FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Applying Quantum Annealing to the Tail Assignment Problem

Luís Noites Martins

Mestrado Integrado em Engenharia Informática e Computação

Supervisor: Ana Paula Rocha, PhD

Co-Supervisor: António Castro, PhD

July 31, 2020

Applying Quantum Annealing to the Tail Assignment Problem

Luís Noites Martins

Mestrado Integrado em Engenharia Informática e Computação

Approved in oral examination by the committee:

Chair: Rosaldo Rossetti, PhD
External Examiner: Rui Maranhão Abreu, PhD
Supervisor: Ana Paula Rocha, PhD

July 31, 2020

Abstract

Airline companies struggle every day trying to schedule crew, flights and aircraft. Tail Assignment is the problem of allocating individual aircraft to a set of flights, while ensuring multiple constraints and aiming to minimise an objective function, such as operational costs. Given the enormous amount of possibilities and constraints involved, this problem has been a study case for the last decade. Many solutions have emerged using classical computing, but with limitations regarding performance. Quantum Annealing (QA) is a heuristic technique for finding global minimum energy levels over an energy landscape using quantum mechanics. Due to its characteristics, it has been proved to have a clear advantage in solving some complex optimisation problems, being a promising technique to apply in multiple fields.

In this study, Tail Assignment Problem was set as a Quadratic Unconstrained Binary Optimisation (QUBO) model, using two different techniques, and was solved using a classical and two hybrid solvers. Tests were run based on extractions from real-world data, analysing the performance of the implementation in terms of time, scalability and quality (i.e., the lowest operational costs) of the obtained solutions.

We concluded that using a library for modelling the problem as well as considering individual flights rather than pre-aggregating them in strings can be a bottleneck when it comes to its scalability. Furthermore, we found out that there was a higher probability of obtaining better solutions for this problem using one of the hybrid solvers when compared with the classical heuristic algorithms such as Simulated Annealing (SA). These findings can serve as foundations for further studies.

Keywords: Quantum Annealing, Tail Assignment Problem, Quantum Computing

Resumo

As companhias aéreas lidam, diariamente, com dificuldades operacionais no planeamento de tripulação, voos e aviões. A alocação de aeronaves a voos é um problema complexo uma vez que é necessário garantir múltiplas restrições tentando minimizar uma certa função de custo, como, por exemplo, custos operacionais. Dada a volumosa quantidade de possibilidades e restrições envolvidas, este problema tem sido um caso de estudo durante a última década. Várias propostas de solução têm surgido utilizando computação clássica, mas com limitações sistemáticas no que diz respeito à performance. Arrefecimento Quântico é uma técnica heurística que permite encontrar um nível mínimo de energia de entre um dado grupo de níveis de energia, utilizando física quântica. Considerando as suas características, esta técnica tem-se revelado vantajosa na resolução de alguns problemas complexos de otimização, sendo a sua aplicação promissora em várias áreas.

Nesta dissertação, o Problema de Alocação de Aeronaves foi definido como um modelo *Quadratic Unconstrained Binary Optimisation* (QUBO), utilizando duas técnicas diferentes, e foi resolvido com recurso a um *solver* clássico e dois *solvers* híbridos. Foram realizados testes utilizando extrações de dados de uma companhia aérea, de modo a analisar a performance da implementação proposta no que concerne a tempo, escalabilidade e qualidade (i.e., custo mínimo operacional) das soluções obtidas.

Concluímos que utilizar uma biblioteca para modelação, bem como considerar os voos de forma individual e não agrupados em *strings*, pode limitar a escalabilidade do problema. Constatámos ainda que a utilização de um dos *solvers* híbridos permitiu obter com maior probabilidade soluções menos dispendiosas quando comparada com a utilização de um algoritmo clássico, como Arrefecimento Simulado. Estas conclusões poderão servir de base para futuros estudos.

Keywords: Arrefecimento Quântico, Alocação de Aeronaves, Computação Quântica

Acknowledgements

I would like to start by thanking my supervisor Ana Paula Rocha and my co-supervisor António Castro for their guidance, advice and constant feedback throughout the whole dissertation.

Secondly, I am also thankful to my family for all the support and advice mainly during the hard times.

Finally, I would like to thank my friends for their companionship, remembering all the laughs and good moments we spent together.

Luis

"If quantum mechanics hasn't profoundly shocked you,
you haven't understood it yet."

Niels Bohr

Contents

List of Figures

List of Tables

Abbreviations

API	Application Programming Interface
ATC	Air Traffic Control
BQM	Binary Quadratic Model
CPU	Central Processing Unit
CSP	Constraint Satisfaction Problem
DOC	Direct Operational Costs
FIFO	First-in, First-out
GPU	Graphics Processing Unit
HSS	Hybrid Solver Service
IATA	International Air Transport Association
IOC	Indirect Operational Costs
LP	Linear Programming
NB	Narrow Body
NISQ	Noisy Intermediate-Scale Quantum Computer
NP	Non-deterministic Polynomial time
OR	Operations Research
QA	Quantum Annealing
QAOA	Quantum Approximate Optimisation Algorithm
QPU	Quantum Processing Unit
QUBO	Quadratic Unconstrained Binary Optimisation
SA	Simulated Annealing
UC	Units of Cost
VQE	Variational Quantum Eigensolver
WB	Wide Body

Chapter 1

Introduction

This chapter starts by giving a brief introduction and explaining the context of this study, in section 1.1. Following, section 1.2 presents the motivation behind this dissertation as well as a succinct definition of the main objective. By the end of the chapter, in section 1.3, the main contributions of this dissertation are presented and, in section 1.4, a pithy description of the structure of the document is given.

1.1 Context

Airlines' sector is currently one of the most competitive industries. With the appearance of new low-cost airlines, new marketing strategies have also appeared offering tickets with reduced prices and new routes, with almost 22.000 regular city-pair flights by the end of 2018. The fierce competition in this sector has been ensuring that airfares remain affordable to travelers. This factor forces airlines to reduce profit margins to keep the clients, having a generated profit per passenger of $6.85 [2].

Although over the last 20 years total costs have dropped more than 50%, this value is still a key factor in the daily operations of the airlines. The operational costs have a big impact on the overall value, with jet fuel representing around 24%. Due to the high fuel costs and environmental concerns, a lot of effort has been done to avoid the usage of unnecessary fuel [1].

Airline scheduling is a complex process composed by various sequential steps, starting from defining which airports and routes the airline company will operate, until the moment an individual aircraft and crew members are assigned to each specific flight. The Tail Assignment Problem is part of this process and aims to efficiently allocate individual aircraft to flights minimising a certain objective function, such as operational costs, while ensuring some constraints [31].

Based on the concept firstly presented by Ray et al. [47], Quantum Annealing (QA) is a technique used for finding a global minimum of an objective function defined over an energy landscape

using quantum mechanics [39]. Such technique has been used to solve complex optimisation problems with a very large set of possibilities aiming to find global minimum solutions [37] [44]. This dissertation focuses on these two subjects, studying and implementing a QA approach to the Tail Assignment Problem.

1.2 Motivation and Objectives

Forced to reduce profit margins, lowering operational costs becomes one of the main goals of airline companies. Due to a large number of routes, aircraft and crew, scheduling and operational management are the most complex and challenging tasks that airlines need to face every day.

Assigning aircraft to pre-defined flights is one of the most critical tasks regarding airline scheduling, since a good allocation of resources can allow important savings. As aircraft are not homogeneous, taking into account the specifics of each aircraft during the scheduling process is crucial for an effective and efficient assignment of resources [31]. The Tail Assignment Problem has been analysed over the last decade, being most of the times solved using traditional Operations Research (OR) algorithms.

Regarding QA, recent studies have shown promising results on applying such technique to various real-world optimisation problems, such as scheduling problems [37] [57] or flow problems [44].

Given the definition and complexity of finding a proper solution for the Tail Assignment Problem, as well as the good results when applying QA to optimisation problems, the main objective of this dissertation is to study the feasibility of solving the Tail Assignment Problem, considering the operational restrictions and the operational costs, using QA in order to evaluate the usefulness of this kind of technique in such complex domain.

1.3 Main Contributions

By the end of this dissertation we were expected to have some contributions mainly by:

- Detailing a model for a complex scheduling problem, the Tail Assignment Problem
- Analysing and comparing the scalability of using two different techniques to model the same problem, using either a library of creating a model from scratch
- Presenting a comparison between different ways of obtaining solutions for the problem using both classical and hybrid approaches

1.4 Document Structure

This dissertation is divided into six chapters. In the current chapter, *Introduction*, we describe the environment of the airline industry and define the scope of this study. Furthermore, we analyse the how promising QA can be and why it may be useful on solving the Tail Assignment Problem. Chapter 2 provides some background knowledge to allow a better understanding of the key concepts used in this dissertation. Additionally, in the same chapter, an analysis of the existing related work is done, focusing on previous implementations of the Tail Assignment Problem as well as the use of quantum computing for solving real-world problems. Chapter 3 presents the main limitations found on the current state-of-the-art and the research questions we aim to answer. Furthermore, it gives a detailed description of the problem addressed in this study, with a brief analysis of the dataset considered for testing. Chapter 4 details the implementation of two techniques for modelling the previously proposed definition of the Tail Assignment Problem to be solved on a quantum annealer. Chapter 5 contains the tests performed to validate the proposed implementation regarding time, scalability and quality of the obtained solutions. In Chapter 6 an overview of the main conclusions of this dissertation is given, detailing, the main contributions and enumerating some guidance for further investigation on this topic. Finally, Appendix A provides the obtained results that were used for the analysis and conclusions of this study.

Chapter 2

Literature Review

In this chapter, we start by presenting some background in section 2.1, followed by an analysis on previous studies on both quantum computing and the Tail Assignment Problem in section 2.2. Finally, in section 2.3 a brief summary is given.

2.1 Background

This section presents some background on various steps that compose the airline scheduling process and how the Tail Assignment Problem is part of it as well as some concepts of quantum computing.

2.1.1 Airline Scheduling Process

The Airline Scheduling Process aims to efficiently create an optimised master schedule combining aircraft, crew flights and operational constraints. Although it may vary in some phases, this process is usually divided into several sequential steps, happening progressively as shown in Figure 2.1. In fact, it is possible to find studies in the literature where a same phase of the process is named differently [31][11]. The scheduling process starts several months before the flight takes place and is usually composed by three main consecutive steps, namely, Timetable Creation or Flight Schedule Generation, Fleet Scheduling and Crew Scheduling [27]. When the airline scheduling process is complete, some non-contemplated situations, like flight delays or adverse meteorological conditions, can still happen and must be handled by the operational control centre in a process called Disruption Management [16].

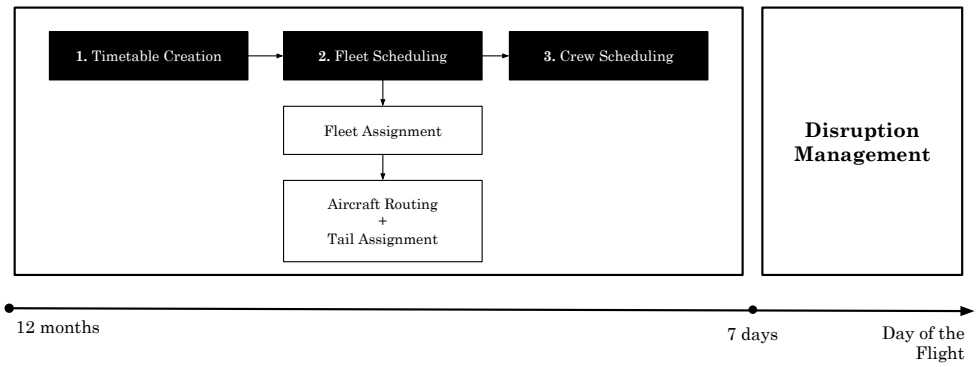


Figure 2.1: Airline Scheduling Process and Disruption Management

2.1.1.1 Key Concepts

Before describing the various steps that compose the airline scheduling process, some concepts must be defined for a better understanding of the context in which they appear.

Activity

An activity is a task that must be performed by an individual aircraft. It may be either a flight or a maintenance task. In the context of this problem, maintenance tasks are considered as pre-assigned activities as they must be performed by a specific aircraft.

Maintenance station

As maintenance tasks may require considerable manpower and heavy machinery, they cannot be performed in every airport. Therefore, a maintenance station is an airport where maintenance tasks must take place.

Routes and Flight strings

A route is a set of flights that are feasible to be executed in sequence by a given aircraft [31]. During the scheduling process, there is, frequently, an advantage in creating partial routes that can be part of bigger routes. These partial routes correspond to aggregated flights. As shown in Figure 2.2, when grouping flights, these are replaced by a new one that starts at the departure airport and scheduled departure time of the first flight and ends at the arrival airport and scheduled arrival time of the last flight. A flight string corresponds to an aggregated flight that starts and ends in a maintenance station [9].

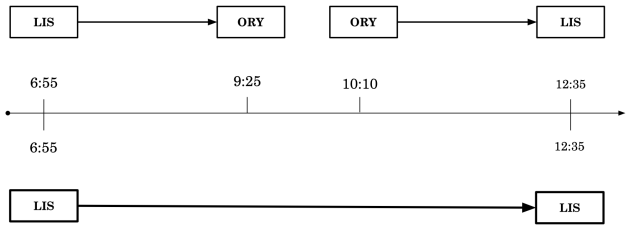


Figure 2.2: Example of aggregating two flights into one. The layout on the top represents the individual flights and the layout on the bottom represents the aggregated flight

Turnaround Time

The turnaround time, sometimes called as turn time, is defined as the minimum time needed for an aircraft to be parked in a gate between two consecutive activities. This period of time is usually necessary for tasks like unloading, loading and cleaning among others [30].

Aircraft Fleet

When considering the different types of existent aircraft it is possible to categorize them in two different fleets, namely *narrow-body* (NB) aircraft and *wide-body* (WB) aircraft. These fleets differ in the fuselage diameter and therefore in the number of existent aisles. On the one hand, a WB aircraft is characterized by a fuselage wide enough to accommodate two aisles with seven or more seats per row. On the other hand, on NB aircraft, the fuselage width has only space for one aisle with up to six seats per row [28]. When it comes to flying time, the two fleets are used in different types of flights. NB are usually used in short-haul (30 minutes to 3 hours) and medium-haul (between 3 hours and 6 hours) flights, while WB are mostly used in long-haul (more than 6 hours) flights.

Aircraft model

Besides being divided into fleets, aircraft may also be characterized according to their model. In fact, two aircraft from a same fleet can have some differences regarding number of seats and associated operational costs among others. Therefore, when planning, it becomes important to take into account the various models of the aircraft that compose the airline fleet.

Operational Costs

Operational costs are the costs related to the operation of a business. In the case of airline companies, these costs can be divided in two general groups: Direct Operational Costs (DOC) and Indirect Operational Costs (IOC). On the one hand, DOC are the costs directly related to aircraft flying operations, such as fuel, navigation and maintenance. On the other hand, IOC are related to the indirect operating expenses such as ticketing, sales and passenger satisfaction, among others [43]. As a matter of fact, DOC can be further divided into two: flight operating expenses

(costs directly related to flying activities such as fuel and navigation costs) and ground operating expenses (the costs that are directly related to the ground activities such as maintenance tasks) [3]. Furthermore direct operating costs are the most significant costs and therefore can highly affect the profitability of an airline company [15].

2.1.1.2 Phases of the Airline Scheduling Process

Timetable Creation

Timetable Creation is the first step and aims to construct a timetable that meets the demand for flights between places at different times. It takes into account the commercial aspects such as which markets are better for the airlines to operate as well as timetable synchronization among different airlines from the same alliance [1] [22].

Fleet Scheduling

After defining a timetable, an aircraft is chosen for each flight [27]. This process is called Fleet Scheduling and is composed by two main stages: Fleet Assignment and Aircraft Routing.

Fleet Assignment is responsible for assigning a specific aircraft type (fleet) for the scheduled flights, taking into account a set of considerations such as forecasted demand and operational costs and maximising the revenue [35]. On the other hand, Aircraft Routing is responsible for determining feasible maintenance routes for each fleet or sub-fleet, according to maintenance rules defined by the aviation authorities [31] [43]. Besides maintenance tasks, routes also take into account the turnaround time.

Aircraft Routing only takes into account general constraints such as fuel capacity or noise level, disregarding specific constraints for individual aircraft such as heavy maintenance tasks. To fill this gap, the Tail Assignment Problem appears as an extension of aircraft routing. In fact, the Tail Assignment Problem is the problem of assigning individual aircraft, also called tail, to flights while satisfying all operational constraints and optimising some objective function [26]. Therefore, solving this problem allows a better allocation and distribution of resources (tails) by the different needs (flights), minimising the operational costs. There are some motivations to perform this assignment earlier such as the possibility of properly consider long-term individual maintenance tasks [31].

Crew Scheduling

Finally the Airline Scheduling Process ends with the Crew Scheduling, which is responsible for assigning crew members to the different existent flights. It is usually performed in two steps, namely crew pairing and crew rostering. In crew pairing, groups of anonymous crew member types are allocated to the pre-calculated flights, considering obligatory labor rules. In the second step, individual crew members are assigned to specific pre-calculated pairings [32].

[1] An airline alliance is an arrangement between different airline companies to coordinate activities regarding the air transportation services for mutual benefits.

2.1.2 Quantum Mechanics

Quantum mechanics is a theory of the physical world that is not deterministic, but probabilistic, with inherent uncertainty [34].

2.1.2.1 Key Concepts

For a better understanding of quantum mechanics and how it can be used for computation, it is important to define some fundamental concepts.

Measurement

A quantum object does not exist in a unique and knowable state. Albeit when observed it only seems to be in a specific state, when not observed it behaves like a wave. The process of observing a quantum object is called "measurement" and consists of collapsing the quantum object from its wave shape to a knowable state, losing information. Because of the loss of information, this process is irreversible and, therefore, a measured quantum object will always have the same value when measured.

Superposition

While representing a wave, a quantum object is said as being in superposition since it can represent two or more states at the same time. This superposition is represented by a linear combination of the contributing states, each one associated with a complex coefficient number that represents their contribution to the final state. The contributing states are said "coherent".

Decoherence

Due to the impossibility of the existence of total isolated systems, the interactions of a system with the environment result in some small measurements, partly collapsing the wave function. This process is called decoherence and it makes the complex coefficient of each state more probabilistic. While using a quantum system, it is important to consider the existence of decoherence as it may affect the reliability of the obtained results.

Entanglement

Under some circumstances, two or more quantum particles can be linked in such a way that the state of one directly affects the state of the others. Linked quantum particles are said to be entangled and due to this characteristic, whenever one of the entangled particles is measured it will collapse the state of the other particles.

2.1.2.2 Quantum Information Science

Quantum mechanics has been having multiple breakthroughs in many fields which, although having different approaches and goals, can not be seen as totally independent. One of these fields is

quantum computing which aims to get answers, with high probability, for problems of interest, based on the manipulation of quantum particles [34].

Quantum computing

Through unusual properties of the quantum world, quantum computing uses a different approach when compared with the classical approach.

Quantum computing is based on quantum particles called qubits. Similarly to a bit in classical computing, a qubit is the basic unit of quantum information. Unlikely a classical bit, a qubit is a two-state quantum-mechanical particle, and therefore it can be found in either one of the basic states or a coherent superposition of both states. While on a classical bit when measured it does not change its state, when it comes to qubits measuring it disturbs its superposition state.

The power of quantum computing comes when the system is composed by n qubits. In fact, a n qubit system requires 2^n coefficients and has 2^n possible states. As mentioned in subsection 2.1.2.1, whenever a qubit in superposition is measured, it collapses to one of the basic states and therefore, as a result, a group of n basic states is obtained.

2.1.3 Quantum Computers

Quantum computers are devices created for accomplishing practical quantum computation [34].

Regarding the existent noise on this type of device, it is possible to detect two different kinds: *fundamental noise* which may appear from a spontaneous change of energy of an object and *systematic noise* which results from uncorrected signal interactions. The latter can occur for multiple reasons such as manufacturing variations or abstraction on the design of the hardware.

Due to the novelty of this topic, a lot of different definitions and considerations can be considered. When it comes to the classification of the different types of existing quantum devices, there is no consensus on the research community regarding which devices are not quantum or how many different classes of quantum computers exist. As presented in Figure 2.3, quantum computers can be divided in two different types, namely Quantum Annealers and Universal Quantum Computers [24]. The main difference between these two types of computers is related to how controllable the system is. While quantum annealers do not allow one to decide what happens during the annealing process, universal quantum computers are based on the idea that each qubit may be changed as desired.

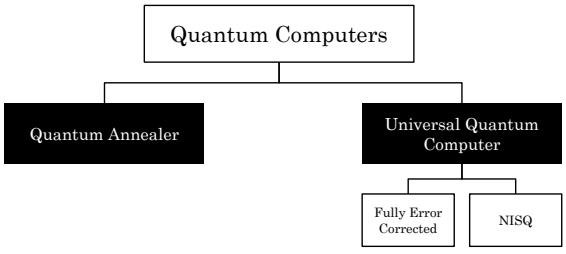


Figure 2.3: Quantum Computers Overview

2.1.3.1 Quantum Annealer

Quantum annealers are computers designed to solve some specific fields of complex problems based on the energetic evolution of a quantum system. To solve a problem in a quantum annealer, it must be firstly defined into its energy levels which together define what is called an energy landscape. These devices make use of a quantum technique called Quantum Annealing (QA) which aims to solve optimisation problems using quantum mechanics [23]. Similarly to Simulated Annealing (SA), QA also aims to find a global minimum of a certain function/energetic landscape, but instead of doing such search based on the ideal of thermal jumps, QA makes use of quantum tunnelling (a property of quantum mechanics based on the fact that each quantum particle is a wave rather that a static particle), to find global minimum solutions rather than just local minimum solutions [39]. Figure 2.4 highlights the main evolution difference between SA and QA.

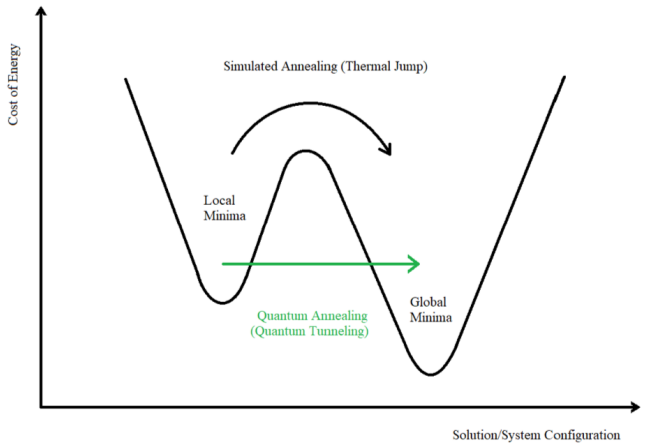


Figure 2.4: Comparison between SA and QA. Adapted from [50]

Defining a problem in terms of its energy levels, can be done through what is called an Hamiltonian. It is a mathematical operator that maps a physical system in terms of its energies. The Hamiltonian is important as it can map different states of the system (eigenstates) to energy levels easily allowing a separation between the lowest energy states and the excited states. The group of different eigenstates that compose a system is called eigenspectrum and represent all the energy levels of the system. The system Hamiltonian can be mapped as shown in equation 2.1,

$$\mathscr{H}_{ising} = \underbrace{-\frac{A(s)}{2}\left(\sum_i \sigma_x^{(i)}\right)}_{\text{Initial Hamiltonian}} + \underbrace{\frac{B(s)}{2}\left(\sum_i h_i \sigma_z^{(i)} + \sum_{i>j} J_{i,j}\sigma_z^{(i)}\sigma_z^{(j)}\right)}_{\text{Final Hamiltonian}} \tag{2.1}$$

where the initial Hamiltonian represents the initial state of the system in which all qubits are set in superposition and the final Hamiltonian, also called problem Hamiltonian, represents the problem aimed to be solved defined on its different energy levels [7].

Regarding the problem Hamiltonian, h and J represent the biases and coupling strengths which are the biases and coupling strengths of the qubits. Biases and coupling strengths can be considered as controllable external factors that can influence the system in such a way that some qubits will be more favourable to fall to a certain state upon measurement. While biases only affect the probability of a certain qubit as an individual, the coupling strengths are related with the probability of two linked qubits to be measured with the same result. Coupling strengths are in some term related to the entanglement phenomena.

Starting from the lowest energy level of the initial Hamiltonian throughout the annealing process the problem Hamiltonian is introduced and therefore other energy levels are also integrated in the system. While annealing, some energy levels may get closer to the ground state, and there is a point where the lowest energy state apart from the ground state gets closer and then diverges again. As presented in Figure 2.5, this minimum distance between the ground state and the first excited state is called minimum gap. The lower the minimum gap, the higher the probability of the system to end up in an excited state. In fact, current quantum annealers not being perfectly isolated some external factors, may cause the system to jump to a state of higher energy. Even though, for the majority of the situations the energy states obtained are still useful.

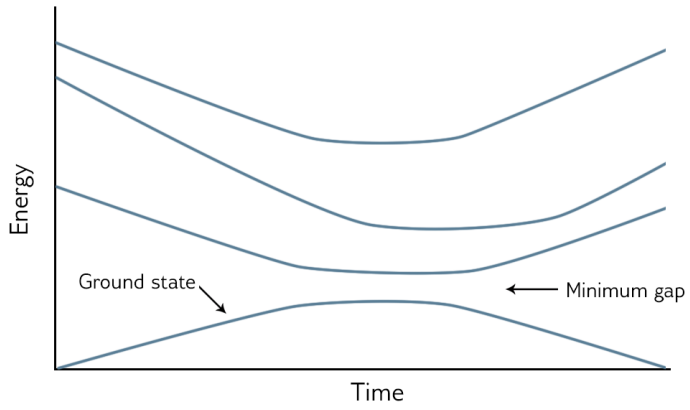


Figure 2.5: Eigenspectrum of a system. Adapted from [7]

The most well-known existent quantum annealers were developed by a Canadian company called D-Wave Systems, which has already developed devices with 2048 qubits. Although this represents a significant accomplishment, these machines still have considerable trade-offs in qubit fidelity, with the presence of some analog errors [40]. Such devices are composed by a processing unit called Quantum Processing Unit (QPU) which is organized in the form of a chimera graph of 256 unit cells. A chimera graph is a structure composed by sets of unit cells. As represented in Figure 2.6, each unit cell is composed by eight qubits each one of them connected to other four qubits.

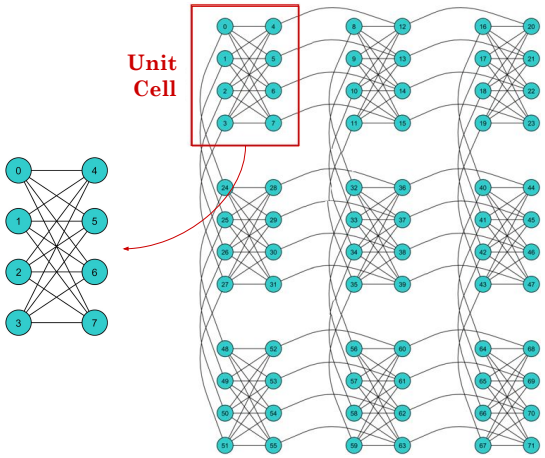


Figure 2.6: Structure of the D-Wave's QPU chimera graph. Adapted from [5]

2.1.3.2 Universal Quantum Computers

Universal Quantum Computers follow a different approach when compared with quantum annealers. Through a well defined and controlled sequence of changes (applying quantum gates) on some qubits of the Hamiltonian of the system, it aims to achieve the desired transformations in such a way that the final state is representative of the problem to solve. This approach is highly similar to the implementation of classical computers.

Since the changes are performed on the scale of the qubit, precision is one of the main issues that need to be addressed. It is possible to distinguish two main kinds of universal quantum computers, namely Noisy Intermediate-Scale Quantum Computer (NISQ) and Fully Error-Corrected Quantum Computer. Thus far, only NISQ (sometimes also referred to as near-term devices) are already implemented as they are designed to tolerate some noise. The best NISQ developed so far has a maximum of 53 qubits [8].

2.1.4 Quantum Algorithms

The power of quantum algorithms comes from the possibility of taking advantage of the entanglement of n qubits. When entangled, n qubits are described by 2^n complex coefficients. Therefore, when updating one of the qubits all the qubits are updated which means that all the 2^n complex numbers that describe the total state of the system are also updated. Thus, it seems that multiple computations are performed with just one step. To take advantage of the computational power of quantum computers, many algorithms have appeared since the end of the last century. Although a lot of time has passed since the first algorithm was proposed by Deutsch and Jozsa [20], only a few algorithms have appeared in this area with the majority of them being proposed to solve well-known and defined mathematical problems. In fact, due to its novelty, the existence of quantum algorithms for real-world case scenarios is still limited.

As some quantum computers may suffer some errors, it is possible to find algorithms that were created to be run on specific devices that guarantee (or not) certain characteristics such as error-correction [34].

Following the same division as the one previously presented for computers, a possible way of categorising the existent quantum algorithms is dividing them by the type of computer they were designed for.

Quantum Annealing Algorithms

These algorithms work by representing the problem using an Hamiltonian. Starting from a low-energy state defined by the initial Hamiltonian (or tunnelling Hamiltonian) where all qubits are in superposition the system evolves in the direction of the problem Hamiltonian where the lowest-energy level represents the solution for the defined problem. Whenever the anneal ends each qubit of the system will no longer be an object in superposition but a well defined classical object.

In order to formulate a realistic problem for being solved in a quantum annealer, it must be possible to be transformed into a Binary Quadratic Model (BQM). A BQM can be formulated using two different approaches, namely Ising Model and Quadratic Unconstrained Binary Optimisation (QUBO) form, with a trivial conversion between them [41]. These approaches aim to model the energy of the system in order to make favourable (less energetic) the best states, so that, whenever the system evolves, the minimum energy is found as well as a solution for the problem. To control the evolution and result of the system, it is possible to control the values of both biases and coupling strengths between qubits. As the system aims to find the lowest energy-state, for each qubit, the higher the value of the bias the lower the probability of such qubit to be part of the solution (take the final value of 1). Similarly, for coupling strengths, the higher the value of the coupling strength between two qubits the lower the probability of both qubits to be part of the solution.

An Ising model is traditionally used in statistical mechanics and each variable can have two different basic states represented by +1 and -1. Equation 2.2 is the representation of Ising model,

$$E_{ising}(s) = \sum_{i=1}^{N} h_i s_i + \sum_{i=1}^{N} \sum_{j=i+1}^{N} J_{i,j} s_i s_j \tag{2.2}$$

where s_i and s_j corresponds to qubits, h_i represents the linear coefficients of the qubits and $J_{i,j}$ are quadratic coefficients representing the strength between each pair of connected (coupled) qubits.

Following the same idea in the QUBO model, each variable can also assume two different states represented by 1 and 0. Equation 2.3 represents a QUBO model,

$$E_{qubo}(a_i, b_{i,j}; q_i) = \sum_{i=1} a_i q_i + \sum_{i<j} b_{i,j} q_i q_j \tag{2.3}$$

where q_i and q_j correspond to qubits, a_i represents the linear coefficients of the qubits and $b_{i,j}$ are quadratic coefficients representing the strength between each pair of connected (coupled) qubits.

Although presented here in a scalar notation, the biggest difference between both notations is that QUBO model can also be redefined using an upper-diagonal matrix whereas Ising model cannot.

In fact, a QUBO model can be expressed by a minimisation problem presented in equation 2.4 where x is a vector of binary variables and Q is an upper-triangular matrix of constants that represent the biases and coupling strengths of the considered binary variable.

$$minimise\ y = x^t Q x \tag{2.4}$$

As many complex problems aim to minimise or maximise a certain objective function, while ensuring multiple constraints, some changes must be considered in order to transform those constraints into the desired unconstrained format. It can be done by introducing quadratic penalties. These penalties are chosen to increase the value of the objective function in a way that while being solved, the resulting QUBO model will search for a solution avoiding incurring in those penalties. Therefore, if no constraint is violated, the resulting QUBO model will still correspond to the minimisation of the objective function [29].

Certain types of constraints have well defined penalty functions. In Table 2.1 are represented the penalty functions associated to some boolean gates. For some of the boolean expressions, as is the case of the $\oplus$ gate, an auxiliar variable may be necessary to ensure the desired penalty.

Table 2.1: Boolean penalty functions

Classical Constraint	**Equivalent Penalty**
$x_3 \Leftrightarrow x_1 \wedge x_2$	$\mathrm{P}(x_1x_2 - 2(x_1+x_2)x_3 + 3x_3)$
$x_3 \Leftrightarrow x_1 \vee x_2$	$\mathrm{P}(x_1x_2 + (x_1+x_2)(1-2x_3) + x_3)$
$x_3 \Leftrightarrow x_1 \oplus x_2$	$\mathrm{P}(2x_1x_2 - 2(x_1+x_2)x_3 - 4(x_1+x_2)a + 4ax_3 + x_1 + x_2 + x_3 + 4a)$

As an example, let us consider the given problem:

$$\begin{gathered} minimise\ y = f(x) \\ \text{subject to constraint} \\ x_3 = x_1x_2 \end{gathered}$$

where x_1, x_2 and x_3 are binary variables. The resulting unconstrained problem could be defined as in equation 2.5.

$$minimise\ y = f(x) + P(x_1x_2 - 2(x_1+x_2)x_3 + 3x_3) \tag{2.5}$$

Therefore, if $f(x)$ is linear or quadratic, this unconstrained problem is defined in the format of a QUBO model, and cases where $x_3 = x_1 * x_2$ will represent lower energy levels than the ones that violate the constraint.

Albeit being a simpler and still good approach, some studies have shown that for some non-deterministic Polynomial time (NP)-complete [2] problems and particularly for 3-satisfiability (3-SAT) [3] problems, the performance of this kind of algorithms decreases exponentially with the size of the problem [56], so there is still a lot of unknowns on how well this type of computing can perform in different situations.

More recently, some algorithms that try to put together QA with machine learning have appeared. These algorithms aim to reduce the time needed to train a Boltzmann machine[4] when compared with classical algorithms [36].

Universal Quantum Algorithms

The algorithms that can run on universal quantum computers are the ones that, due to their granularity and detail, allow one to know with high accuracy all the transformations that happen while computing the solution. Besides the algorithm proposed by Deutsch and Jozsa [20], other important algorithms were proposed by Shor [52] for integer factorization in polynomial time and by Grover [33] to find a unique input for an unknown function that produces a particular output.

Furthermore, and due to the current limitations of universal quantum computers, some other algorithms have been developed to run on a hybrid approach taking care of the possible existent errors in the quantum devices. The quantum approximation algorithms are the ones that aim to solve the problem by using an approximate or heuristic approach. These algorithms are mainly designed considering a hybrid quantum-classical approach, providing useful approximation results for the problem using low resources. Variational quantum algorithms are the group of algorithms that run interactively starting from a guess as input and trying to improve the input on the next iterations until reaching a good result. These algorithms normally run on both quantum and classical devices, using the quantum computer to perform the optimisation step, while the classical computer verifies whether or not a new iteration should be executed [34]. Examples of variational algorithms are the Variational Quantum Eigensolver (VQE), which divides a big problem in the sum of small problems and treat them independently [46], or the Quantum Approximate Optimisation Algorithm (QAOA), that tries to solve an optimisation problem through a variational guess of the wave function that satisfies the problem [21].

[2] NP-complete is the set of NP problems that are reducible between each other in polynomial time. A NP problem is a decision problem for which it is not possible to find a solution in polynomial time, but an existent solution can be verified in polynomial time.

[3] A 3-SAT problem is a satisfiability problem that was proven by Miller et al. [42] to be NP-complete.

[4] A Boltzmann machine is a stochastic neural network that can be defined based on a Hamiltonian

2.2 Related Work

In this section, an analysis of the related work is performed in order to understand what is the state-of-the-art on both Tail Assignment Problem and quantum computing fields. Firstly and due to the numerous existent studies, a succinct review is performed on the different approaches already used to solve the Tail Assignment Problem. Furthermore, a brief analysis is done on the research studies that apply quantum computing to real-world problems. Due to the shortage of studies on this field, a wider approach is considered, taking into account related research topics such as scheduling problems.

2.2.1 The Tail Assignment Problem

Contrarily to the other steps of the airline scheduling process, the Tail Assignment Problem has been studied more intensively only over the last decade.

Grönkvist [31] studied an hybrid approach, using constraint programming and a branch-and-price algorithm, a well-kwown algorithm used in OR for solving Linear Programming (LP) problems which obtains a solution using branch-and-bound together with a price problem [10]. By defining the Tail Assignment Problem as a set partitioning problem based on pre-calculated routes, it starts by modelling the problem as a Constraint Satisfaction Problem (CSP). Such modelling is important for generating an initial feasible solution that is later optimised using a branch-and-price algorithm solving the problem for a fixed period of time and taking into account some specific activities and irregular schedules. The combination of both algorithms allows improving initial solutions that respect operational constraints such as maintenance, turnaround times and pre-assigned activities. Testing the effectiveness of such approach, the authors applied the model in a real-world scenario helping an airline company to reduce costs minimising the need for aircraft leasing. Furthermore, it also revealed that the Tail Assignment Problem is a NP-hard problem.

In Borndörfer et al. [13] the author presents an approach for solving the Tail Assignment Problem in such a way that the resulting schedule is robust to flight delays. Making use of a branch-and-price algorithm that take advantage of historical data for short-haul flights, the resulting model is thought to be used together with a column generation algorithm.

Following also a branch-and-price algorithm, Ruther et al. [49] presents a solution for the Tail Assignment Problem together with the crew pairing. This approach is thought to be executed only few days before the operation, creating scheduled routes for specific aircraft instead of generic ones. Considering specific aircraft details such as location, maintenance plan and flying history, aircraft routes can be designed using updated information. This approach was tested using real world data, proving to be a valid solution.

In Froyland et al. [25] the authors solved a recoverable version Tail Assignment Problem for being used during the Disruption Management phase, by creating a model for recover from flight delays while ensuring minimum recovery costs. Such model was solved using Benders decomposition, a technique for solving linear programming, by breaking down the problem into two stages solving two smaller sub-problems [12].

In Montoito [43] the author presents a solution for the Tail Assignment Problem using SA with an adaptive neighbourhood local search approach. Starting by obtaining an initial feasible solution using a First-In-First-Out (FIFO) approach, it made use of a SA algorithm to minimise the operational costs. As result, this study pointed out that, when applying it to a real case scenario of a Portuguese airline company, it can save thousands of euros when compared with a solution where all aircraft are considered to be homogeneous.

In Yadav [59] the authors considered a multi-objective optimisation using a genetic algorithm in order to minimise the operational cost while maximising the flight distance. When compared with a lagrangian relaxation, this approach performs better returning optimal paths which are essential for facing operational issues. Despite the good results presented, this study took into account a small dataset.

Albeit the Tail Assignment Problem takes into account individual aircraft characteristics, some studies have focused on creating schedules by only solving Fleet Scheduling without taking into consideration Tail Assignment, and therefore, different aircraft from a same model end up being considered homogeneous. Here, we detail two of those studies. In Barnhart et al. [9] the authors presented a solution for solving the aircraft routing problem together with the fleet assignment through the generation of flight strings. The problem was then solved making use of a branch-and-price algorithm. In Zhao et al. [60] the authors reveal a solution for the fleet assignment problem considering different aircraft models and their characteristics as well as the scheduled flights. This approach aims to maximise the final revenue while finding out a suitable solution in a timely manner.

2.2.2 Quantum Computing

Considering the different algorithms mentioned in subsection 2.1.4, QA algorithms are the ones with more applications in real-world problems, with some few implementations of QAOA appearing in the latest years. Due to the few research developments in this area, no studies were found applying QA to the Tail Assignment Problem. Therefore, a wider analysis was performed searching for another kind of problems that could have an implementation using this approach.

Tran et al. [54] present an hybrid quantum-classical approach. Through the division of complex problems in smaller sub-problems, the authors used a quantum annealer to obtain strong candidate solutions for each of the considered sub-problem. On the other hand, the classical processor keeps a global search tree while ensuring that the global problem keeps all the necessary constraints that are relaxed on the considered sub-problems. Some empirical experiments were done mapping three different scheduling real problems into a QUBO formulation, concluding that, for most of the cases, a quantum annealer can be useful for pruning and guidance on the search process in a global search tree approach.

In Venturelli et al. [57] the authors propose a solution for the Job Shop scheduling problem using a QA approach. By modelling the problem as a BQM they concluded that, although the used quantum annealer was limited for solving the problem, the usage of hybrid approaches and meta-heuristics were fruitful and should exist further investigation on such area. Furthermore, they also

concluded that pruning unnecessary variables is an important step as it may highly reduce the size of the problem.

Moreover, Stollenwerk et al. [53] tried to implement flight gate assignment minimising the total time for transit passengers. Converting this NP-hard problem to a QUBO model, they tested how a quantum annealer, in this case a D-Wave 2000Q, could perform solving this problem. Due to hardware limitations, only a small portion of the real data from a mid-sized German airport was considered, concluding that extracting problem instances from data can lead to a QUBO model with distributed coefficients, reducing the success probability.

Another study was also performed in D-Wave 2000Q quantum annealer, but this time for solving the Nurse Scheduling Problem with hard constraints Ikeda et al. [37]. In this study, the authors started by using a QUBO model and then translating it to an Ising model. It concluded that satisfiable solutions could be achieved using QA but only for small samples due to hardware constraints. They also shown that, for some cases, applying reverse annealing, i.e., a technique to refine good known local solutions, on the results from the QA implementation improved the final solution. Finally, a test was performed trying to decompose the big problem in smaller problems using an existent library called qbsolv that iterates over the smaller problems using a Tabu search technique. This approach also helped to get satisfiable solutions for the problem.

Analyzing now the application of QA on another type of real-world problems not involving scheduling, only few results were found. On their study Neukart et al. [44] implemented a simplified version of the traffic flow problem for the German company Volkswagen testing it in a D-Wave 2000Q device. Using the library qbsolv this study shows that QA is suitable for this kind of problems.

Some other QUBO models were also defined for well-known problems such as the graph partitioning problem [55], the maximum clique problem [17], the traveling salesman problem [45] or the minimum multi-cut problem [18]. Furthermore, in Lucas [41] the author proposes an Ising formulation for multiple well-known NP-hard and NP-complete problems, that can be run on a quantum annealer.

When it comes to other quantum approaches, Vikstål et al. [58] present a partial implementation of the Tail Assignment Problem using QAOA. In this study the goal was to find a feasible solution considering some constraints. Due to the limited capabilities of the used quantum computer, a limited number of pre-calculated routes (using branch-and-price) was considered, ensuring that one of these routes was a solution with all the flights assigned. It also considered that all the aircraft would be used. As a goal, the implementation was converted to an Exact Cover/Set Partitioning Problem, where each route was selected by exactly one tail and no flights were included in two of the selected routes. Furthermore, it was able to find a solution satisfying all the constraints disregarding the costs.

2.2.3 Key findings

For the Tail Assignment Problem, as presented in Table 2.2 it is possible to understand that heuristic algorithms were used for all the studies with branch-and-price being the most common. Thus, such heuristic algorithm returned satisfactory results for the majority of the cases. It is also noticeable that the minimisation of costs is one of the main objectives in most of the studies. Multiple other studies have been developed regarding aircraft routing and fleet assignment, with most of them using the same OR algorithms for solving the proposed problems.

Table 2.2: Summary of the Tail Assignment related work

Reference	Algorithm	Optimisations	Use Real Data	Remarks
Grönkvist, 2005 [31]	Constraint Programming + Branch-and-price	Minimise medium-haul connections	Yes	1. Fast initial solution 2. Allow reducing aircraft leasing 3. Can get highly quality solutions
Borndörfer et al., 2010 [13]	Stochastic programming + Branch-and-price	Minimise the probability of delay propagation	Yes	Generalization for other problems (e.g. crew scheduling)
Ruther et al., 2017 [49]	Branch-and-price	Minimise routing + Minimise pairing cost	Yes	Combined: aircraft routing + crew pairing + tail assignment
Froyland et al., 2013 [25]	Benders decomposition	Minimise recovery costs	Yes	Possibility of reducing recovery costs upon disruption
Montoito, 2016 [43]	SA	Minimise operational costs	Yes	1. Fast initial solution 2. Allows savings when compared to the current implementation for the considered airline company
Yadav, 2017 [59]	Genetic	Minimise operational costs + Maximise flight distance	Unknown	Small dataset

Regarding the applications of quantum computing, one of the main issues to point out is the existent hardware limitations that restrict implementations to be simplified and tested with only a few amount of data. As presented in Table 2.3, most of the existent studies solving complex real-world problems used a QA approach. The majority of them used hybrid techniques with a quantum computer running the most difficult calculus while a classical algorithm was managing the direction of the evolution of the solution. Some good results were obtained, suggesting that further research should be considered.

Table 2.3: Summary of the Quantum Computing related work

Reference	Device	Addressed Problem	Algorithm
Tran et al., 2016 [54]	Quantum Annealer	Mars Lander Task Scheduling Airport Runway Scheduling	Hybrid
Venturelli et al., 2016 [57]	Quantum Annealer	Job-Shop Scheduling	Hybrid
Neukart et al., 2017 [44]	Quantum Annealer	Traffic Flow	Hybrid
Stollenwerk et al., 2019 [53]	Quantum Annealer	Flight Gate Assignment	QA
Ikeda et al., 2019 [37]	Quantum Annealer	Nurse Scheduling	QA + Hybrid
Vikstål et al., 2019 [58]	Universal Quantum Computer	Tail Assignment	QAOA

Summing up, it is possible to conclude that the Tail Assignment Problem is a problem that has been studied over the last decade most of the times in a perspective of minimising operational costs and making use of heuristic algorithms commonly associated to OR. Furthermore, it is also understandable that although having some scalability issues, QA has successfully been applied to multiple complex optimisation problems of considerable sizes, being a hot research topic in the last years. Furthermore, the unique application of a quantum approach to the Tail Assignment Problem was developed for a universal quantum computer and revealed the need of a lot of pre-processing being only able to find feasible solutions rather than cost optimised solutions. Since QA has been proven to be extremely useful solving multiple NP-hard problems and, to the best of our knowledge, has never been applied to the Tail Assignment Problem, this study aims to fulfil such gap.

2.3 Summary

The Tail Assignment Problem is the problem of assigning individual aircraft to flights considering specific maintenance needs and regulatory constraints while minimising operational costs. Such a problem has been solved using multiple OR algorithms.

Quantum computing is a sub-field of quantum information science, that aims to solve computing problems using quantum mechanics. Due to its inherent properties, it follows the fundamental principles of quantum mechanics such as measurement, superposition, entanglement, among others. QA is a technique that aims to solve optimisation problems on real quantum devices considering the inherent existent noise. Multiple algorithms have been developed for the different quantum computers being the NISQ and quantum annealers the ones with algorithms already tested in physical devices. Regarding the application of QA to the Tail Assignment Problem, no studies were found so the present study aims to fulfil such a gap.

Chapter 3

Problem Statement

Section 3.1 summarises the identified main limitations in the studies found in the literature. A detailed description of the problem is given in section 3.2. In section 3.3, the three main research questions we aim to answer in this dissertation are presented. Finally, in section 3.4, we analyse the dataset used posteriorly to test the quality of the implemented solutions.

3.1 Current Limitations

As presented in Chapter 2, the existent solutions offer great alternatives for the Tail Assignment Problem, but some of them require the calculus of an initial feasible solution and sometimes also pre-calculated routes are required. In fact, since solving the whole problem at once is highly complex. Usually, an iterative approach is used to improve an existent solution through heuristic algorithms, which may end up not searching in the proper direction, finding sub-optimal solutions. As the majority of these algorithms are somehow probabilistic and due to the lack of knowledge of which is the best solution, it becomes difficult to conclude when the algorithm should put a stop to the incessant search for an even better solution.

3.2 Proposal

The purpose of this dissertation is to model the Tail Assignment Problem for being solved in a quantum annealer. Instead of focusing on an implementation that would cover all the possible constraints and scenarios, in this study, we focus on a simplified version of the problem. Our goal is to obtain a scalable modelling for a QA approach that can be useful for understanding the viability of further investigation on solving the full version of the problem using such technique.

As described in subsection 2.1.1, the Tail Assignment Problem aims to solve the problem of assigning timetable/scheduled flights or pre-defined routes to a tail, so that all flights have an assignment and all constraints are met.

We define the Tail Assignment Problem as the problem of assigning a set of scheduled flights to a set of tails, ensuring that multiple operational constraints are satisfied while minimising the associated objective function.

This section describes the constraints and optimisation criteria and the assumptions used to narrow the scope of the problem.

3.2.1 Constraints

As previously proposed by Grönkvist [31], the considered proposal of current work for the Tail Assignment Problem takes into account some specific constraints defined by four different groups: *Assignment Constraints*, *Connection Constraints*, *Maintenance Constraints*, and *Flights Restriction Constraints*. The main goal of using such constraints is to guarantee that the obtained solutions are feasible for a simplified real-world application.

3.2.1.1 Assignment Constraints

Assignment constraints are responsible for ensuring that no activities are left unassigned and that each activity is attributed only to one tail. Following this constraint, it is possible to guarantee that each and every flight is allocated exclusively to one aircraft.

3.2.1.2 Connection Constraints

Connection constraints are the most basic ones and can be as simple as ensuring that two activities may connect one another, i.e., can be operated in sequence. To operationally define these constraints, network modelling techniques are often used. These techniques can be classified as cyclic if they repeat for a finite period of time, or non-cyclic if initial and final activities are defined. Regarding the network modelling, there are two different techniques to be considered: connection network and time-line network [51] [9].

The connection network is a modelling technique that represents each activity as a node of a direct graph and each legal pair of nodes is connected by an arc. A pair of activities is considered legal and an arc may be established between them, if, not only the outbound (or departure) airport of the second activity is the same as the inbound (or arrival) airport of the first activity, but, also, the departure time of the second activity is later than the arrival time of the first activity, considering the turnaround time. If a connection does not satisfy the defined constraints then it is considered illegal and, therefore, not represented in the graph [31]. Figure 3.1 is an example of a connection network involving four airports, seven flights and three tails.

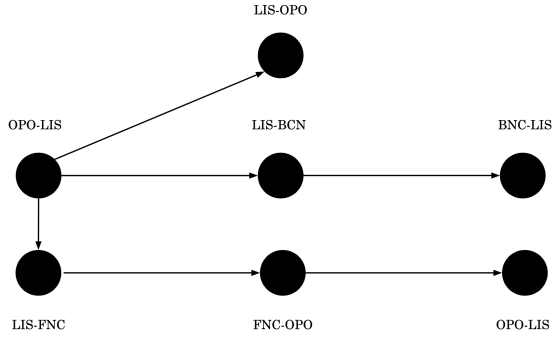


Figure 3.1: Connection network example

On the other hand, as firstly presented by Hane et al. [35], the time-line network represents each airport as a time-line in which each node represents the departure or arrival of a flight to or from that airport. In Figure 3.2, there is a representation of a time-line network of the previous example.

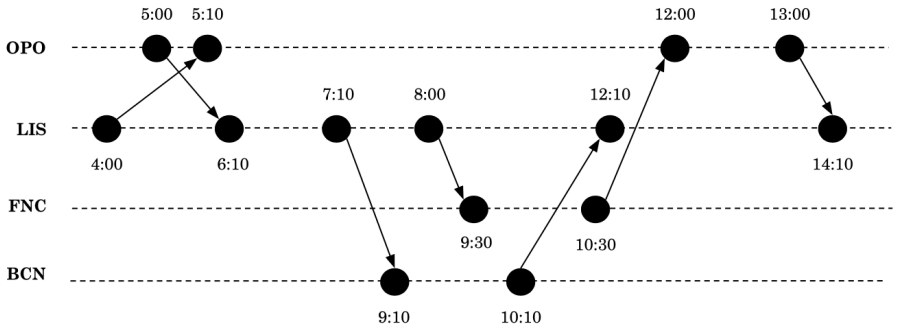


Figure 3.2: Time-line network example

Aiming to solve a problem involving scheduled activities, in this dissertation we considered a non-cyclic connection network in which nodes represent activities and edges represent possible connections between them. In other words, two connected nodes represent a pair of activities that can be performed by the same tail and each group of connected nodes represent a valid sequence of flights that can be assigned to the same tail. The latter is, in the context of this dissertation, called as valid path. As activities must happen at a specific date and time, no backward edges are valid.

3.2.1.3 Maintenance Constraints

Maintenance constraints are one of the most important operational constraints. Despite being generally set by national and international agencies, each airline is responsible for, following pre-determined guidelines, define types (also known as checks) of maintenance and time intervals in which they should take place. Maintenance tasks are usually divided into two categories: pre-assigned tasks and minor maintenance tasks. On the one hand, minor maintenance tasks can be

performed in short intervals of time and are not pre-defined, but rather depend on tail usage. On the other hand, the pre-assigned maintenance tasks are defined by a long term maintenance plan and are usually more time consuming as they require the aircraft to be on the ground for long periods. An example of different maintenance checks may be found in Table 3.1.

Table 3.1: Example of maintenance checks and intervals. Adapted from [43]

Maintenance Checks	Interval	Category
T	Before each flight	Minor
T1	36 hours	
T2	8 days	
A	4 months	Pre-assigned
B	8 months	
C	24 months	

In this study, we only considered pre-assigned maintenance tasks, in which a maintenance plan, including the location and time of each task, is provided beforehand. Such constraints ensure that each tail is available and on time at the proper airport to perform the scheduled maintenance task.

3.2.1.4 Flight Restriction Constraints

Another type of constraints that should be taken into account while creating a valid schedule is Flight Restriction Constraints. These constraints ensure, among other things, what is known as curfews. Curfews limit the possibility of certain flights to be operated by certain tails as these tails are not allowed to land on the destination airport due to noise or aircraft size restrictions. Furthermore, Flight Restriction Constraints exclude the assignment of flights to tails that do not have enough fuel tanks or seats, or that do not meet the required fleet to operate it.

In this study, we considered that each flight has a minimum required fleet as well as a minimum number of seats corresponding to the demands which must be met. Therefore, tails belonging to a smaller fleet or not having the required number of seats were considered as being unfit to perform the flight.

3.2.2 Optimisation criteria

As mentioned in subsection 2.2.1, several studies consider the optimisation criteria as the minimisation of a certain objective function. Although there are multiple approaches to this problem, many studies use this function to lower the operational costs associated with a given solution [43]. Other studies have used this objective function to minimise the number of medium length connections [31].

In this dissertation, we set an objective function to minimise some of the operational costs, as presented in subsection 3.2.2.1. Although not being part of the objective function, other costs are included as part of the total cost of any obtained solution as described in subsection 3.2.2.2.

3.2.2.1 Objective Function

As presented in Chapter 2, operational costs may be divided into two main groups: IOC and DOC. In this study, we only included the DOC related to the execution of flights by tails. In fact, as scheduled maintenance tasks associated with each tail are inherent and immutable, no *Maintenance Tasks' Costs* were considered in our objective function. Furthermore, costs related to the time a tail remains on the ground not performing any activity, here called *Standby Parking Costs*, are also not included in the objective function.

Figure 3.3 is a representation of the various costs taken into account in our study. Although these costs are all related to the execution of a flight by a given tail, for a matter of simplicity, we divided them into three parts: *Ground costs*, *Takeoff/Landing costs*, and *Flying costs*.

Ground costs are fixed for different airports and depend on the chosen aircraft model. They are branched in two parts: Airport Handling fee and Turnaround Parking cost. Airport Handling fee is a fixed value corresponding to the cost of loading and unloading the aircraft. Turnaround Parking cost depends on the amount of time an aircraft takes to complete loading and unloading procedures.

Takeoff/Landing costs correspond to the Takeoff and Landing fees that an airline company must pay for the usage of the airport runway. As different airports may have different Takeoff and Landing fees, we consider these costs as airport-dependent. Thus, Takeoff and Landing fees depend on the departure and arrival airports, respectively.

Finally, the *Flying costs* are associated with the air-time and are split into two parts as well: Jet Fuel costs and Air Traffic Control (ATC) costs. Jet Fuel cost is the monetary value of the fuel needed for a certain aircraft to perform the desired flight. ATC cost is the value charged for the monitoring done by the air traffic controllers as well as taxes that each country charges for flying over the country air space during the flight. ATC cost is considered as being time-dependent and Jet fuel cost as distance-dependent. The flight duration is given by the scheduled block time (i.e., the difference between the scheduled departure time and the scheduled arrival time). The flight distance is the length between the two airports it connects.

Therefore the objective function is defined as in equation 3.1, where F represents the set of all flights that must be performed.

$$\begin{aligned} ObjectiveFunction = \sum_{f}^{F} (&Airport\ Handling\ Costs_f + Turnaround\ Parking\ Costs_f + \\ &+ Takeoff\ Fees_f + Fuel\ and\ ATC\ costs_f + Landing\ Fees_f) \end{aligned} \tag{3.1}$$

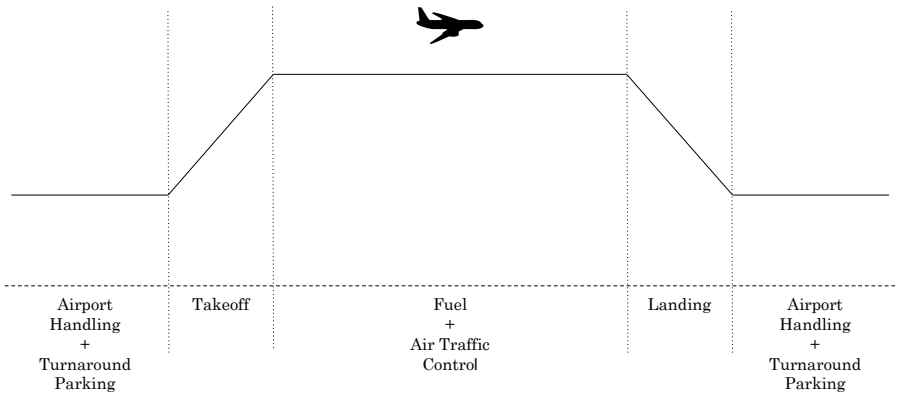


Figure 3.3: Distribution of the considered operational costs on the different moments of a flight

3.2.2.2 Solution Total Cost

Although some of the costs are not considered by the objective function, they still need to be included in the total cost of any solution. As presented in equation 3.2, the total cost of the solution is obtained by adding *Maintenance Tasks' Costs* and *Standby Parking Costs* to the costs associated with the execution of the flights, as previously defined in the objective function.

$$\begin{aligned} Solution\ Total\ Cost = Objective\ Function + \sum_{f}^{F} Maintenance\ Tasks'\ Costs_f + \\ + \sum_{t}^{T} Standby\ Parking\ Costs_t \end{aligned} \tag{3.2}$$

3.2.3 Assumptions

Due to the specific scope of this dissertation, multiple aspects of a real-world application were not granted. Besides the simplifications previously presented, we took into consideration some assumptions to easily deal with the problem.

i) While calculating a solution for the problem, we assumed that it was always possible to obtain a valid solution assigning all the flights;

ii) A pre-defined turnaround time value was chosen, disregarding tail or flight characteristics;

iii) Tails did not have an initial pre-defined location.

The first assumption is taken as it is not desirable that all flight are part of the schedule. In fact, leaving flight without any tail assigned can bring negative effects to the airline company, as it would require to compensate the clients that would not be able to fly to their desired destination.

The second assumption was taken into account the assumptions also considered by Montoito [43] where the turnaround time was considered as a fixed value corresponding to the average of this time in different situations. Furthermore, Grönkvist [31] also indentifies that such time does not vary too much so it can be considered as an average in some cases for a matter of simplicity.

Finally, the third assumption was considered since there was no knowledge on the initial location of each one of the tails.

3.3 Research Questions

Considering the previously identified issues and taking into account the encouraging results of solving combinatorial problems using QA [48], some research questions arise. In this study, we formulated three main questions to guide us throughout the modelling of the Tail Assignment Problem for such a technique.

Those questions were:

RQ1. Can the Tail Assignment Problem be modelled to run on a quantum annealer?

RQ2. Is this approach scalable?

RQ3. Are the results obtained from such modelling favourable?

To answer the first question we start by detailing our definition of the problem in section 3.2, followed by presenting a possible modelling of it in Chapter 4. Aiming to answer questions two and three, in Chapter 5, we run some experiments and analysed the obtained results.

3.4 Dataset Analysis

For a proper validation of the proposed solution, a dataset was provided by the Portuguese airline company *TAP Air Portugal*[1]. It is composed of 1965 short and medium-haul flights that together connect 25 different airports resulting in a total of 55 unique city pairs (i.e., pairs of locations that are connected by at least one flight). Each airport is identified by its IATA (International Air Transport Association) code. Moreover, the considered airline fleet is composed of 42 different aircraft and 30 maintenance tasks of types A and C. This dataset is representative of part of the flights performed by this company during September 2009.

3.4.1 Flights Analysis

With 1965 short and medium-haul flights, the dataset corresponds to 30 days' data summing up a total of 4003 block hours. Block hours are defined by the time interval between the moment an aircraft closes the door at the departure airport to perform a flight until the moment it is opened again upon arriving at the destination.

[1] http://www.flytap.com/

Among other information, each flight is characterized by both scheduled time of departure and arrival, as well as an expected number of seats demand (i.e., the minimum number of seats that the tail performing the flight must have).

Almost 39% of the flights are Portuguese domestic flights, i.e., flights with Portuguese departure and arrival airports. When considering international activities, almost 60% of the flights are in European soil, mainly between Portugal and another country. The most common foreign airports on our dataset are Spain, France and Belgium, as shown in Figure 3.4.

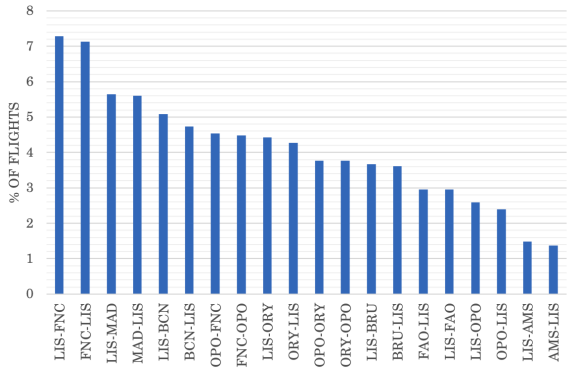


Figure 3.4: Top 20 most frequent connections in percentage from total number of observations

Regarding the distribution of flights by location, as shown in Figure 3.5, the influx is roughly the same when comparing the departure and arrival airports. Almost 80% of the flights leave or arrive in Lisbon. This is understandable as the airline company is well known for operating in a hub-and-spoke network, i.e., the majority of the flights depart or arrive in a hub [43]. A hub is a term that defines one of the main airports in which an airline company operates. In fact, a hub may also be a maintenance station as maintenance tasks can be held there. A spoke is a term used for referring to all the airports where the airline company operates and that are not hubs.

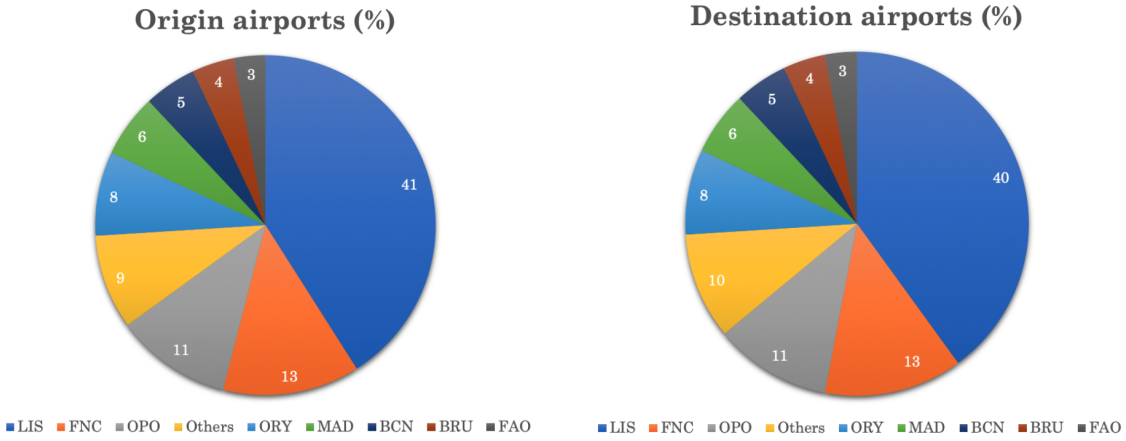


Figure 3.5: Percentage of departures and arrivals by airport

As presented in Figure 3.6 and considering the flying time as the difference between the scheduled departure and arrival times, it is relevant to highlight that the average block time is of two hours. 90% of them are short-haul flights taking between 30 minutes and three hours of flying time.

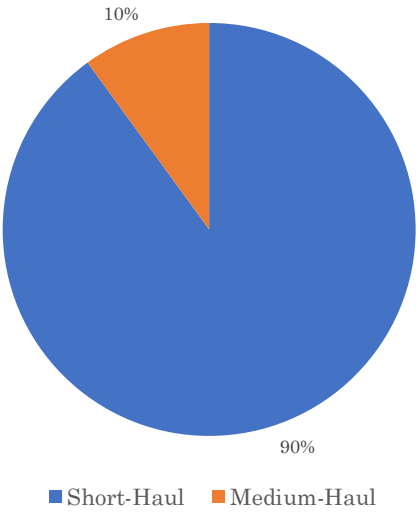


Figure 3.6: Percentage of flights of short-haul and medium-haul

Analysing the daily distribution of flights, as presented in Figure 3.7, it is possible to understand that days 24 and 25 may be considered as outliers since the number of flights on those two days is significantly smaller when compared with the other days of the month. Disregarding those two days, there is an average of 69 flights per day. Taking into account the considered airline fleet of 42 aircraft, each aircraft should perform almost two daily flights on average.

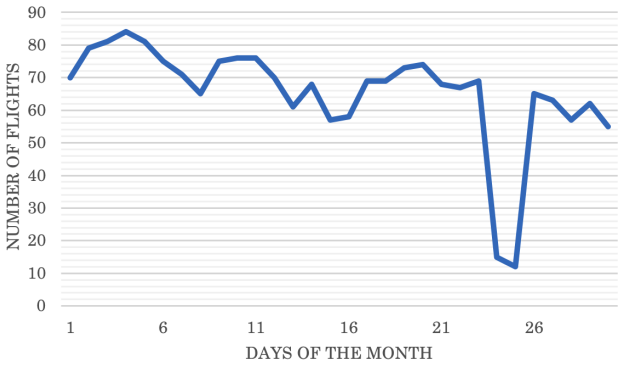


Figure 3.7: Distribution of the flights by scheduled day of departure

Moreover, as presented in Figure 3.8, the majority of the flights happen between 6 a.m (6h) and 10 p.m (22h), so aircraft are expected to have a more intensive use during these hours.

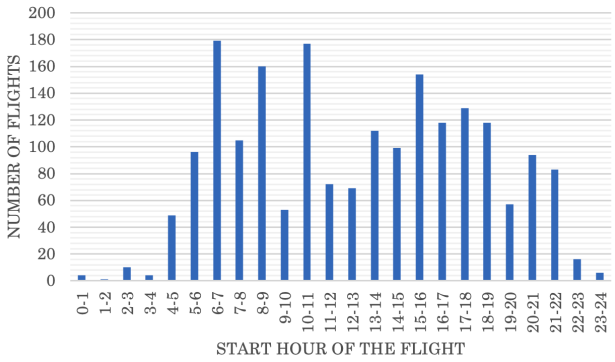


Figure 3.8: Distribution of flights by scheduled hour of departure

3.4.2 Aircraft and Maintenance Analysis

To perform this set of flights, the dataset is composed by 42 NB aircraft, divided into 3 different models 319, 320 and 321, with passenger seats' capacity of 138, 136 and 210, respectively. Figure 3.9 (a) represents the percentage of tails by aircraft model, considering the total fleet of the dataset, while Figure 3.9 (b) represents the cumulative percentage of flights that can be performed by tails from each aircraft model regarding flights' demand and seats capacity of the different models. Analysing Figure 3.9, it is possible to conclude that tails of the 321 model are expected to have a higher number of flights assigned per aircraft when comparing with tails of other models.

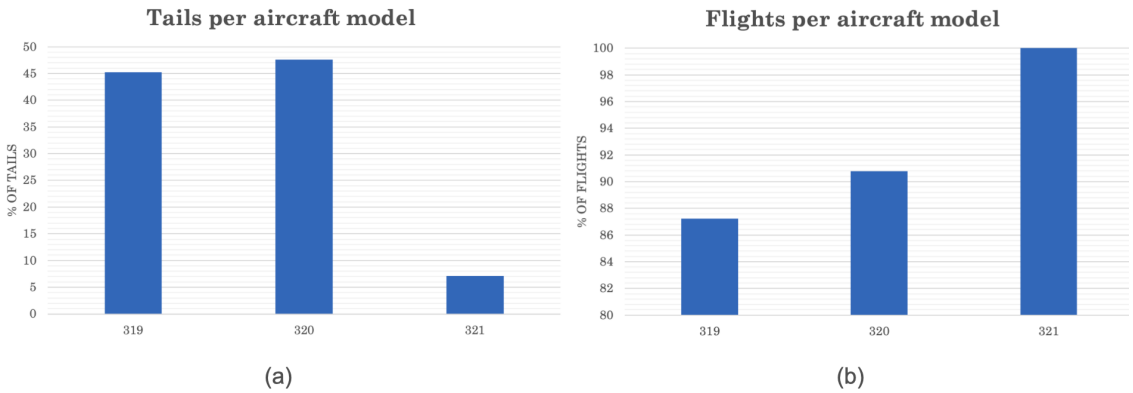


Figure 3.9: (a) Percentage of tails by aircraft model; (b) Percentage of flights that can be performed by aircraft model

Regarding pre-assigned long term maintenance tasks (i.e., maintenance tasks of type A, B or C), which are the only maintenance tasks our implementation focuses on, the dataset includes 30 maintenance tasks to be performed by 27 different aircraft. In fact, the dataset only has type A and

C maintenance tasks, with an average duration of one day and one hour for type A maintenance tasks and of approximately 15 days for type C maintenance tasks.

According to the literature, type A maintenance tasks are more frequent than type C maintenance tasks [43]. Such tendency was also found in our dataset, which is composed of 28 type A maintenance tasks and two type C maintenance tasks.

Analysing the distribution of the type A maintenance tasks, it is possible to conclude that there is approximately one aircraft daily performing such activity. Looking into the start and end times of the maintenance tasks, it is possible to understand that the majority of them start at the beginning of a day, ending at the end of that same day as represented in Figure 3.10. This schedule organization allows the airline company to take the opportunity of having the biggest possible number of tails available when the number of flights is higher.

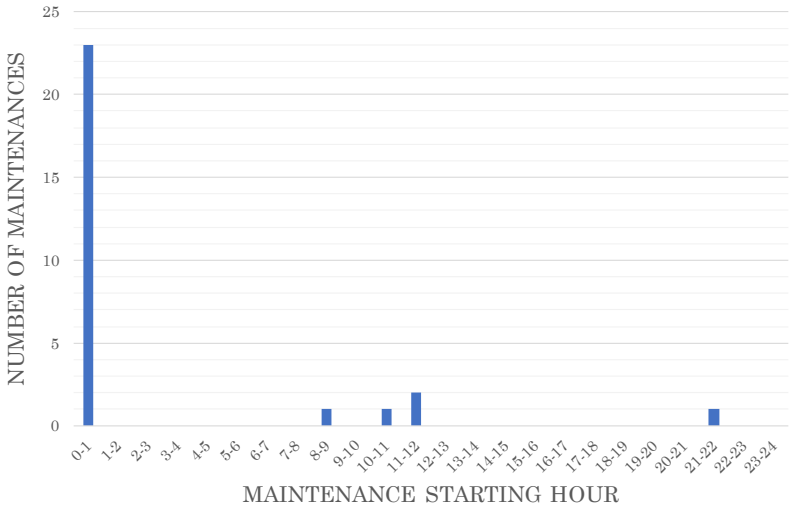


Figure 3.10: Distribution of type A maintenance tasks by scheduled starting hour

Although the existing dataset only includes one maintenance station, in order to enlarge the scope of our implementation, in this study we considered the possibility of the existence of multiple maintenance stations.

3.4.3 Data Aggregation

Taking into account that the airline company operates in a hub-and-spoke model, to reduce the number of flights it may be important to aggregate some flights. Therefore, and following the idea presented by Montoito [43], we decided to put together flights that have a high probability of being operated by the same tail. Bearing this in mind, we proceeded to partially aggregate data, grouping flights with only one possible previous connection. Regarding costs, the cost of carrying out the aggregated activity corresponds to the sum of the costs of the consecutive flights included in the aggregation. The only exception of this algorithm are flights starting at one of the hubs.

Such flights are not considered to be aggregated with any previous one as it could unnecessarily interfere with the existent pre-assigned maintenance tasks.

For our dataset, by doing such aggregation, it was possible to reduce the number of flights by 15% for a total of 1677 flights, with 288 of them corresponding to aggregations. Analysing the results presented in Figure 3.11, it is worth mentioning the relevance of some of the aggregated flights. In fact, almost 94% of the aggregated flights correspond to flight strings, i.e individual flights that are aggregated in such a way that the aggregated activity starts and ends in a maintenance station (LIS).

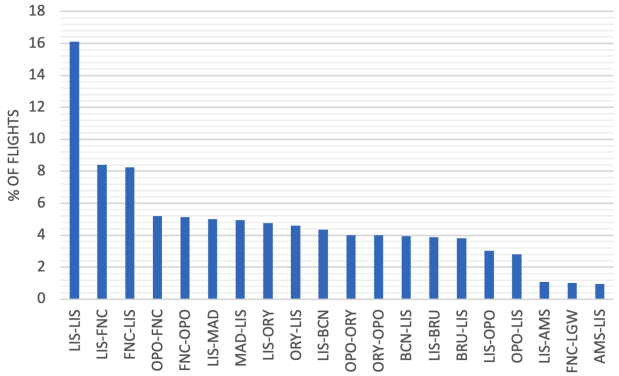


Figure 3.11: Top 20 most frequent connections considering aggreagted flights. Connections where the departure and arrival airport are the same represent aggreagted flights, whereas the others represent individual flights

Despite the good results with such aggregation, the number of non-aggregated flights has still a big importance in the overall set. Thus, other heuristics aiming to ensure more groups were also tested. Using additional aggregation heuristics, it would be possible to increase the number of groups and therefore, reduce the number of individual flights. However, when verifying such aggregations, it would require a high number of flights not to be carried out (, position flights) to guarantee a possible valid schedule. Since one of the main goals of this study is to calculate a feasible solution for the existent flights, we decided to keep the original grouping rules that guarantee a valid schedule for the flights of the dataset. Since the used dataset does not include all the flights performed by the airline company during the considered period, it may be the case where the missing flights would allow further aggregation.

3.5 Summary

In this chapter, we defined the scope of the dissertation, presenting the research questions that guided the study. Furthermore, we detailed the proposal of the problem to be implemented, specifying the constraints and operational criteria. Some assumptions were further defined to reduce the complexity of the considered problem. Finally, we characterized the dataset analysing the distribution regarding flights, aircraft and maintenance tasks.

Chapter 4

A Quantum Annealing Approach

QA is a technique commonly used to solve a variety of hard optimisation problems, finding low-energy states through quantum mechanics. In this chapter, we detail the whole pipeline needed for solving the presented version of the Tail Assignment Problem on a quantum annealer. Section 4.1 gives an overview of the various phases that should be considered. Section 4.2 presents two different ways of modelling the problem into the desired format. Section 4.3 gets an overview of the necessary transformations for a problem to be solved on a real quantum annealer. Section 4.4 shows multiple ways to solve the problem. Finally, section 4.5 presents a summary of the chapter.

4.1 Pipeline Overview

To solve a problem in a quantum annealer, it is necessary to rethink and adapt it to better fit the required strict specifications of these devices. When mapping a problem to be solved using a QA technique, three main phases must be considered: *Modelling*, *Embedding* and *Solving by Sampling*.

Modelling is the first phase and one of the hardest parts of the entire pipeline. We must start by setting the whole problem into a Binary Quadratic Model (BQM), ensuring that the desired possible solutions correspond to the lowest energy levels.

Having a proper BQM, it is then necessary to transform it into a format that will properly fit within the requirements of the quantum annealer. Such a process is called minor embedding or simply *Embedding*.

Finally, it is necessary to solve the problem obtaining the desired solution. Since the existent quantum annealers are not fully adiabatic, i.e., do not guarantee that the lowest energy level is always found, solving a problem implies running it multiple times, obtaining different solutions/ samples. *Solving by Sampling* allows a good finding of a proper solution, since it may discover different energy levels in different samples.

As *Modelling* is one of the hardest and most distinct parts among the different studies in this field, we gave deeper attention to it, developing two different ways of modelling the Tail Assignment Problem. On the other hand, and given the scope and duration of this dissertation, *Embedding* and *Solving by Sampling* were analysed in a softer way.

Figure 4.1 presents an overview of the three phases considered for employing QA to the Tail Assignment Problem. The first phase, *Modelling*, comprehends three steps for transforming the problem into a BQM. The third step comprises two different techniques, where the first is detailed in section 4.2.1 and the second is detailed in section 4.2.2.

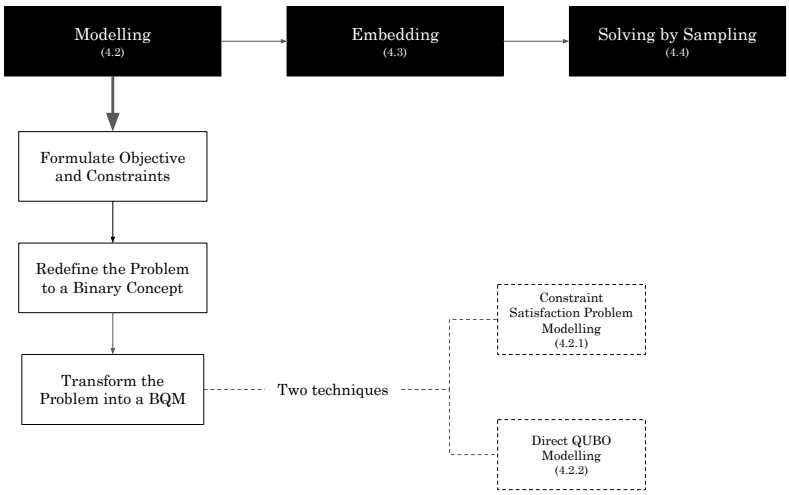


Figure 4.1: Overview of the three phases that compose the QA approach

4.2 Modelling

Answering to research question RQ1., for a problem to run in a quantum annealer, a BQM must be formulated. A proper BQM entails the sequential execution of three main steps: *Formulate the Objective and Constraints*, *Redefine the Problem to a Binary Concept* and finally *Transform the Problem into a BQM*.

Formulate the Objective and Constraints

Starting by *Formulate the Objective and Constraints*, we specify and systematize the optimisation criteria and groups of constraints. Taking into account the optimisation criteria previously presented, we state the following definition:

Objective: To minimise the considered operational costs of the schedule, i.e., to get a solution such that each flight is performed by the tail with the lowest execution cost.

To ensure that all the scenarios are analysed, we dive deeper on the specification of each group of **constraints**:

- **Assignment Constraints**: each flight must only be performed by one tail;

- **Connection and Maintenance Constraints**: a tail must have a valid schedule regarding arrival and departure locations and times of activities. This group of constraints can be divided into three subgroups as follows:
  - **Impossible Pairing Activities**: activities that have no valid path between them and therefore cannot be performed by the same tail, i.e., a given tail cannot perform both activities while ensuring a proper solution;
  - **Impossible Flights due to Maintenance**: as maintenance is obligatory, tails cannot perform flights that have no valid path for all their maintenance tasks;
  - **Activity Path Consistency**: all consecutive activities performed by a tail must follow a valid path;
- **Flights Requirement Constraints**: since each flight has a pre-defined minimum fleet and a minimum number of seats required, a tail that meets such requirements is needed to perform the flight.

Redefine the Problem to a Binary Concept

After formulating and detailing the problem to be solved, to *Redefine the Problem to a Binary Concept* it is necessary to convert it from an optimisation problem into a decision problem.

Through the analysis of the available literature on this particular matter, it is possible to identify solutions in which binary scenarios were defined based on whether or not some route had to be assigned to a certain tail [58]. However, this approach required a lot of pre-processing to consider all the possible routes and had scalability issues associated. Therefore and as an effort to minimise the pre-processing burden of the previously presented approach, in this study our option was to consider individual flights and tails instead of routes.

Redefining the problem into binary scenarios, we state it as follows: "Should flight X be performed by tail Y?". Each variable of the problem is then represented as the possible assignment of a certain flight to a specific tail.

Considering a set of F flights labeled as $f = 1, ..., F$ and a set of T tails labeled as $t = 1, ..., T$, we define the binary variables $q_{f,t}$ as the variables of the problem's domain.

In addition to the definition of the problem's possibilities in terms of binary variables, it is still necessary to transform the objective and constraints to use these binary variables.

Transform the Problem into a BQM

To *Transform the Problem into a BQM* it is necessary to start by defining which type of model to use. In this study, our option is to do an implementation based on a QUBO model and, therefore, the previously defined binary variables are redefined as $q_{f,t} \in \{0, 1\}$, where $q_{f,t} = 1$ represents the assignment of flight f to tail t.

Regarding the implementation of constraints, we start by setting a connection network graph. As we only consider pre-assigned maintenance tasks, it is necessary to define not one, but multiple

connection network graphs, one for each tail. Each one of these graphs includes all activities (flights and maintenance tasks) that can or have to be performed by a specific tail. In these graphs, a maintenance task is considered as equivalent to a flight, since both of them have departure and arrival airports, as well as a certain duration in which they occur. Nonetheless, a maintenance activity has some particularities. The departure and arrival airports are the same and correspond to the location where the maintenance task takes place, whereas the departure and arrival times correspond to the initial and end time of the maintenance task.

So, while implementing a QUBO model we only deal with variables that represent the assignment of flights to tails that may be able to perform them. As each graph only includes flights compatible with the obligatory maintenance tasks of the tail, **Impossible Flights due to Maintenance** are guaranteed and, therefore, do not need to be implemented. Furthermore, for **Flight Requirement Constraints**, as each tail's graph only includes flights that can be performed by such tail, this group of constraints is also assured and do not require further specification.

Figure 4.2 represents an example of three tails and eight flights. For a clear distinction between the representation of flights and tails, in this example we consider tails as letters and flights as numbers. Flights not represented on the graphs are not part of the problem. For example, as tail A cannot perform flights 7 and 8, the corresponding variables are not part of the modelling: $q_{7,A}$; $q_{8,A}$. Throughout this chapter, we use this example as the illustrative example.

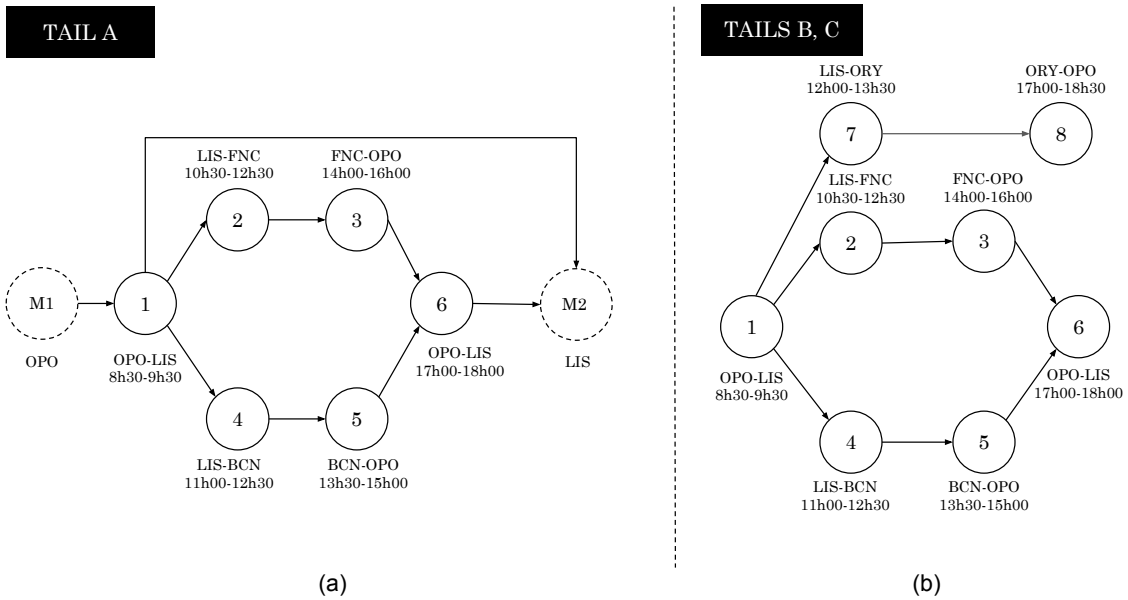


Figure 4.2: Illustrative example of connection network graphs for three tails and eight flights. (a) tail A can perform flights 1 to 6 and must perform two obligatory maintenance tasks, M1 in OPO and M2 in LIS; (b) tails B and C can perform flights 1 to 8 and have no maintenance associated. Flights 8 and 6 are not connected as they overlap in time

Regarding maintenance and since some scalability issues have appeared in the literature for problems with a large number of variables [54], in this implementation we do not set maintenance

tasks as part of the QUBO model. However, as they are obligatory, variables representing the assignment of flights to tails are restricted to ensure that the final solution has, for each tail, a schedule that guarantees the viability of its maintenance tasks.

To map the objective and constraints into a valid QUBO model, two techniques were considered, depending on whether they made use of an existent library for Constraint Satisfaction Problems, as described in section 4.2.1, or directly set the problem in the form of a QUBO model, as described in section 4.2.2. A detailed description of each technique is presented in the following subsections.

4.2.1 Constraint Satisfaction Problem Modelling

As previously presented, the Tail Assignment Problem has multiple constraints that must be met aiming to get a valid solution. Therefore, the implementation of the problem as a CSP was the first approach to be considered in this study.

Due to the complexity of finding a good mapping between a CSP and a QUBO model, some libraries provide a manner of doing such transformation programmatically. Dwave's *dwavebinarycsp*[1] library allows one to create a valid QUBO model, mapping individual constraints. Constraints are defined as a set of variables and valid configurations in the form of tuples. Each tuple is given by a group of binary values (one for each variable of the constraint) representing the cases that are valid for such constraint.

Let's consider for instance a pair of variables γ and β. If these two variables must be mutually exclusive (i.e., they cannot be assigned as 1 at the same time), then the valid configurations would be $\{ (1,0), (0,1) \}$.

To transform a given constraint into a QUBO model, D-Wave's library makes use of a penalty model function to set biases and coupling strengths between variables, creating a BQM that represents the desired constraint. The final QUBO model is a result of multiple small models. In the end, the modelling of the CSP is reached and the feasible solutions have lower energy levels than the unfeasible ones.

Being our main goal to get a valid solution with a minimum cost rather than just a feasible solution, after converting the CSP into the desired format, we add the defined objective to the QUBO model.

In the following sections a detailed explanation on the implemented constraints and objective function is presented.

4.2.1.1 Assignment Constraints

Each flight can only be assigned to one tail which is the one responsible for performing it. Therefore, variables representing the assignment of a same flight for different tails must be mutually exclusive. Considering the illustrative example presented in Figure 4.2, ensuring that flight 7 is assigned to either tail B or C, is implemented as follows:

[1] https://docs.ocean.dwavesys.com/projects/binarycsp/

- Constraint:
  - Variables: { $q_{7,B}$; $q_{7,C}$ }
  - Tuples: { (1,0), (0,1) }

Where (1,0) represents setting $q_{7,B} = 1$ and $q_{7,C} = 0$ and (0,1) represents setting $q_{7,B} = 0$ and $q_{7,C} = 1$

Following the same idea, other constraints were added for the rest of the flights that can be performed by more than one tail.

4.2.1.2 Connection and Maintenance Constraints

Connection constraints aim to ensure that two consecutive activities assigned to the same tail are part of a valid solution. In this way, considering two activities, the arrival airport of the first activity must be the same as the departure airport of the second activity and the first flight has to end before the scheduled departure time of the second activity, taking into account the obligatory minimum turnaround time between activities. In other words, recalling the concept of connection network, two non-connected nodes represent two activities that cannot be assigned together to the same tail and each valid path represents a sequence of activities that can be part of a same schedule of one tail. As activities can be compared to nodes in the network connection graph, saying that two activities are connected is similar to saying that the nodes that represent such activities are connected. Regarding the specific case of maintenance, if two maintenance tasks for the same tail do not take place on the same location, then a valid sequence of flights that connect both maintenance tasks must be assigned to such tail.

Analysing the connections in each tail's graph, we can extract: which pairs of flights are impossible to be assigned simultaneously; which paths are valid; and which flights must be assigned to ensure that all maintenance tasks can be performed. As this group of constraints is tail-dependent, it must be analysed and implemented tail by tail.

To guarantee a proper solution, two subgroups of constraints can be defined: one to remove the possibility of non-pairable flights and the second to obligate two indirectly connected flights or maintenance tasks to follow a valid path, when assigned to the same tail.

4.2.1.2.1 Impossible Pairing Activities

Some activities may not be possible to be assigned together to the same tail, since there is no way for it to perform both activities. This group of constraints can be modelled as pairs of variables that cannot be simultaneously part of a same solution. In this case, each pair is composed by two variables that represent the assignment of non-connected activities for the same tail. Considering the illustrative example presented in Figure 4.2, tail *A* cannot perform together flights 2 and 4, so the following constraint was added:

- Constraint:
  - Variables: { $q_{2,A}$; $q_{4,A}$ }
  - Tuples: { (0,0), (0,1), (1,0) }

Other equivalent constraints also were considered for the variables represented in the remaining pairs of non-connected nodes of each graph.

4.2.1.2.2 Activity Path Consistency

This subgroup of constraints aim to ensure the chosen schedule is valid regarding that consistency. Three different subgroup of constraints can be defined: *Path consistency between non-consecutive flights*, *Path consistency between maintenance tasks* and *Path consistency between flight and maintenance task*

Path consistency between non-consecutive flights

This subgroup of constraints aim to ensure that if two non-consecutive flights are assigned to the same tail t, then all the flights of one of the existent valid paths between these two flights, are also assigned to tail t. Doing it iteratively, for each indirectly connected pair of flights (saying flights α and θ), it is only necessary to guarantee that at least one of the flights that is directly connected to θ and has a valid path from α, is also assigned to tail t.

Considering the illustrative example presented in Figure 4.2, for tail A, ensuring a valid path between flight 1 and 6, would obligate one of the valid paths between these two flights to be assigned to that same tail, namely flights 2 – 3 or flights 4 – 5. Implementing such constraint can be done by setting three iterative sub-constraints (see equations 4.1, 4.2, 4.3). As represented in equations 4.1 and 4.2, a valid path between flights 1 and 3 would require flight 2 to be assigned to tail A and a valid path between flight 1 and 5 would require flight 4 to be performed by that same tail. Besides, as shown in equation 4.3, a valid path between flight 1 and flight 6 would require flights 3 or 5 to be assigned to tail A.

$$q_{1,A} \wedge q_{3,A} \implies q_{2,A} \tag{4.1}$$

$$q_{1,A} \wedge q_{5,A} \implies q_{4,A} \tag{4.2}$$

$$q_{1,A} \wedge q_{6,A} \implies q_{3,A} \vee q_{5,A} \tag{4.3}$$

Using *dwavebinarycsp* library, this subgroup of constraints can be modelled taking advantage of pre-defined factories such as the implementation of the boolean gates *and* or *or*. Each of these gates take three variables as argument (two input and one output) where the output variable takes the value of the desired relationship between the two inputs.

The implementation of each one of the sub-constraints defined above is divided into three steps, named as $initialfinal_{and}$, $intermediate_{or}$ and $combined_{tuple}$.

In the illustrative example, to implement the relationship presented in equation 4.3 these steps go as follows:

1. $initialfinal_{and}$: set an *and* gate between the initial ($q_{1,A}$) and the final ($q_{6,A}$) variables, generating an auxiliary variable as output ($aux1$): $and(q_{1,A}; q_{6,A}; aux_1)$

2. $intermediate_{or}$: set an *or* gate between the variables representing intermediate flights that are part of a valid path between the initial and the final flight and directly connected to the final flight ($q_{3,A}$; $q_{5,A}$), also generating an auxiliary variable ($aux2$): $or(q_{3,A}; q_{5,A}; aux_2)$

3. $combined_{tuple}$: set a tuple with a configuration obligating that if the initial and final flights are assigned to the same tail (which means $aux_1 = 1$) then, at least one of the variables defined in the second step is also assigned as 1 (which means $aux_2 = 1$):

   Variables: { aux_1, aux_2 }

   Tuples: { (0,0), (0,1), (1,1) }

Since it must be done iteratively to ensure the connectivity of all possible valid paths, it required similar sub-constraints for the other indirectly connected pairs of flights.

Path consistency between maintenance tasks

If two maintenance tasks of the same tail are not to be performed on the same location, then a sequence of flights that allow such tail to be on time in each one of the desired airports must be assigned to it.

Considering the illustrative example presented in figure 4.2, tail A must perform an initial maintenance task in airport of Porto (OPO) and later another maintenance task in airport of Lisbon (LIS). Therefore, it is necessary to ensure that flight 1 and flights 1 or 6 are assigned to tail A. The resulting boolean expression is presented in equation 4.4.

$$q_{1,A} \wedge (q_{1,A} \vee q_{6,A}) \tag{4.4}$$

As flight 1 must be assigned to tail A, the representative variable is removed from the problem and variables representing the assignment of the same flight to tails B and C are not needed for the modelling.

Path consistency between flight and maintenance task

Regarding valid paths from a specific flight to obligatory maintenance tasks, it is necessary to guarantee that any flight that is not directly connected to an existent maintenance task, is only

assigned to the tail that must perform that maintenance task if the valid path between both activities (flight and maintenance task) is part of the same assignment. For example, in the illustrative example presented in figure 4.2, for tail A to perform flight 3, it is necessary that it also performs flights 2 and 6 to be able to connect to both maintenance tasks M1 and M2, as represented in the boolean condition of equation 4.5. Implementing such constraints can be done using the same steps like the ones presented for implementing equation 4.3.

$$q_{3,A} \implies q_{2,A} \wedge q_{6,A} \tag{4.5}$$

When implementing all the previously defined groups of constraints, the CSP is then transformed into a QUBO model with proper values for biases and coupling strengths ensuring that feasible solutions correspond to the lowest energy levels. As some constraints might be redundant D-Wave's library establishes which ones are really necessary in order to minimise the number of variables in the final QUBO model. For the illustrative example, the resulting QUBO model includes 43 variables, 24 of them being auxiliary variables.

4.2.1.3 Objective Function

As defined in subsection 3.2.2, the objective function aims to minimise the execution cost of the selected schedule. In fact, the problem defined with the previous constraints only ensures that a valid solution is selected disregarding the associated costs. As a result, the presented QUBO model has to be adapted to minimise an objective function given in equation 3.1. To implement it, each variable get its bias reduced in such a way that variables representing cheaper assignments have a lower bias value.

4.2.2 Direct QUBO modelling

Although using external libraries may be easier to get a formulation of the problem, when scaling it, the generation of a proper QUBO model can become an important bottleneck. Therefore, a direct modelling may help. In this context, a QUBO model is designed representing the Tail Assignment Problem, keeping the same rules and constraints as defined in 4.2.1.

The QUBO model is set as a sum of six different terms that represent both the constraints and the objective function, presented in equation 4.6.

$$H = \gamma H_A + \eta H_{B_1} + \lambda H_{B_2} + \tau H_{B_3} + \phi H_{B_4} + \psi H_C \tag{4.6}$$

From this equation γ, η, λ, τ, ϕ and ψ represent positive real-valued numbers used to tune the relative importance of each term in the global QUBO model. H_A is related to the assignment constraints. On its turn, terms H_{B_1}, H_{B_2}, H_{B_3} and H_{B_4} assure connection and maintenance constraints. Finally, H_C is the term responsible for setting a lower energy value to solutions with minimum value of the objective function. To assign values to the different tuning parameters, it is important to take into account that some of them may incorrectly affect the others. For example, the value of ψ must be chosen sufficiently small not to violate any of the constraints of the problem. We considered the following notation:

T : set of all tails

F : set of all flights

M : set of all maintenance tasks

$T_f \subset T$: subset of T that can perform a given flight f

$F_t \subset F$: subset of F that can be performed by a given tail t

$I_f \subset Ft$: subset of Ft that cannot be assigned together with a given flight f

$IC_f \subset Ft$: subset of Ft that is indirectly connected to a given f on the corresponding tail's connection network graph

$M_t \subset M$: subset of M that must be performed by a given tail t

4.2.2.1 Assignment Constraints

Since it is desirable that all flights are assigned to exactly one tail, a quadratic penalty is introduced for schedules not meeting such condition. The implementation of this constraint follows equation 4.7.

$$H_A = \sum_{f}^{F} \left(\sum_{t}^{T_f} q_{f,t} - 1 \right)^2 \tag{4.7}$$

Such constraint is set by defining a negative bias to each one of the individual variables $q_{f,t}$ and a positive coupling strength, for each pair of variables $\{ q_{f,t} ; q_{f,t'} \}$, where the latter corresponds to the assignment of a same flight to different tails.

4.2.2.2 Connection and Maintenance Constraints

As previously described, connection and maintenance constraints aim to guarantee that the obtained solution is valid regarding sequences of flights while ensuring all pre-defined maintenance tasks can be performed by the correspondent tail.

4.2.2.2.1 Impossible Pairing Activities

This subgroup of constraints is set to penalise the cases where two flights (f, f') that have no valid path between them are assigned simultaneously to the same tail (t). As defined in equation 4.8, it penalises variables that represent non-pairable flights.

$$H_{B_1} = \sum_{t}^{T} \sum_{f}^{F_t} \sum_{f'}^{I_f} q_{f,t} q_{f',t} \tag{4.8}$$

4.2.2.2.2 Activity Path Consistency

To ensure that the chosen schedule is valid regarding path consistency, an extra penalty is added on paths that are not valid. Three different subgroup of constraints can be defined: *Path consistency between non-consecutive flights*, *Path consistency between maintenance tasks* and *Path consistency between flight and maintenance task*

Path consistency between non-consecutive flights

To ensure path consistency between two non-consecutive flights, equation 4.9 penalises all pairs of indirectly connected flights (f, f') assigned to a tail (t) that do not have a valid path assigned to that same tail. It is done by penalizing cases where $p(f,f',t)$, which corresponds to the penalty function of the boolean expression $p_b(f,f',t)$ in equation 4.10, has a value different than 1.

$$H_{B_2} = \sum_{t}^{T} \sum_{f}^{F_t} \sum_{f'}^{IC_f} (p(f,f',t) - 1)^2 \tag{4.9}$$

$$p_b(f,f',t) = ((q_{f,t} \wedge q_{f',t}) \oplus 1) \vee w \tag{4.10}$$

In equation 4.10, $w = q_{f_1,t} \vee q_{f_2,t} \vee ... \vee q_{f_n,t}$ and $[f_1,...,f_n] \in PF$ where PF represents the set of flights that can be reached from flight f and are directly connected to flight f', when both flights are assigned to tail t.

Translating the desired boolean expression defined in 4.10 to a penalty function and taking into account the translations presented in Chapter 2, we use the conversions defined in Table 4.1, where x_3 represents the output auxiliary variables that take the value of the relationship between the two inputs (x_1 and x_2). As previously presented, the first two penalty functions can be found in the literature, whereas the third penalty function was constructed for this study based on the penalty functions of the trivial boolean relationships.

Table 4.1: Boolean penalty functions needed for the QA approach

Classical Constraint	Equivalent Penalty
$x_3 \Leftrightarrow x_1 \wedge x_2$	$P(x_1x_2 - 2(x_1+x_2)x_3 + 3x_3)$
$x_3 \Leftrightarrow x_1 \vee x_2$	$P(x_1x_2 + (x_1+x_2)(1-2x_3) + x_3)$
$x_3 \Leftrightarrow (x_1 \oplus 1) \vee x_2$	$P(-x_1x_2 - x_3 + 2x_1x_3 - 2x_2x_3 - x_1 + 2x_2 + 1)$

Path consistency between maintenance tasks

To guarantee a valid path between obligatory maintenance tasks, a new penalty is added for the cases where none of the flights that guarantee such path is assigned to the considered tail, as represented in equation 4.11.

$$H_{B_3} = \sum_{t}^{T} \sum_{m,m'}^{M_t} (g(m,m',t) - 1)^2 \tag{4.11}$$

In this equation $g(m,m',t)$ corresponds to the penalty function of the boolean expression $g_b(m,m',t)$ presented in equation 4.12

$$g_b(m,m',t) = (q_{f_1,t} \vee q_{f_2,t} \vee ... \vee q_{f_n,t}) \wedge (q_{f'_1,t} \vee q_{f'_2,t} \vee ... \vee ...q_{f'_n,t}) \tag{4.12}$$

where $[f_1,\ldots,f_n] \in MPF$ and $[f'_1,...,f'_n] \in MNF$. On one hand, MPF represents the set of flights that are part of a valid path between the two maintenance tasks (m and m') and are directly connected to m on the tail's connection network graph. On the other hand, MNF represents the set of flights that are part of a valid path between two maintenance tasks (m and m') and are directly connected to m' on the corresponding tail's connection network graph. Setting boolean expression defined in equation 4.12 can be done using the penalty functions from Table 4.1.

Path consistency between flight and maintenance task

Finally, regarding valid paths from a specific flight to obligatory maintenance task, it is necessary to guarantee that any flight that is not directly connected to an existent maintenance task, is only assigned to the tail that must execute such maintenance task if the valid path between both activities is part of the same assignment. Equation 4.13 sets a penalty for solutions that do not follow such constraint,

$$H_{B_4} = \sum_{t}^{T} \sum_{f}^{F_t} \sum_{m}^{M_t} (r(f,m,t) - 1)^2 \tag{4.13}$$

where $r(f,m,t)$ corresponds to the penalty function of the boolean expression $r_b(f,m,t)$ presented in equation 4.14,

$$r_b(f,m,t) = (q_{f,t} \oplus 1) \vee (q_{f'_1,t} \vee q_{f'_2,t} \vee \ldots \vee q_{f'_n,t}) \tag{4.14}$$

where $[f'_1,\ldots,f'_n] \in MCF$, with MCF representing the set of flights that are directly connected to maintenance task m and have a valid path to flight f on the corresponding tail's connection network graph.

Analysing all constraints as a whole, some redundancies may appear. Therefore, to minimise the number of variables of the problem, while implementing each constraint, it is necessary to verify whether or not that constraint is already assured.

Figure 4.3 shows a QUBO model of the illustrative problem for some of the constraints, in the form of an upper-triangular matrix. Furthermore, every coefficient is multiplied by the tuning parameter associated to the constraint it represents.

	2,A	3,A	4,A	5,A	6,A	2,B	3,B	4,B	5,B	6,B	7,B	8,B	2,C	3,C	4,C	5,C	6,C	7,C	8,C
2,A	-1		1	1		2							2						
3,A		-1	1	1			2							2					
4,A			-1					2							2				
5,A				-1					2							2			
6,A					-1					2							2		
2,B						-1		1	1		1	1	2						
3,B							-1	1	1		1	1		2					
4,B								-1			1	1			2				
5,B									-1		1	1				2			
6,B										-1							2		
7,B											-1							2	
8,B												-1							2
2,C													-1		1	1		1	1
3,C														-1	1	1		1	1
4,C															-1			1	1
5,C																-1		1	1
6,C																	-1		
7,C																		-1	
8,C																			-1

Figure 4.3: Partial QUBO model matrix for the illustrative example: 2,A corresponds to the variable represented by the assignment of flight 2 to tail A. The colored entries correspond to the coefficients from Equations (4.7) and (4.8) and empty entries correspond to coefficient 0.

4.2.2.3 Objective Function

Each variable $q_{f,t}$ gets its bias increased depending on how smaller is the operational cost of the assignment it represents when compared with the maximum cost possible for performing flight f. Equation 4.15 represents the associated penalty function,

$$H_C = \sum_{f}^{F} \sum_{t}^{T_f} \frac{execcost_{f,t}}{maxcost_f} q_{f,t} \tag{4.15}$$

where $execcost_{f,t}$ represents the cost of performing flight f on tail t, and $maxcost_f$ represents the maximum cost of performing flight f in any of the possible tails.

Therefore for each flight, H_C will take the value of 1 for the most expensive assignment.

4.3 Embedding the Problem

A QUBO model corresponds to the representation of a problem using logical variables. Thus, to solve the problem using a quantum annealer, it is necessary to transform it to meet the hardware requirements of the device. Furthermore, it is relevant to note that a QUBO model can be seen as a graph where the variables correspond to the graph nodes and the possible links between them represent the constraints between variables. Through a process called minor-embedding, logical variables are transformed into Ising variables and the associated coefficients are mapped considering the limits defined by the quantum processor [18].

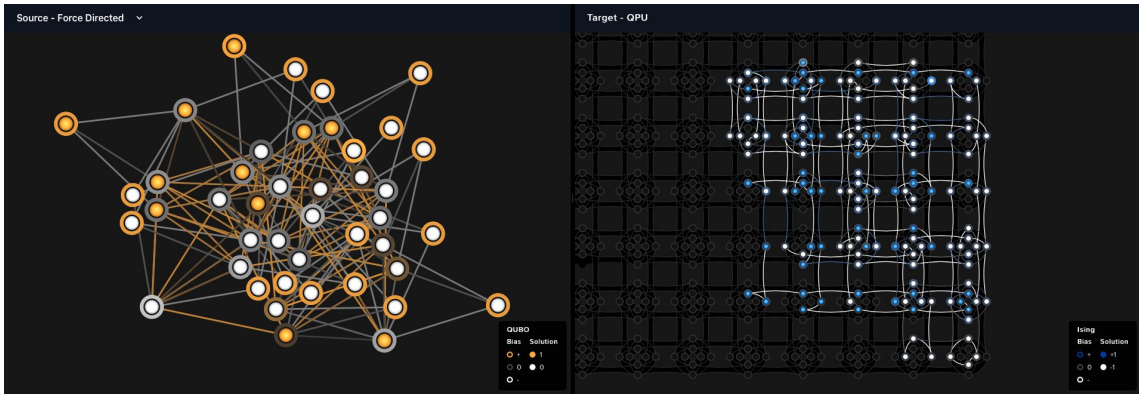
Considering the scope of this work, we used the lower-noise quantum annealer D-Wave 2000Q, specially designed for problem-solving based on QUBO models [38]. This device is composed by 2048 qubits connected in groups of 8, in the form of a chimera graph. Given the low number of connections present in a chimera graph, two or more qubits can be forced to represent the same variable to ensure the integrity of all the desired interactions. Thus, the number of variables that would initially be necessary to represent a problem could correspond to a greater number of qubits required and, therefore, raise scalability issues.

In order to embed our modelling of the Tail Assignment Problem, we resorted to a randomized embedding algorithm first developed by Cai et al. [14] and implemented as part of D-Wave's toolchain. Starting from the graph that represents the QUBO model, this algorithm defines heuristics, trying to find the best way to adapt it to the desired shape. Because it is a method with some randomness to obtain reliable results, it becomes important to run the algorithm several times, ensuring that different embeddings are found.

Considering the illustrative example presented in Figure 4.2, embedding it using this algorithm would require 156 qubits as presented in Figure 4.4. This figure was obtained using D-Wave's problem inspector [2].

A recently published study obtained better results regarding the number of qubits and time needed to find a proper embedding of a problem using the same device. However, no implementation of the algorithm developed is yet available [19].

[2] https://docs.ocean.dwavesys.com/projects/inspector/en/latest/



(a) (b)

Figure 4.4: Embedding of the illustrative example. (a) representation of the QUBO's model corresponding graph; (b) representation of the nodes required for embedding the problem's illustrative example.

4.4 Solving by Sampling

In this dissertation, we solved the problem by sampling low energy states from the defined QUBO model. As it is not possible to guarantee that the best solution is found by solving the problem at the first try, multiple runs must be performed to obtain multiple samples. By the end of this process, it is expected that, among the obtained samples, the state with the lowest energy is present.

Three different types of solvers can be defined: Classical solvers, Quantum Solvers and Hybrid Solvers.

4.4.1 Classical solvers

Classical solvers are based on the traditional heuristic searching algorithms, such as SA or Tabu Search or on brute-force approaches such as the Exact method [4] [6].

These solvers are important as they can be used for validating the BQM and as a comparison for other methods, such as the quantum solvers. Comparing multiple solvers may give a better understanding on the possibility of reaching quantum advantage for solving the proposed problem. Running on Central Processing Units (CPUs) or Graphics Processing Units (GPUs), these solvers do not require the problem to be embedded.

In this study we make use of an implementation of SA called *SimulatedAnnealingSampler* [3] here named as *SASampler*. This implementation is based on the idea that starting from a randomly generated solution, changes in the value of binary variables depend on how much each variable can alter the energy of the obtained solution.

[3] https://docs.ocean.dwavesys.com/projects/neal

4.4.2 Quantum solvers

Quantum solvers are solvers that run on quantum devices, solving problems that were previously defined and embedded to fit in it. As quantum computers are expensive and rare the current existent solutions work as remote devices, accessible through an application programming interface (API).

Using the API it is possible to submit a problem in the form of either a QUBO or Ising model, which must perfectly fit on the quantum annealer. When the computer receives the problem, biases and coupling strengths are associated with the qubits and couplers, respectively, and the quantum annealing process starts. In this study we do not make directly use of these solvers.

4.4.3 Hybrid solvers

Current existent quantum devices are limited in the number of qubits they provide as well as the number of connections between those qubits. An hybrid approach is one of the ways to decompose the problem into multiple sub-problems and still take advantage of the quantum computer to solve complex sub-problems.

In this dissertation, we took advantage of two different solvers proposed on D-Wave's toolchain: D-Wave Hybrid Solver Service (*HSS*) and *Kerberos*.

HSS [4] is a cloud-based solver developed by D-Wave for solving a problem modelled as a BQM with up to 10000 variables, using state-of-the-art classical algorithms together with a QA, being the latter used to solve parts of the problem. Classical algorithms run on remote classical computers whereas the quantum annealing part makes use of the D-Wave 2000Q QPU. As *HSS* is a paid product, no details are available regarding either the algorithms uses or how they are combined.

On its turn, *Kerberos* [5] is a sampler that runs two classical algorithms together with a QPU sub-problem sampler (which samples from variables that have a higher-energy impact) all in parallel, returning the best solution of the three approaches. For the classical algorithms it makes use of the local machine where the algorithm is being executed, whereas for the quantum annealing part it makes use of the D-Wave 2000Q QPU.

4.5 Summary

In this chapter, we presented how the proposed definition of the Tail Assignment Problem, as described in Chapter 3, can be redefined and modelled to be solved in a quantum annealer. Regarding modelling, and answering to research question RQ1., two different techniques were described: one using a built-in library for modelling CSP and another for directly constructing a QUBO model. Moreover, we presented the embedding process that was considered for mapping the QUBO model into the quantum annealer chimera structure, and the three categories of solvers that we took into account to solve the problem.

[4] https://docs.dwavesys.com/docs/latest/doc_leap_hybrid.html
[5] https://docs.ocean.dwavesys.com/projects/hybrid/en/latest/intro/using.html

Chapter 5

Tests and Results

In this chapter we perform some tests for analysing the implemented modelling techniques and the performance of the considered solvers using different metrics. Section 5.1 presents the tuning parameters used in the Direct QUBO modelling when running the desired sets of tests. Section 5.2 states the algorithms used for choosing the data used in the different tests. The following sections present the three groups of tests performed. Firstly, in section 5.3 a set of tests was run for analysing the performance of each modelling technique. Secondly, in section 5.4 we verified how a dataset composed by flight strings instead of individual flights could affect the scalability of the problem. Finally, in section 5.5 a third set of tests was run in order to evaluate the quality of the obtained solutions. By the end of the chapter, section 5.6 presents an overview of the main results and conclusions obtained with the performed tests, while section 5.7 summarises the chapter.

Moreover, for all the tests that ran on a local machine, we made use of a computer with an Intel(R) Core(TM) i7 CPU @ 2.2 GHz and 16GB of RAM.

5.1 Tuning Parameters

The quality of the obtained QUBO model highly depends on how well it can relate to the definition of the problem by penalising invalid solutions. In Table 5.1 we present the values of the tuning parameters used during the performed tests. Since no rules were possible to set on the values of these parameters, they were found empirically. Nonetheless, some relationship was possible verified. Terms that relate fewer variables must have higher tuning values than the ones involving a larger number of variables. Furthermore, the parameter of the objective function term must be sufficiently small not to violate any constraint.

Table 5.1: Tuning parameters' values for QUBO model

γ	η	λ	τ	ϕ	ψ
5	8.5	3	4	4	0.3

5.2 Data Selection

To run the desired sets of tests, we started by using the initial dataset as described in Chapter 3. However, due to the amount of data and a large number of possible solutions, it was not viable to obtain any results in useful time. Therefore, an algorithm was establish for selecting smaller datasets while keeping the proportions of the initial dataset.

As the problem includes two different types of data (activities and tails), two algorithms were used with similarities between them, as described in the following sections.

5.2.1 Tail Selection

Starting by analysing the dataset regarding aircraft and since all of them belong to the same fleet (NB), to keep the existent proportions, the algorithm was defined to focus on the aircraft models.

As previously identified, the initial dataset has three different aircraft models with a bigger number of tails from models *319* and *320*.

To keep such proportion in the partial datasets, Algorithm 1 selects a given number of tails, where each model has a probability of being chosen according to the percentage of aircraft of that same model present in the initial dataset. After choosing the aircraft, no preference is given to any tail and, therefore, tails from the same model have an equal probability of being chosen to be part of the partial dataset.

Algorithm 1 Tails selection based on aircraft models' proportions

Input: $allTails$ (list of all tails), $numDesiredTails$ (number of tails to select)
Output: $selectedTails$ (list of tails selected)

$percentages, tailsByModel, models \leftarrow$ PERCENTAGESAIRCRAFTMODELS($allTails$)
$selectedTails = []$
while $length(selectedTails) < numDesiredTails$ **do**
    $rModel \leftarrow random(0,1)$
    $sumProbabilities = 0$
    for $model$ in $models$ **do**
        $sumProbabilities = sumProbabilities + percentages[model]$
        if $percentages[model] <= sumProbabilities$ **then**
            $rTail \leftarrow random(1, len(tailsByModel[model]))$
            $selectedTails.append(allTails[rTail])$
        end if
        Break
    end for
end while

5.2.2 Flight Selection

To choose activities that should be considered on the partial dataset, it is just necessary to select flights, since maintenance tasks are automatically selected when choosing tails through the previous algorithm. To selecting flights, as it is expected that they fit on a valid schedule for the pre-selected tails, we must take into account the number of seats available on the chosen tails as well as the pre-assigned maintenances they have to perform.

To keep proportion on the selected flights when compared to the initial dataset, they are chosen based on the number of flights of each unique city-pair. Therefore, the bigger the number of flights of a unique city-pair, the higher the probability of a flight of such unique city-pair to be part of the partial dataset. To ensure that it is possible to obtain a valid schedule for the chosen flights and tails, we thought of this process as a generation of a feasible schedule for the pre-selected tails. An initial flight was assigned to each tail so that all tails are used. After such assignment, and to ensure path consistency between activities, the remaining flights were obtained always based on the flights previously selected. Algorithm 2 was used to choose a valid set of flights to be performed by the pre-selected tails. Such algorithm works by iteratively selecting a tail and making use of the auxiliary Algorithm 3 to choose an individual flight. The latter selects the individual flights based on the unique city-pairs' proportions in a way that it can be part of a valid schedule for the tail previously selected.

Algorithm 2 Flights selection based on unique city-pairs' proportions

Input: $tails$ (list of tails)

Output: $selectedFlights$ (matrix where for each row (tail) there is a list with the flight chosen for that tail)

1: $schedule = [[]]$
2: **for** $i \leftarrow 1$ to $length(tails)$ **do**
3: $\quad schedule \leftarrow$ SELECTFLIGHT$(i, schedule)$
4: **end for**
5: **for** $i \leftarrow 1$ to $numberDesiredFlights - length(tails)$ **do**
6: $\quad rTail \leftarrow random(1, length(tails))$
7: $\quad schedule \leftarrow$ SELECTFLIGHT$(rTail, schedule)$
8: **end for**
9: $selectedFlights \leftarrow$ MERGEFLIGHTS$(schedule)$
10: **return** $selectedFlights$

Algorithm 3 Flight selection based on unique city-pairs' proportions

Input: *tail* (tail to consider for adding new flight), *schedule* (existent schedule)

Output: *schedule* (matrix where for each row (tail) there is a list with the flight chosen for that tail)

```
1:  function SELECTFLIGHT(tail, schedule[[ ]])
2:      percentages, flightsByCityPair, cityPairs ←PERCENTAGESCITYPAIRSBYTAIL(tail, schedule[tail])
3:      assigned = False
4:      while not assigned do
5:          rCityPair ← random(0,1)
6:          sumProbabilities = 0
7:          for cityPair in cityPairs do
8:              sumProbabilities = sumProbabilities + percentages[cityPair]
9:              if percentages[cityPair] <= sumProbabilities then
10:                 rFlight ← random(1, length(flightsByCityPair[cityPair]))
11:                 schedule[tail].append(flightsByCityPair[cityPair][rFlight])
12:                 if VALIDASSIGNMENTWITHMAINTENANCES(schedule) then
13:                     assigned = True
14:                     Break
15:                 else
16:                     schedule[tail].removeLast()
17:                 end if
18:             end if
19:         end for
20:     end while
21:     return schedule
22: end function
```

5.3 Modelling Performance

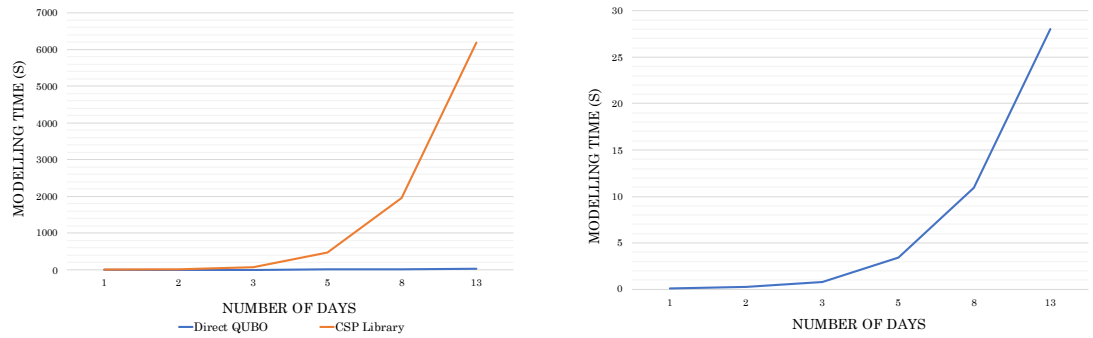
Since a major focus of this study is the modelling process, we started by setting some tests on each of the two modelling techniques developed (using a CSP library or Direct QUBO modelling). To verify the scalability of each technique, multiple datasets were extracted from the initial dataset considered. As presented in Table 5.2, we ran modelling tests for 12 different datasets, that are characterized in this table by the number of tails, number of maintenance tasks, number of days and number of flights. In fact, to analyse how each one of the modelling techniques would perform when considering a different number of both tails and activities, we defined two sets of tails (8 and 10) and for each of them, six different sets of flights. Each set of tails was obtained making use of the Algorithm 1. After selecting the set of tails, the sets of flights were calculated making use of Algorithm 3 and based on the assumption that each tail would perform an average of two flights

per day. As the modelling time could vary when executing the same test, we ran three shots for each dataset using as a result the average of the three different runs. Furthermore, all the tests were executed making use of the local machine as previously described.

Table 5.2: Modelling scalability test datasets

#Tails	#Maintenance Tasks	#Days	#Flights
8	0	1	16
	0	2	32
	0	3	48
	2	5	80
	3	8	128
	5	13	208
10	0	1	20
	0	2	40
	1	3	60
	2	5	100
	4	8	160
	7	13	260

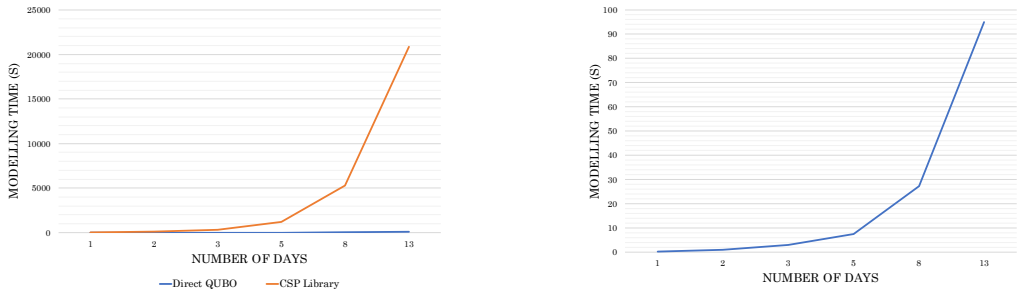
In Figure 5.1a, it is represented the time needed for both techniques to model the different datasets of eight tails. Since the modelling time for the Direct QUBO modelling is significantly smaller than the modelling time using the CSP library, Figure 5.1b represents the same modelling time but only for the Direct QUBO modelling. Although for both models an increasing number of activities (represented by the number of days) requires a significant increase in the time needed for modelling, when comparing the modelling time of both techniques for the same number of activities (number of days), CSP library takes much more time than the Direct QUBO modelling technique, with the latter needing a maximum of 28 seconds to obtain a QUBO model for the largest dataset. This difference may indicate that using the CSP library for modelling can represent a scalability bottleneck.



(a) **(b)**

Figure 5.1: Time for both modelling techniques considering the 8 tails datasets

In Figure 5.2a is plotted the modelling time needed for both techniques regarding the datasets of 10 tails. Similarly to the previous analysis, Figure 5.2b shows the modelling time only for Direct QUBO modelling as it is significantly smaller when compared to the cases using the CSP library. Analysing the growth rate, it is possible to understand that the modelling time increases significantly with the number of activities for both techniques. When comparing it for the same number of activities (same number of days), it is possible to conclude, once again, that Direct QUBO modelling is the fastest technique.



(a) **(b)**

Figure 5.2: Modelling time for both modelling techniques considering the 10 tails datasets

Summing up, regarding time modelling, Direct QUBO modelling was the technique that performed the best, for all the datasets. As we intend to solve a problem that, in a real-world scenario, can quickly grow in size and complexity, for the following tests we used Direct QUBO modelling in order to avoid having a bottleneck on this step.

5.4 Flight Aggregations' Impact

As proposed in subsection 3.4.3, and since the considered dataset belongs to an airline company that is well-known for operating in a hub-and-spoke network, aggregating flights may have a big impact on the number of variables needed for modelling the problem.

As maintenances are obligatory, and therefore, must be part of the solution, to verify the impact of aggregating flights, we decided to focus on flight strings, i.e., aggregations that start and end in the maintenance station (LIS). As identified in subsection 3.4.3, aggregating flights that were likely to be performed by the same tail, departing and arriving to a maintenance station, resulted in a total of 270 flight strings, each one of them composed by two flights (one departing from LIS and one arriving to LIS).

To analyse the impact that this aggregation may have in the scalability of the problem, we considered two datasets of flight strings.

Starting by choosing the tails for each one of the datasets, we made use of Algorithm 1, selecting a group of five tails and another group of 10 tails. For the first group, just one maintenance task was pre-assigned, whereas, for the second group, nine maintenance tasks were considered. Regarding flights, we randomly selected, from the previously identified flight strings, a set of 144 individual flights (72 flight strings) distributed over 15 days to be performed by the first group of tails and a set of 532 individual flights (266 flight strings) distributed over 30 days to be performed by the second group. The dataset including five tails was named Dataset A, while the one including 10 tails was named Dataset B.

In order to test the possible advantage of using flight strings over considering individual flights, we ran tests for both datasets (A and B), comparing the modelling time and needed number of variables when obtaining a QUBO model.

Table 5.3 represents the modelling time for each one of the datasets regarding both flight strings and individual flights. It is notorious that the time needed for modelling the problem is bigger for cases where individual flights were considered. The obtained results shown that using a flight string approach largely surpasses the alternative approach.

Table 5.3: Comparison the of modelling time, in seconds, of two datasets regarding flight strings and individual flights

	Modelling time (s)	
Dataset	**Flight strings**	**Individual flights**
A	0.182154542	12.97299304
B	8.417376267	662.8983683

Focusing now on the number of variables used to represent the problem as a QUBO model, as presented in Table 5.4, individual flights require a bigger number of variables when compared with the number of variables needed by the flight strings' cases. As the current quantum annealers

are limited in the number of qubits, minimising the number of variables to solve a problem is an important achievement for making it scalable.

Table 5.4: Comparison of the number of variables of the QUBO model of two dataset regarding flight strings and individual flights

	Number of variables	
Dataset	**Flight strings**	**Individual flights**
A	346	49818
B	49818	1086416

It is then possible to conclude that focusing on flight strings may be the best way to solve the proposed problem. In fact, Montoito [43] also concluded that aggregating flights before solving the problem would highly reduce the complexity of the problem.

5.5 Solvers' Performance

After analysing the best modelling technique and the best way of considering the flights (aggregated or not), in this final set of tests we aim to understand whether there is a real advantage of using a quantum annealer on solving such complex problem. As we were not able to test the initial dataset as it is, in order to have a bigger test scenario, we created some extra flight strings. This new aggregation was based on the idea that the tail that flies from a certain hub to a spoke is the one that performs the flight back (from the spoke to the initial hub). Thus, a new dataset was considered composed by 442 flight strings corresponding to 844 individual flights and 10 type A maintenance tasks, to be performed by 15 tails over 30 days. This dataset was named Dataset C.

Considering the solvers presented in section 4.4, for this analysis we used three different solvers, being one of them a classical solver and the other two hybrid solvers. The chosen classical solver was the *SimulatedAnnealingSampler* whereas the hybrid solvers were *HSS* and *KerberosSampler*. As previously referred, all solvers must be executed multiple times for each dataset as they work by sampling solutions of low energy levels. Therefore, we ran 10 shots per dataset for each one of the different solvers. Furthermore, for two of the solvers we were able to define some running properties, setting *SimulatedAnnealingSampler* to run 500 annealing times per shot, and *KerberosSampler* to a maximum of 500 iterations per shot. Having no previous knowledge on which values should be set on these properties and since the access to the quantum computer was limited, these values were decided based on the previous study developed by Ikeda et al. [37]. Appendix A, details the results obtained for the 10 shots ran for each one of the datasets regarding the different solvers.

To verify the performance of each solver, we analysed the results considering two characteristics: solving time and quality.

5.5.1 Solving Time

As important as finding a good solution for the problem, it is to understand the amount of time needed to reach it. Therefore, we started by analysing how much time each solver needs to find a solution for the different datasets. This analysis is important to understand which solvers can obtain a timely solution to the problem.

In Table 5.5, it is represented the time, in seconds, each solver takes for solving the QUBO model associated with each one of the datasets. When comparing the three solvers, for each one of the datasets, we can conclude that *Kerberos* is the solver that takes more time for all the datasets. This finding is reasonable since this solver is a hybrid solver that runs, in parallel, multiple iterations of classical algortihms in the local machine while communicating with the remote quantum computer for also trying to solve the same problem. In fact, the communication with the remote device has some overhead. Another explanation may be that, as the quantum annealer is a shared device, when submitting the problems to be solved, they go into a queue where they need to wait for their turn. Furthermore, for all solvers, the solving time increases with the size of the problem. Comparing *SASampler* with *HSS*, the latter outperforms for all the datasets, being the difference more significant as the size of the problem increases. In fact, for the largest dataset, *HSS* required only 20% of the time required by *SASampler* to retrieve a solution.

Table 5.5: Comparison of solving time for the three solvers and three datasets

	Solving time (s)		
Dataset	***SASampler***	***Kerberos***	***HSS***
A	3.5475	38.8842	3.2872
B	23.4909	98.7532	4.0734
C	101.0860	643.8522	20.5036

5.5.2 Solutions' Quality

Once we aim to obtain a valid solution with a minimum operational cost, we set the quality of a solution based on this goal. In fact, to verify the quality of the solutions we ran multiple tests analysing the solutions obtained by the different solvers regarding feasibility and cost. Furthermore, we also analysed the accuracy of obtaining a solution of minimum total cost by minimising the objective function as defined in subsection 3.2.2.

Finding a feasible solution

Regarding the quality of the obtained solution, we started by analysing the probability of each solver to find a feasible solution for the different scenarios. Taking into account the definition of the problem, finding a solution means finding a schedule where all flights are assigned to any tail while ensuring that all constraints are met. As presented in Table 5.6, for dataset A, all solvers sistematically found a valid solution for the problem. For dataset B, *HSS* was the one performing

the worst, only being able to find a valid solution for 80% of the shots, while the other two solvers found a feasible solution for all the shots. Regarding dataset C, *SASampler* and *Kerberos* found a solution less than 50% of the times while *HSS* was able to find it for all the 10 shots. Analysing the results, it indicates that *SASampler* and *Kerberos* may have some issues solving bigger problems. Thus, *HSS* presents better results, being able to find feasible solutions for the majority of the tests, but having some issues regarding dataset B.

Table 5.6: Probability of finding valid solutions for the three solvers and three datasets

	Prob. valid solutions		
Dataset	***SASampler***	***Kerberos***	***HSS***
A	1	1	1
B	1	1	0.8
C	0.2	0.3	1

Finding an optimal solution

More than being able to find a solution, in the Tail Assignment Problem we aim to find the best or at least a good solution. Starting by obtaining the best solution, none of the problems represented in the three datasets was possible to be solved using an Exact Solver, i.e., a solver that finds the optimal solution using a brute-force approach. Then, we considered as the best/optimal solution, the one with minimum energy found by any of the three solvers for the 10 shots.

In Table 5.7 is presented the probability of the different solvers to find a solution with the minimum value of the objective function for each one of the datasets. Here we name the cost associated with the objective function as OF cost. Although none of the solvers was always able to find a solution of minimum OF cost, some of them had a better performance than the others. In fact, as the size of the dataset grows, the probability of the different solvers to find this solution becomes smaller. *Kerberos* is the solver with the worst results, as it was only able to find a solution of minimum OF cost for the smallest dataset. *SASampler* was able to find such solution 20% of the times for the two smallest datasets. However, for dataset C, it only found the solution in one of the 10 shots. Finally, *HSS* had the best performance, being able to find a solution of minimum OF cost in at least two shots for each dataset.

Table 5.7: Probability of finding a minimum OF cost solution for the three solvers and three datasets

	Prob. minimum OF cost solutions		
Dataset	***SASampler***	***Kerberos***	***HSS***
A	0.2	0.2	0.3
B	0.2	0	0.2
C	0.1	0	0.2

Since the objective function as defined in subsection 3.2.2 minimises the cost of each individual flight, no considerations were taken regarding maintenance tasks' costs and *Standby Parking Costs*. Therefore, two minimum OF cost solutions can, in fact, have different total costs. Thus, we now analyse the probability of each solver to find the solution with minimum total cost. As presented in Table 5.8, *Kerberos* was not able to find any of the solutions that have a minimum total cost and *SASampler* was only able to do it for the smallest dataset. Finally, *HSS* was able to find that solution for all the three datasets, but only in 10% of the shots. Once more, *HSS* was the one with the best results.

Table 5.8: Probability of finding the solution of minimum total cost for the three solvers and three datasets

	Prob. minimum total cost solutions		
Dataset	***SASampler***	***Kerberos***	***HSS***
A	0.1	0	0.1
B	0	0	0.1
C	0	0	0.1

Verifying the accuracy of the objective function

As seen in some cases, the minimisation of the OF cost do not correspond to a minimisation of the solution's total cost. Therefore, we analysed if any relationship existed between the OF cost and the total cost of the same solution. Figure 5.3 presents the comparison between the OF cost and the total cost for the valid solutions obtained from all the three solvers. Figures 5.3a, 5.3b and 5.3c correspond to the solutions obtained for dataset A, B and C, respectively. As previously presented in Table 5.6, not all solvers were able to find valid solutions for all the 10 shots and therefore some datasets have a bigger number of solutions to be considered than the others.

A global analysis shows that, for the different datasets, an increment on the OF cost also corresponds to an increment on the total cost roughly on the same dimension, which may indicate that in general, a minimisation of the OF cost corresponds to a minimisation of the total cost. However, as previously identified, some exceptions can be found. For example, a deeper analysis of Figure 5.3c showed that the three solutions with lowest OF cost have the same value of OF cost but one of them has a total cost smaller than the other two. The existence of such difference becomes more evident when analysing Figure 5.3b, where solution 13 has the same OF cost of solution 12 but a smaller total cost. As only a few of these situations were found for all the three datasets, it is possible to conclude that, in general, the minimisation of the OF cost can be a good representation of the minimisation of the total cost, even though not with total accuracy.

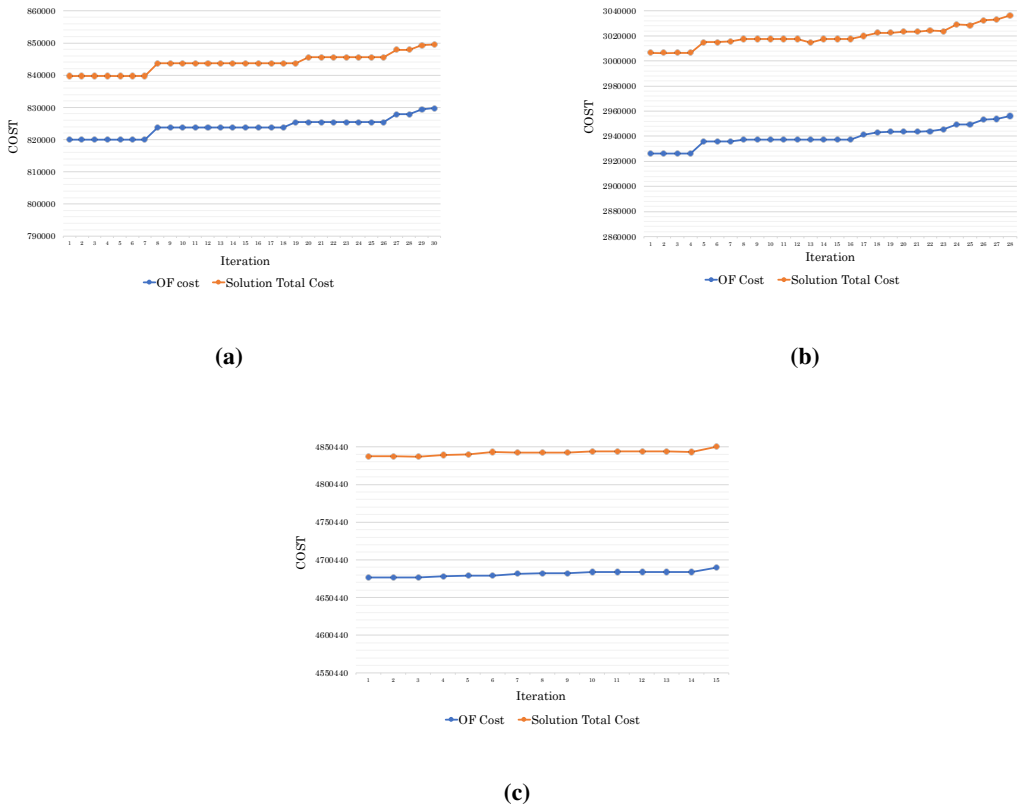


Figure 5.3: Comparison of OF costs and total costs of the found solutions (a) Comparison of the costs for solutions regarding dataset A, (b) Comparison of the costs for solutions regarding dataset B, (c) Comparison of the costs for solutions regarding dataset C

Finding non-optimal solutions

Finally, as the probability of finding the best solution was quite small, we evaluated how good each one of the non-optimal solutions was. Starting by analysing dataset A, in Figure 5.4 (a) it is represented the box plot of the OF costs of the obtained solutions for each one of the three solvers. As expected, since all solvers were able to find the solution of minimum energy, the minimum OF cost of 819971.5437 units of cost (UC) is the same for all solvers. However, some distinctions can be found between them when considering the other solutions. For *SASampler*, the majority of the solutions found had their OF costs close to the most expensive solution found by this solver. In fact, 75% of the obtained solutions had an OF cost of more than 823733.4813 UC with the most expensive solution having an OF cost of 825470.6937 UC. For *Kerberos*, 75% of the obtained solutions have a OF cost of more than 823756 UC. Finally, for *HSS*, the maximum OF cost obtained was equal to the maximum cost obtained for *SASampler* as well as the cost of the median solution. However, the 50% cheapest solutions (i.e., with lower OF cost) were more dispersed when compared to the other solvers and, therefore, the probability of finding a cheaper non-optimal solution was bigger.

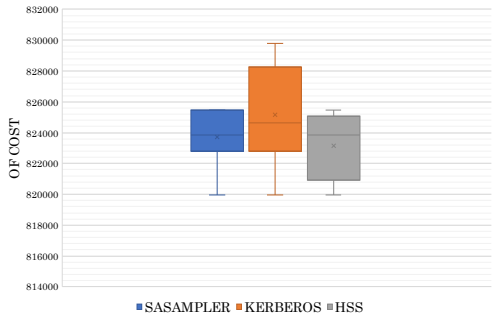


Figure 5.4: Box plot of the OF costs regarding the obtained solutions from the three solvers for dataset A

Regarding dataset B, Figure 5.5 represents the box plot of the OF costs of the solutions obtained for the three different solvers. For *SASampler* and *Kerberos* some outliers were identified. In fact, the majority of the valid solutions obtained had similar OF costs and therefore, the solution with minimum OF cost obtained for *SASampler* and the solution with maximum OF cost obtained for *Kerberos* were considered outliers. For *HSS*, the solutions found showed a higher variability with 25% of the solutions having an OF cost of less than 29.334.216 UC. In fact, when outliers were not considered, the solution with the smallest OF cost found by any of the other solvers had the same cost. However, it is also relevant to note that, for *HSS*, the most expensive solutions found have an associated OF cost considerably higher when compared with the most expensive solution obtained by any of the other two solvers. Summing up, *HSS* has a bigger variability on finding solutions and also a bigger probability of finding cheaper solutions when compared with the other solvers used. Nonetheless, this solver also has some drawbacks as it found a big number of expensive solutions.

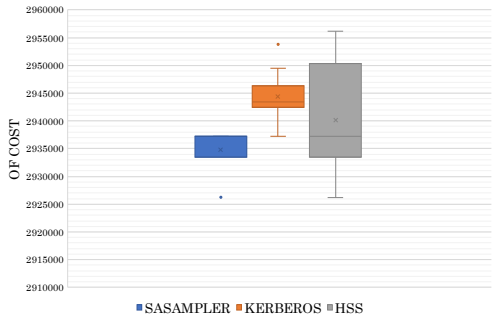


Figure 5.5: Box plot of the OF costs regarding the obtained solutions from the three solvers for dataset B

Finally, regarding dataset C, we tried to verify if the same previous conclusions were also true for bigger datasets. In Figure 5.6 is represented the box plot of the OF costs of the solutions

obtained by the three solvers. For *SASampler* and *Kerberos*, the obtained solutions are highly concentrated in terms of OF costs indicating that their OF costs are similar. This may have happened because both of these solvers were only able to find a few valid solutions. Furthermore, for *SASampler*, the solutions found had lower OF costs when compared with the solutions found by *Kerberos*. Regarding *HSS*, it was able to find valid solutions for all the 10 shots. Following the same trend verified for the other datasets, *HSS* found solutions with a big range of OF costs. Although the most expensive solution found for dataset C was obtained by this solver, 50% of the solutions found had a lower OF cost when compared with any of the solutions obtained by *Kerberos*. Comparing *HSS* with *SASampler*, both of them were able to find the minimum cost solution. However, as previously analysed, *SASampler* was able to find a valid solution only 20% of the times. Thus, it may indicate that using *SASampler* for big datasets can lead to multiple unfeasible solutions.

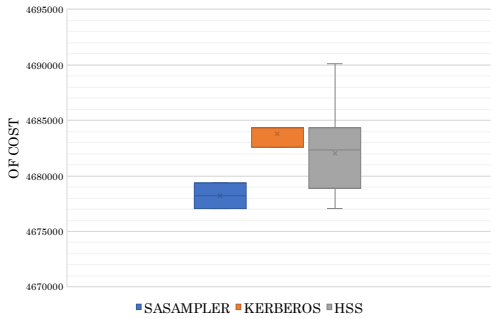


Figure 5.6: Box plot of the OF costs regarding the obtained solutions from the three solvers for dataset C

To sum up, when comparing the quality of the solutions obtained by the three solvers, it is possible to understand that none of the solvers revealed being perfect for finding the best solutions, despite diverging on the performance when solving the problem considering the different datasets. In fact, although all the solvers were able to find feasible solutions for all the datasets, none of them was able to find the solution with minimum total cost in the majority of the shots. *Kerberos* was the one that performed the worst, as it was not able to find any optimal solution for the biggest datasets and the non-optimal results obtained were also non-satisfactory. *SASampler* revealed better results, being able to find the minimum energy solution at least once for the different datasets. However, for bigger datasets, this solver revealed not to be so useful as the majority of the solutions found were either unfeasible or considerably more expensive than the minimum energy solution. Finally, *HSS* was the solver that performed the best overall, even just being able to find a feasible solution in 80% of the times for the second dataset. The most relevant advantages of this solver were that it was able to find solutions considerably faster than any other of the two solvers, obtaining a big number of different solutions. Such characteristics reveal that this solver may have great possibilities to obtain good non-optimal solutions.

5.6 Results Overview

Modelling is a crucial and complex step in the process of solving a problem in a quantum annealer. A bad choice in the modelling technique can represent a bottleneck on the applicability of a quantum annealing approach to a problem. Therefore, it is possible to conclude that, although it requires some mathematical knowledge, a direct formulation of the QUBO model is essential for having a proper BQM in useful time. That may be the reason why, in the last years, multiple studies have been focusing on finding the best way to model different problems into a BQM format.

Regarding flights data, we concluded that aggregating flights in strings would be a valuable saver of time and number of needed variables, which is accordingly to the conclusions of Montoito [43]. In fact, when considering only flight strings and just one maintenance station, no constraints regarding path consistency are needed to be considered as all activities start and end in the same airport. Therefore, the problem ends up being similar to solving a set partitioning problem (with some extra constraints) which have been proved to be NP-hard [41]. Such simplification represents a trade-off since, when aggregating flights, some decisions are taken beforehand which may affect in some way the quality of the obtained solution.

Finally, comparing the performance of the three solvers considered, it is possible to conclude that *Kerberos* was the one performing the worst, which may indicate it may not be the best choice for this type of problems. Using the *HSS* solver allowed to obtain the best results with a high probability of finding good non-optimal solutions. In fact, although *Kerberos* and *HSS* are both considered as hybrid solvers, as identified in section 4.4, *Kerberos* runs classical algorithms in parallel with a pure quantum approach while *HSS* uses them together for obtaining the results. Such characteristic may be the reason for these solvers to performance so differently. Comparing *SASampler* with *HSS*, the latter was able to find a bigger number and variety of solutions, which despite not being optimal are also not much more expensive than the best solution found. A reason for that may lay on the fact that, due to the complexity of the energy landscape, *SASampler* ended up finding a solution that represents a local minimum energy rather than a global minimum energy. On the other hand, *HSS*, using state-of-the-art classical algorithms together with a quantum annealer and its properties such as quantum tunnelling, was able to a find a bigger variety of solutions which may have been the reason why it was able to find the minimum energy solution obtained for any of the three datasets. Furthermore, this solver also revealed being faster than any other solver obtaining solutions. Such characteristic is a good indicator regarding the performance of the solver on scalability. Analysing the accuracy of minimising the OF cost rather than the total cost, it was possible to conclude that, although for some solutions a smaller OF cost did not reflect a smaller total cost, for the majority of the possible solutions that was true. So, using the OF cost for finding a solution may represent a good approximation.

Regarding the comparison with other studies, some have proposed solving the Tail Assignment Problem modelling as a set partitioning problem, presenting good results for both classical approach and quantum computing approach. For classical approaches no comparison can be made

since the datasets' size, the number of constraints and the minimisation objective function have different scopes. Considering the study developed by Vikstål et al. [58], which also presented satisfactory results, it is also not possible to have an accurate comparison as it is based on a total different quantum approach. In fact, this study aimed to find a feasible solution, based on the decision version of the set partitioning problem from a small set of options, rather than an optimal solution.

Regarding the proposed research question RQ2., we concluded that scalability is possible but it comes with some trade-offs. Furthermore, not being possible to know whether or not a solution is a real optimal solution, when comparing the results, it was possible to conclude that *HSS* performed the best. In fact, *HSS* was able to find a high variety of solutions with many of them not being much more expensive when compared with the best solution found.

For research question RQ3., we concluded that using a quantum annealing approach or a classical algorithm separately may not lead to good results as *Kerberos* and *SASampler* did not provide positive results. However, setting together classical algorithms with quantum annealing can have some advantages as it may be able to find lower energy solutions for complex problems.

5.7 Summary

To test the proposed approach, we performed multiple tests to understand how good each one of the modelling techniques presented in the previous chapter could be and whether or not such approaches could be scaled. Not being able to perform tests for the entire dataset, we defined algorithms for selecting both tails and flights while keeping the proportions of the initial dataset. Analysing the modelling time for both techniques, we concluded that Direct QUBO modelling performs the best. Furthermore, evaluating the impact of using flight strings rather than individual flights, we concluded that flight strings can be a valuable simplification for solving a scalable version of the problem, even though it introduces a trade-off. Finally, we evaluated the performance of three different solvers (*SASampler*, *Kerberos* and *HSS*), when solving the proposed definition of the Tail Assignment Problem, as described in Chapter 3. We concluded that *HSS* performed the best and that it can be a useful solver to find good non-optimal solutions.

Chapter 6

Conclusions and Future Work

This chapter presents an overview of all the work done, gathering what was learnt with it. Section 6.1 compiles the main difficulties felt while developing this study. Following, section 6.2 collects what are the main contributions of this study. By the end of the chapter, section 6.3 summarises the conclusions of this dissertation and section 6.4 presents the possible future work that could increase its scientific contributions and applicability on the real-world.

6.1 Main Difficulties

During the development of this study, we had to deal with multiple difficulties. The Tail Assignment Problem is a complex problem that requires background on the domain and context it happens. On the other hand, quantum computing and more specific quantum annealing, is a quite novel approach when it comes to its usage on real-world problems. As it relies upon quantum mechanics properties, which are quite far from the classical physics, it requires some study on the field to understand the concepts that are crucial for successfully model a problem to be solved on a quantum annealer. Furthermore, as existent quantum annealers are only accessible remotely and the provider sets limits on the usage of it, test cases were limited.

6.2 Main Contributions

The implementation of a solution for the problems identified in Chapter 3 brought some contributions to the current state-of-the-art of the application of quantum annealing to real-world problems. The following contributions were identified:

> **Modelling of a complex problem**: as identified in Chapter 2 only a few real-world problems were modelled and implemented for being solved on quantum annealers. Therefore, the modelling techniques as proposed in Chapter 4 bring some new considerations on how to model complex constraints. Furthermore, as both modelling techniques were highly detailed

it also contributes as an implementation tutorial of some complex restrictions on a BQM format;

Comparison between solvers: although multiple studies have already compared SA algorithms with quantum algorithms, from the best of our knowledge, it is the first study comparing the performance of *HSS* with other solvers regarding solution quality;

Scalability analysis: this study also provides some knowledge of scalability regarding the time needed for both modelling and solving as well as the number of variables that compose the model when increasing the size of the problem;

Different approach to solve a complex problem: although a study already applied a quantum approach to find a feasible solution to Tail Assignment Problem, from the best of our knowledge, this is the first implementation of this problem using a QA approach.

6.3 Conclusions

Currently airline companies are facing real threats on their sustainability. New and improved operational changes are critical for minimising costs and increasing profit margins. A correct planning and scheduling is fundamental for a proper usage of the resources.

In this dissertation, we have studied one of the steps of the Airline Scheduling Process, called Tail Assignment. The Tail Assignment Problem aims to find a proper assignment of individual aircraft to flights minimising a certain objective function, such as operational costs. As presented in Chapter 3 the study of such problem is quite recent with some research focusing on finding feasible solutions rather than good solutions. Furthermore, multiple studies have been implementing classical algorithms for solving the problem although it may hinder the possibility of finding real good solutions.

Given the promises of quantum computing to solve complex problems faster than classical computers, we made use of quantum annealing, a quantum technique focused on solving optimisation and sampling problems based on the energy minimisation of a certain quantum system. To narrow the scope of the proposed problem we defined three research questions to guide this study. Analysing each one of these questions we were able to obtain some answers.

RQ1. **Can the Tail Assignment Problem be modelled to run on a quantum annealer?**

To answer this question, we understood that it could be done modelling the Tail Assignment Problem. In fact, we used two different techniques, as presented in Chapter 4. Such modelling set the problem as a minimisation problem, aiming to find the solution with minimum operational cost while respecting the demand and requirements of each flight.

RQ2. **Is this approach scalable?**

This question was addressed in Chapter 5. We started by running some tests to verify the scalability of the chosen approach. We concluded that despite being time-consuming and requiring some non-trivial mathematical knowledge, formulating the problem by directly constructing a

QUBO model is crucial for scalability. Moreover, some pre-calculus such as flight aggregation is an important step to reduce the size and complexity of the addressed problem.

RQ3. **Are the results obtained from such modelling favourable?**

Finally, also in Chapter 5, we ran tests to verify the quality of the solutions obtained by three different solvers. We concluded that solving the problem based on hybrid solvers that use both classical algorithms and a quantum annealer may have some advantages over pure classical or quantum solvers. However, it is still not completely clear if such hybrid solvers can always outperform classical solvers. The hybrid solver HSS was able to obtain solutions faster than any other solver, finding various low energy solutions. Such performance represents a relevant achievement when considering complex problems that must be solved in useful time.

6.4 Future Work

Throughout this study, we opted to implement multiple simplifications to narrow the scope of the problem in analysis. Thus, it is possible to point out some considerations to be taken for further investigation on this topic.

As mentioned in Chapter 2, some of the premises assumed do not portray all the real scenarios. A non-consideration of minor maintenance tasks is not realistic as they have to occur in a real scenario. Furthermore, a robust approach may be a pivotal achievement as flight delays are frequent and tight schedules can be significantly affected by that. Since the implemented solution required some flight aggregations to be scalable, it could be interesting to analyse how some of the constraints could be adapted to require fewer variables.

Additionally, to understand the effectiveness in a deeper level of the proposed modelling techniques, when applied to multiple solvers using different scenarios, it would be important to run more tests using different datasets.

Finally, as *HSS* revealed to perform better for solving this problem than using only a classical algorithm such as a SA algorithm, further studies on hybrid solvers could be relevant for a better understanding on the real advantage of using such technique to solve complex problems.

Appendix A

List of Results

In this chapter, we present the list of results obtained by the different solvers when solving the Direct QUBO model for each one of the datasets A, B and C using three different solvers, namely *SASampler*, *Kerberos* and *HSS*.

As identified in Chapter 5, we run 10 shots for every dataset and every solver. Although on the following tables data is ordered by energy levels, they may not have been obtained by this order.

A.1 Results for Dataset A

Dataset A was composed by 76 flight strings and 5 tails, resulting in a total of 346 variables to be considered on the QUBO model. Table A.1 shows the obtained results for the different tests that were performed for this dataset considering the three different solvers.

Table A.1: Results obtained for dataset A considering the three solvers

Solver	Iteration	Energy	Model Time(s)	Solving Time (s)	OF Cost (UC)	Total Cost (UC)	# Not Assigned Flights
SASAMPLER	1	277.20800	0.182379248	3.418465715	819971.5437	839704.0831	0
	2	277.20800	0.183967898	3.791254646	819971.5437	839804.0831	0
	3	277.22476	0.185946707	3.519480983	823733.4813	843702.714	0
	4	277.22476	0.184651006	3.465722741	823733.4813	843690.0963	0
	5	277.23178	0.202914295	3.706763998	823826.1241	843781.8299	0
	6	277.23178	0.212139207	3.696696741	823826.1241	843781.8299	0
	7	277.24358	0.18055937	3.460139989	825470.6937	843774.5078	0
	8	277.24358	0.184979075	3.516830012	825470.6937	845497.5939	0
	9	277.24358	0.183510812	3.479651053	825470.6937	845497.5939	0
	10	277.24358	0.182628264	3.420000051	825470.6937	845497.5939	0
KERBEROS	1	277.20800	0.173452469	57.584294475	819971.5437	839804.0831	0
	2	277.20800	0.169569042	38.163124937	819971.5437	839804.0831	0
	3	277.22476	0.187474081	62.002601734	823733.4813	843702.714	0
	4	277.23178	0.183153713	35.795245440	823826.1241	843781.8299	0
	5	277.23178	0.179027627	38.232482310	823826.1241	843781.8299	0
	6	277.24358	0.181130402	37.692339423	825470.6937	845497.5939	0
	7	277.26004	0.179498475	26.742519705	827849.4297	847905.5197	0
	8	277.26004	0.18636867	37.856658652	827849.4297	847905.5197	0
	9	277.27108	0.180948455	31.269128661	829454.8177	849330.0738	0

Solver	Iteration	Energy	Model Time(s)	Solving Time (s)	OF Cost (UC)	Total Cost (UC)	# Not Assigned Flights
KERBEROS	10	277.28947	0.18447192	23.503666601	829790.4717	849507.3201	0
HSS	1	277.20800	0.181881708	3.748502878	819971.5437	839804.0831	0
	2	277.20800	0.181847183	2.994759000	819971.5437	839704.0831	0
	3	277.20800	0.167866505	3.489082735	819971.5437	839804.0831	0
	4	277.22476	0.176534859	3.562318308	823733.4813	843702.714	0
	5	277.23178	0.171819635	3.632817820	823826.1241	843781.8299	0
	6	277.23178	0.17750968	2.990932000	823826.1241	843781.8299	0
	7	277.23178	0.182717825	3.000000000	823826.1241	843774.5078	0
	8	277.24358	0.179352934	2.994826000	825470.6937	845497.5939	0
	9	277.24358	0.177528564	3.460187231	825470.6937	845497.5939	0
	10	277.24358	0.178806633	2.998657000	825470.6937	845497.5939	0

A.2 Results for Dataset B

Dataset B was composed by 266 flight strings and 10 tails, resulting in a total of 2281 variables to be considered on the QUBO model. Table A.2 shows the obtained results for the different tests that were performed for this dataset considering the three different solvers.

Table A.2: Results obtained for dataset B considering the three solvers

Solver	Iteration	Energy	Model Time(s)	Solving Time (s)	OF Cost (UC)	Total Cost (UC)	# Not Assigned Flights
SASAMPLER	1	2039.71047	8.417376267	22.972970963	2926194.49	3006255.595	0
	2	2039.71047	8.417376267	22.493104370	2926194.49	3006255.595	0
	3	2039.72371	8.417366264	24.512780496	2935830.679	3015022.714	0
	4	2039.72371	8.417376267	22.422219177	2935830.679	3015562.184	0
	5	2039.73218	8.417376267	25.128687121	2937235.948	3017581.831	0
	6	2039.73218	8.417376267	22.509806887	2937235.948	3017581.831	0
	7	2039.73218	8.417356262	23.260578529	2937235.948	3014485.13	0
	8	2039.73218	8.417376267	24.693772434	2937235.948	3017581.831	0
	9	2039.73218	8.417376267	23.678797619	2937235.948	3017581.831	0
	10	2039.73218	8.417396267	23.236093505	2937235.948	3017581.831	0
KERBEROS	1	2039.73218	8.417376267	86.218200000	2937235.95	3017581.83	0
	2	2039.75175	8.417376267	81.849700000	2941127.48	3019931.65	0
	3	2039.76362	8.417386267	184.564000000	2942916.07	3022144.49	0
	4	2039.77127	8.41737627	69.364400000	2943432.76	3022220.44	0
	5	2039.77127	8.417356267	104.436100000	2943432.76	3023309.04	0
	6	2039.77127	8.417376267	121.022500000	2943432.76	3023309.04	0
	7	2039.78376	8.417376273	68.767600000	2943978.65	3024039.04	0
	8	2039.79185	8.417396267	87.879400000	2945364.22	3023789.13	0
	9	2039.80235	8.417376367	116.098200000	2949452.13	3028880.6	0
	10	2039.85496	8.417376267	67.332300000	2953769.43	3032743.4	0
HSS	1	2039.71047	8.417376267	2.264967892	2926194.49	3006255.595	0
	2	2039.71047	8.417376267	2.329698372	2926194.49	3006155.595	0
	3	2039.72371	8.417376162	2.307319753	2935830.679	3015022.714	0
	4	2039.73218	8.417376267	2.283741608	2937235.948	3017581.831	0
	5	2039.73218	8.417376267	2.282452149	2937235.948	3017581.831	0
	6	2039.80235	8.417376267	5.849336000	2949452.125	3028511.296	0
	7	2039.82422	8.417356267	5.843157000	2953167.46	3032536.911	0
	8	2039.86442	8.417376267	5.860483000	2956149.825	3036300.28	0

Solver	Iteration	Energy	Model Time(s)	Solving Time (s)	OF Cost (UC)	Total Cost (UC)	# Not Assigned Flights
HSS	9	2040.66512	8.417396261	5.858956000	2924642.954	3004719.01	1
	10	2040.71922	8.417376267	5.853812000	2941976.437	3021706.773	1

A.3 Results for Dataset C

Dataset C was composed by 442 flight strings and 15 tails, resulting in a total of 6185 variables to be considered on the QUBO model. Table A.3 shows the obtained results for the different tests that were performed for this dataset considering the three different solvers. As the modelling part took some time, for this dataset we just obtained the QUBO model once. After having the model, we used the tools described in section **??**, for loading it to be used in the desired tests.

Table A.3: Results obtained for dataset C considering the three solvers

Solver	Iteration	Energy	Solving Time (s)	OF Cost (UC)	Total Cost (UC)	# Not Assigned Flights
SASAMPLER	1	5784.19864	100.154251365	4677048.483	4838237.242	0
	2	5784.22466	96.953693783	4679382.511	4840318.173	0
	3	5788.90319	97.660469203	4642268.115	4804183.413	5
	4	5788.914	98.477399727	4642511.93	4804641.808	5
	5	5788.93134	103.870623174	4643242.454	4805385.218	5
	6	5788.94236	107.815065962	4646350.925	4808385.474	5
	7	5788.9511	106.474855474	4649438.461	4811399.324	5
	8	5788.97157	106.333038408	4652544.13	4814894.782	5
	9	5789.75886	91.850000524	4627741.741	4790077.442	6
	10	5789.77964	101.270924260	4628998.034	4791183.444	6
KERBEROS	1	5784.24124	744.938298277	4682608.642	4842829.156	0
	2	5784.25794	564.658136782	4684362.438	4844689.744	0
	3	5784.25794	724.398569635	4684362.438	4843689.744	0
	4	5787.92842	860.718904894	4639651.196	4801264.914	4
	5	5787.99219	835.921923767	4642622.392	4804439.086	4
	6	5788.00456	513.751813763	4652731.983	4814801.417	4
	7	5788.00456	506.135730028	4652731.983	4814801.417	4
	8	5788.0487	509.829274747	4662796.9	4824635.309	4
	9	5788.88417	751.960776046	4635350.899	4797470.881	5
	10	5788.89005	426.208075702	4639193.868	4800939.368	5
HSS	1	5784.19864	20.589524000	4677048.48	4838237.24	0
	2	5784.19864	20.446742000	4677048.48	4837237.24	0
	3	5784.21274	20.444211000	4678752.46	4839571.37	0
	4	5784.22466	20.507462000	4679382.51	4843318.17	0
	5	5784.23177	20.530345000	4682069.1	4842631.34	0
	6	5784.24124	20.522170000	4682608.64	4842829.16	0
	7	5784.25794	20.478465000	4684362.44	4844689.74	0
	8	5784.25794	20.474923000	4684362.44	4844509.74	0
	9	5784.25794	20.518151000	4684362.44	4844689.74	0
	10	5784.28347	20.523844000	4690094.7	4850433.4	0

References

[1] Network Operations Report, Annex I. Technical report, EUROCONTROL, 2018. URL https://www.eurocontrol.int/sites/default/files/2019-11/nm-annual-network-operations-report-2018-annex-I.pdf. Retrieved by 2020-06-22.

[2] Annual Review. Technical report, IATA, 2019. URL https://www.iata.org/contentassets/c81222d96c9a4e0bb4ff6ced0126f0bb/iata-annual-review-2019.pdf. Retrieved by 2020-06-22.

[3] Airline Cost Management Group. Technical report, IATA, 2019. URL https://www.iata.org/contentassets/3b5a413027704ce08976fe1890fb43e2/acmg-instructions-manual.pdf. Retrieved by 202-06-20.

[4] D-wave systems: exact solver, 2019. URL https://docs.ocean.dwavesys.com/en/stable/docs_dimod/reference/sampler_composites/samplers.html#exact-solver. Retrieved by 2020-06-29.

[5] D-wave systems: D-wave qpu architecture - chimera, 2019. URL https://docs.dwavesys.com/docs/latest/c_gs_4.html. Retrieved by 2020-06-22.

[6] D-wave systems: dwave-tabu, 2019. URL https://docs.ocean.dwavesys.com/projects/tabu. Retrieved by 2020-06-29.

[7] D-wave systems: Underlying quantum physics, 2019. URL https://docs.dwavesys.com/docs/latest/c_gs_2.html. Retrieved by 2020-06-20.

[8] Frank Arute, Kunal Arya, Ryan Babbush, Dave Bacon, Joseph C. Bardin, Rami Barends, Rupak Biswas, Sergio Boixo, Fernando G.S.L. Brandao, David A. Buell, Brian Burkett, Yu Chen, Zijun Chen, Ben Chiaro, Roberto Collins, William Courtney, Andrew Dunsworth, Edward Farhi, Brooks Foxen, Austin Fowler, Craig Gidney, Marissa Giustina, Rob Graff, Keith Guerin, Steve Habegger, Matthew P. Harrigan, Michael J. Hartmann, Alan Ho, Markus Hoffmann, Trent Huang, Travis S. Humble, Sergei V. Isakov, Evan Jeffrey, Zhang Jiang, Dvir Kafri, Kostyantyn Kechedzhi, Julian Kelly, Paul V. Klimov, Sergey Knysh, Alexander Korotkov, Fedor Kostritsa, David Landhuis, Mike Lindmark, Erik Lucero, Dmitry Lyakh, Salvatore Mandrà, Jarrod R. McClean, Matthew McEwen, Anthony Megrant, Xiao Mi,

Kristel Michielsen, Masoud Mohseni, Josh Mutus, Ofer Naaman, Matthew Neeley, Charles Neill, Murphy Yuezhen Niu, Eric Ostby, Andre Petukhov, John C. Platt, Chris Quintana, Eleanor G. Rieffel, Pedram Roushan, Nicholas C. Rubin, Daniel Sank, Kevin J. Satzinger, Vadim Smelyanskiy, Kevin J. Sung, Matthew D. Trevithick, Amit Vainsencher, Benjamin Villalonga, Theodore White, Z. Jamie Yao, Ping Yeh, Adam Zalcman, Hartmut Neven, and John M. Martinis. Quantum supremacy using a programmable superconducting processor. *Nature*, 574(7779):505–510, 2019. ISSN 14764687. doi: 10.1038/s41586-019-1666-5.

[9] Cynthia Barnhart, Natashia L. Boland, Lloyd W. Clarke, Ellis L. Johnson, George L. Nemhauser, and Rajesh G. Shenoi. Flight string models for aircraft fleeting and routing. *Transportation Science*, 32(3):208–220, 1998. doi: 10.1287/trsc.32.3.208.

[10] Cynthia Barnhart, Ellis L. Johnson, George L. Nemhauser, Martin W. P. Savelsbergh, and Pamela H. Vance. Branch-and-price: Column generation for solving huge integer programs. *Operations Research*, 46(3):316–329, 1998. doi: 10.1287/opre.46.3.316.

[11] Cynthia Barnhart, Peter Belobaba, and Amedeo R. Odoni. Applications of operations research in the air transport industry. *Transportation Science*, 37(4):368–391, 2003. ISSN 00411655. doi: 10.1287/trsc.37.4.368.23276.

[12] J. F. Benders. Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik*, 4(1):238–252, 1962. doi: 10.1007/BF01386316.

[13] Ralf Borndörfer, Ivan Dovica, Ivo Nowak, and Thomas Schickinger. Robust tail assignment. Technical Report 10-08, ZIB, Takustr. 7, 14195 Berlin, 2010.

[14] Jun Cai, William G. Macready, and Aidan Roy. A practical heuristic for finding graph minors. pages 1–16, 2014. arXiv:1406.2741 [quant-ph].

[15] Mark Anthony Camilleri. *Aircraft Operating Costs and Profitability*, pages 191–204. Springer International Publishing, Cham, 2018. ISBN 978-3-319-49849-2. doi: 10.1007/978-3-319-49849-2_12.

[16] Antonio J. M. Castro and Eugenio Oliveira. *Web Intelligence and Intelligent Agents*, chapter Disruption Management in Airline Operations Control An Intelligent Agent-Based Approach, pages 107–132. INTECH, March 2010. ISBN 978-953-7619-85-5.

[17] Guillaume Chapuis, Hristo Djidjev, Georg Hahn, and Guillaume Rizk. Finding maximum cliques on the d-wave quantum annealer. *Journal of Signal Processing Systems*, 91(3):363–377, 2019. doi: 10.1007/s11265-018-1357-8.

[18] William Cruz-Santos, Salvador E. Venegas-Andraca, and Marco Lanzagorta. A QUBO Formulation of Minimum Multicut Problem Instances in Trees for D-Wave Quantum Annealers. *Scientific Reports*, 9(1):1–12, 2019. ISSN 20452322. doi: 10.1038/s41598-019-53585-5.

[19] Prasanna Date, Robert Patton, Catherine Schuman, and Thomas Potok. Efficiently embedding qubo problems on adiabatic quantum computers. *Quantum Information Processing*, 18: 117, 2019. doi: 10.1007/s11128-019-2236-3.

[20] David Deutsch and Richard Jozsa. Rapid solution of problems by quantum computation. *Proceedings of the Royal Society of London. Series A: Mathematical and Physical Sciences*, 439(1907):553–558, 1992. doi: 10.1098/rspa.1992.0167.

[21] Edward Farhi, Jeffrey Goldstone, and Sam Gutmann. A Quantum Approximate Optimization Algorithm. pages 1–16, 2014. arXiv:1411.4028 [quant-ph].

[22] Pablo E. Fernandez de la Torre. Airline alliances: The airline perspective. *Massachusetts Institute of Technology, Dept. of Aeronautics and Astronautics, Flight Transportation Laboratory Report R99-1*, pages 1–48, 1999.

[23] A.B. Finnila, M.A. Gomez, C. Sebenik, C. Stenson, and J.D. Doll. Quantum annealing: A new method for minimizing multidimensional functions. *Chemical Physics Letters*, 219(5): 343 – 348, 1994. ISSN 0009-2614.

[24] Robert C. Foster, Brian Weaver, and James Gattiker. Applications of Quantum Annealing in Statistics, 2019. arXiv:1904.06819 [stat.CO].

[25] Gary Froyland, Stephen J. Maher, and Cheng Lung Wu. The recoverable robust tail assignment problem. *Transportation Science*, 48(3):351–372, 2013. ISSN 15265447. doi: 10.1287/trsc.2013.0463.

[26] Sami Gabteni and Mattias Grönkvist. A hybrid column generation and constraint programming optimizer for the tail assignment problem. In J. Christopher Beck and Barbara M. Smith, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 89–103, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg. ISBN 978-3-540-34307-3.

[27] Sami Gabteni and Mattias Grönkvist. Combining column generation and constraint programming to solve the tail assignment problem. *Annals of Operations Research*, 171(1):61, 2008. doi: 10.1007/s10479-008-0379-1.

[28] Moshe Givoni and Piet Rietveld. Airline's choice of aircraft size – explanations and implications. *Transportation Research Part A: Policy and Practice*, 43(5):500 – 510, 2009. ISSN 0965-8564. doi: 10.1016/j.tra.2009.01.001.

[29] Fred Glover, Gary Kochenberger, and Yu Du. Quantum bridge analytics i: a tutorial on formulating and using qubo models. *4OR*, 17(4):335–371, 2019.

[30] F. Gomez and D. Scholz. Improvements to Ground Handling Operations and their Benefits to Direct Operating Costs. *Deutscher Luft- und Raumfahrtkongress*, pages 1–11, 2009.

[31] Mattias Grönkvist. *The tail assignment problem*. PhD thesis, Chalmers University of Technology and Göteborg University, 2005.

[32] Tobias Grosche. *Airline Scheduling Process*, pages 7–46. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009. ISBN 978-3-540-89887-0. doi: 10.1007/978-3-540-89887-0_2.

[33] Lov K. Grover. A fast quantum mechanical algorithm for database search. *Proceedings of the Annual ACM Symposium on Theory of Computing*, Part F129452:212–219, 1996. ISSN 07378017. doi: 10.1145/237814.237866.

[34] Emily Grumbling and Mark Horowitz, editors. *Quantum Computing: Progress and Prospects*. The National Academies Press, Washington, DC, 2019. ISBN 978-0-309-47969-1. doi: 10.17226/25196.

[35] Christopher A. Hane, Cynthia Barnhart, Ellis L. Johnson, Roy E. Marsten, George L. Nemhauser, and Gabriele Sigismondi. The fleet assignment problem: Solving a large-scale integer program. *Mathematical Programming*, 70(1):211–232, 1995. doi: 10.1007/BF01585938.

[36] Feng Hu, Ban-Nan Wang, Ning Wang, and Chao Wang. Quantum machine learning with d-wave quantum computer. *Quantum Engineering*, 1(2):e12, 2019. doi: 10.1002/que2.12. e12 QUE-2019-0007.R1.

[37] Kazuki Ikeda, Yuma Nakamura, and Travis S. Humble. Application of quantum annealing to nurse scheduling problem. *Scientific Reports*, 9(1):12837, 2019. doi: 10.1038/s41598-019-49172-3.

[38] M. W. Johnson, M. H.S. Amin, S. Gildert, T. Lanting, F. Hamze, N. Dickson, R. Harris, A. J. Berkley, J. Johansson, P. Bunyk, E. M. Chapple, C. Enderud, J. P. Hilton, K. Karimi, E. Ladizinsky, N. Ladizinsky, T. Oh, I. Perminov, C. Rich, M. C. Thom, E. Tolkacheva, C. J.S. Truncik, S. Uchaikin, J. Wang, B. Wilson, and G. Rose. Quantum annealing with manufactured spins. *Nature*, 473(7346):194–198, 2011. ISSN 00280836. doi: 10.1038/nature10012.

[39] Tadashi Kadowaki and Hidetoshi Nishimori. Quantum annealing in the transverse ising model. *Phys. Rev. E*, 58:5355–5363, Nov 1998. doi: 10.1103/PhysRevE.58.5355.

[40] James King, Sheir Yarkoni, Jack Raymond, Isil Ozfidan, Andrew D. King, Mayssam Mohammadi Nevisi, Jeremy P. Hilton, and Catherine C. McGeoch. Quantum Annealing amid Local Ruggedness and Global Frustration. *Journal of the Physical Society of Japan*, 88(6), 2019. ISSN 13474073. doi: 10.7566/JPSJ.88.061007.

[41] Andrew Lucas. Ising formulations of many NP problems. *Frontiers in Physics*, 2(February): 1–14, 2014. ISSN 2296424X. doi: 10.3389/fphy.2014.00005.

[42] Raymond E. Miller, James W. Thatcher, and Jean D. Bohlinger, editors. *Reducibility among Combinatorial Problems*, pages 85–103. Springer US, Boston, MA, 1972. ISBN 978-1-4684-2001-2. doi: 10.1007/978-1-4684-2001-2_9.

[43] Francisco Montoito. Application of the Simulated Annealing with Adaptive Local Neighborhood Search to the Tail Assignment Problem The Case Study of TAP. Master's thesis, Instituto Superior Técnico, Universidade de Lisboa, October 2016.

[44] Florian Neukart, Gabriele Compostella, Christian Seidel, David von Dollen, Sheir Yarkoni, and Bob Parney. Traffic flow optimization using a quantum annealer. *Frontiers in ICT*, 4:29, 2017. ISSN 2297-198X. doi: 10.3389/fict.2017.00029.

[45] Christos Papalitsas, Theodore Andronikos, Konstantinos Giannakis, Georgia Theocharopoulou, and Sofia Fanarioti. A QUBO model for the traveling salesman problem with time windows. *Algorithms*, 12(11):1–18, 2019. ISSN 19994893. doi: 10.3390/a12110224.

[46] Alberto Peruzzo, Jarrod McClean, Peter Shadbolt, Man Hong Yung, Xiao Qi Zhou, Peter J. Love, Alán Aspuru-Guzik, and Jeremy L. O'Brien. A variational eigenvalue solver on a photonic quantum processor. *Nature Communications*, 5(2):1–10, 2014. ISSN 20411723. doi: 10.1038/ncomms5213.

[47] P. Ray, B. K. Chakrabarti, and Arunava Chakrabarti. Sherrington-kirkpatrick model in a transverse field: Absence of replica symmetry breaking due to quantum fluctuations. *Phys. Rev. B*, 39:11828–11832, Jun 1989. doi: 10.1103/PhysRevB.39.11828.

[48] Eleanor G. Rieffel, Davide Venturelli, Bryan O'Gorman, Minh B. Do, Elicia M. Prystay, and Vadim N. Smelyanskiy. A case study in programming a quantum annealer for hard operational planning problems. *Quantum Information Processing*, 14(1), 2014. ISSN 15700755. doi: 10.1007/s11128-014-0892-x. arXiv:1407.2887 [quant-ph].

[49] Sebastian Ruther, Natashia Boland, Faramroze G. Engineer, and Ian Evans. Integrated aircraft routing, crew pairing, and tail assignment: Branch-and-price with many pricing problems. *Transportation Science*, 51(1):177–195, 2017. doi: 10.1287/trsc.2015.0664.

[50] Aasish Sharma and Pradip Maharjan. Quantum annealing as an optimized simulated annealing: A case study. 12 2018. doi: 10.13140/RG.2.2.33538.53448.

[51] Hanif D. Sherali, Ebru K. Bish, and Xiaomei Zhu. Airline fleet assignment concepts, models, and algorithms. *European Journal of Operational Research*, 172(1):1–30, 2006. ISSN 03772217. doi: 10.1016/j.ejor.2005.01.056.

[52] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing*, 26(5):1484–1509, 1996. ISSN 00975397. doi: 10.1137/S0097539795293172.

[53] Tobias Stollenwerk, Elisabeth Lobe, and Martin Jung. Flight Gate Assignment with a Quantum Annealer. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 11413 LNCS:99–110, 2019. ISSN 16113349. doi: 10.1007/978-3-030-14082-3_9.

[54] Tony T. Tran, Minh Do, Eleanor G. Rieffel, Jeremy Frank, Zhihui Wang, Bryan O'Gorman, Davide Venturelli, and J. Christopher Beck. A hybrid quantum-classical approach to solving scheduling problems. In *SOCS*, 2016.

[55] Hayato Ushijima-Mwesigwa, Christian F. A. Negre, and Susan M. Mniszewski. Graph partitioning using quantum annealing on the d-wave system. In *Proceedings of the Second International Workshop on Post Moores Era Supercomputing*, PMES'17, page 22–29, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450351263. doi: 10.1145/3149526.3149531.

[56] Wim Van Dam, Michele Mosca, and Umesh Vazirani. How powerful is adiabatic quantum computation? *Annual Symposium on Foundations of Computer Science - Proceedings*, pages 279–287, 2001. ISSN 02725428. doi: 10.1109/SFCS.2001.959902. arXiv:0206003 [quant-ph].

[57] Davide Venturelli, Dominic J. J. Marchand, and Galo Rojo. Job Shop Scheduling Solver based on Quantum Annealing. pages 1–16, 2016. arXiv:1506.08479 [quant-ph].

[58] Pontus Vikstål, Mattias Grönkvist, Marika Svensson, Martin Andersson, Göran Johansson, and Giulia Ferrini. Applying the Quantum Approximate Optimization Algorithm to the Tail Assignment Problem. pages 1–9, 2019. arXiv:1912.10499 [quant-ph].

[59] Krishna Kumar Yadav. An Effective and Efficient Solution to Tail Assignment Problem. *International Journal of Advanced Research in Computer Science and Software Engineering*, 7(4):378–386, 2017. ISSN 22776451. doi: 10.23956/ijarcsse/v7i4/0239.

[60] Jun Zhao, Yaohua Li, and Na Tan. Study on fleet assignment problem model and algorithm. *Mathematical Problems in Engineering*, 2013:581586, 2013. doi: 10.1155/2013/581586.