with 125 variables

production, and at the end outputs the resulting sum of what was produced in each unit and its cost.

With this, two test cases were made where they have same amount of different solutions and the same parameters for both the random generator and D-WAVE.

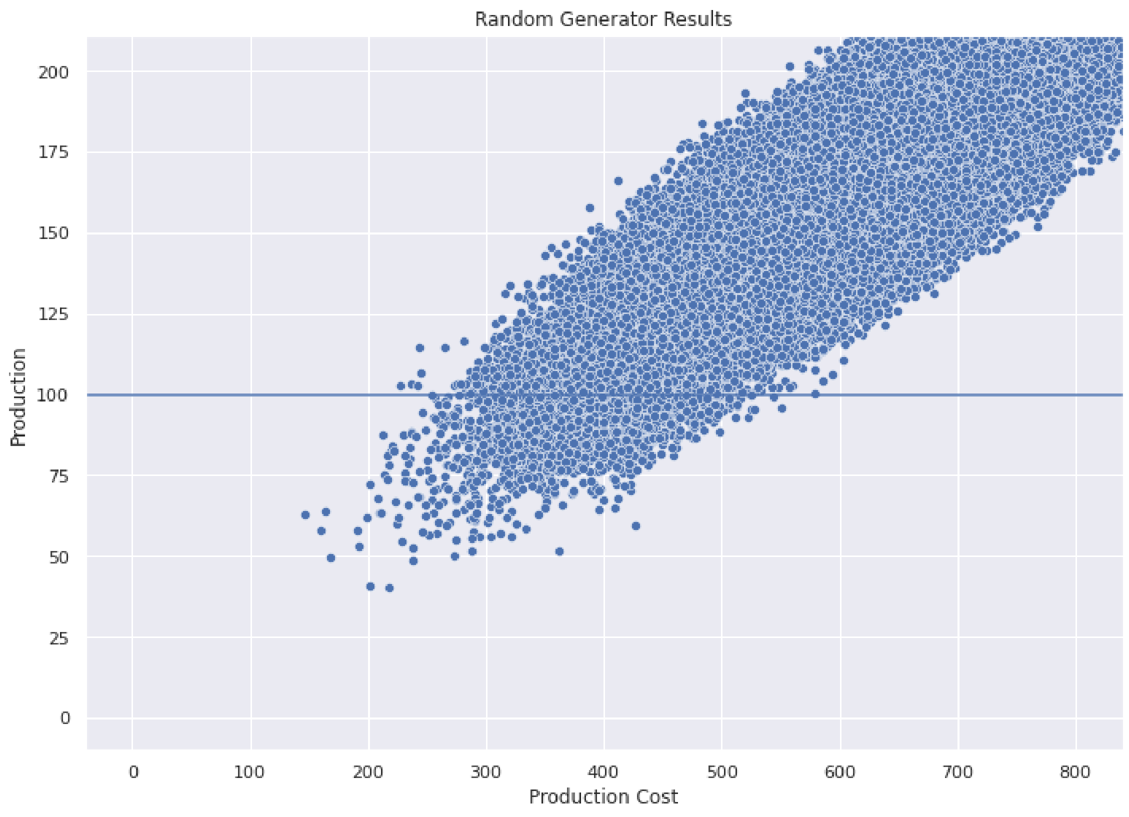


Figure 4.3: Results from random solution generator

Both figures show a line for a production value equal to 100 that corresponds to the required production to meet demand, to allow for an easier distinction from the feasible and unfeasible solutions each test case has.

Figures 4.3 and 4.4 show the results relating to the Random solution program and D-WAVE respectively and, at a glance it is possible to see that the random solution test case has many more valid solutions than the D-WAVE counterpart. This can be seen as a consequence to the random solution program always maintains the first constraint, therefore allowing to reach a bigger production value more easily. Even with this in mind, the results from D-WAVE still tend to be closer to the most optimal solutions returned by the classical solver.

4.1.1 Findings and Important Factors

Before discussing some initial results on how exactly the various solutions returned by the problem differ, some important factors and their function will be explained.

Independent Variables

- DeltaP_A - related to the Delta value acting on the first constraint, derived from a percentage value of the biggest bias in the QUBO matrix, previously mentioned in section 3.1.1.

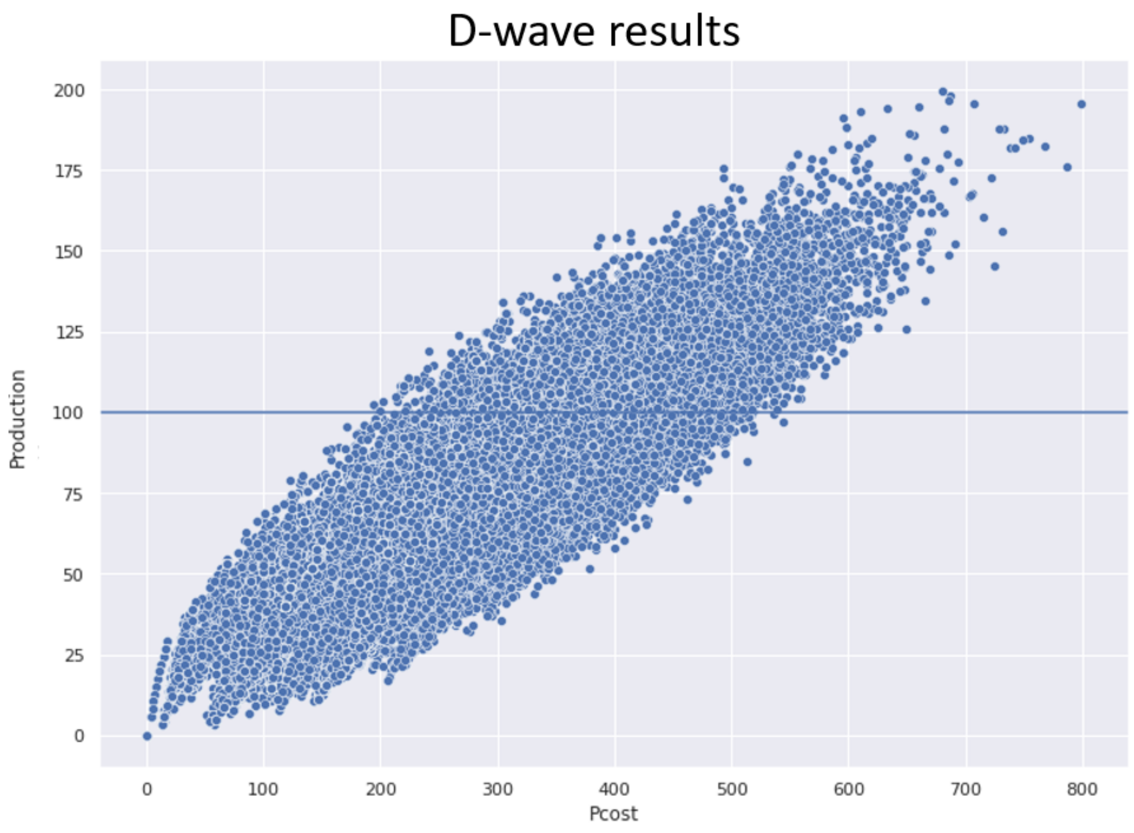


Figure 4.4: Results from D-WAVE

- Delta_B - also a Delta value that is related to the second constraint.
- Slack - mentioned as Constraint Type, it represents whether the problem uses an equality or inequality constraint type for the second constraint
- Grids - the amount of segments a unit's production is split into. More segments means a better representation of reality while also increasing complexity
- Dataset - the set of data being used as an input for the problem.
- Embedding - the type of embedding being used in the Quantum Processor
- Shots - the amount of solutions to be returned from D-WAVE during a run

Dependent Variables

- Chain_break - the occurrence, in percentage, of chain breaks throughout the different shots of a certain execution
- Restriction1 - the amount of broken units, units that have more than one production level, in a solution
- Restriction2 - the deviation of production from the required Demand

With this in mind, some initial hypothesis can be made with the preliminary experiments. Throughout the different tests, some parameters stood out on how they impacted the problem: Grids, Slack and DeltaP_A. Testing out different values for these parameters showed some visible differences in the results and means that statistical analysis will most likely show a statistical difference.

While the actual quality of impact varied, considering that changing the constraint type to an inequality and increasing the DeltaP_A value has shown some improvement in the amount of valid solutions, while increasing the Grids used seemed to cause a negative impact to the number of valid solutions.

As for the other parameters the impact was not so noticeable without resorting to variance tests, and may end up having no significant statistical difference in the worst case.

4.2 Formulating a Test Case

Before beginning the actual experimental analysis, some thought had to be put into balancing the amount of variance for the independent variables and the allowed computational budget that D'Wave offered for the processing, as to be able to cover a big scope of possible variance and possible intersections between the independent variables considered.

In order to check whether changing the default "regular" embedding had any effect, two kinds of embedding, regular and clique, were used.

Then, two different datasets were used to compare behaviors when inputs vary to a certain extent. These datasets are generated randomly by using the same methods detailed previously in section 4.1 using the Demand and the Units as input as a basis for range of possible values picked for each unit's production, to allow the random values to reach a valid solution, this code can be checked in the git repository for further details.

The amount of units were fixed at 10 and the only factor that varied was the amount of grids, from 5 to 10, and the Constraint Type, from equality to inequality. This means that, with the combination of these factors, the number of variables considered were either 70 at it's minimum, or 125 at it's maximum. Note that the amount of Grids for this test scenario were fixed at 5 since it failed to find a a clique embedding for the test cases with 10 grids. A special analysis of the impact of the Grids can be found in section 4.4.3 at the end of this experimental analysis.

The deltas for each constraint assumed values with exponential growth that allowed for a broader coverage in the quality of solutions while also letting some irregular cases to be observed in the jumps between values used. The starting value for DeltaP_A was chosen as 0.5 due to it having a bigger impact on the problem, as was previously observed in the previous section, than Delta_B which starts at 1.0, with the bigger values for Delta_B intended to cover a bigger range of sets to confirm whether there is a significant variance or not.

The value for chain strength is a single percentage, instead of comparing different percentage values. As per D-WAVE's guide on the parameter [47] this value is near, in this case using 75% of, the biggest absolute bias used in the final matrix of the QUBO formulation sent to D-WAVE. Using values too close to this absolute bias tended to result in a negative influence when few variables are used.

The following list shows the possible values of parameters that the problem can take:

- Embedding: Regular and Clique

- Dataset: 1 and 2
- Units: 10
- Grids: 5
- Constraint type: equality and inequality
- DeltaP_A: 0.5 and 1 and 2 and 4
- Delta_B: 1 and 2 and 4 and 8
- Chain Strength: 75%
- Shots: 4000

While chain strength was not a varying factor considered for this test case, due to budget constraints and the bigger focus on the independent variables related to the formulation, the importance of changing the default value is not to be understated.

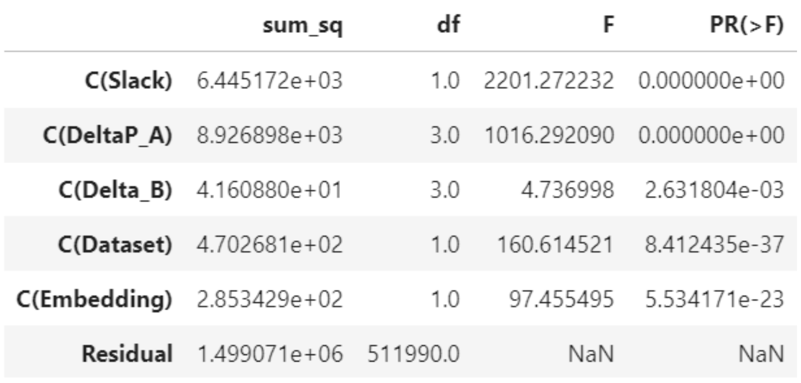
It is crucial to note that the default value for this factor is not at all appropriate for any problem sent to D-WAVE that has a large amount of variables. By default, D-WAVE applies a routine that tries to find a fitting value for this strength when it predicts that a chain break can occur but, when it fails to do so it returns the default value one. This value seriously hinders problems at a higher complexity, while for problems that deal with qubits numbering below 10 may not see an actual impact. This default value of 1 is usually what was observed during the executions done so far and so it was set as an actual value before sending the problem over to D-WAVE.

Per D-WAVE's documentation on the matter [47], as the weights in the QUBO are scaled automatically to fit between $[-1, 1]$, it is important to keep the chain strength at a value so that it is not to small to keep up with the big weights of the QUBO values, and that it's not too big to overshadow the weights of the QUBO and hinder the formulation for the problem. With this, the program in this thesis uses a percentage value for chain strength, set at 75%, that picks a value for the chain strength according to the biggest weight in the QUBO. With this, the given chain strength at any time will have a value of 75% of the biggest weight of the QUBO, where some simple tests showed that it stood an acceptable performance. This factor was not expanded upon in this test case due to budget constraints, as well as the impact being obvious enough, since the varying chain break values can also be used to derive conclusions to the influence of chain strength aside from largely lowering the amount of broken chains.

4.3 Analyzing results

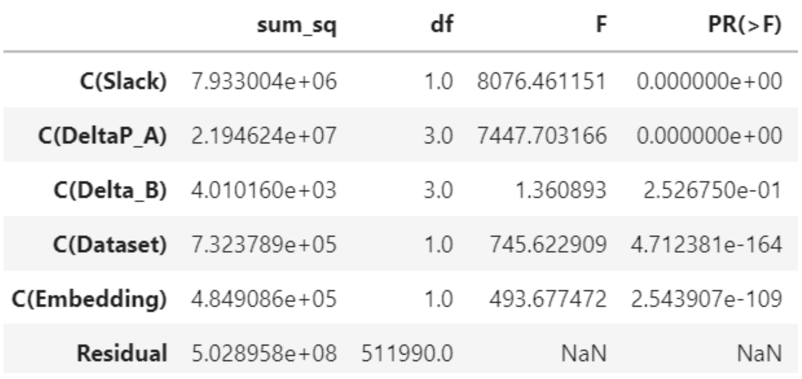
To analyze the results returned by D-WAVE, the data of the solutions is saved to a text file in a format that allows for importing in other platforms. In this case Jupyter was used to create a notebook to import, analyze and create graphs and plots of the data for analysis as well as hypothesis testing. The most relevant kinds of data that were collected were are Pcost (the production cost for a given solution), Restriction1 (amount of units that have more than one production level) and Restriction2 (the production deviation when compared to demand). Due to the rare amount of solutions that passed the constraints, some special decisions were made. All of the solutions produced were collected regardless of passing the constraints, although in case the first constraint was broken the units responsible for this breakage were considered as being off and not producing anything for the purposes of the experimental analysis.

A 5-way ANOVA (with no interactions) was performed for the five main factors described, in particular the ones that had different values, described in the list of section 4.2 in order to gain some insight into the role of each factor in the performance. This process was made with the tools available in the statsmodel library for python, more specifically the formula.api and stats.anova modules. The ANOVA tables can be found in Figures 4.5, 4.6 and 4.7 using as dependent variables the number of violations of the first constraint, second constraint and the amount of chain break, respectively.



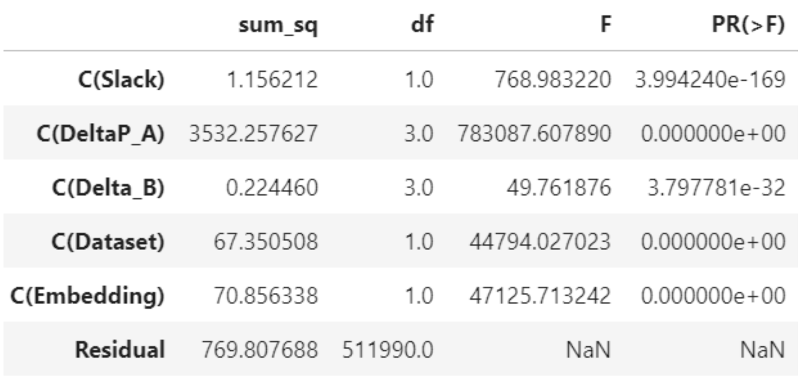
	sum_sq	df	F	PR(>F)
C(Slack)	6.445172e+03	1.0	2201.272232	0.000000e+00
C(DeltaP_A)	8.926898e+03	3.0	1016.292090	0.000000e+00
C(Delta_B)	4.160880e+01	3.0	4.736998	2.631804e-03
C(Dataset)	4.702681e+02	1.0	160.614521	8.412435e-37
C(Embedding)	2.853429e+02	1.0	97.455495	5.534171e-23
Residual	1.499071e+06	511990.0	NaN	NaN

Figure 4.5: A 5-way Anova test using the values related to the first restriction as a basis



	sum_sq	df	F	PR(>F)
C(Slack)	7.933004e+06	1.0	8076.461151	0.000000e+00
C(DeltaP_A)	2.194624e+07	3.0	7447.703166	0.000000e+00
C(Delta_B)	4.010160e+03	3.0	1.360893	2.526750e-01
C(Dataset)	7.323789e+05	1.0	745.622909	4.712381e-164
C(Embedding)	4.849086e+05	1.0	493.677472	2.543907e-109
Residual	5.028958e+08	511990.0	NaN	NaN

Figure 4.6: A 5-way Anova test using the values related to the second restriction as a basis



	sum_sq	df	F	PR(>F)
C(Slack)	1.156212	1.0	768.983220	3.994240e-169
C(DeltaP_A)	3532.257627	3.0	783087.607890	0.000000e+00
C(Delta_B)	0.224460	3.0	49.761876	3.797781e-32
C(Dataset)	67.350508	1.0	44794.027023	0.000000e+00
C(Embedding)	70.856338	1.0	47125.713242	0.000000e+00
Residual	769.807688	511990.0	NaN	NaN

Figure 4.7: A 5-way Anova test using the values from chain break as a basis

Overall, the factors that have the biggest impact on the parts responsible for the quality of the solutions returned to the formulated problem appear to be DeltaP_A and Constraint type for the constraint performance shown in tables 4.5,4.6, while Dataset and Embedding also have a big impact on chain break as shown in table 4.7.

With just the results from the ANOVA test there is no data on the quality of the impact of each factor in the overall problem, that is, whether the impact deteriorates the number of feasible solutions or actually improves them. To verify the actual impact of these factors further analysis will be done, starting with some simple exploratory analysis, and then proceeding with experimental analysis and statistical tests to check if the hypothesis holds.

4.4 Testing hypothesis

Before heading onto the grit of the analysis an important distinction must be made. There are two kinds of parameters for this problem: parameters relating to the formulation, that influence the behavior of the problem being formulated, and parameters related to the D-WAVE machines, that control how our formulation is implemented onto and performs on the quantum annealer.

4.4.1 Formulation Parameters

These are the parameters responsible for the accuracy of the representation of the original formulated problem. Most of the factors used belong to this category and many of them have a great impact on how the program behaves. A more detailed analysis follows.

Constraint Type

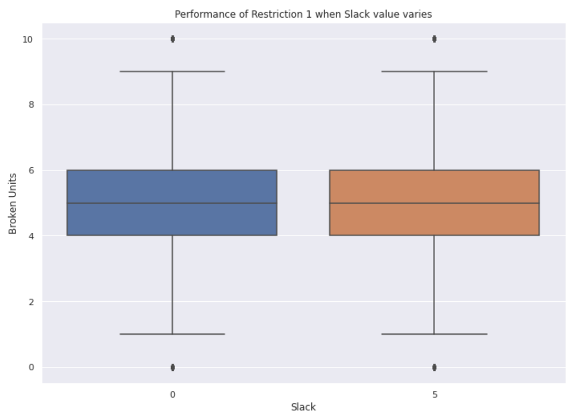


Figure 4.8: Boxplot of broken units for the different constraint types

The analysis confirms the expected result that changing the second constraint to become an inequality would net positive results. This relaxation means that it becomes easier to find a solution that produces enough to meet the Demand constraint, as shown by Figure 4.9, while the first constraint sees a small improvement, about 0.22 less broken units albeit not noticeable in Figure 4.8, when using the second constraint as inequality. The means of chain break occurrences sees a slight increase when opting for the inequality type, but even so, the amount of occurrences seems to be more concentrated on lower values when

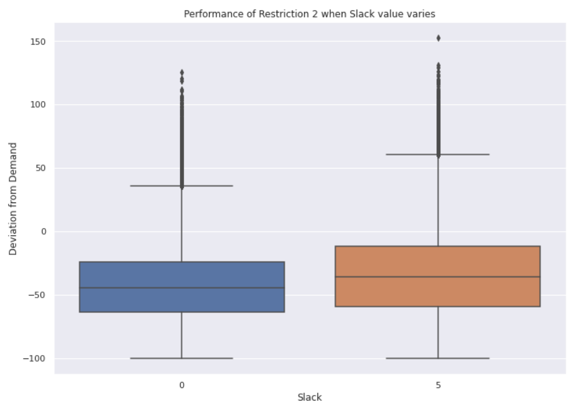


Figure 4.9: Boxplot of deviation from Demand for the different constraint types

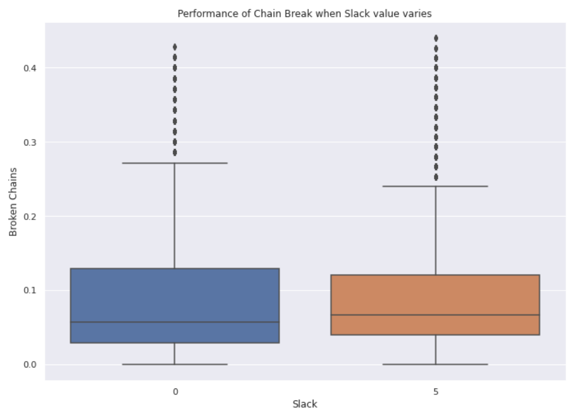


Figure 4.10: Boxplot of broken chains for the different constraint types

comparing to the broad coverage of the equality constraint type, as shown by the Figure 4.10

DeltaP_A

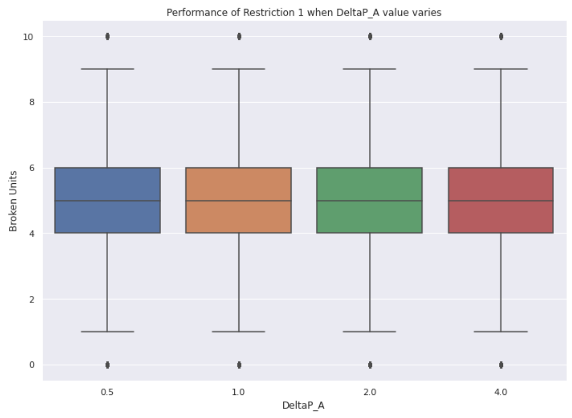


Figure 4.11: Boxplot of broken units for the different DeltaP_A values

This, alongside Delta_B, is one of the most important factors to analyze, given their role

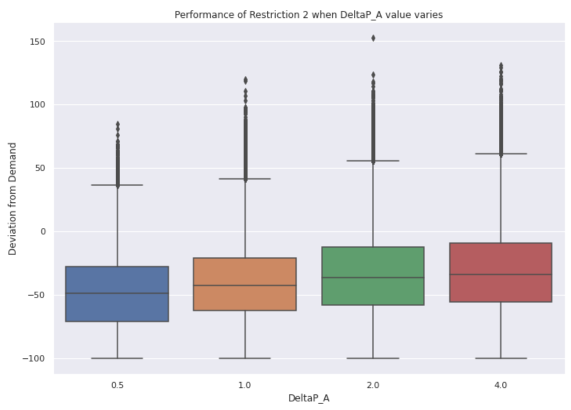


Figure 4.12: Boxplot of deviation from Demand for the different DeltaP_A values

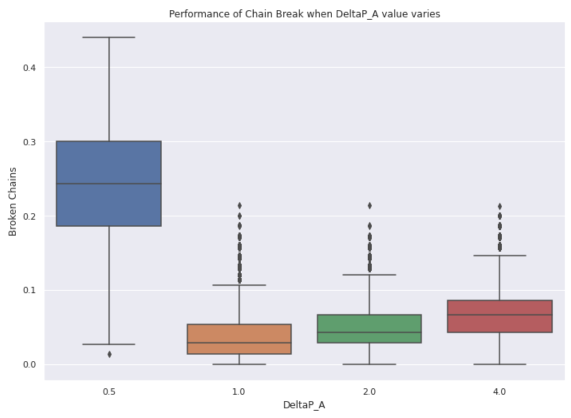


Figure 4.13: Boxplot of broken chains for the different DeltaP_A values

for enforcing and strengthening their respective constraints and this analysis serves to verify if they actually do so.

To start off, it's difficult to see the results that Figure 4.11 shows since there are only 10 values and the results hover around the same area. Even so, it's possible to see some variation through the median of each part.

The results from the figure are inconclusive but the ANOVA test rejected the null hypothesis, so there must be some variation happening. For a more detailed analysis, a pairwise Tukey HSD test, using the statsmodels library in python, was performed for the three factors.

Group1	Group2	Diff	P-value
0.5	1.0	-0.1146	0.001000
0.5	2.0	-0.3007	0.001000
0.5	4.0	-0.3161	0.001000
1.0	2.0	-0.1862	0.001000
1.0	4.0	-0.0154	0.001000
2.0	4.0	-0.0154	0.1057

Table 4.1: A Tukey test with the values of DeltaP_A when considering the values from the first constraint as a basis

With the Tukey HSD test from 4.1 it becomes easier to compare the actual variations between each part, it also returns a P-value that tells whether or not to reject the null hypothesis that the sets of data share the same median. Through the differences between each set, it's possible to observe that the broken units tend to decrease as the value of DeltaP_A increases, which means that the delta is correctly affecting the constraint it is responsible for. On a closer look, it can be seen that the last case comparing the DeltaP_A value sets of 2.0 and 4.0 returns a P-value that leads to the acceptance of the null hypothesis. Meaning that there is no statistical difference between both sets, and perhaps even little point to increasing the DeltaP_A value beyond 2.0.

With regards to how the second constraint behaves, Figure 4.12 shows some clear variation between each value. With this, raising the value of DeltaP_A has a visible positive effect, decreasing the gap between the energy demand and produced energy to be closer to zero. Through the Tukey test in table 4.2 the individual differences between groups can be seen, and there is always some gain from increasing the Delta, unlike the previous case of the first constraint, where no statistical difference can be seen beyond the value 2.

Group1	Group2	Diff	P-value
0.5	1.0	7.429	0.001
0.5	2.0	14.0476	0.001
0.5	4.0	17.0028	0.001
1.0	2.0	6.6186	0.001
1.0	4.0	9.5738	0.001
2.0	4.0	2.9552	0.001

Table 4.2: A Tukey test with the values of DeltaP_A when considering the values from the second constraint as a basis

In regards to the values of chain break shown in Figure 4.13 and the Table 4.3, using values below one for the Delta has a considerable negative effect on the amount of broken chains. Aside from this irregularity, the amount of broken chains increases progressively alongside the Delta value, meaning that is somewhat of a negative effect on the chains as the Delta increases. Although this is a negative effect, considering the gains to the other factors and the fact that chain break is still below acceptable values, the gains that outperform this negative impact means that the optimal decision is to increase the Delta value as the overall effect on the amount of valid solutions increases.

Group1	Group2	Diff	P-value
0.5	1.0	-0.2037	0.001
0.5	2.0	-0.1919	0.001
0.5	4.0	-0.1757	0.001
1.0	2.0	0.0118	0.001
1.0	4.0	0.028	0.001
2.0	4.0	0.0162	0.001

Table 4.3: A Tukey test with the values of DeltaP_A when considering the values from the chain break as a basis

The take from this is that there is no reason to raise the Delta value to a high value like 2.0 or 4.0 as there is an overall positive effect on the amount of valid solutions to the problem.

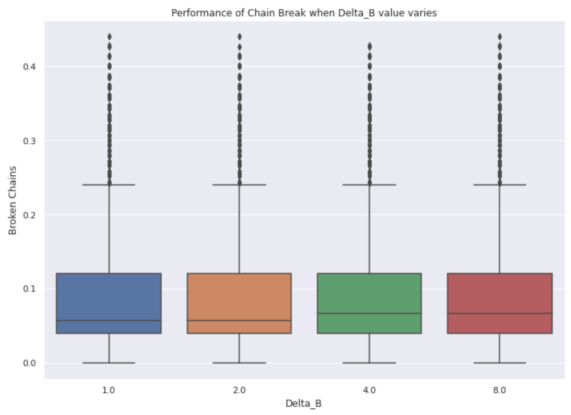


Figure 4.14: Boxplot of broken chains for the different Delta_B values

Delta_B

With regards to the effects of Delta_B, things get more complicated. With the acceptance of the null hypothesis in the Anova test related to both the first and second constraint, the statistical impact of this Delta is on the verge of being null. Even so, the Anova test of the impact on chain break rejects the null hypothesis, meaning that there must be some variation to analyze.

Group1	Group2	Diff	P-value
1.0	2.0	0.0004	0.7063
1.0	4.0	0.0011	0.0111
1.0	8.0	0.0017	0.001
2.0	4.0	0.0008	0.1696
2.0	8.0	0.0013	0.0018
4.0	8.0	0.0006	0.4045

Table 4.4: A Tukey test with the values of DeltaP_B when considering the values from broken chains as a basis

Now, with the results from the Tukey HSD test shown in Table 4.4, the situation is grim.

At first glance, half the combinations lead to the rejection of the null hypothesis, while the other half accept it. Even without the p-values, it's already possible to see how little the differences between each group actually is, meaning that there is a minimal impact on the amount of chain breaks.

This means that potentially, the Delta can be discarded as the statistical impact in the overall problem is insignificant.

Dataset

With figures 4.15, 4.16 and 4.17, the difference in performance between datasets can be seen. Overall they stay largely the same with some minor variations, with the second constraint seeing some negative impact on the second set of inputs, this means that the kind of dataset used has some impact on the valid solutions for the problem. Along with the second constraint, the amount of chain break has also seen some deterioration, while the values resulting from the first constraint remain largely the same. But, seeing as these

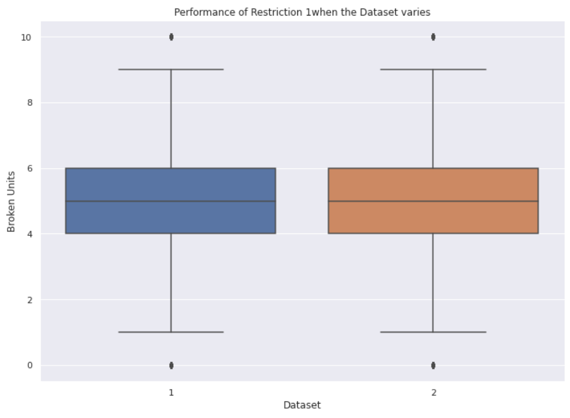


Figure 4.15: Boxplot of broken units for the different Datasets used

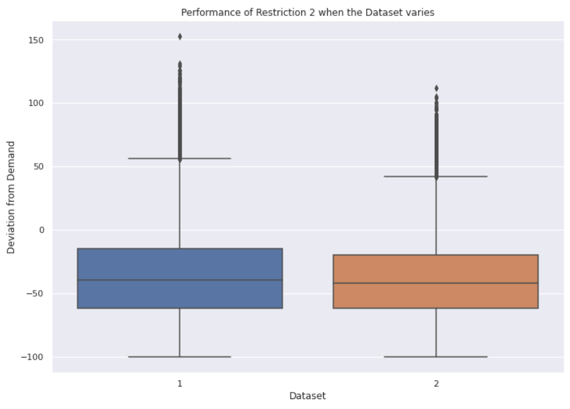


Figure 4.16: Boxplot of deviation from Demand for the different Datasets used

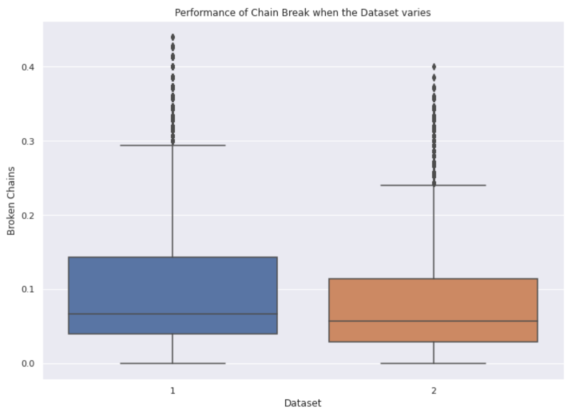


Figure 4.17: Boxplot of broken chains for the different Datasets used

inputs are related to the actual working values that most likely cannot be changed to improve the problem, not much can be done to improve this factor's impact towards the positive side.

4.4.2 D-Wave Parameters

These are the parameters responsible for actually interacting with the behavior of the D-WAVE machines. In this case only the embedding being used was considered as a varying value, while chain strength was used with a fixed value. Other factors exist, like annealing schedule, but were not considered due to the current budget restraints.

Embedding

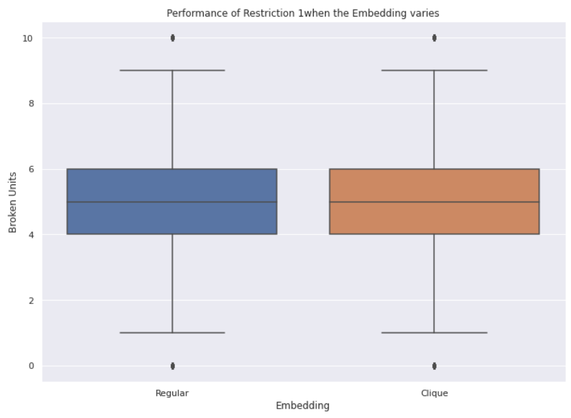


Figure 4.18: Boxplot of broken units for the different embeddings used

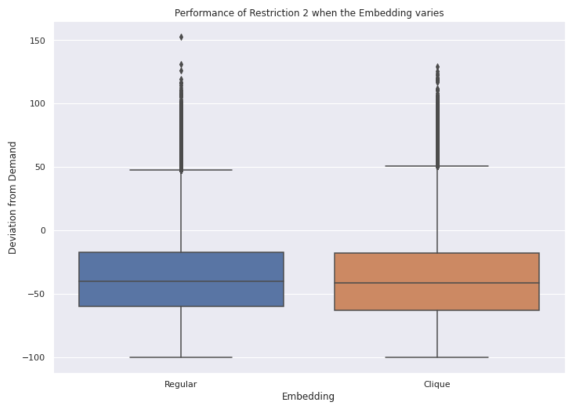


Figure 4.19: Boxplot of deviation from Demand for the different embeddings used

The embedding used, is of course vital to the performance of the problem since it's related to how the problem is mapped onto the QPU. As shown by the figures 4.18, 4.19 and 4.20, the clique embedding closely comes out on top. If comparing the actual means, the clique embedding has a better performance on all but the second constraint, where it is a whole 2 points lower when compared to the regular embedding. As it stands, it could be worth using for the improvement of the chain break behavior, specially if used with the high values of DeltaP_A that increase the chain break amount while also improving the second constraint.

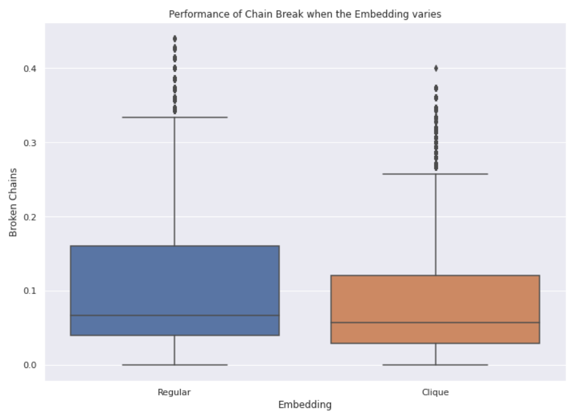
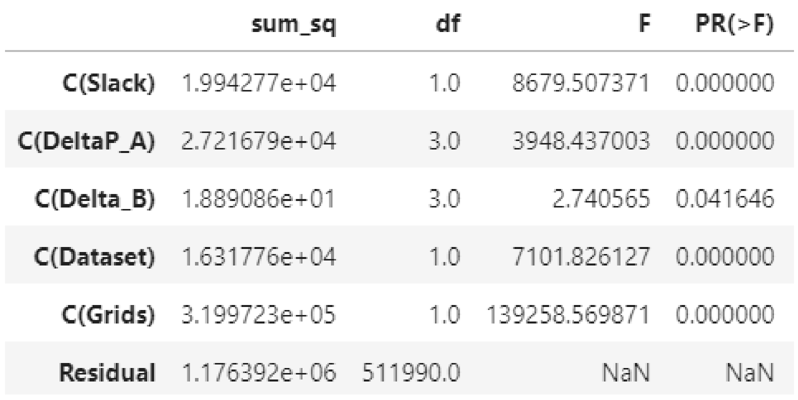


Figure 4.20: Boxplot of broken chains for the different embeddings used

4.4.3 Isolated Parameter

Since no embedding was found for the test cases with 10 Grids for the Clique Embedding, the Grids parameter was removed from the main analysis. This was done to keep the number of parameters from both the problem formulation and the QPU more balanced. But, there is still some relevant analysis to be gained from this parameter, which will be included in this extra section.



	sum_sq	df	F	PR(>F)
C(Slack)	1.994277e+04	1.0	8679.507371	0.000000
C(DeltaP_A)	2.721679e+04	3.0	3948.437003	0.000000
C(Delta_B)	1.889086e+01	3.0	2.740565	0.041646
C(Dataset)	1.631776e+04	1.0	7101.826127	0.000000
C(Grids)	3.199723e+05	1.0	139258.569871	0.000000
Residual	1.176392e+06	511990.0	NaN	NaN

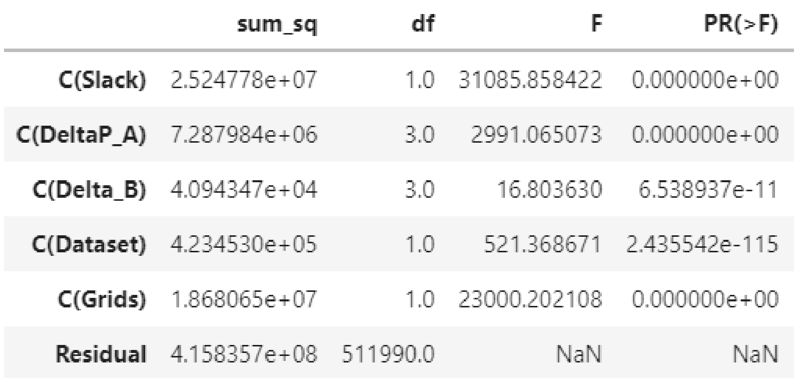
Figure 4.21: A 5-way Anova test using the values related to the first restriction as a basis

This new 5-way Anova test that replaces embedding with grids shows just how much of an impact varying the number of grids in the problem can have. With the grids parameter having the biggest impact on the performance of the first constraint and the chain break values, and the second most impact on the performance of the second constraint. The results from this Anova test can be observed in figures 4.21, 4.23, and 4.22 respectively.

The great impact grids has on the first constraint is to be expected since it is closely tied to the amount of production levels each unit has and, with more levels to check against the constraint, more errors, broken units, are prone to happen.

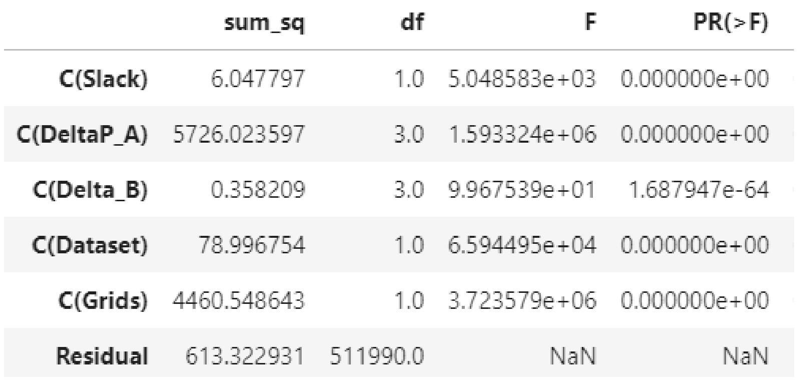
The boxplots from Figures 4.24, 4.25 and 4.26 show that as the amount of grids increase, the solution quality deteriorates.

While increasing the amount of grids considered makes the linearization more accurate, it also deteriorates the solution quality since it has a big impact on the number of variables



	sum_sq	df	F	PR(>F)
C(Slack)	2.524778e+07	1.0	31085.858422	0.000000e+00
C(DeltaP_A)	7.287984e+06	3.0	2991.065073	0.000000e+00
C(Delta_B)	4.094347e+04	3.0	16.803630	6.538937e-11
C(Dataset)	4.234530e+05	1.0	521.368671	2.435542e-115
C(Grids)	1.868065e+07	1.0	23000.202108	0.000000e+00
Residual	4.158357e+08	511990.0	NaN	NaN

Figure 4.22: A 5-way Anova test using the values related to the second restriction as a basis



	sum_sq	df	F	PR(>F)
C(Slack)	6.047797	1.0	5.048583e+03	0.000000e+00
C(DeltaP_A)	5726.023597	3.0	1.593324e+06	0.000000e+00
C(Delta_B)	0.358209	3.0	9.967539e+01	1.687947e-64
C(Dataset)	78.996754	1.0	6.594495e+04	0.000000e+00
C(Grids)	4460.548643	1.0	3.723579e+06	0.000000e+00
Residual	613.322931	511990.0	NaN	NaN

Figure 4.23: A 5-way Anova test using the values related to the chain break as a basis

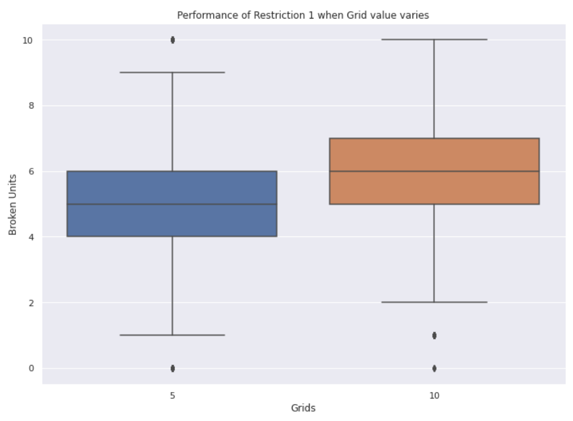


Figure 4.24: Boxplot of broken units when Grids used varies

for the problem.

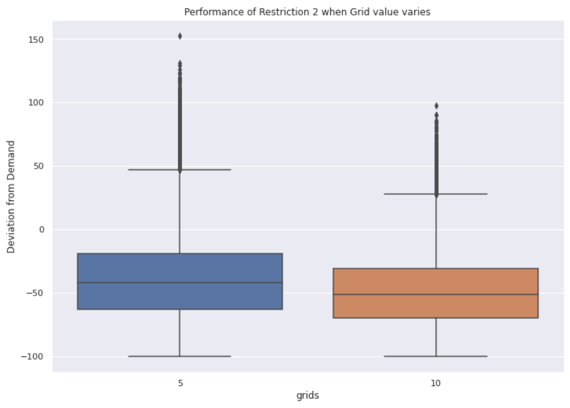


Figure 4.25: Boxplot of deviation from Demand when Grids used varies

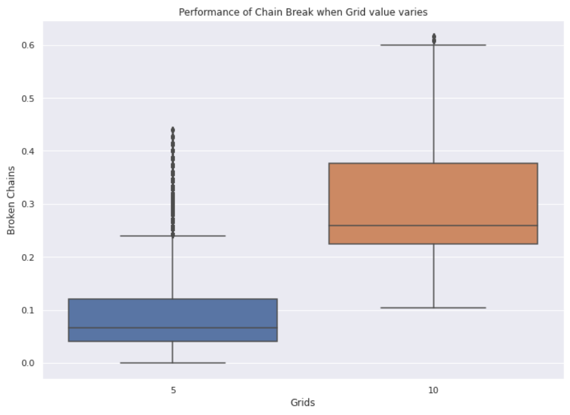


Figure 4.26: Boxplot of broken chains when Grids used varies

Chapter 5

Conclusion

This thesis was conducted with the goal to expand the current implementations of Unit Commitment on quantum computers and study the impact of the various parameters of the problem. To that end, a test scenario was designed to allow an experimental analysis to be conducted and verify the hypothesis that the parameters do influence the results. The results from the test scenario show that both the parameters related to the problem formulation and the parameters from the quantum annealer have a considerable impact on the solutions to the problem.

From the problem formulation parameters, the results from the test scenario show that the Grids, Constraint type and the Delta of the first constraint do have an influence on the quality of the solutions returned by the QPU. Aside from these that were showcased in the test scenario, there is also the number of variables that the problem is considering, or the scope of the problem, which is the biggest influencer on the complexity of the problem and is also expected to have a great impact. The results from the experimental analysis were to be expected, since these parameters are closely related to the complexity of the problem, as well as the weight of each constraint. As for the remaining parameter, the Delta associated with the second constraint, the results show that it does not have a statistical difference when comparing the different samples, which means more focus should be put into fine tuning the other parameters instead.

When it comes to the parameters from the QPU, a very important case stands out. The chain strength parameter has a default value that is destructive to problems at a higher qubit complexity. It's imperative that this parameter is set correctly, as the default and a wrongly set value can a great negative impact on solution quality. This is due to the nature of chains and the part they play in representing a specific variable throughout several regions of the problem. Aside from this parameter two different kinds of embedding were also tested, but the impact is in general overshadowed by the formulation parameters, although a considerable impact still exists when it comes to influencing chain break occurrence.

A possible follow-up to this work could be to study the interactions between each parameter since there could be specific interactions that have a great impact on the solution quality.

From the results achieved in this thesis it can be concluded that an approach that provides an undeniable advantage through quantum computing on a real world scale for the Unit Commitment Problem is still currently out of reach, specially since the formulation used for this thesis is already a simplification of the original problem. Even so, the potential is there, since the QPU can return a great amount of solutions in a short time and allow for a much greater flexibility in picking the best solution for a real world scenario, since an unexpected occurrence can invalidate a solution that would take a classical computer several hours, if not days, to compute.

References

[1] Qiskit. Bloch sphere. `https://qiskit.org/textbook/ch-states/introduction.html`. Accessed: 6 January 2021.

[2] Wikipedia. Comparison of d-wave systems. `https://en.wikipedia.org/wiki/D-Wave_Systems#Comparison_of_D-Wave_systems`. Accessed: 15 January 2021.

[3] Catherine C. McGeoch, Richard Harris, Steven P. Reinhardt, and Paul I. Bunyk. Practical annealing-based quantum computing. *Computer*, 52(6):38–46, 2019.

[4] Andrew Steane. Quantum computing. *Reports on Progress in Physics*, 61(2):117, 1998.

[5] Jozef Gruska et al. *Quantum computing*, volume 2005. McGraw-Hill London, 1999.

[6] Peter W Shor. Quantum computing. *Documenta Mathematica*, 1(1000):467–486, 1998.

[7] Tony Hey. Quantum computing: an introduction. *Computing & Control Engineering Journal*, 10(3):105–112, 1999.

[8] Eleanor Rieffel and Wolfgang Polak. An introduction to quantum computing for non-physicists. *ACM Computing Surveys (CSUR)*, 32(3):300–335, 2000.

[9] Han-Sen Zhong, Hui Wang, Yu-Hao Deng, Ming-Cheng Chen, Li-Chao Peng, Yi-Han Luo, Jian Qin, Dian Wu, Xing Ding, Yi Hu, Peng Hu, Xiao-Yan Yang, Wei-Jun Zhang, Hao Li, Yuxuan Li, Xiao Jiang, Lin Gan, Guangwen Yang, Lixing You, Zhen Wang, Li Li, Nai-Le Liu, Chao-Yang Lu, and Jian-Wei Pan. Quantum computational advantage using photons. *Science*, 370(6523):1460–1463, 2020.

[10] Ashley Montanaro. Quantum speedup of monte carlo methods. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 471(2181):20150301, Sep 2015.

[11] Dan Stahlke. Quantum interference as a resource for quantum speedup. *Physical Review A*, 90(2):022302, 2014.

[12] Shah Muhammad Hamdi, Syed Tauhid Zuhori, Firoz Mahmud, and Biprodip Pal. A compare between shor's quantum factoring algorithm and general number field sieve. pages 1–6, 2014.

[13] Mirko Amico, Zain H. Saleem, and Muir Kumph. Experimental study of shor's factoring algorithm using the ibm q experience. *Physical Review A*, 100(1), Jul 2019.

[14] IBM. Qiskit front page. `https://qiskit.org/`. Accessed: October 2021.

[15] D-Wave. D-wave front page. `https://www.dwavesys.com/`. Accessed: October 2021.

[16] Amazon. Aws amazon braket front page. `https://aws.amazon.com/braket/`. Accessed: October 2021.

[17] Qutech. Quantum inspire front page. `https://www.quantum-inspire.com/`. Accessed: October 2021.

[18] Microsoft. Azure quantum front page. `https://azure.microsoft.com/en-us/services/quantum/`. Accessed: October 2021.

[19] University of Bristol. Quantum in the cloud. `http://www.bristol.ac.uk/physics/research/quantum/engagement/qcloud/`. Accessed: October 2021.

[20] Arute, F, Arya, K, Babbush, and R et al. Quantum supremacy using a programmable superconducting processor. *Nature*, 574:505–510, 2019.

[21] Edwin Pednault, John Gunnels, Dimitri Maslov, and Jay Gambetta. `https://www.ibm.com/blogs/research/2019/10/on-quantum-supremacy/`. Accessed: October 2021.

[22] Kevin Hartnett. A new law to describe quantum computing's rise? `https://www.quantamagazine.org/does-nevens-law-describe-quantum-computings-rise-20190618/`. Accessed: September 2021.

[23] University of Queensland. Improved fabrication technique paves way for improved quantum devices. `https://phys.org/news/2021-09-fabrication-technique-paves-quantum-devices.html`. Accessed: September 2021.

[24] RM Brady. Optimization strategies gleaned from biological evolution. *Nature*, 317(6040):804–806, 1985.

[25] Marco Dorigo, Mauro Birattari, and Thomas Stutzle. Ant colony optimization. *IEEE computational intelligence magazine*, 1(4):28–39, 2006.

[26] James Kennedy and Russell Eberhart. Particle swarm optimization. 4:1942–1948, 1995.

[27] Iztok Fister Jr, Xin-She Yang, Iztok Fister, Janez Brest, and Dušan Fister. A brief review of nature-inspired algorithms for optimization. *arXiv preprint arXiv:1307.4186*, 2013.

[28] Xin-She Yang. *Nature-inspired optimization algorithms.* Academic Press, 2020.

[29] Akshay Ajagekar and Fengqi You. Quantum computing for energy systems optimization: Challenges and opportunities. *Energy*, 179:76 – 89, 2019.

[30] F. A. C. C. Fontes*, D. B. M. M. Fontes**, and L. A. Roque***. An optimal control approach to the unit commitment problem. 2012.

[31] Qiaozhu Zhai Xiaohong Guan and A. Papalexopoulos. Optimization based methods for unit commitment: Lagrangian relaxation versus general mixed integer programming. *IEEE Power Engineering Society General Meeting*, 2(1095-1100), 2003.

[32] Idriss Abdou and Mohamed Tkiouat. Unit commitment problem in electrical power system: A literature review. *International Journal of Electrical and Computer Engineering (IJECE)*, 8:1357, 06 2018.

[33] Ana Viana João Pedro Pedroso. A new milp-based approach for unit commitment in power production planning. *International Journal of Electrical Power & Energy Systems*, 44(1):997 – 1005, 2013.

[34] K Srikanth, Lokesh Kumar Panwar, BK Panigrahi, Enrique Herrera-Viedma, Arun Kumar Sangaiah, and Gai-Ge Wang. Meta-heuristic framework: Quantum inspired binary grey wolf optimizer for unit commitment problem. *Computers & Electrical Engineering*, 70:243 – 260, 2018.

[35] Saleh Y. Abujarad, M.W. Mustafa, and J.J. Jamian. Recent approaches of unit commitment in the presence of intermittent renewable energy resources: A review. *Renewable and Sustainable Energy Reviews*, 70:215 – 223, 2017.

[36] Yuntao Ju, Jiankai Wang, Fuchao Ge, Yi Lin, Mingyu Dong, Dezhi Li, Kun Shi, and Haibo Zhang. Unit commitment accommodating large scale green power. *Applied Sciences*, 9(8):1611, Apr 2019.

[37] Nikolaos V. Sahinidis. Mixed-integer nonlinear programming. *Optimization and Engineering*, 20(301-306), 2018.

[38] M. Carrion and J. M. Arroyo. A computationally efficient mixed-integer linear formulation for the thermal unit commitment problem. *IEEE Transactions on Power Systems*, 21(3):1371–1378, 2006.

[39] Kuk-Hyun Han and Jong-Hwan Kim. Quantum-inspired evolutionary algorithm for a class of combinatorial optimization. *IEEE Transactions on Evolutionary Computation*, 6(6):580–593, 2002.

[40] Qiskit. Representing qubit states. `https://qiskit.org/textbook/ch-states/representing-qubit-states.html`. Accessed: 15 January 2021.

[41] Asher Peres and Daniel R. Terno. Quantum information and relativity theory. *Reviews of Modern Physics*, 76(1):93–123, Jan 2004.

[42] Wilen, C D, Abdullah, S, Kurinsky, and N A et al. Correlated charge noise and relaxation errors in superconducting qubits. *Nature*, 594:369–373, 2021.

[43] D-Wave. What is quantum annealing? `https://docs.dwavesys.com/docs/latest/c_gs_2.html`. Accessed: 15 January 2021.

[44] D-Wave. Qubo expansion. `https://youtu.be/Q4FE4jou5CA?t=1440`. Accessed: December 2020.

[45] D-Wave. D-wave qpu architecture: Topologies. `https://docs.dwavesys.com/docs/latest/c_gs_4.html`. Accessed: August 2021.

[46] D-Wave. D-wave solutions and products. `https://www.dwavesys.com/solutions-and-products/systems/`. Accessed: August 2021.

[47] D-Wave Systems. Programming the d-wave qpu: Setting the chain strength. `https://www.dwavesys.com/media/vsufwv1d/14-1041a-a_setting_the_chain_strength.pdf`. Accessed: February 2021.

This page is intentionally left blank.

Appendices

This page is intentionally left blank.

Appendix A

```python
        def qubo(params):
        print(f"Creating Qubo dict w/ deltaP_A {params['deltaP_A']} & delta_B {params['delta_B']}")
        delta_B = params["delta_B"]
        deltaP_A = params["deltaP_A"]

        additive_a = 0
        additive_b = 0

        # objective function
        for i in range(0,U):
            for k in range(1,N):                                            # python range stops b4 last term, so N = Grids + 2 is the same as N + 1 in summation terms
            Qubo[str(i*N+k), str(i*N+k)] += float( cost(i,k) )
            for m in range(k+1,N):          # extra func (from sumation square expansion)
                Qubo[str(i*N+k), str(i*N+m)] += float( 2*C[i]*prod(i,k)*prod(i,m) )

        # constraint 2
        for i in range(0,U):
            for k in range(1,N):
            Qubo[str(i*N+k), str(i*N+k)] += float( delta_B*( prod(i,k)**2 ) )       # first term: sum xi **2
            Qubo[str(i*N+k), str(i*N+k)] += float( delta_B*( -2*Demand*prod(i,k) ) ) # third term: -2*C*sum xi
            for ii in range(i,U):
                for kk in range(1,N):
                if ii == i and k >= kk: continue
                Qubo[str(i*N+k), str(ii*N+kk)] += float( delta_B*( 2*prod(i,k)*prod(ii,kk) ) ) # second term: 2*sum xi*xj, where j > i
            for s in range(scope-Slack, scope):
                Qubo[str(i*N+k), str(s)] += float( delta_B*( 2*prod(i,k)*prod_slack(s - scope + Slack) ) )# Slack variables | s - scope + Slack -> range from 0 to 5 (Slack variables)

        for s in range(scope-Slack, scope):
            Qubo[str(s), str(s)] += float( delta_B * (prod_slack(s - scope + Slack)**2) )# first factor +
            Qubo[str(s), str(s)] += float( delta_B* -2 * Demand * prod_slack(s - scope + Slack) )# third factor -

            for s2 in range(s+1,scope):
            Qubo[str(s), str(s2)] += float( delta_B* 2 * prod_slack(s - scope + Slack) * prod_slack(s2 - scope + Slack) )

        additive_b += delta_B*( Demand**2 )

        max_bias = 0
        for value in Qubo.values():
            if abs(value) >= max_bias:
            max_bias = value

        delta_A = max_bias * deltaP_A
        params["delta_A"] = delta_A
        max_bias_final = max_bias

        # restriction 1
        for i in range(0,U):
            for k in range(0,N):
            Qubo[str(i*N+k), str(i*N+k)] += float( delta_A*( 1 ) )  # 1**2
            Qubo[str(i*N+k), str(i*N+k)] += float( delta_A*( -2 ) ) # -2*1

            for kk in range(k+1,N):
                Qubo[str(i*N+k), str(i*N+kk)] += float( delta_A*( 2 ) ) # 2*1*1
            additive_a += delta_A*(1) #1**2

        for value in Qubo.values():
            if abs(value) >= max_bias_final:
            max_bias_final = value

        #print_qubo(Qubo)

        params["additive_a"] = float( additive_a )
        params["additive_b"] = float( additive_b )
        params["max_bias_B"] = float( max_bias )
        params["max_bias_final"] = float( max_bias_final )

        print(f"Created Qubo dict w/ delta_A {params['delta_A']:.2f} & max_bias_B {params['max_bias_B']:.3f} & max_bias_final {params['max_bias_final']:.3f}")
        return Qubo,Qubo_obj,Qubo_res1,Qubo_res2,Qubo_slack
```

Listing 1: Qubo Function