POLITECNICO DI TORINO

Department of Control and Computer Engineering

Master's Degree in
Ingegneria Informatica (Computer Engineering)

Master's Degree Thesis

Seating Arrangement Optimization in COVID-19 Era: a quantum computing approach

Supervisor:
Prof. Roberto Tadei

Co-Supervisors:
Dr. Daniele Manerba
Dr. Edoardo Fadda

Candidate:
Ilaria Gioda s257312

Company Supervisor
Data Reply S.R.L.
Dr. Emanuele Gallo

Academic Year 2020/2021

To my family and to all those who have remained by my side over these years, giving me strength and reminding me every single day who I really was and what I was capable of.

Acknowledgements

Devo ammettere che mi fa un po' strano essere arrivata a scrivere questa parte, però sono felice di aver finalmente uno spazio tutto mio in cui poter guardare indietro a questi cinque anni passati ed esprimere ciò che ho tenuto per me durante questo tortuoso percorso.

Sono tante le persone che vorrei ringraziare. Cinque anni di università sono un lungo periodo e quello che ho capito è che ogni singolo momento vissuto è stato dato dalle persone che ho incontrato e che mi sono state vicine. Ognuna di esse nel suo piccolo mi ha dato un qualcosa, anche solo un sorriso, una risata o un "siamo tutti sulla stessa barca", donando colore e dolci ricordi alla mia avventura universitaria.

Questo lavoro, così come tutti i traguardi raggiunti, è dedicato prima di tutto alla mia famiglia. Alla mia famiglia, per avermi supportato in tutti questi anni di clausura, adattandosi perfettamente ad ogni situazione e ad ogni mio umore, positivo o negativo che fosse. A mio papà, che mi ha impedito di distrarmi, esortandomi volta per volta a non paragonarmi agli altri, ma a proseguire dritta e convinta per la mia strada. A mia mamma, che mi ha convinta a distrarmi, riuscendo a preservare la propria sanità mentale di madre ansiosa opponendosi egregiamente alla mia insanità mentale di studentessa ansiosa. A mio fratello, per avermi sopportato con grande grande pazienza e per avarmi lasciato lavorare con la finestra aperta anche di domenica mattina durante le sue poche ore di sonno. Un pensiero e ringraziamento va anche a nonna Lina, ai miei cugini, ai miei zii e al resto dei miei nonni, ovunque essi siano. Non avrei potuto chiedere una famiglia migliore di questa e ringrazio ogni singolo giorno per la spensieratezza e l'amore che mi avete dato in tutti questi miei anni di vita.

Il secondo ringraziamento va ai miei amici di sempre: Umberto, Lorenzo, Simone, Chiara, Erika, Vanessa ed Elena. Se c'è una cosa che ho compreso, soprattutto in questi mesi di lontananza, è che ciò che definisce veramente Ilaria Gioda siete proprio voi: ognuno di voi ha contribuito e contribuisce tuttora ad un lato del mio carattere e non sarei la persona che sono adesso senza il vostro costante appoggio ed affetto. Aggiungo un grandissimo grazie a Giulia, mia fedele collega e (s)fortunata compagna di scleri in questo sfiancante viaggio universitario: non so come avrei fatto senza di te e le tue carbonare.

Un grande ringraziamento vorrei poi farlo a tutti i membri della mia associazione studentesca, il Mu Nu Chapter di IEEE-HKN. Grazie a voi sono cresciuta come persona e vi ringrazio per essere stati il mio piccolo angolo di ciel sereno tra i nuvoloni che hanno coperto parte del mio percorso magistrale.

Un sincero ringraziamento va a tutti i ragazzi di Data Reply con i quali ho lavorato in questi mesi di tirocinio, in particolar modo Davide Caputo, Blanca Silva Fernandez e Luca Asproni: grazie per avermi dato l'opportunità di lavorare con voi e per avermi accompagnato nell'esplorazione di

un ramo dell'informatica ancora sconosciuto a molti.

In ultimo, vorrei rivolgere il mio pensiero alle persone che più di tutti gli altri mi sono state vicine con i loro preziosi consigli nello svolgimento di questo lavoro. I miei più cari ringraziamenti vanno al mio relatore, il professor Roberto Tadei, che fin dall'inizio mi ha dato fiducia e ha creduto in me per questa tesi da molti considerata troppo innovativa. La sua presenza costante e le sue celeri risposte alle mie domande sono state di incoraggiamento e mi hanno infuso sicurezza nei momenti di difficoltà spesso incontrati in questo percorso. Un sentito e doveroso grazie va anche ai miei due co-relatori, il professor Daniele Manerba e il dottor Edoardo Fadda, per la loro gentilezza e disponibilità nel dare una chiara direzione al mio lavoro di tesi.

Ilaria

Abstract

The 2020 year has been a turning point for the entire humanity. The COVID-19 pandemic has not only revolutionized our way of thinking, but it has changed the way we live, leading to having limitations on any aspect of our daily life, from the minor things to the biggest. This extraordinary situation has afflicted the railway world, too, since the movements have been limited drastically and new rules have been introduced. This thesis focuses on analyzing passenger transport's current situation on high-speed trains in this particular historical period we are experiencing, the COVID-19 Era. It aims to create hypothetical scenarios of passengers' distribution inside a train wagon with current health and hygiene rules. The main goal of what has been defined as the Seating Arrangement Optimization problem is to fill the train wagon as much as possible and therefore transport the most significant number of people by maximizing the number of passengers belonging to the same family sitting in nearest seats. In this work, two different optimization models for formalizing the identified problem are presented and compared: a Mixed-Integer Linear Programming (MILP) model, one of the most common formulations in literature in the optimization field, and a Quadratic Unconstrained Binary Optimization (QUBO) model, a novel type of problem that can be solved on a quantum computer. In particular, for the resolution of the first type of model, the Gurobi commercial solver is used while, for the latter one, the QBSolv and a classical-quantum hybrid solver provided by D-Wave Systems Inc., a leader company in the quantum computing field, are considered. Quantum computing is an innovative research field currently still under development, on which time and money have begun to be invested only in recent years. What makes this research field attractive is the particular way with which quantum technology operates and the great potential it can offer to solve real-world problems. Due to the current physical capabilities, quantum computers' computational resources are not sufficient to execute too large optimization problems. Still, even with small instances like the ones considered in this work, it can be seen how much they can find highly optimized solutions in a short time.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Context

The COVID-19 pandemic has changed the world. Not only has it changed our way of thinking, but it has changed our lives, starting from big up to small things. What initially looked like a few weeks in between turned out to be months and months of changes, starting from social relationships up to the adaptation of any activity in our life to compliance with health and hygiene rules created to counteract the invisible evil that has swooped in and is constantly lurking over us. Every sector in our society had to readjust and, in some cases, even innovate itself to ensure people could and can lead a life as close as possible to what we were used to before all this took over and conditioned us to the core.

The COVID-19 health emergency has brought new challenges for the public transport industry too. Consequently, many Italian train companies have found themselves forced to reduce the number of high-speed trains due to the lockdown and the impediments to travel between Italian regions. Moreover, railway companies had to implement strategies to ensure maximum safety for passengers and employees both in the stations during boarding and alighting from trains and on the vehicles themselves: usage of protective masks, sanification of convoys and shared environments with special disinfectants, constant air exchange inside enclosed spaces, installation of hand sanitizer gel dispensers and, finally, a social distancing between people, through special markers and procedures [34].

Due to the Italian Government's new regulations, new passenger positioning strategies have been adopted to counter the spread of the COVID-19 virus. The most recent guidelines issued for long-distance rail transport in Italy are contained in Annex 15 of the Decree of the President of the Council of Ministers [22] dated back to 7 September 2020. Among them, some rules regard the personal distancing between passengers and their assignment to seats within high-speed trains during the travel:

- adoption of nominative train tickets to identify all passengers and manage eventual suspected or confirmed cases of infected people on board;
- physical distancing of one meter on board with the application of markers on non-usable seats also ensured through a preventive and preferably online booking mechanism;

- exclusion of the usage of opposite seats, i.e., one in front of another, in cases in which it is not possible to permanently guarantee the interpersonal distance of at least one meter;
- granting an exception from the interpersonal distancing of one meter only in cases in which the use of adjacent or opposite seats is limited exclusively to the occupation by passengers belonging to the same family and/or cohabiting in the same housing unit;
- filling coefficient of the vehicles not exceeding 80% of the seats allowed by the vehicle registration certificate.

Currently, the high-speed rail operators choose to arrange passengers in alternate seats, leading to a total filling capacity of the wagons of 50%. As a result of this choice, the railway companies suffer a drastic reduction in their earnings due to the mismatch between the costs necessary for the activation of the railway transport lines and the revenues obtained from ticket sales. The purpose of this thesis is to investigate how much the filling of a wagon and, generalizing, the filling of an entire train, could be improved if new constraints granted within the limits of the Decree of the President of the Council of Ministers were introduced. With this optimization, more people could be transported on a high-speed train. The railway company in question could significantly increase its earnings, still complying with the rules for protecting its passengers' health.

1.2 Initial idea

The work conducted in this document investigates a novel seat allocation criterion to improve the current condition of passenger transport on high-speed trains, limited by the many restrictions imposed by the health and hygiene regulations on social distancing and the maximum filling capacity inside a train due to the COVID-19 pandemic. In particular, this problem's goal is to find, given a set of people, an adequate allocation of seats that maximizes the number of transported passengers by assigning in close seats people of the same family or stable affections, such that the railway company's profit is increased. The considered problem has been called the Seating Arrangement Optimization problem. The entire study has been conducted with the support of the Quantum Computing Team of Data Reply S.r.l., a consulting company specializing in data science and quantum computing.

The problem is formalized in two different optimization models. A Mixed-Integer Linear Programming (MILP) model, one of the most common formulations in literature in the optimization field, is firstly presented. Among the solvers currently available to resolve this type of problem, the Gurobi Optimizer, a paid commercial solver, has been chosen. A Quadratic Unconstrained Binary Optimization (QUBO) model, a novel type of formulation for optimization problems, is considered.

The peculiarity of the Quadratic Unconstrained Binary Optimization formulation lies in the fact that this model is designed to be solved by a quantum computer through a process called quantum annealing. Quantum computing is an innovative research field for processing information and algorithms with its foundations in quantum physics instead of a traditional one. It relies on the quantum bit's fundamental concept (qubit): qubits can assume both 0 and 1 values like classical bits, but they present a further third state, called superposition, which includes the co-presence of the two states at the same time. By exploiting the quantum properties of these elements, quantum computers offer great potential in computational terms. In the next few years, they are expected to surpass the traditional computers in solving classical complex algorithms, which are now intractable or requiring too much time to be executed. Building quantum computers is still a challenge: the many limitations concerning the technology they are built with and their environment proved to

make quantum machines expensive and challenging to develop and maintain. Due to this, quantum computers' computational resources are not currently sufficient to execute too significant problems, leading to having the quantum advantage still far from being realized.

Many researchers and influential companies such as Google, IBM, NASA, and Rigetti have devoted time and money to research in this new area in recent years. Beyond optimization, many theoretical studies have already been conducted in other various fields such as machine learning [31], chemistry [25], and cybersecurity [32]. The work carried out in this thesis aims to be among the first to analyze and solve a real-world optimization problem using a quantum annealer, i.e., a quantum computer exploiting the process of quantum annealing.

The tools used to solve and analyze the Seating Arrangement Optimization problem are made available a few years ago by D-Wave Systems Inc., a leader company in the quantum computing field. After having formulated the Quadratic Unconstrained Binary Optimization problem, the QBSolv and a classical-quantum hybrid solver provided by the quantum company are used. The differences and advantages between those solvers and the classical optimizer Gurobi are compared to solution quality and time to solution.

1.3 Structure of the thesis

This thesis is organized as follows. In Chapter 2, a review of the literature is performed. Chapter 3 presents the Seating Arrangement Optimization problem and the ad-hoc dataset that has been realized for the software implementation. Chapter 4 outlines the MILP formulation of the problem. Chapter 5 reports details of the MILP model implementation with the Gurobi commercial solver. Chapter 6 analyzes the QUBO formulation of the problem. Chapter 7 presents the basic concepts of quantum annealing, D-Wave quantum technologies, and available solvers. Chapter 8 describes and compares the results of the experimental analysis. Finally, Chapter 9 is the last chapter that contains conclusions and possible future works.

Chapter 2

Literature review

The work carried out in this thesis presents a new avenue in the literature. It presents three types of substantial contributions that bring innovation to the researches in optimization and, specifically, in the railway field:

- the analysis of a real-world problem, the allocation of passengers in seats on a train, through a new type of formulation, the Quadratic Unconstrained Binary Optimization (QUBO) one, designed for being solved using the novel metaheuristic of quantum annealing;
- the comparison between this novel formulation and the classical Mixed-Integer Linear Programming (MILP) one;
- the resolution of a problem created due to the restrictions imposed in the historical period in which we have been living only for a year, the COVID-19 Era.

All these three characteristics lead this work to be unique, and, to the best of our knowledge, this thesis is the first study that has been conducted in this specific field with the current regulations.

The analysis conducted in this document comes close to one of the research branches of the passenger transport literature called seat inventory control, one technique for revenue management. The problems that belong to this category focus on maximizing the total revenue obtained from loading people into the considered mean of transport, efficiently balancing the seats available and sold at different fares. Many studies in this category refer to the airline industry, but some belong to the railway world as reviewed and compared in [2] by Amstrong and Meissner. For example, Peng-Sheng You in [28] developed a hybrid heuristic approach for finding the booking limits for the rail network seat inventory problem and maximizing the total expected sales revenue from selling tickets of all itineraries. Another example of study is the one conducted by Yuan and Nie [38], who focus on the seat allocation problem for the Chinese railway by considering a ticket booking mechanism of China Railway Corporation, called the seat-based control, with consideration of customer behavior such as the customer arrival time (the time when a customer starts to book a ticket) and purchase preference. Although the work of this thesis approaches the research area of seat inventory control, it cannot be entirely classified in this category of problems: it does not focus on the fares of passenger tickets, and it does not propose an actual pricing policy, but the main objective for the allocation of passengers is the observance of social distancing due to COVID-19 regulations.

Many studies have been carried out regarding the novel mathematical QUBO formulation in recent years on this topic. In 2014 Kochenberger et al. conducted a survey [20] that groups in chronological order a good percentage of works on Unconstrained Binary Quadratic Program (UBQP) problems, predecessors of the QUBO ones, up to that year. As reported in their document, researchers are focusing mainly on the formulation of traditional problems in the QUBO form to understand the benefits of this formulation: many of them try to model problems directly in the QUBO form, while others provide for the re-cast of already studied and implemented problems from their original formulations to this novel form. The most famous case in literature is the one conducted by Lucas [23]. He studies how to rewrite many well-known problems from their classical form to the Ising one (a formulation equivalent to the QUBO, mainly used in physics). Another example is the O'Gorman et al. study [27], which shows two different QUBO mapping techniques for a general class of planning problems based on graph-coloring.

In addition to the reformulation of mathematical models, most of the works in the research branch of combinatorial optimization for quantum computers such as quantum annealers focus on the study of concepts, technologies, and possible improvements for the moment in which quantum hardware will be suitable for solving even more complex problems and will prove its supremacy over classical computers [30, 35]. Lewis and Glover in [21] focus on the pre-processing of QUBO by providing methods that can reduce the size of the Q matrix to improve the time to find good solutions. Goh et al. in [16] study how to improve the limitations of the QUBO formulation on large permutation-based optimization problems by proposing a divide-and-conquer framework which also includes the usage of machine learning. Venturelli et al. in [36] first formulate the job-shop scheduling problem in the QUBO form and then design a quantum annealing-based solver for running it on a D-Wave Two machine and comparing its performance with two exhaustive classical algorithms.

Although the literature involving quantum technologies has begun to spread, few practical pieces of research present the analysis of actual cases with a resolution on quantum annealers. The works that come closest to the study in this document are implementations of optimization problems related to real-world applications using recent technologies made by D-Wave Systems. Both works are based on using a hybrid quantum-classical approach to define the problem in the QUBO formulation and the use of the QBSolv solver. The first study is Volkswagen's Traffic Flow Optimization [26], which deals with a real-world application for managing and minimizing congestions on-road segments within a particular city based on the number of cars and their routes. Another work is the Capacitated Vehicle Routing Problem [14], whose research consists in planning tours for vehicles to supply a given number of customers as efficiently as possible, comparing on different datasets a local optimization on CPU with a remote optimization on the QPU of a quantum annealer.

While there has been much research on the QUBO formulation, to the best of our knowledge, no researchers have taken into consideration the comparison between a model in this novel form and one in the classical MILP formulation. This research tries to fill this gap in the literature, and it is probably among the first to make a direct comparison between the Gurobi and the D-Wave Systems solvers.

Chapter 3

The Seating Arrangement Optimization problem

This chapter introduces the case study that has been examined, the Seating Arrangement Optimization problem. In the first section, a description of the problem is presented. In contrast, in the second one, the structure of the dataset that has been created specifically for the software implementation of the two optimization models is analyzed.

3.1 Problem description

The issue of seat allocation has always been a concern of the industries in the railway world. Until last year, the biggest problem consisted of filling the train without particular restrictions but, with the advent of the COVID-19 health emergency, new rules have been imposed on companies. The Seating Arrangement Optimization problem arises from the need to solve one of the further complications that have upset the usual procedures: the allocation of passengers considering the current hygiene regulations for social distancing to protect people's health. In particular, it aims to propose a new seat allocation criterion for allocating people on high-speed trains. The main objective is to fill the train wagon as much as possible and transport the most significant number of people by maximizing the number of passengers belonging to the same family sitting in the nearest seats. We assume that the situation in which our analysis takes place is static: just one train segment, i.e., trip between two adjacent stations, is considered so that the number of passengers and their social relationships are known beforehand without any changes during the travel.

The fundamental concepts of the problem can be summarized as follows:

- Passenger;
- Booking ID;
- Train;
- Wagon;
- Seat.

A set of passengers that has to be transported on a high-speed train is given. During the ticket reservation procedure, each passenger is associated with a unique identifier, the booking ID, which can be shared or not with other passengers. The important assumption of the problem is that people with the same booking ID who made the ticket reservation belong to the same family or live in the same house. This condition, therefore, assumes they can be excluded from the social distancing impositions prescribed by the regulations against the spread of the COVID-19 virus.

A high-speed train is then considered: it is composed of several wagons, each one consisting of a certain number of seats. Each seat is represented by a pair of coordinates, a row, and a column number, which collocate it into a sort of grid. A graphic representation of a typical high-speed train's wagon can be seen in Figure 3.1 below.

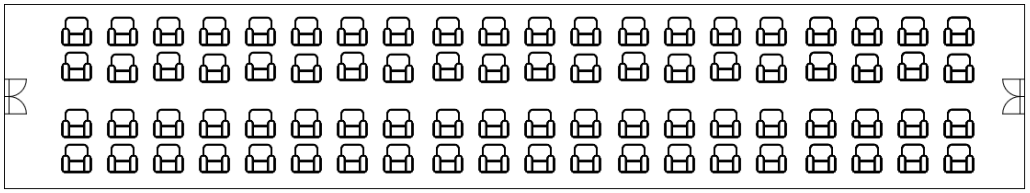

Figure 3.1: Graphical representation of a typical high-speed train wagon

Although the focus of the problem can be extended to the entire train, it should be noted that the analysis that has been carried out and reported in this document refers to only one wagon since the computational resources and capabilities used to obtain the results are not enough to run effectively large-scale data models.

The allocation of seats inside the wagon brings a series of constraints that must be respected to ensure that the identified solution can be considered accurate and truly applicable to solve the Seating Arrangement Optimization problem. All the limitations and goals that have been considered for the modeling of the problem are summarized and reported below:

- allocation of a seat to each one of the considered passengers;
- allocation of a maximum of one seat for each of these passengers (avoid that a passenger has more than one seat assigned to him);
- allocation of a maximum of one passenger to each seat (avoid different passengers being assigned the same seat);
- allocation of distant and not adjacent (in front/behind/left/right) seats to people belonging to different families (identified by different booking IDs).

3.2 Dataset

This section shows the dataset structure for the software implementation phase of the Seating Arrangement Optimization problem and reports the steps followed during its construction. As previously mentioned, the analysis carried out in this thesis refers to a single wagon, so the dataset's structure also reflects this aspect.

The dataset has been created ad-hoc for the problem and consists of three different tables, one for each of the current case study's fundamental key concepts: `SEAT`, `PASSENGER`, and `BOOKING`.

As the name suggests, `SEAT` is the table that contains the data relating to the single seats of the train wagon. The form chosen for this table is the following:

`SEAT(seat_id, row, column)`

Each seat is uniquely identified by an alphanumeric string, the `seat_id`, and has two integers, `row` and `column`, which allows knowing its spatial location within the wagon.

The second table of the dataset is one of the passengers, `PASSENGER`. This table allows collecting all the information regarding the people who have to be transported on the train and, more precisely, in the target wagon. The created table is in the form:

`PASSENGER(passenger_id, name, surname, booking_id, email, birthdate)`

Each row of this table corresponds to a different passenger, uniquely identified by an alphanumeric string, the `passenger_id`, and another ID is associated from the ticket reservation phase, the `booking_id`. The booking ID can be either single (if the passenger is traveling alone) or shared between several passengers (if more than one ticket has been purchased during the same booking session). The rest of the information contained in each row can be considered excessive to the current analysis. Still, it has been decided to keep them to make the passengers' data similar to those of a real dataset.

The third and last table is the reservations table, `BOOKING`. This table was created to maintain the list of booking IDs present in the entire dataset and comes in the form:

`BOOKING(booking_id, reference_email)`

Each row of the table corresponds to a specific booking, uniquely identified by an alphanumeric string, the `booking_id`, and associated to a particular reference email with the passenger who purchased the tickets during that booking session. Notice that the `booking_id` field of the `PASSENGER` table represents a foreign key that references the `booking_id` primary key of the `BOOKING` table.

The next step after defining the dataset's structure has been to generate pertinent data during the software implementation phase. We have chosen to fill the dataset as follows.

The first set of data to be created was the one related to seats. First, a search on the Internet had to be carried out to create a plausible dataset: images of wagons of actual Italian train companies (Italo and Trenitalia) have been searched, and the number of seats inside them has been counted to have an indicative and realistic estimate of how many seats are contained inside a wagon of a real generic train. After this research, a standard wagon of a Frecciarossa train composed of 80 seats has been taken as reference: the seats in this wagon are placed in a 4x20 grid, made up of 4 horizontal (the rows) and 20 vertical (the columns) rows. A spatial representation of the seats can be observed in Figure 3.2.

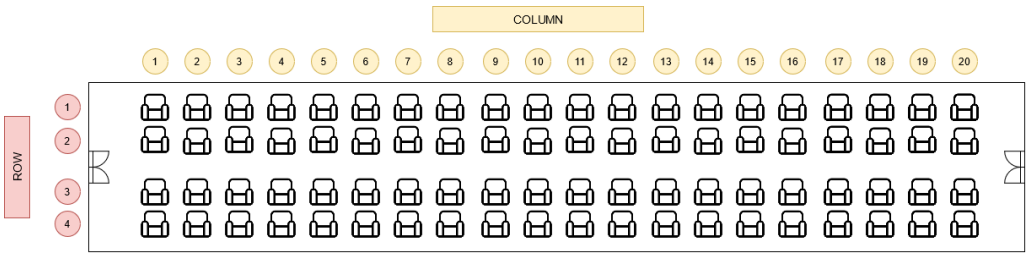

Figure 3.2: Graphical representation of seats with row and column numbers inside a train wagon

The next step has been the creation of all data regarding passengers and reservations. Taking as a reference a credible number of passengers for a high-speed train, the `PASSENGER` table was filled with 1000 passengers, generated with the help of a fake generator [3]. Each passenger was given SSN code as passenger ID, and afterward, it was necessary to associate a specific booking ID. At this point, the creation of the `BOOKING` table proved to be necessary: assuming that the number of family members is 3/4 on average and having a total of passengers equal to 1000, the average number of families on a train and the number of reservations is about 250/330. We decided to use 300 distinct bookings to make a reasonably homogeneous assignment based on this reasoning. By making use of the `np.random.choice` function from the Numpy Python library, one of the 300 booking IDs was randomly assigned to each of the 1000 passengers of the train. The resulting final dataset presents the following characteristics:

- 290 booking IDs (10 out of 300 were not assigned to any passenger due to the usage of the `np.random.choice` function);
- minimum number of passengers with the same booking ID: 1;
- maximum number of passengers with the same booking ID: 8.

Chapter 4

Mixed-Integer Linear Programming formulation

The first mathematical formulation taken into consideration for the analysis of the Seating Arrangement Optimization problem is the Mixed-Integer Linear Programming (MILP) one. In the first section, this chapter describes what a MILP problem is and what its main characteristics are. In contrast, the second one describes how this formulation is used to model the Seating Arrangement Optimization problem. The source code created during the software implementation phase to be executed with the usage of the Gurobi solver, whose details can be explored in Chapter 5, can be found in Appendix A.

4.1 What a Mixed-Integer Linear Programming problem is

A Mixed-Integer Linear Programming (MILP) problem is an optimization problem whose goal is to minimize or maximize a particular objective function, composed of decision variables suitably combined in the form of a mathematical expression. The MILP problems category is included in the more general one of Linear Programming problems, which means that both objective function and constraints are linear combinations of the decision variables. Still, some of these variables are forced to assume integer values in the solution.

In general, a MILP model comes in the standard form

$$\begin{aligned} & \min f(x_1, ..., x_n, y_1, ..., y_q) & \\ \text{s.t.} \quad & g_j(x_1, ..., x_n, y_1, ..., y_q) = 0 & \forall j = \{1, ..., m\} \\ & x_i \geq 0 & \forall i = \{1, ..., n\} \\ & y_k \in \mathbb{Z}^+ & \forall k = \{1, ..., q\} \end{aligned}$$

where x_i and y_k are the variables of the problem.

4.2 Model formulation

In this section, the formulation of the MILP model of the Seating Arrangement Optimization problem is described. First of all, the description of the problem variables is reported, then the

structure of the objective function and the linear constraints is analyzed.

Note that during the definition of the whole model, the following notations have been adopted:

Set of indices:

- R = {1, ..., n_r}: set of seats row numbers;
- C = {1, ..., n_c}: set of seats column numbers;
- K: set of booking IDs.

Parameters:

- n_k: number of passengers with booking ID k

4.2.1 Decision variables

The MILP model created for the problem under consideration presents three types of decision variables.

The first variables, $x_{(r,c),k}$, could be defined as "structural variables": they have been created to declare whether a person with a certain booking ID has been assigned to a certain seat of the wagon or not. For this reason, the variables x are restricted to take only the decision values "yes" / "no", i.e., 1 / 0. Furthermore, as seen later in this document, the same variables have been used in the same way and with the same purpose within the Quadratic Unconstrained Binary Optimization model of the problem. In particular, this first type of variables is defined as follows:

$$x_{(r,c),k} = \begin{cases} 1 & \text{if a passenger with booking ID } k \text{ is assigned to seat with row and column } \textit{(r,c)} \\ 0 & \text{otherwise} \end{cases}$$

$$r \in R,\ c \in C,\ k \in K$$

The second and third types of variables used in the MILP model, $v_{(r,c),k}$ and $w_{(r,c),k}$, can be considered as "auxiliary variables" or "indicators variables". Like the x, these variables can only assume 0 or 1, but these values are strictly related to the ones assumed by the structural variables x. Specifically, the variables v have been created such that they take the 1 value whenever two different passengers, with the same booking ID, are assigned to seats, one next to the other (i.e., two seats whose row numbers are one after the other), while the variables w have been created in such a way that they take the value 1 in the case of two different people, with the same booking ID, assigned to seats facing each other (i.e. two seats whose columns are one after the other). The formal definition of the mentioned auxiliary variables is reported below:

$$v_{(r,c),k} = \begin{cases} 1 & \text{if } x_{(r,c),k} \text{ and } x_{(r+1,c),k} \text{ are both equal to 1} \\ 0 & \text{otherwise} \end{cases}$$

$$r \in R \setminus \{n_r\},\ c \in C,\ k \in K$$

$$w_{(r,c),k} = \begin{cases} 1 & \text{if } x_{(r,c),k} \text{ and } x_{(r,c+1),k} \text{ are both equal to 1} \\ 0 & \text{otherwise} \end{cases}$$

$$r \in R,\ c \in C \setminus \{n_c\},\ k \in K$$

4.2.2 Objective function

The Seating Arrangement Optimization problem's objective is to maximize the number of people with the same booking ID seated close together to fill as many seats as possible inside the wagon. To impose this goal mathematically, the auxiliary variables v and w are used since, by definition, they imply the co-presence of people with the same booking ID seated nearby in the allocation of seats within the solution. The mathematical formulation of the objective function is now reported:

$$\max \sum_{k} \sum_{(r,c)} (v_{(r,c),k} + w_{(r,c),k})$$

4.2.3 Constraints

During the analysis of the Seating Arrangement Optimization problem, six constraints have been identified for the MILP model: four main, conceptually related to the problem under study, plus two written such that they tie the auxiliary variables v and w to the structural variables x and allow the formers to assume the values 0 or 1 based on the value of the latter.

The mathematical formulation of the constraints of the MILP problem is reported below.

1. Each passenger with a certain booking ID is assigned to one seat:

$$\sum_{(r,c)} x_{(r,c),k} = n_k \quad k \in K \tag{4.1}$$

2. Each seat is assigned to one passenger with a certain booking ID at most:

$$\sum_{k} x_{(r,c),k} \leq 1 \quad r \in R,\ c \in C \tag{4.2}$$

3. Two seats, one next to the other (same column), must not be assigned to passengers with different booking ID:

$$x_{(r,c),k} + x_{(r+1,c),k'} \leq 1 \quad r \in R \setminus \{n_r\},\ c \in C,\ k \neq k' \in K \tag{4.3}$$

4. Two seats, one in front of the other (same row), must not be assigned to passengers with different booking ID:

$$x_{(r,c),k} + x_{(r,c+1),k'} \leq 1 \quad r \in R,\ c \in C \setminus \{n_c\},\ k \neq k' \in K \tag{4.4}$$

5. If two passengers with the same booking ID are not assigned to two adjacent seats (one next to the other), the relative variable v must assume the value 0:

$$x_{(r,c),k} + x_{(r+1,c),k} \geq 2 \cdot v_{(r,c),k} \quad r \in R \setminus \{n_r\},\ c \in C,\ k \in K \tag{4.5}$$

6. If two passengers with the same booking ID are not assigned to two adjacent seats (one in front of the other), the relative variable w must assume the value 0:

$$x_{(r,c),k} + x_{(r,c+1),k} \geq 2 \cdot w_{(r,c),k} \quad r \in R,\ c \in C \setminus \{n_c\},\ k \in K \tag{4.6}$$

Note that the objective function and all the constraints defined above imply the variables belonging to their respective domains:

$$x_{(r,c),k} \in \{0,1\} \quad r \in R,\ c \in C,\ k \in K$$

$$v_{(r,c),k} \in \{0,1\} \quad r \in R \setminus \{n_r\},\ c \in C,\ k \in K$$

$$w_{(r,c),k} \in \{0,1\} \quad r \in R,\ c \in C \setminus \{n_c\},\ k \in K$$

Chapter 5

Solving the problem with Gurobi

After having formulated the Seating Arrangement problem in the MILP form, the software implementation has been performed. Among the solvers currently available to resolve this type of problem, the Gurobi Optimizer has been chosen. Gurobi is a paid commercial solver that allows finding optimal solutions for both linear and quadratic optimization problems efficiently and quickly. Among its various features, the Gurobi Optimizer offers a development environment and a series of APIs in different programming languages, including Python, which define a mathematical model and run it over a specific dataset. In this chapter, we show the steps which have been followed and the commands that have been used to create and run our model in the Gurobi environment.

5.1 Solving the MILP problem with the Gurobi solver

We now describe the main aspects of the software implementation of the Seating Arrangement Optimization problem in its MILP form using the functions provided by the Gurobi Python interface, `gurobipy`.

The core of the realized program is the `Model` object on which all the other functions are called. First, the variables (instances of the `Var` class) are created, then the constraints (objects belonging to the `Constr` class) and the objective function are defined. The optimization then starts by calling the `optimize()` method, which allows to run the Gurobi solver and look for solutions that can solve the analyzed problem. In addition to the functions for defining the model, the Gurobi Python interface also offers a specialized data structure, the tuplelist, which allows to maintain and manage Python tuples efficiently through the specific function `select()` [19]. Specifically, this data structure has been used to store the tuples related to the x, v, and w variables of the problem.

After realizing and executing the program, two further steps have been carried out: early solver termination and providing an initial feasible solution. The first change consists of a maximum time limit for the solver to search for the solution by modifying the TimeLimit parameter in the `Model` object [18]. The second one has been performed to provide an initial feasible solution to the solver such that it could find others more efficiently and in fewer time [17]. Note that, in this step, the MIP start values have been specified but just for the x variables, leaving the optimizer to compute the values for the remaining variables by itself. Furthermore, the initial feasible solution that has

been provided to Gurobi has been the one found by the QBSolv solver, whose details are described in the following chapters.

The main steps followed during software implementation of the MILP model with the Gurobi Python interface `gurobipy` are now summarized. As previously mentioned, the entire code can be found in Appendix A.

1. Upload the passengers, seats, and booking ID data and put them within Pandas dataframes to be able to manage them more easily
2. Create a Python dictionary to maintain the correspondence between each booking ID and the number of passengers associated with it
3. Create the Model object

   ```
   m = Model()
   ```

4. Define the binary structural variables x

   ```
   x = m.addVars(x_tuplelist, vtype=GRB.BINARY, name='x')
   ```

5. Provide the solver, if needed, an initial feasible solution for the x variables

   ```
   x[r, c, k].start = 1.0
   ```

6. Create the auxiliary binary variables v and w and define them as result of logic AND operation between x variables, i.e., such that they assume value 1 when the values of the x operands are both 1

   ```
   v = m.addVars(v_tuplelist, vtype=GRB.BINARY, name='v')
   w = m.addVars(w_tuplelist, vtype=GRB.BINARY, name='w')
   m.addGenConstrAnd(value, [x[r, c, k], x[r+1, c, k]])
   m.addGenConstrAnd(value, [x[r, c, k], x[r, c+1, k]])
   ```

7. Specify the problem constraints by calling the function

   ```
   m.addConstr()
   ```

8. Set the linear objective function after having created the Linear Expression object `obj` as sum of the v and w variables

   ```
   m.setObjective(obj, GRB.MAXIMIZE)
   ```

9. Set a time limit in order to stop the solver and obtain a solution after a certain fixed time

   ```
   m.Params.TimeLimit = TIME_LIMIT
   ```

10. Call the Gurobi Optimizer and compute the solution

    ```
    m.optimize()
    ```

11. Count the number of solutions found by the solver

    ```
    nSolutions = m.SolCount
    ```

12. Retrieve the optimal solution found by the solver

    ```
    solution = m.getAttr('x', x)
    ```

Chapter 6

Quadratic Unconstrained Binary Optimization formulation

This chapter presents the Seating Arrangement Optimization problem written in another type of formulation, the Quadratic Unconstrained Binary Optimization (QUBO) one. Initially, a description of the general characteristics of a QUBO problem is reported, then the formulation of the model of our case study is presented in detail. The source code created during the software implementation phase can be found in Appendix B.

6.1 What a Quadratic Unconstrained Binary Optimization problem is

A Quadratic Unconstrained Binary Optimization (QUBO) problem is a type of optimization problem which aims to find an optimal solution, i.e., a combination of variables that satisfies a series of constraints imposed by the context of the problem by minimizing a quadratic function in the form

$$f(x) = \sum_i Q_{i,i} x_i + \sum_{i<j} Q_{i,j} x_i x_j \tag{6.1}$$

where x are the variables of the problem, $Q_{i,i}$ are the diagonal or linear coefficients, and $Q_{i,j}$ are the off-diagonal or quadratic coefficients.

In addition to the functional form, a Quadratic Unconstrained Binary Optimization problem can also be described by the more compact form

$$\min_{x \in \{0,1\}^N} x^T Q x \tag{6.2}$$

where x is a vector of variables of size N and Q a NxN matrix called QUBO matrix.

The first characteristic of a Quadratic Unconstrained Binary Optimization model, as already suggested by its name, is the type and the relationship of the variables x that define the problem. In the QUBO the variables can only be binary, i.e., they can only assume the values 0 and 1, and they

appear in the objective function both in the linear form, i.e., individually x_i, and in the quadratic form, i.e., one multiplied by the other $x_i x_j$.

The second feature of this type of optimization problem is the presence of the Q matrix: it is an upper-triangular squared matrix NxN composed of real values that allow describing the relationship between the variables. The Q matrix, also called QUBO matrix, is the critical element of a QUBO formulation as it enables to summarize all the constraints to which the problem must undergo in a single compact form, and the "Unconstrained" word in the QUBO name refers precisely to this peculiarity.

The study of a given optimization problem in the Quadratic Unconstrained Binary Optimization formulation can be interesting since this particular format is understood and then well suited to be solved by a quantum computer. A more accurate description of the link between a QUBO model and quantum computing can be found in the next chapter.

6.2 Model formulation

The QUBO formulation of the Seating Arrangement Optimization problem is now analyzed. First of all, the decision variables have been identified, and then the objective function and constraints have been formulated, early in the mathematical form and then in the QUBO one. The substantial difference between this formulation and the MILP one can be seen in the fact that in the QUBO model, a single function H is created and has to be minimized. It keeps within it both what we want to optimize and what we want to impose as a limitation for the search of the solution, i.e., the constraints. By expanding the single terms of this function, we can deduce the QUBO matrix Q, which summarizes all the characteristics necessary to solve the problem under examination.

As can be seen in the following subsections, the same notations already considered in Chapter 4 when modeling the Seating Arrangement Optimization problem in the MILP form have been adopted.

Set of indices:

- R = {1, ..., n_r}: set of seats row numbers;
- C = {1, ..., n_c}: set of seats column numbers;
- K: set of booking IDs.

Relevant parameters:

- n_k: number of passengers with booking ID k

6.2.1 Decision variables

The QUBO model of the Seating Arrangement Optimization problem presents a single type of binary variables, the variables $x_{(r,c),k}$. These variables have the same meaning as the variables x saw in the MILP model described in the previous chapter: they indicate whether a passenger with a certain booking ID k has been assigned to the seat with r and c coordinates. The formal notation of these variables is shown below:

$$x_{(r,c),k} = \begin{cases} 1 & \text{if a passenger with booking ID } k \text{ is assigned to seat with row and column } (r,c) \\ 0 & \text{otherwise} \end{cases}$$

$$r \in R,\ c \in C,\ k \in K$$

6.2.2 Objective function

The objective function that has to be minimized in the QUBO form of the Seating Arrangement problem contains everything that characterizes the considered case. The concatenation of five terms gives it, one (the term E) representing what we want to minimize, and four (the terms A, B, C, and D), which correspond to quadratic penalties introduced to impose the constraints identified in the problem under consideration. Specifically, the purpose of the penalties is to prevent the optimizer from choosing solutions that violate the constraints: these penalties involve the addition of a positive quantity, therefore not favorable to the minimization objective, in case of infeasible solutions, while they are equal to zero in case of feasible solutions [15]. In particular, the objective function of the QUBO model is defined as follows:

$$H = \lambda_A \cdot H_A + \lambda_B \cdot H_B + \lambda_C \cdot H_C + \lambda_D \cdot H_D + \lambda_E \cdot H_E \tag{6.3}$$

where λ_A, λ_B, λ_C, λ_D, and λ_E are the parametric coefficients to be calibrated during the software implementation of the QUBO problem.

As already described, the single terms that makeup H correspond to both constraints and the actual function we want to minimize, i.e.:

(A) Assign a seat to a person (constraint);

(B) Assign a person to a seat (constraint);

(C) Empty seats next to people (left/right) with different booking ID (constraint);

(D) Empty seats next to people (in front/behind) with different booking ID (constraint);

(E) Nearest seats to people with same booking ID (to be optimized).

The algebraic and QUBO formulations of such terms are left to the next section.

6.2.3 Algebraic and QUBO formulations of QUBO terms

This section describes in detail the terms that make up the objective function of the QUBO model. First the algebraic formulation of each term has been written and then, starting from it, the QUBO formulation in the form of penalty has been deduced. To pass from one formulation to the other, the table found in the paper [15] and reported in Table 6.1 has been used. The domain to which the decision variables belong is defined as follows:

$$x_{(r,c),k} \in \{0,1\} \quad r \in R,\ c \in C,\ k \in K$$

(A) Assign a seat to a person (constraint)

- Textual formulation:

Table 6.1: Some known penalties (where P is a positive scalar value)

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

"Each passenger with a certain booking ID is assigned to one seat"

- Algebraic formulation:

$$\sum_{(r,c)} x_{(r,c),k} = n_k \quad k \in K$$

- QUBO formulation:

$$H_A = \sum_k (n_k - \sum_{(r,c)} x_{(r,c),k})^2$$

(B) Assign a person to a seat (constraint)

- Textual formulation:

"Each seat is assigned to one passenger with a certain booking ID at most"

- Algebraic formulation:

$$\sum_k x_{(r,c),k} \leq 1 \quad r \in R,\ c \in C$$

- QUBO formulation:

$$H_B = \sum_{(r,c)} \sum_{k,k'} x_{(r,c),k} \cdot x_{(r,c),k'}$$

(C) Empty seats next to people (left/right) with different booking ID (constraint)

- Textual formulation:

"Two seats, one next to the other (same column), must not be assigned to passengers with different booking ID"

- Algebraic formulation:

$$\sum_{k,k'} x_{(r,c),k} \cdot x_{(r+1,c),k'} = 0 \quad r \in R \setminus \{n_r\},\ c \in C,\ k \neq k' \in K$$

- QUBO formulation:

$$H_C = \sum_{(r,c)} \sum_{k,k'} x_{(r,c),k} \cdot x_{(r+1,c),k'}$$

(D) Empty seats next to people (in front/behind) with different booking ID (constraint)

- Textual formulation:

  "Two seats, one in front of the other (same row), must not be assigned to passengers with different booking ID"

- Algebraic formulation:

$$\sum_{k,k'} x_{(r,c),k} \cdot x_{(r,c+1),k'} = 0 \quad r \in R,\ c \in C \setminus \{n_c\},\ k \neq k' \in K$$

- QUBO formulation:

$$H_D = \sum_{(r,c)} \sum_{k,k'} x_{(r,c),k} \cdot x_{(r,c+1),k'}$$

(E) Nearest seats to people with same booking ID (to be optimized)

- Textual formulation:

  "People with same booking ID are assigned to nearest seats"

- Algebraic formulation:

$$\max \sum_{k} \sum_{(r,c)} x_{(r,c),k} \cdot x_{(r+1,c),k} + \sum_{k} \sum_{(r,c)} x_{(r,c),k} \cdot x_{(r,c+1),k}$$

- QUBO formulation:

$$H_E = -\left(\sum_{k} \sum_{(r,c)} x_{(r,c),k} \cdot x_{(r+1,c),k} + \sum_{k} \sum_{(r,c)} x_{(r,c),k} \cdot x_{(r,c+1),k} \right)$$

6.3 The QUBO matrix

The next step after identifying and writing the problem's constraints consists of creating the key and peculiar element of a QUBO problem, the Q matrix. In the current section, the structure of the matrix is first analyzed, after which the steps taken for its creation in the case of the Seating Arrangement Optimization problem are described. In the end, the `calculate_matrix()` function realized during the software implementation phase is shown.

6.3.1 Structure of QUBO matrix

In a QUBO problem, the Q matrix, also known as the QUBO matrix, represents the model's heart. In the form of constant values, it maintains the numerical relationships between all the variables, originating from the constraints to which the problem must subject. This element is also essential at the computational level since its dimensions impact the complexity of the problem itself in terms of quality and time to solution [21].

The QUBO matrix comes in the form of a square NxN matrix, where N is the number of binary variables that constitute the problem and can be found both as symmetric and as upper-triangular matrix [15]. Within this work, it was decided to use the second version of this matrix, i.e., the one composed of real values but equal to 0 under the main diagonal. In particular, in the QUBO model of the Seating Arrangement Optimization problem, it is composed of two distinct types of

constants: the numbers on its diagonal correspond to the coefficients of the linear terms in the objective function (6.1), i.e., the coefficients that multiply the simple variables $x_{(r,c),k}$, while the non-zero values in the upper part of the matrix regulate the relationship between different variables and coincide with the coefficients of the quadratic terms of (6.1), i.e., the coefficients that multiply $x_{(r,c),k} \cdot x_{(r,c),k'}$. A plot of an example QUBO matrix can be seen in Figure 6.1.

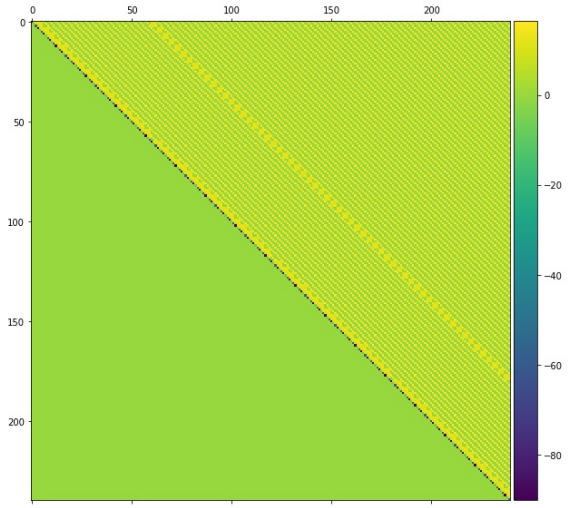

Figure 6.1: Example of the upper-triangular QUBO matrix of size 240x240 created from the instance consisting of 80 seats, 3 booking IDs, and 11 passengers

6.3.2 Expansions of QUBO terms

During the QUBO model's software implementation, the Q matrix was created through the `calculate_matrix()` function. To realize this function, it has been necessary to identify the relationship between the problem's variables, i.e., the multiplicative coefficients of the equation (6.1). First of all, the single QUBO terms of the function (6.3) have been expanded. Then, after the coefficients have been found, they have been multiplied by the parametric coefficients λ_A, λ_B, λ_C, λ_D and λ_E, whose purpose is to give more or less weight to each QUBO penalty such that the constraints are imposed when searching for the solution.

This part of the document reports the expansions of the terms that make up the objective function (6.3): for each of them, the QUBO formulation has been considered and then expanded to explicit the multiplicative factors and use them during the creation of the QUBO matrix.

Expansion of term A

$$H_A = \sum_k (n_k - \sum_{(r,c)} x_{(r,c),k})^2$$

Given a certain booking ID $\bar{k}$, assigned to $n_{\bar{k}}$ passengers, we take into consideration seats

$(r1, c1)$ and $(r2, c2)$:

$$\begin{aligned}
H_A &= (n_{\bar{k}} - (x_{(r1,c1),\bar{k}} + x_{(r2,c2),\bar{k}}))^2 = \\
&= (n_{\bar{k}} - x_{(r1,c1),\bar{k}} - x_{(r2,c2),\bar{k}})^2 = \\
&= n_{\bar{k}}^2 + x_{(r1,c1),\bar{k}}^2 + x_{(r2,c2),\bar{k}}^2 - 2 \cdot n_{\bar{k}} \cdot x_{(r1,c1),\bar{k}} - 2 \cdot n_{\bar{k}} \cdot x_{(r2,c2),\bar{k}} \\
&+ 2 \cdot x_{(r1,c1),\bar{k}} \cdot x_{(r2,c2),\bar{k}} = \\
&= n_{\bar{k}}^2 + x_{(r1,c1),\bar{k}} + x_{(r2,c2),\bar{k}} - 2 \cdot n_{\bar{k}} \cdot x_{(r1,c1),\bar{k}} - 2 \cdot n_{\bar{k}} \cdot x_{(r2,c2),\bar{k}} \\
&+ 2 \cdot x_{(r1,c1),\bar{k}} \cdot x_{(r2,c2),\bar{k}} = \\
&= (1 - 2 \cdot n_{\bar{k}}) \cdot x_{(r1,c1),\bar{k}} + (1 - 2 \cdot n_{\bar{k}}) \cdot x_{(r2,c2),\bar{k}} + 2 \cdot x_{(r1,c1),\bar{k}} \cdot x_{(r2,c2),\bar{k}}
\end{aligned}$$

The multipliers deriving from term A are the following:

- DIAGONAL: $\lambda_A \cdot (1 - 2n_{\bar{k}})$
- OFF DIAGONAL: $\lambda_A \cdot (2)$

Expansion of term B

$$H_B = \sum_{(r,c)} \sum_{k,k'} x_{(r,c),k} \cdot x_{(r,c),k'}$$

Given a certain seat $(\bar{r}, \bar{c})$, we consider a passenger with booking ID q and another one with booking ID p $(p \neq q)$:

$$\begin{aligned}
H_B &= x_{(\bar{r},\bar{c}),q} \cdot x_{(\bar{r},\bar{c}),p} = \\
&= x_{(\bar{r},\bar{c}),q} \cdot x_{(\bar{r},\bar{c}),p} + 0 \cdot x_{(\bar{r},\bar{c}),q} + 0 \cdot x_{(\bar{r},\bar{c}),p}
\end{aligned}$$

The multipliers deriving from term B are the following:

- DIAGONAL: $\lambda_B \cdot (0)$
- OFF DIAGONAL: $\lambda_B \cdot (1)$

Expansion of term C

$$H_C = \sum_{(r,c)} \sum_{k,k'} x_{(r,c),k} \cdot x_{(r+1,c),k'}$$

Given a certain seat $(\bar{r}, \bar{c})$, we take into consideration a passenger with booking ID q and another one with booking ID p $(p \neq q)$:

$$\begin{aligned}
H_C &= x_{(\bar{r},\bar{c}),q} \cdot x_{(\bar{r}+1,\bar{c}),p} = \\
&= x_{(\bar{r},\bar{c}),q} \cdot x_{(\bar{r}+1,\bar{c}),p} + 0 \cdot x_{(\bar{r},\bar{c}),q} + 0 \cdot x_{(\bar{r}+1,\bar{c}),p}
\end{aligned}$$

The multipliers deriving from term C are the following:

- DIAGONAL: $\lambda_C \cdot (0)$
- OFF DIAGONAL: $\lambda_C \cdot (1)$

Expansion of term D

$$H_D = \sum_{(r,c)} \sum_{k,k'} x_{(r,c),k} \cdot x_{(r,c+1),k'}$$

Given a certain seat $(\bar{r}, \bar{c})$, we take into consideration a passenger with booking ID q and another one with booking ID p ($p \neq q$):

$$\begin{aligned} H_D &= x_{(\bar{r},\bar{c}),q} \cdot x_{(\bar{r},\bar{c}+1),p} = \\ &= x_{(\bar{r},\bar{c}),q} \cdot x_{(\bar{r},\bar{c}+1),p} + 0 \cdot x_{(\bar{r},\bar{c}),q} + 0 \cdot x_{(\bar{r},\bar{c}+1),p} \end{aligned}$$

The multipliers deriving from term D are the following:

- DIAGONAL: $\lambda_D \cdot (0)$
- OFF DIAGONAL: $\lambda_D \cdot (1)$

Expansion of term E

$$H_E = -(\sum_{k} \sum_{(r,c)} x_{(r,c),k} \cdot x_{(r+1,c),k} + \sum_{k} \sum_{(r,c)} x_{(r,c),k} \cdot x_{(r,c+1),k})$$

Let's suppose to have two passengers with the same booking ID $\bar{k}$, and let's consider a seat $(\bar{r}, \bar{c})$:

$$\begin{aligned} H_E &= -(x_{(\bar{r},\bar{c}),\bar{k}} \cdot x_{(\bar{r}+1,\bar{c}),\bar{k}} + x_{(\bar{r},\bar{c}),\bar{k}} \cdot x_{(\bar{r},\bar{c}+1),\bar{k}}) = \\ &= -x_{(\bar{r},\bar{c}),\bar{k}} \cdot x_{(\bar{r}+1,\bar{c}),\bar{k}} - x_{(\bar{r},\bar{c}),\bar{k}} \cdot x_{(\bar{r},\bar{c}+1),\bar{k}} \end{aligned}$$

The multipliers deriving from term E are the following:

- DIAGONAL: $\lambda_E \cdot (0)$
- OFF DIAGONAL: $\lambda_E \cdot (-1)$

6.3.3 The calculate_matrix() function

We now analyze in detail the `calculate_matrix()` function and the steps taken during the software implementation to get to call it. This particular function aims to calculate the Q matrix of a specific instance of the QUBO model of the Seating Arrangement Optimization problem.

After an initial phase of reading the dataset and preparing the input data, a dataframe with all the possible "seat - booking ID" combinations called `variables_QUBO` has been generated. The fundamental step during the construction of this dataframe has been the assignment of an increasing numeric ID to each row, i.e., "seat - booking ID" combination, to identify the corresponding row/column index within the QUBO matrix.

Starting from the `variables_QUBO` dataframe, another one, named `full_input` and consisting of all possible combinations of assignments of two booking IDs to two specific seats (e.g., a passenger with booking ID A assigned to seat (1,3) and a passenger with booking ID B assigned to seat (2,18)) has been created. At this point, the numeric ID assigned to each row of `variables_QUBO` proved to be essential in the `full_input` dataframe since it allows to identify and then assign the correct coefficient $Q_{i,j}$ to each combination of variables.

To obtain the Q matrix in its canonical form as an upper-triangular matrix, a further operation has been carried out immediately before the calculation of the QUBO matrix's coefficients: the filtering of the "seat - booking ID - seat - booking ID" combinations based on their ID in the `full_input` dataframe.

```
full_input = full_input.loc[full_input['id2']>=full_input['id1']]
```

The `calculate_matrix()` function is finally called and applied to each of the remaining rows of this dataframe. In particular, it calculates the value associated with the specific row passed as a parameter within the QUBO matrix, and it returns an object in the form {(id1, id2): associated value}. After retrieving all the returned values, the actual matrix Q is created through the iteration on them.

Since the QUBO matrix dimensions have a rather significant impact on the entire program's computation time, two strategies of applying the `calculate_matrix()` function have been tried during the software implementation phase: one sequential, i.e., the function has been applied over the entire dataframe one line at a time by the same processor,

```
QUBO = full_input.apply(calculate_matrix, axis = 1)
```

and one parallelized [1], i.e., the function has been applied simultaneously on multiple rows of the dataframe thanks to the split into smaller dataframes and execution in several parallel processes on different cores

```
QUBO = parallelize_dataframe(full_input, apply_calculate_matrix)
```

The difference in the two strategies' computational times can be seen in more detail in Chapter 8.

The complete source code of the `calculate_matrix()` is now reported.

```
1  def calculate_matrix(row):
2      """
3      Function that calculates the value in the QUBO matrix for the specific row
       passed as parameter
4
5      Returns:
6        array containing a single dictionary --> dictionary containing the entry '(
       id1, id2) -> value'
7      """
8
9      output=[]
10
11     #diagonal
12     if row.id1 == row.id2:
13         output.append({(row.id1,row.id2): coeff_a*(1-2*row.n_passengers1)})
14
15     #off diagonal
16     else:
17         if row.seat1 == row.seat2:
18                 output.append({(row.id1,row.id2): coeff_b*(1)})
19         else:  # row.seat1 != row.seat2
20             if row.bookingID1 != row.bookingID2:
```

```
21                 if ((row.row2 == row.row1 + 1) | (row.row1 == row.row2 + 1)) & (
    row.column1 == row.column2):
22                     output.append({(row.id1,row.id2): coeff_c*(1)})
23                 elif ((row.column2 == row.column1 + 1) | (row.column1 == row.
    column2 + 1)) & (row.row1 == row.row2):
24                     output.append({(row.id1,row.id2): coeff_d*(1)})
25                 else:
26                     output.append({(row.id1,row.id2): 0})
27             else:   # row.bookingID1 == row.bookingID2
28                 if (((row.column1 == row.column2) & ((row.row2 == row.row1 + 1) |
     (row.row1 == row.row2 + 1))) |
29                     ((row.row1 == row.row2) & ((row.column2 == row.column1 + 1) |
     (row.column1 == row.column2 + 1)))):
30                     output.append({(row.id1,row.id2): coeff_a*(2) + coeff_e*(-1)
    })
31                 else:
32                     output.append({(row.id1,row.id2): coeff_a*(2)})
33
34     return output
35
36
37 def apply_calculate_matrix(sub_df):
38     new_sub_df = sub_df.apply(calculate_matrix, axis = 1)
39     return new_sub_df
40
41
42 def parallelize_dataframe(df, func, n_cores=4):
43     """
44     Function that uses parallilelized apply over a given dataframe
45
46     Returns:
47         modified dataframe
48     """
49
50     df_split = np.array_split(df, n_cores)
51     pool = Pool(n_cores)
52     df = pd.concat(pool.map(func, df_split))
53     pool.close()
54     pool.join()
55
56     return df
```

6.4 Coefficients calibration

A brief description of how the calibration of the parametric coefficients λ_A, λ_B, λ_C, λ_D and λ_E (`coeff_a`, `coeff_b`, `coeff_c`, `coeff_d` and `coeff_e` in the code) has been carried out is now reported.

The primary purpose of these parameters is to weigh each QUBO term within the objective function (6.3) such that, in the case of one of the first four terms A, B, C, and D, the associated constraint is always respected, and, in the case of term E, the minimization request is satisfied. Therefore, the calibration of these coefficients is necessary to ensure that the solver can always find good candidate solutions, even in new instances of the problem.

Expressed as pseudo-code, the high-level steps followed during the calibration of the λ coefficients are as follows:

1. Initialization: start by using a small-sized instance of the problem (e.g., 80 seats and 11 passengers) and assign small real random values to λ_A, λ_B, λ_C, λ_D, and λ_E;
2. Run the code and get a particular solution from the solver;
3. Check the solution;
4. If infeasible solution, i.e., solution that does not respect at least one of the A, B, C, D constraints, check its energy:
   4.1. If the energy is greater than the energy of a feasible solution already found, then the solver has failed to find the solution with the lowest energy;
   4.2. If the energy is less than the energy of a feasible solution already found, then the current configuration of the λ parameters is not working; increase the λ parameter for which the solution violates the associated constraint (e.g., infeasible solution because it violates the constraint C, then increases the λ_C coefficient).
5. Continue repeating steps 2., 3. and 4. until only feasible solutions for the current instance of the problem are found;
6. Repeat the entire process by increasing the dimension of the considered instance until only feasible solutions are found.

Chapter 7

Solving the problem with Quantum Computing

This chapter briefly presents the main concepts of Quantum Computing. It reports the strategies undertaken to solve the Seating Arrangement Optimization problem using the tools provided by D-Wave Systems Inc., a leader company in the quantum computing sector. Initially, the concept of Quantum Annealing and its relationship with Quadratic Unconstrained Binary Optimization problems are described. Afterward, a brief overview of the technologies on which the quantum computers owned by D-Wave Systems Inc. are based is given. Finally, the software implementation strategies for solving the QUBO model of our problem on the CPU, through the QBSolv solver, and on the QPU, through the usage of a Leap's quantum-classical hybrid solver, are described.

7.1 QUBO and Quantum Annealing

What makes the study of Quadratic Unconstrained Binary Optimization algorithms so interesting is the link that this class of problems has with one of the most innovative research fields of recent years, Quantum Computing. Quantum Computing is a research field based on a technology that exploits quantum physics instead of classical.

Nowadays, quantum technology is currently still young and under development, and there are different approaches for the construction of quantum computers. However, we can identify two main categories of computers, which differ in the type of structure and the applications for which they are designed [24]. The first category includes Universal or Gate-model quantum computers. The systems belonging to this class of quantum machines are equipped with a particular circuitry that manipulates the qubits' state. Due to the many limitations concerning the technology they are built with and the environment that surrounds them, universal quantum computers offers only several qubits of the order of dozens, and currently, they are used to solve small instances of problems in various areas such as machine learning [31] and chemistry [25]. The second type of quantum computer is equipped with hardware with more available qubits (in the order of thousands) and has optimization as its primary purpose. The systems belonging to this class of quantum computers take the name of Quantum Annealers. Nowadays, several companies, including D-Wave Systems Inc., are involved worldwide in the quantum computing research field: they have been building and, only in recent years, have been granting access to this type of technology for

the resolution of small-to-medium-sized algorithms. Since optimization is the focus of the research considered in this document, it was decided to use a D-Wave Quantum Annealer's technology, whose description of processor architecture and topology is left to the next section. A photograph of the most recent D-Wave Quantum Annealer is shown in Figure 7.1.

Figure 7.1: D-Wave Advantage Quantum Annealer [37]

Quantum annealers owe their name to the process which physically takes place within them and which, by exploiting the intrinsic effects of quantum physics, can be used to solve a specific optimization problem, quantum annealing. In detail, quantum annealing is a meta-heuristic that has as objective the search for the global minimum in the presence of many local minima of a function describing a physical system's energy by using the process based on quantum fluctuations. Quantum annealing exploits the physical concept that everything in nature tends to evolve towards a lower energy state. Since quantum physics follows this reasoning, the quantum process is used to find lower energy states corresponding to optimal solutions for the problem under observation.

The quantum annealing and the quantum systems that implement it are based on characteristics that make their computation principle unique and innovative. The first and most important of which is the quantum bit (qubit). Like a classic computer bit, a qubit can assume one of the two binary values 0 or 1, but the novelty of this element lies in the fact that there is a third further state, called superposition, given by the co-presence of both values at the same time. At the beginning of the annealing process, each qubit finds itself in the superposition state, and its value can assume with equal probability the 0 or 1 state. This probability can be modified and directed towards one of the two states following an external magnetic field, called bias.

The second important concept of quantum annealing is entanglement: qubits do not work alone but take part in a linking process that involves them. That binds them together, making each one's state-dependent on one of the others. At the hardware level, this linking is done through

what is called a coupler: the purpose of this element is to create a sort of single entity formed by pairs of qubits so that these two assume either the same value (both 0 or both 1) or the opposite one (one 0 and the other 1, or vice-versa). Therefore, this unique object has 4 states (00, 01, 10, 11), whose energies depend on the bias of the two single qubits that compose it and the strength of the coupling between them.

What a user of a quantum computer is allowed to do is programming the biases and the coupling strengths in such a way as to create an energy landscape which then allows searching for the lower energy state during the annealing process. Thanks to this way of acting, quantum annealers' technologies are explicitly designed to solve combinatorial problems. It is possible to identify a direct correspondence between the physical implementation of a quantum processing unit (QPU), the hardware processor of a quantum annealer, and a QUBO problem: given a generic QUBO model, each binary variable, whose possible values are 0,1, is represented on QPU by a qubit, whose states are called spin up, spin down, while the linear and quadratic coefficients $Q_{i,i}$, and $Q_{i,j}$ correspond respectively to the biases and coupling strengths on a QPU. Despite this correspondence, usually, QUBO problems can not be directly mapped to the QPU due to the physical interconnections and qubits available on the hardware. In practice, they undergo a translation process called Minor Embedding, which allows the mapping of each logical variable of the problem into a set of physical qubits.

7.2 Architecture of D-Wave Quantum Processing Unit (QPU)

A brief overview of the architecture of a D-Wave quantum processor inside a quantum annealer is now presented. A photograph of a D-Wave processor can be seen in Figure 7.2.

A Quantum Processing Unit (QPU) is the elaboration hardware component of a D-Wave quantum annealer, formed by a series of interconnected qubits via couplers arranged to form a lattice. Depending on the arrangement and the number of qubits that compose it, two versions of D-Wave Systems QPU are now available: the D-Wave 2000Q and the Advantage QPU [11].

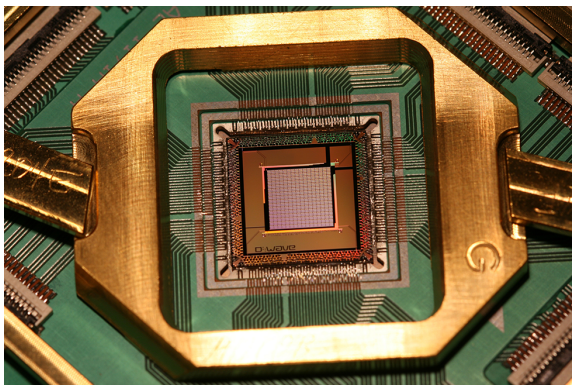

Figure 7.2: D-Wave QPU [37]

The first type of Quantum Processing Unit, the D-Wave 2000Q, features a topology called Chimera [6]. In this topology, the qubits are arranged with a vertical or horizontal orientation. They are interconnected by internal (binding vertical to horizontal qubits) and external couplers (binding horizontal/vertical qubits to each other within the same row/column). Putting together 8 qubits,

4 vertical and 4 horizontal, the Chimera topology's elementary element, called Chimera or unit cell, is obtained. A graphical representation of these interconnections can be seen in Figure 7.3. This basic structure is then replicated NxN times until the entire lattice is obtained: the D-Wave 2000Q QPU consists of a lattice of 16x16 unit cells (from which it also takes the name of C16 Chimera graph) for a total of 2048 qubits and 6016 couplers.

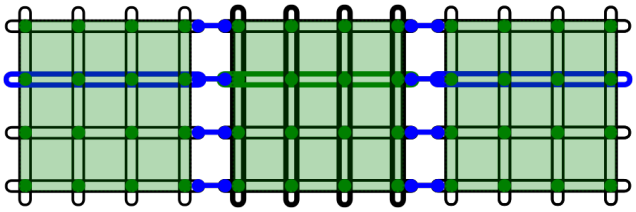

Figure 7.3: Graphical representation of the lattice interconnection of the Chimera topology, as presented in the D-Wave Systems Documentation [11]. In this figure, three-unit cells, highlighted in green, are shown: each one is formed by 4 vertical qubits (four of them highlighted in black) and 4 horizontal qubits (two of them highlighted in blue and one in green). Internal couplers are represented by green dots while external ones by blue dots.

The Chimera topology can also be seen as a graph, having qubits as nodes and couplers as edges. In the Chimera graph each unit cell corresponds to a $K_{4,4}$ bipartite sub-graph, as shown in Figure 7.4.

The second type of Quantum Processing Unit, made available by D-Wave Systems Inc. at the end of 2020, is the Advantage QPU, whose topology is called Pegasus [5]. Being an evolution of its predecessor, the Pegasus topology has a structure similar to D-Wave 2000Q but consists of a higher number of qubits (5640 in total). Here the qubits, besides being still arranged vertically and horizontally, are also put in pairs and connected by odd couplers, leading to having for each of them a total of 15 connections to other different qubits (more than double compared to the 6 of the Chimera topology). Overall, the Advantage QPU has a lattice structure of 16x16 unit cells, also called the P16 Pegasus graph. A graphical representation of the Pegasus topology can be seen in Figure 7.5.

7.3 Solving the QUBO problem with D-Wave Systems Compute Resources

After having analyzed the general characteristics of quantum computing and quantum processing units, we now present the tools provided by D-Wave Systems for solving problems formulated as QUBO. In the last years, the leader of quantum computing has made available to developers a Software Development Kit, the Ocean SDK, containing a series of open-source Python tools for solving hard problems with quantum computers [9]. This suite offers the possibility to solve the second type of model covered in this document by providing some solvers working on the Central Processing Unit (CPU) and others exploiting a Quantum Processing Unit (QPU). Both resolution approaches were used and compared in the analysis of the Seating Arrangement Optimization problem.

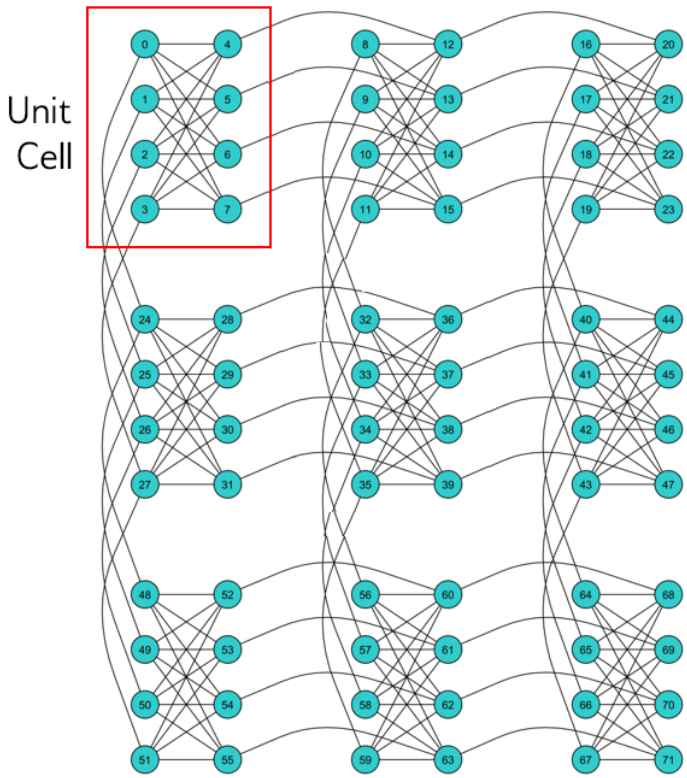

Figure 7.4: Representation of a Chimera topology as Chimera graph from [29], with the detail of a unit cell

7.3.1 Solving the QUBO problem with the QBSolv solver

The first tool that D-Wave Systems provides for solving a QUBO form is QBSolv [10]. QBSolv is an open-source solver made available in recent years (January 2017) by the leading company in quantum computing which runs on the CPU just like the traditional solvers. Its goal is to solve big QUBO problems with high connectivity that otherwise would not be resolved on QPU due to their size. This type of solver strategy consists of partitioning significant QUBO problems into smaller components and applying a specified sampling method (the classical Tabu search algorithm, by default) independently to each of these pieces to find the minimum value required for the optimization. Further technical details on QBSolv can be found in [4].

Within the Python code realized for implementing the Seating Arrangement Optimization problem, after appropriately adapting the QUBO matrix to conform with the solver's input, the sampler is called through the specific function of the `dwave_qbsolv` library:

```
response = QBSolv().sample_qubo(out)
```

This `QBSolv()` function returns a solution, so the latter and its energy can be retrieved, respectively employing the following functions:

```
output = list(response.samples())[0]
energy = response.to_pandas_dataframe()['energy'][0]
```

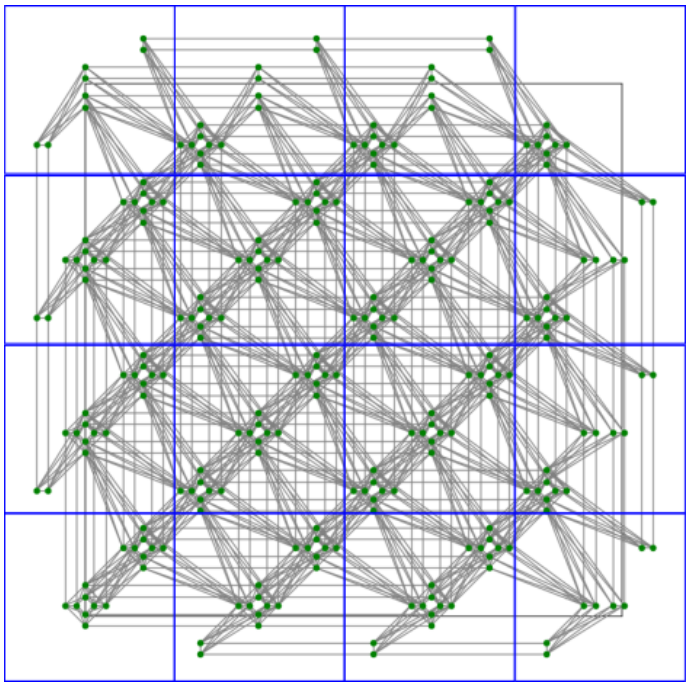

Figure 7.5: Representation of a Pegasus topology as Pegasus graph from [11]

7.3.2 Submitting a QUBO problem to a D-Wave Quantum System

The other important feature that D-Wave Systems has made available to users since 2018 is a cloud service, the D-Wave's Leap, and a set of Python APIs, the Solver API (SAPI). Using these two tools, any developer can access the cloud service and submit any problem to a D-Wave Quantum System. A representation of interaction between a user and a D-Wave quantum computer can be seen in Figure 7.6.

At the time of writing this document, there are two types of solvers made available by the D-Wave Systems company: a solver that directly uses a QPU (to be chosen between an Advantage system and a D-Wave 2000Q lower-noise system) [12] and one that exploits a quantum-classical hybrid system [7]. Both types of sampler accept as input a Binary Quadratic Model (BQM), a specific format that can be constructed starting from the Q matrix of a common Quadratic Unconstrained Binary Optimization problem using the function

```
bqm = BinaryQuadraticModel.from_qubo(qubo)
```

Due to the practical limitations of the QPU (number of qubits and their physical layout on the chip) and the size of the Seating Arrangement Optimization problem instances, it has not been possible to use the first category of solvers. Therefore a quantum-classical hybrid system has been used. A quantum-classical hybrid system is a solver that uses both classical and quantum resources: specifically, it analyzes and subdivides significant BQM problems using classical technology to break them up into sub-problems and establish which of them should be solved with classical algorithms and which can be effectively performed on a quantum processing unit, by interacting directly with it for their resolution [33].

To submit the instances of the Seating Arrangement Optimization problem written in the QUBO formulation to the D-Wave quantum-classical hybrid system, the following SAPIs have been used:

```
sampler = LeapHybridSampler()
results = sampler.sample(bqm)
```

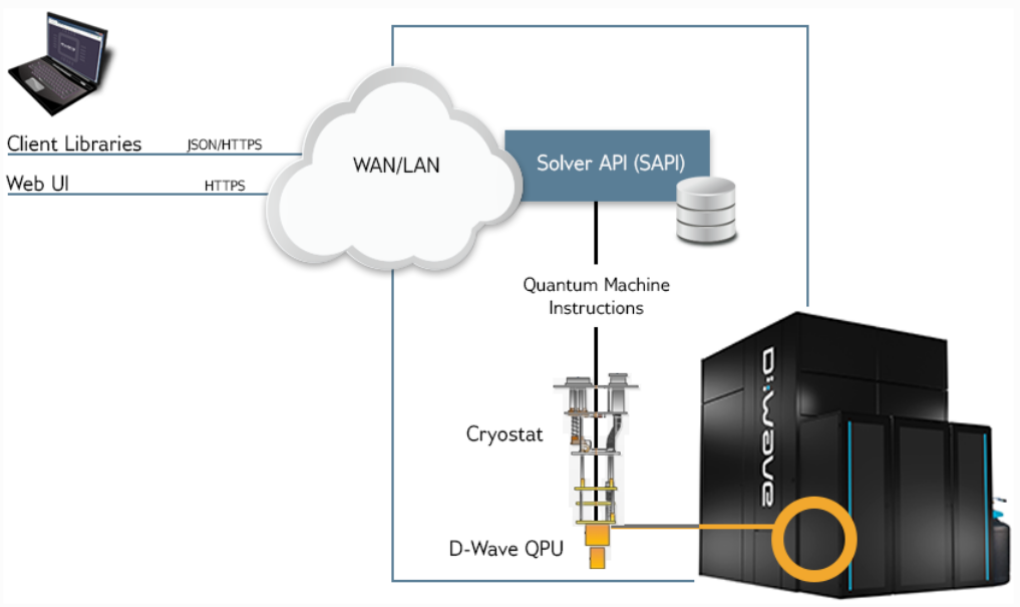

Figure 7.6: Interaction scheme between the user and the D-Wave quantum computer [37]

Chapter 8

Results comparison

In this chapter, the results obtained by running the software programs containing the two algebraic formulations, MILP and QUBO, on different instances of the Seating Arrangement Optimization problem are reported and analyzed under various aspects. The first comparison is on the number of variables needed by the two models on different problem instances. The second one is on the optimal solutions found by the three solvers, Gurobi, QBSolv, and D-Wave Leap's cloud-based quantum-classical hybrid solver (from now on referred to as D-Wave Hybrid Solver). Finally, the last comparison is on the computational times needed in search of the solution. First, the model creation time is analyzed in detail, then the total time taken by the entire program to be executed is considered, and, finally, the running time of the two D-Wave solvers is compared.

8.1 Number of variables

The first comparison that can be made between the two mathematical models designed to formulate the Seating Arrangement Optimization problem is on the number of variables. As can be seen in Figure 8.1, by maintaining the number of available seats fixed at 80, the total number of variables in both models increases as the number of considered booking IDs grows. The substantial difference that immediately stands out in the chart is that the MILP model requires several higher variables, almost three times greater, than the one needed by the QUBO model. The reason for this phenomenon lies in the fact that the two models have in common the structural variables x, one for each "seat - booking ID" combination. Still, the MILP model has two additional variables, one v and one w for each of these combinations (excluding those coming from the limit cases of the seats at the borders of the wagon). In any case, note that the choice of which instances of the problem to use for the comparison of the two models is strictly related and constrained by the number of variables of the QUBO model: for structural reasons related to the QBSolv solver, only instances leading to a maximum of 5000 variables can be solved on CPU with such optimizer.

8.2 Solutions obtained by solvers

This section shows the three solvers' solutions: Gurobi, QBSolv, and D-Wave Hybrid Solver. The numerical results for the Seating Arrangement Optimization problem's various instances can be seen in Table 8.1. In the analyses that have been carried out, it was decided to use a wagon

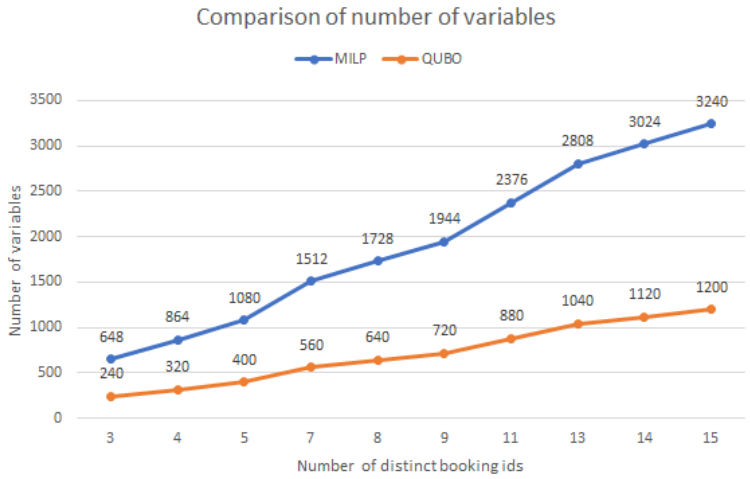

Figure 8.1: Comparison of the number of variables in MILP and QUBO models

consisting of 80 seats and several booking IDs, and therefore, of passengers, gradually increasing. To ensure that the constraints were respected, the results obtained at each run have been checked automatically through functions implemented precisely for this purpose and visually through plots to show the assignment of seats inside the wagon, as can be seen, for example, in Figure 8.2.

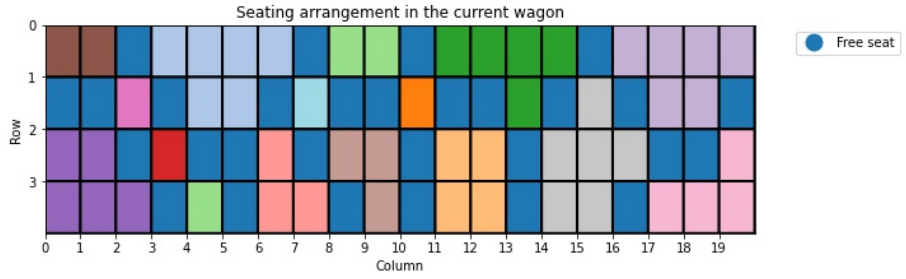

Figure 8.2: Example of graphical representation of a wagon with seating arrangement found by the QBSolv solver (case of 80 seats, 15 booking IDs, and 51 passengers; each color represents a different booking ID)

The obtained results lead to an improvement in passenger transport's current situation: all three solvers manage to find at least an acceptable seating arrangement up to 15 booking IDs for a total of 51 passengers, bringing therefore to have a filling percentage of the seats equal to 63,75%. Beyond this value, however, none of the three solvers was able to find a feasible solution that respects all the imposed constraints. Furthermore, a difference in the three solvers' behavior can be noted in this limit case: Gurobi, not finding any solution, stops and does not provide any result, while instead QBSolv and the D-Wave Hybrid solver still offer a solution, even if an infeasible one. Taking advantage of this common feature of the two D-Wave Systems's solvers, one could think of an optimization approach that considers several wagons and optimizes the remaining passengers smartly from an assignment of a certain wagon within the next one.

In general, the three solvers seem to find the same solutions, respecting the imposed constraints

Table 8.1: Optimal solutions obtained by running the MILP model instances with the Gurobi optimizer, the QUBO model instances with QBSolv and the QUBO model instances with the D-Wave Hybrid classical-quantum solver

Seats	Passengers	Distinct booking IDs	Solver	Nb. of passengers with an assigned seat	Nb. of passengers with same booking ID assigned to close seats	Total minimum energy
80	11	3	Gurobi	11	11	-
			QBSolv	11	10	-464,300
			D-Wave Hybrid	10	11	-464,300
80	16	4	Gurobi	16	16	-
			QBSolv	16	15	-682,800
			D-Wave Hybrid	16	15	-682,800
80	19	5	Gurobi	19	18	-
			QBSolv	19	18	-762,000
			D-Wave Hybrid	19	18	-762,000
80	23	7	Gurobi	23	20	-
			QBSolv	23	21	-849,400
			D-Wave Hybrid	23	21	-849,400
80	28	8	Gurobi	28	25	-
			QBSolv	28	26	-1067,900
			D-Wave Hybrid	28	26	-1067,900
80	34	9	Gurobi	34	32	-
			QBSolv	34	32	-1382,000
			D-Wave Hybrid	34	32	-1382,000
80	39	11	Gurobi	39	35	-
			QBSolv	39	37	-1496,700
			D-Wave Hybrid	39	37	-1496,700
80	44	13	Gurobi	44	39	-
			QBSolv	44	41	-1646,900
			D-Wave Hybrid	44	41	-1646,900
80	50	14	Gurobi	50	43	-
			QBSolv	50	45	-1947,500
			D-Wave Hybrid	50	47	-1958,300
80	51	15	Gurobi	51	42	-
			QBSolv	51	46	-1953,000
			D-Wave Hybrid	51	47	-1961,100

and, most of the time, placing passengers with the same booking ID in close seats. A difference that has been noticed between the Gurobi Optimizer, and the two solvers of D-Wave Systems is that the first one always provides the same solution as it does not have stochastic components inside it, while the other two find a different feasible solution at each run, even though it may not necessarily correspond to the globally optimal one.

Finally, another noteworthy observation is reported. For most of the problem's analyzed instances, the D-Wave Hybrid Solver finds solutions with the same energy as those found by QBSolv optimizer. This means that the solver running on the CPU performs well in terms of solution quality, even

without quantum hardware usage. However, there are two cases, i.e., the ones corresponding to the instances with 14 and 15 distinct booking IDs (respectively 50 and 51 passengers) in which D-Wave Hybrid Solver finds two lower energy and better solutions than those found by the QBSolv sampler.

8.2.1 Case of large instances of the MILP model

Unlike the two solvers by D-Wave Systems, which can find solutions for any instance of the Seating Arrangement Optimization problem up to the limit case of 15 booking IDs/51 passengers, the Gurobi solver could not find even one feasible solution for solving the two largest instances (14 booking IDs/50 passengers and 15 booking IDs/51 passengers) in the established time. An interesting observation came out by providing this solver a feasible solution found by the QBSolv optimizer by executing the QUBO model with the same size and on the same data. Thanks to this starting point, the MILP optimizer proved to find at least an acceptable assignment of seats, thus providing an optimal solution to the problem. In Table 8.2, we report the results obtained by providing and not providing an initial solution to the Gurobi solver in the case of these two large instances.

Table 8.2: Comparison of two large instances of the MILP model providing and not providing an initial solution to the Gurobi solver

Seats	Passengers	Distinct booking IDs	Initial solution provided to Gurobi	Nb. of solutions found by Gurobi	Nb. of passengers with an assigned seat	Nb. of passengers with same booking ID assigned to close seats
80	50	14	Not provided	0	-	-
			Provided	3	50	43
80	51	15	Not provided	0	-	-
			Provided	4	51	42

8.2.2 Impact of term E in the QUBO model

Finally, the impact that the term E, i.e., the term that forces the maximization of the number of nearest seats assigned to passengers with the same booking ID, has in the QUBO model of the Seating Arrangement Optimization problem is shown. Figure 8.3 and Figure 8.4 show the QBSolv solver gives two solutions: the first of them was obtained by creating a model in which the coefficient `coeff_e` is present and therefore contributes with its term to the objective function, while the second provides a model in which the coefficient `coeff_e` has been set to 0. The results are evident: the solution found in the first case consists of an assignment of seats that take into account the booking ID information considering people with the same booking ID seated together. In contrast, the second one implies passengers to be arranged independently from their booking ID. In both cases, the constraints are respected. Still, the first case assignment is better as a possible re-arrangement of passengers would lead to the release of new empty seats that can be used to allocate other passengers.

8.3 Computational time

The last comparison that is now described is on the computational times needed to search for the solution. Computational experiments have been implemented in a desktop computer with 1.8 GHz

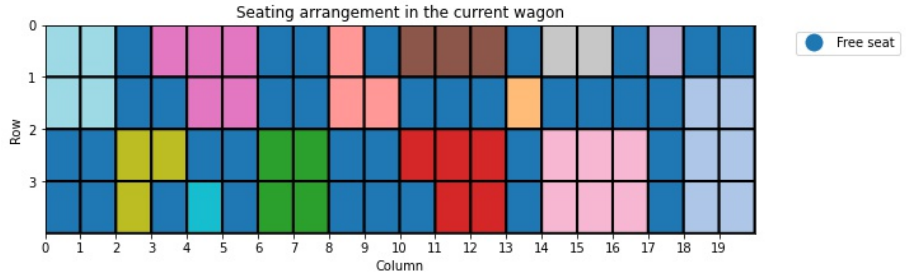

Figure 8.3: Graphical representation of a wagon with seating arrangement found by the QBSolv solver providing a QUBO model with term E (each color represents a different booking ID)

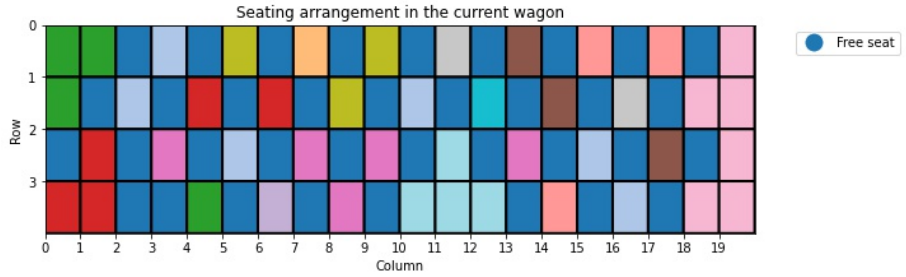

Figure 8.4: Graphical representation of a wagon with seating arrangement found by the QBSolv solver providing a QUBO model without term E (each color represents a different booking ID)

Intel Core i7-8550U processor.

The two software programs reported in Appendix A and Appendix B appear different from each other, but, despite this, we can recognize the same high-level steps:

1. load data;
2. create the model;
3. run the solver;
4. check the solution.

We first examine the difference between the MILP and QUBO model creation times for various instances of the Seating Arrangement Optimization problem. As already seen in Chapter 5, the primary operations for creating the MILP model are all those preceding the call of the `m.optimize()` function and consist in the creation of the `Model` object, definitions of the `Var` variables and the `Constr` constraints and, finally, the specification of the objective function. On the other hand, in the QUBO implementation, variables and constraints do not have to be explicitly created, so the heart of the implementation is creating the `full_input` dataframe containing all the combinations "seat - booking ID - seat - booking ID" and the QUBO matrix by calling the `calculate_matrix()` function. The times that have been obtained are shown in Figure 8.5.

As expected, both models' creation times grow as the number of distinct bookings and, therefore, the number of variables increases. As can be noted, the time for creating the MILP model is much less than the one for making the `full_input` dataframe and the QUBO matrix. The results seem

reasonable: in the latter model, a whole new dataframe must be made by merging all the "seat - booking ID" combinations, and then it must be analyzed entirely to calculate the coefficients within the QUBO matrix.

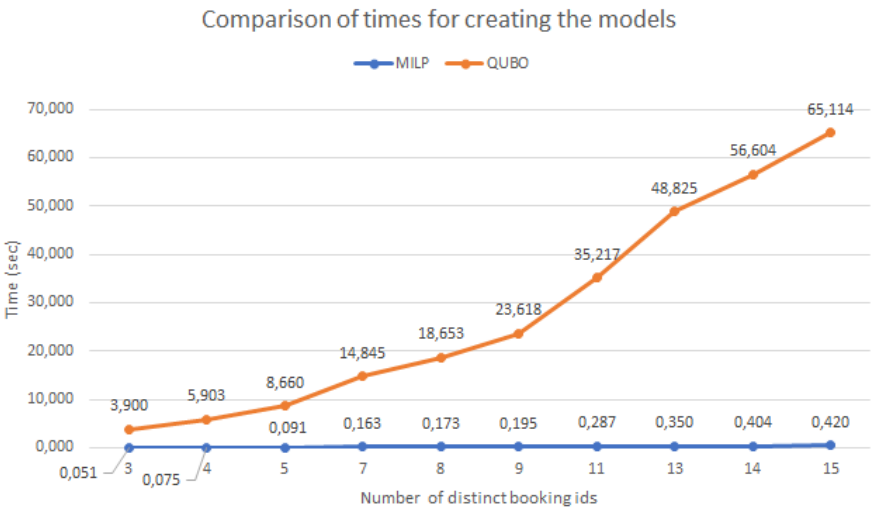

Figure 8.5: Comparison of times for the creation of the MILP and QUBO models

Note that all the times detected for creating the QUBO model and shown in Figure 8.5 have considered the application of the `calculate_matrix()` function to the `full_input` dataframe according to the parallelized approach. Initially, the sequential strategy was adopted through the application of this function one line at a time. Still, since very long computational times were needed, we moved to the parallelized approach, i.e., multiple processes that apply the same function simultaneously on various lines. The performances have improved significantly, leading to a halving of the time for the matrix creation and, therefore, the entire program's execution time. An example of time comparison can be seen in the table Table 8.3.

Table 8.3: Comparison of times for sequential and parallelized apply of calculate_matrix() function in QUBO program

Seats	Passengers	Distinct booking IDs	Approach	Time for applying calculate_matrix() function (sec)	Time for running QUBO program with QBSolv (sec)
80	44	13	Sequential Parallelized	133,101 49,205	148,123 64,708

We now compare the overall time that was necessary for the execution of the two realized software programs. Figure 8.6 compares for each instance of the Seating Arrangement Optimization problem the total time taken for the execution of the program containing the MILP model (executed with the Gurobi solver, without providing any initial solution) and the one served for the one with the QUBO model (performed with the QBSolv solver).

The two approaches for obtaining the solution are different. The QBSolv solver executes and gives the minimum energy solution that it can find in a particular run: this may not be necessarily the

best result since, in another run, the solver could find an even lower energy solution. Instead, the Gurobi solver involves searching for all possible solutions and then providing the optimal one, potentially leading to very long execution times. During the analysis, having set the time limit to 300 seconds, the search by Gurobi is interrupted, and the best solution among those found at the time of the stop is then provided. For this reason, the time taken by the program containing the MILP model and executed with Gurobi remains approximately constant at 300 seconds, while the time taken by the program, including the QUBO model and performed by the QBSolv solver, appears to grow linearly as the number of booking ID taken into account increases.

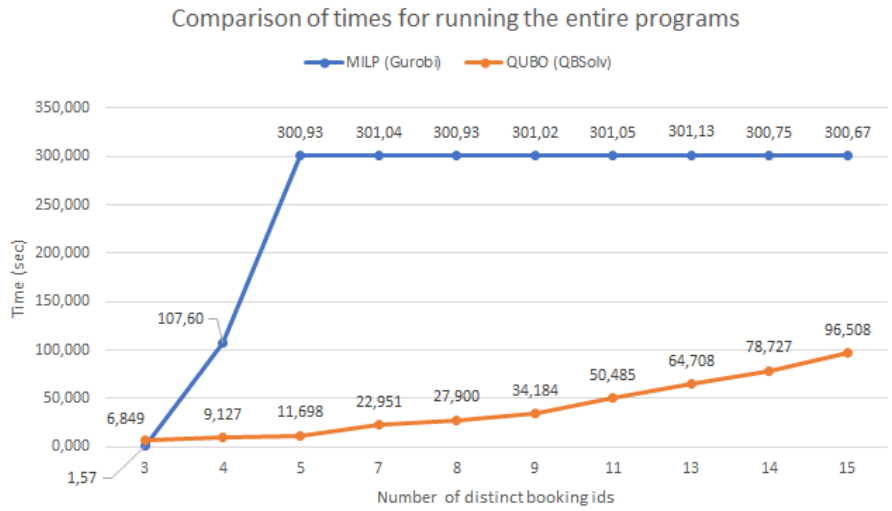

Figure 8.6: Comparison of times for running the entire MILP and QUBO programs

Finally, the time taken to find the QBSolv solver's solution and the time to access and use the D-Wave Leap's cloud-based quantum-classical hybrid solver are compared. As can be seen in Table 8.4, also in this case, for both solvers, the time increases as the size of the considered instance of the problem grows. The difference between these two compared solvers is that the first optimizer, QBSolv, works locally on the CPU. In contrast, the second one, D-Wave Hybrid Solver, requires remote access via the Internet to a physically remote system shared between multiple users. For this reason, the usage of this type of systems provides for additional time-consuming steps: first of all, internet latency required to access the remote machine, then the problem management time by the classic resources of the solver, and finally the wait for execution on QPU due to the presence of instructions of other users which must be run on the same quantum processor. More details on QPU access times can be found in the D-Wave Systems documentation [13]. Therefore, all these conditions lead to an overall time more significant than that employed by the solver, which uses the CPU locally. The real-time use of QPU by the Hybrid Quantum-Classical solver is indicated in the column of the table called "QPU access time" [8]: this time is short compared to the previous two times as the QPU executes instructions in times in the order of microseconds instead of seconds, leading to obtaining a solution for the problem in a short time.

Table 8.4: Comparison of computational times of QBSolv solver and D-Wave Leap's cloud-based quantum-classical hybrid solver

Seats	Passengers	Distinct booking IDs	QBSolv (D-Wave CPU solver) (sec)	LeapHybridSampler (D-Wave quantum-classical hybrid solver) (sec)	QPU access time (D-Wave quantum-classical hybrid solver) (sec)
80	11	3	1,267	6,505	0,040579
80	16	4	1,567	7,792	0,041579
80	19	5	1,801	9,865	0,041494
80	23	7	4,561	15,776	0,042629
80	28	8	5,327	18,214	0,042623
80	34	9	5,976	18,525	0,042694
80	39	11	8,680	19,593	0,042617
80	44	13	10,560	20,819	0,042623
80	50	14	17,527	20,968	0,042624
80	51	15	18,848	22,041	0,042695

Chapter 9

Conclusions and future developments

In this work, we studied a possible seat allocation criterion to solve social distancing on high-speed trains that emerged with the recent hygiene and health regulations due to the COVID-19 pandemic. The problem under consideration, the Seating Arrangement Optimization problem, was analyzed using two mathematical models, a linear model belonging to the Mixed-Integer Linear Programming (MILP) class and a quadratic one appertaining to the Quadratic Unconstrained Binary Optimization (QUBO) category. After describing the problem's details and presenting the ad-hoc dataset created for the case study, the two models were formulated mathematically, with the definition of the decision variables, the constraints, and the objective functions. Particular emphasis was placed on the QUBO matrix, a fundamental element that contains all the essential information of a QUBO problem. A brief description of the matrix was first presented, and then the steps followed for its creation were exposed. Subsequently, the two mathematical models were implemented via software using the Python language. Three solvers were considered and compared to find possible solutions for various instances of the problem. On the one hand, the Gurobi optimizer was used to solve the problem in the MILP formulation. On the other hand, two solvers made available by the D-Wave Systems company were used to solve the problem represented as QUBO model.

This work's main purpose was to study the problem that emerged in this particular period, the COVID-19 Era, in a novel formulation suitable to be executed on a quantum computer. Specifically, we aimed to investigate this innovative computation technique, quantum computing, and analyze the advantages and disadvantages that derive from it. For this reason, a brief overview of quantum computing's branch of interest was exposed. First, quantum annealers, i.e., quantum systems suitable for solving optimization problems, were presented. We then reported a description of quantum annealing, i.e., the process through which problems are solved on quantum annealers, its key elements, qubits, and entanglement's quantum property. Finally, the two types of D-Wave Systems Quantum Processing Unit (QPU) present within the current quantum annealers were described. After presenting the tools made available by D-Wave Systems for solving QUBO problems on Central Processing Unit (CPU) and on quantum-classical hybrid remote systems, the results obtained by executing the three solvers on several instances of the Seating Arrangement Optimization problem were shown.

The novel QUBO formulation offers the possibility of modeling large problems, as it involves several variables that are significantly lower than the one required by a MILP formulation. Despite this advantage, a model of this type has a limitation on the number of variables that the QBSolv solver can accept since it can solve problems up to 5000 variables on a CPU. Furthermore, due to quantum systems' technological limitations related to the noise and the limited number of qubits, it is impossible to map a large problem directly on a QPU fully. Therefore, the classical resources of a quantum-classical hybrid system are currently still needed to deal with very complex problems.

Regarding optimal solutions, the D-Wave Systems solvers do not offer substantial improvements over the Gurobi optimizer, even if they perform slightly better on larger instances by assigning more people with the same booking ID in close seats. Nonetheless, the results obtained from the analysis of the various instances are interesting. Through both models' resolution, it was possible to obtain a maximum percentage of filling the train wagon (63.75%) higher than the one currently implemented by the Italian railway companies (50%).

However, the D-Wave Systems tools' real potential lies in the operations' speed to find the solution. The times needed by the two solvers of the leader company of quantum computing, both on the CPU and the remote quantum-classical hybrid system, are significantly lower than those required by the Gurobi optimizer: in a few seconds, without any external intervention, both solvers can find a feasible solution, even for significant instances of the problem. Furthermore, the innovative capabilities of a QPU are evident in its calculation times: the QPU executes instructions in times of the order of microseconds. This computational advantage cannot be fully exploited. These fractions of seconds are completely obscured by the times required to send the binary quadratic model to the remote system, manage the problem by the classic resources, and wait in the queue to access the quantum processor.

Regarding the Seating Arrangement Optimization problem, some future developments could be made. Currently, the problem considers the allocation of passengers to specific seats belonging to a single wagon. In the future, the analysis could be extended by enlarging the focus of the problem to more wagons, therefore to the whole train, with the allocation of more passengers. Furthermore, the dataset could be modified and extended, by adding a new field, the `wagon_id`, to the `SEAT` table to identify the belonging of a seat to a specific train wagon.

A further study could be done by conducting a more realistic analysis that considers the train's route, origin, destination, and intermediate stations. This case would include passengers getting on and off the train at different stations, so the issue of seat assignment should be reformulated to consider this additional condition.

To conclude, the two software programs realized in this work could be combined to create a single microservice such that, given a list of seats and passengers to be transported as input, a certain allocation of seats in the wagon is provided as output by using one of the two models based on the size of the provided instance. Furthermore, this application could also be developed and tested with a dataset from a real railway company.

Appendix A

Software implementation of the MILP model

In this appendix, the complete code written in the Python language and realized during the software implementation phase for the Mixed-Integer Linear Programming model of the Seating Arrangement Optimization problem is reported.

```
1  from gurobipy import *
2  import pandas as pd
3  import numpy as np
4  import time
5  import matplotlib
6  import matplotlib.pyplot as plt
7  from matplotlib.patches import Patch
8  from matplotlib.lines import Line2D
9
10 TIME_LIMIT = 300
11 start = False
12
13 pass_str = "51"
14 bookingids_str = "15"
15 filenameLog = "log_80_x_"+pass_str+"_"+bookingids_str+"bookingids_nostart.log"
16 filenamePlotWagon = "mip_wagon_80_x_"+pass_str+"_"+bookingids_str+"
       bookingids_nostart.jpg"
17
18 # Initial feasible solution of 51 passengers - 15 booking IDs
```

```
QUBO_sol = {(1, 2, 'BOOKING11'): 1.0, (1, 3, 'BOOKING11'): 1.0, (1, 5, 'BOOKING5
    '): 1.0, (1, 6, 'BOOKING5'): 1.0, (1, 8, 'BOOKING15'): 1.0, (1, 9, 'BOOKING15
    '): 1.0, (1, 10, 'BOOKING15'): 1.0, (1, 12, 'BOOKING1'): 1.0, (1, 13, '
    BOOKING1'): 1.0, (1, 15, 'BOOKING13'): 1.0, (1, 16, 'BOOKING13'): 1.0, (1,
    17, 'BOOKING13'): 1.0, (1, 19, 'BOOKING4'): 1.0, (1, 20, 'BOOKING4'): 1.0,
    (2, 1, 'BOOKING9'): 1.0, (2, 3, 'BOOKING11'): 1.0, (2, 5, 'BOOKING5'): 1.0,
    (2, 7, 'BOOKING6'): 1.0, (2, 9, 'BOOKING15'): 1.0, (2, 10, 'BOOKING15'): 1.0,
     (2, 12, 'BOOKING1'): 1.0, (2, 13, 'BOOKING1'): 1.0, (2, 14, 'BOOKING1'):
    1.0, (2, 16, 'BOOKING13'): 1.0, (2, 18, 'BOOKING4'): 1.0, (2, 19, 'BOOKING4')
    : 1.0, (2, 20, 'BOOKING4'): 1.0, (3, 1, 'BOOKING9'): 1.0, (3, 2, 'BOOKING9'):
     1.0, (3, 4, 'BOOKING2'): 1.0, (3, 6, 'BOOKING12'): 1.0, (3, 8, 'BOOKING3'):
    1.0, (3, 10, 'BOOKING15'): 1.0, (3, 12, 'BOOKING1'): 1.0, (3, 15, 'BOOKING8')
    : 1.0, (3, 17, 'BOOKING20'): 1.0, (4, 1, 'BOOKING9'): 1.0, (4, 2, 'BOOKING9')
    : 1.0, (4, 3, 'BOOKING9'): 1.0, (4, 5, 'BOOKING10'): 1.0, (4, 7, 'BOOKING3'):
     1.0, (4, 8, 'BOOKING3'): 1.0, (4, 9, 'BOOKING3'): 1.0, (4, 11, 'BOOKING10'):
     1.0, (4, 13, 'BOOKING8'): 1.0, (4, 14, 'BOOKING8'): 1.0, (4, 15, 'BOOKING8')
    : 1.0, (4, 16, 'BOOKING8'): 1.0, (4, 18, 'BOOKING7'): 1.0, (4, 19, 'BOOKING7
    '): 1.0, (4, 20, 'BOOKING7'): 1.0}

START_PROGRAM = time.time()

## FUNCTIONS ##
def check_booking_ids_of_neighbors(wagon, R, C, r, c, bookingID):
    """
    Function that checks in the solution if a passenger has been assigned to a
    seat next to other passengers
    with different booking IDs

    Returns:
      0, 0 --> if the passenger is not seated next to anyone with different
    booking ID
      1, 0 --> if the passenger has on his/her right (row+1) one passenger with
    different booking ID
      0, 1 --> if the passenger has in front of him/her (col+1) one passenger
    with different booking ID
      1, 1 --> if the passenger has on his/her right (row+1) and in front of him/
    her (col+1) two passengers
                with different booking IDs
    """

    right = 0
    front = 0
    if bookingID != 0:
        if r+1 < R:
            if ((wagon[r+1][c] != 0) & (bookingID != wagon[r+1][c])):
                right = 1
        if c+1 < C:
            if ((wagon[r][c+1] != 0) & (bookingID != wagon[r][c+1])):
                front = 1
```

```
46      return right, front
47
48  ## IMPORT DATA ##
49
50  # Import SEATS data
51  seats = pd.read_csv('./data/seats.csv').drop_duplicates()
52  seats['joining_col'] = 1
53  print("Total number of seats: " + str(len(seats)))
54  print(seats)     # Display dataframe
55
56  R = seats['row'].drop_duplicates().tolist()          # Set of row numbers
57  C = seats['column'].drop_duplicates().tolist()       # Set of column numbers
58  nr = max(R)
59  nc = max(C)
60
61  # Import PASSENGERS data
62  passengers = pd.read_csv('./data/passengers.csv').drop_duplicates()
63  print("(BEFORE FILTERING) Total number of passengers: " + str(len(passengers)))
64  print("(BEFORE FILTERING) Total number of distinct BOOKING IDs: " + str(
        passengers["booking_id"].nunique()))
65
66  # Select just the passengers with booking ID in {'BOOKING1', 'BOOKING2', ..., '
        BOOKING15'}
67  passengers = passengers.loc[(passengers['booking_id'] == 'BOOKING1') |
68                              (passengers['booking_id'] == 'BOOKING2') |
69                              (passengers['booking_id'] == 'BOOKING3') |
70                              (passengers['booking_id'] == 'BOOKING4') |
71                              (passengers['booking_id'] == 'BOOKING5') |
72                              (passengers['booking_id'] == 'BOOKING6') |
73                              (passengers['booking_id'] == 'BOOKING7') |
74                              (passengers['booking_id'] == 'BOOKING8') |
75                              (passengers['booking_id'] == 'BOOKING9') |
76                              (passengers['booking_id'] == 'BOOKING10') |
77                              (passengers['booking_id'] == 'BOOKING11') |
78                              (passengers['booking_id'] == 'BOOKING12') |
79                              (passengers['booking_id'] == 'BOOKING13') |
80                              (passengers['booking_id'] == 'BOOKING15') |
81                              (passengers['booking_id'] == 'BOOKING20')
82                             ]
83  print("(AFTER FILTERING) Total number of passengers: " + str(len(passengers)))
84  print("(AFTER FILTERING) Total number of distinct BOOKING IDs: " + str(passengers
        ["booking_id"].nunique()))
85  print(passengers)     # Display dataframe
86
87  bookings = passengers[['passenger_id', 'booking_id']].groupby('booking_id').agg('
        count').reset_index()
88  bookings.columns = ['booking_id', 'n_passengers']
89  bookings['joining_col'] = 1
90  print(bookings)     # Display dataframe
```

```
91
92  K = bookings['booking_id'].drop_duplicates().tolist()      # Set of booking IDs
93
94  k_to_nk = {}
95  for index, line in bookings.iterrows():
96      k_to_nk[line['booking_id']] = line['n_passengers']
97
98  print(k_to_nk)
99
100 ## PREPARE DATA ##
101
102 # Create dataframe with all the combinations booking_id - seat
103 combs = seats.merge(bookings, on='joining_col').drop('joining_col', axis=1)[['row
        ', 'column', 'booking_id']]
104 combs['ID'] = range(len(combs))
105 print(combs)     # Display dataframe
106
107 ## GUROBI MILP MODEL ##
108
109 #### CREATE MODEL ####
110 START_CREATE_MODEL = time.time()
111 m = Model()
112
113 #### VARIABLES ####
114 tuples_x = []
115 tuples_v = []
116 tuples_w = []
117 for index, line in combs.iterrows():
118     r = line['row']
119     c = line['column']
120     k = line['booking_id']
121
122     t = (r, c, k)
123
124     tuples_x.append(t)
125     if r != nr:
126         tuples_v.append(t)
127     if c != nc:
128         tuples_w.append(t)
129
130 x_tuplelist = tuplelist(tuples_x)
131 x = m.addVars(x_tuplelist, vtype=GRB.BINARY, name='x')
132
133 v_tuplelist = tuplelist(tuples_v)
134 v = m.addVars(v_tuplelist, vtype=GRB.BINARY, name='v')
135
136 w_tuplelist = tuplelist(tuples_w)
137 w = m.addVars(w_tuplelist, vtype=GRB.BINARY, name='w')
138
```

```
# Provide the modeler a initial feasible solution
if start == True:
    for key, value in QUBO_sol.items():
        r, c, k = key
        x[r, c, k].start = value

m.update()

#### CONSTRAINTS ####
## DEFINITION OF AUXILIARY VARIABLES ##
for key, value in v.items():
    r, c, k = key
    m.addGenConstrAnd(value, [x[r, c, k], x[r+1, c, k]])

for key, value in w.items():
    r, c, k = key
    m.addGenConstrAnd(value, [x[r, c, k], x[r, c+1, k]])

## CONSTRAINT A ##
constrsA = [m.addConstrs(quicksum(x[r, c, k] for r in R for c in C) == k_to_nk[k]
     for k in K)]

## CONSTRAINT B ##
constrsB = [m.addConstrs(quicksum(x[r, c, k] for k in K) <= 1 for r in R for c in
     C)]

## CONSTRAINT C AND CONSTRAINT LINK 1 ##
constrC = []
constrLink1 = []
for key, value in x.items():
    r, c, k1 = key
    if r != nr:     # check if last row
        constrLink1.append(m.addConstr(x[r, c, k1] + x[r+1, c, k1] >= 2 * v[r, c,
     k1]))
        for k2 in K:
            if k1 != k2:
                constrC.append(m.addConstr(x[r, c, k1] + x[r+1, c, k2] <= 1))

## CONSTRAINT D AND CONSTRAINT LINK 2 ##
constrD = []
constrLink2 = []
for key, value in x.items():
    r, c, k1 = key
    if c != nc:     # check if last column
        constrLink2.append(m.addConstr(x[r, c, k1] + x[r, c+1, k1] >= 2 * w[r, c,
     k1]))
        for k2 in K:
            if k1 != k2:
                constrD.append(m.addConstr(x[r, c, k1] + x[r, c+1, k2] <= 1))
```

```
184
185 m.update()
186
187 #### OBJECTIVE FUNCTION ####
188 obj = LinExpr()
189 for k in K:
190     for r, c, k in v_tuplelist.select('*', '*', k):
191         obj += v[r, c, k]
192     for r, c, k in w_tuplelist.select('*', '*', k):
193         obj += w[r, c, k]
194 m.setObjective(obj, GRB.MAXIMIZE)
195
196 END_CREATE_MODEL = time.time()
197
198 #### OPTIMIZE MODEL ####
199 m.Params.TimeLimit = TIME_LIMIT
200 m.Params.LogFile = "./logs/" + filenameLog
201 m.optimize()
202
203 print("Runtime of solver: " + str(m.Runtime))
204
205
206 ## RESULTS ##
207
208 # Get the solution found by the solver
209 print('Number of variables: ' + str(m.NumVars))
210
211 # Print number of solutions stored
212 nSolutions = m.SolCount
213 print('Number of solutions found: ' + str(nSolutions))
214
215 if(nSolutions == 0):
216     print("No feasible solution found.")
217 else:
218     solution = m.getAttr('x', x)
219
220     # Create a matrix that will be used for the graphical representation of the
    wagon
221     wagon = np.zeros(shape=(nr, nc))
222
223     # Print and check solution
224     check_A = 0
225     check_no_seat_assigned = 0
226     for k in K:
227         print("Given booking ID {} of {} passengers...".format(k, k_to_nk[k]))
228         assigned = 0
229         for r in R:
230             for c in C:
231                 if solution[r, c, k] > 0:
```

```
232                     print("... one passenger assigned to seat ({},{})".format(r,
    c))
233                     assigned += 1
234         if assigned != k_to_nk[k]:
235             check_A += 1
236             check_no_seat_assigned += (k_to_nk(k) - assigned)
237 
238     print("")
239 
240     check_B = 0
241     for r in R:
242         for c in C:
243             print("Given seat ({},{})...".format(r, c))
244             occupied = 0
245             for k in K:
246                 if solution[r, c, k] > 0:
247                     print("... assigned one passenger with booking ID {}".format(
    k))
248                     id_num = k.replace("BOOKING", "")
249                     wagon[r-1, c-1] = int(id_num)
250                     occupied += 1
251             if occupied == 0:
252                 print("... no passenger assigned")
253             elif occupied > 1:
254                 check_B += 1
255 
256     check_C = 0
257     check_D = 0
258     for r in R:
259         for c in C:
260             ret_c, ret_d = check_booking_ids_of_neighbors(wagon, nr, nc, r-1, c
    -1, wagon[r-1][c-1])
261             check_C += ret_c
262             check_D += ret_d
263 
264 
265     # Feasibility check results
266     print('Number of infeasibilites due to term A: ' + str(check_A))
267     print('Number of infeasibilites due to term B: ' + str(check_B))
268     print('Number of infeasibilites due to term C: ' + str(check_C))
269     print('Number of infeasibilites due to term D: ' + str(check_D))
270     print('Number of passengers with a seat: {}'.format(str(len(passengers) -
    check_no_seat_assigned)))
271     print('Number of passengers with same booking ID assigned to nearest seats: %
    g' % m.objVal)
272 
273 
274     # Graphical representation of the solution
275     plt.figure(figsize=(10, 3))
```

```
276     cmap = matplotlib.colors.ListedColormap(matplotlib.cm.get_cmap("tab20").
        colors[1:])
277
278     plt.pcolor(wagon, edgecolors='k', linewidths=2, cmap=cmap)
279     plt.gca().invert_yaxis()
280     plt.xticks(np.arange(nc))
281     plt.yticks(np.arange(nr))
282     plt.xlabel('Column')
283     plt.ylabel('Row')
284     plt.title('Seating arrangement in the current wagon')
285
286     legend_elements = [Line2D([0], [0], marker='o', color='w', label='Free seat',
287                               markerfacecolor=cmap.colors[0], markersize=15)]
288     plt.legend(handles=legend_elements,  bbox_to_anchor=(1.05, 1), loc='upper
        left')
289
290     plt.savefig("./plots/" + filenamePlotWagon, bbox_inches='tight')
291
292 END_PROGRAM = time.time()
293
294 print("Time to create the model: {} seconds".format(END_CREATE_MODEL -
        START_CREATE_MODEL))
295 print("Total time to run the program: {} seconds".format(END_PROGRAM -
        START_PROGRAM))
```

Appendix B

Software implementation of QUBO model

In this appendix, the complete code written in the Python language and realized for the Quadratic Unconstrained Binary Optimization model of the Seating Arrangement Optimization problem in the software implementation phase is reported.

```
1  import pandas as pd
2  import numpy as np
3  import math
4  import time
5  import matplotlib
6  import matplotlib.pyplot as plt
7  from dwave_qbsolv import QBSolv
8  from multiprocessing import Pool
9  from matplotlib.patches import Patch
10 from matplotlib.lines import Line2D
11
12 pass_str = "16"
13 bookingids_str = "4"
14 filenameQ = "Q_80_x_"+pass_str+"_"+bookingids_str+"bookingids.txt"
15 filenameVariablesQUBO = "variables_80_x_"+pass_str+"_"+bookingids_str+"bookingids
       .csv"
16 filenamePlotWagon = "qubo_wagon_80_x_"+pass_str+"_"+bookingids_str+"bookingids.
       jpg"
17
18 # QUBO coefficients
19 coeff_a = 8.2
20
21 coeff_b = 11.58
22 # coeff_b = 0
23
```

```
24 coeff_c = 10.56
25 # coeff_c = 0
26
27 coeff_d = 10.83
28 # coeff_d = 0
29
30 coeff_e = 2.7
31 # coeff_e = 0
32
33
34 def calculate_matrix(row):
35     """
36     Function that calculates the value in the QUBO matrix for the specific row
       passed as parameter
37
38     Returns:
39       array containing a single dictionary --> dictionary containing the entry '(
       id1, id2) -> value'
40     """
41
42     output=[]
43
44     #diagonal
45     if row.id1 == row.id2:
46         output.append({(row.id1,row.id2): coeff_a*(1-2*row.n_passengers1)})
47
48     #off diagonal
49     else:
50         if row.seat1 == row.seat2:
51                 output.append({(row.id1,row.id2): coeff_b*(1)})
52         else:  # row.seat1 != row.seat2
53             if row.bookingID1 != row.bookingID2:
54                 if ((row.row2 == row.row1 + 1) | (row.row1 == row.row2 + 1)) & (
       row.column1 == row.column2):
55                     output.append({(row.id1,row.id2): coeff_c*(1)})
56                 elif ((row.column2 == row.column1 + 1) | (row.column1 == row.
       column2 + 1)) & (row.row1 == row.row2):
57                     output.append({(row.id1,row.id2): coeff_d*(1)})
58                 else:
59                     output.append({(row.id1,row.id2): 0})
60             else:   # row.bookingID1 == row.bookingID2
61                 if (((row.column1 == row.column2) & ((row.row2 == row.row1 + 1) |
        (row.row1 == row.row2 + 1))) |
62                     ((row.row1 == row.row2) & ((row.column2 == row.column1 + 1) |
        (row.column1 == row.column2 + 1)))):
63                     output.append({(row.id1,row.id2): coeff_a*(2) + coeff_e*(-1)
       })
64                 else:
65                     output.append({(row.id1,row.id2): coeff_a*(2)})
```

```
66
67      return output
68
69
70  def apply_calculate_matrix(sub_df):
71      new_sub_df = sub_df.apply(calculate_matrix, axis = 1)
72      return new_sub_df
73
74
75  def parallelize_dataframe(df, func, n_cores=4):
76      """
77      Function that uses parallilelized apply over a given dataframe
78
79      Returns:
80          modified dataframe
81      """
82
83      df_split = np.array_split(df, n_cores)
84      pool = Pool(n_cores)
85      df = pd.concat(pool.map(func, df_split))
86      pool.close()
87      pool.join()
88
89      return df
90
91  def check_booking_ids_of_neighbors(wagon, R, C, r, c, bookingID):
92      """
93      Function that checks if in the solution a passenger has been assigned to a
        seat next to other passengers
94      with the same booking ID
95
96      Returns:
97        0 --> if the passenger is not seated next to anyone with the same booking
        ID
98        1 --> if the passenger is seated next to at least one passenger with the
        same booking ID
99      """
100
101     if bookingID != 0:
102         if r-1 >= 0:
103             if bookingID == wagon[r-1][c]:
104                 return 1
105         if r+1 < R:
106             if bookingID == wagon[r+1][c]:
107                 return 1
108         if c-1 >= 0:
109             if bookingID == wagon[r][c-1]:
110                 return 1
111         if c+1 < C:
```

```
112             if bookingID == wagon[r][c+1]:
113                 return 1
114 
115     return 0
116 
117 START_PROGRAM = time.time()
118 
119 ## Data import ##
120 
121 # Import seats data
122 seats = pd.read_csv('../data/seats.csv').drop_duplicates()
123 seats['joining_col'] = 1
124 print("Total number of seats: " + str(len(seats)))
125 print(seats)       # Display dataframe
126 
127 # Import passengers data
128 passengers = pd.read_csv('../data/passengers.csv').drop_duplicates()
129 passengers['joining_col'] = 1
130 print("(BEFORE FILTERING) Total number of passengers: " + str(len(passengers)))
131 print("(BEFORE FILTERING) Total number of distinct BOOKING IDs: " + str(
        passengers["booking_id"].nunique()))
132 
133 # Select just the passengers with booking ID in {'BOOKING1', 'BOOKING2', ..., '
        BOOKING15'}
134 passengers = passengers.loc[(passengers['booking_id'] == 'BOOKING1') |
135                             (passengers['booking_id'] == 'BOOKING2') |
136                             (passengers['booking_id'] == 'BOOKING3') |
137                             (passengers['booking_id'] == 'BOOKING4') |
138                             (passengers['booking_id'] == 'BOOKING5') |
139                             (passengers['booking_id'] == 'BOOKING6') |
140                             (passengers['booking_id'] == 'BOOKING7') |
141                             (passengers['booking_id'] == 'BOOKING8') |
142                             (passengers['booking_id'] == 'BOOKING9') |
143                             (passengers['booking_id'] == 'BOOKING10') |
144                             (passengers['booking_id'] == 'BOOKING11') |
145                             (passengers['booking_id'] == 'BOOKING12') |
146                             (passengers['booking_id'] == 'BOOKING13') |
147                              (passengers['booking_id'] == 'BOOKING15') |
148 #                             (passengers['booking_id'] == 'BOOKING19') |
149                              (passengers['booking_id'] == 'BOOKING20')
150                            ]
151 
152 print("(AFTER FILTERING) Total number of passengers: " + str(len(passengers)))
153 print("(AFTER FILTERING) Total number of distinct BOOKING IDs: " + str(passengers
        ["booking_id"].nunique()))
154 print(passengers)       # Display dataframe
155 
156 # Import boookings data
```

```
157 bookings = passengers[['passenger_id', 'booking_id']].groupby('booking_id').agg('
        count').reset_index()
158 bookings.columns = ['booking_id', 'n_passengers']
159 bookings['joining_col'] = 1
160 print(bookings)       # Display dataframe
161
162 ## Data preparation ##
163
164 # Create dataframe with all the combinations seat - bookingid
165 combs = seats.merge(bookings, on='joining_col').drop('joining_col', axis=1)[['
        seat_id', 'row', 'column', 'booking_id', 'n_passengers']]
166 combs['ID'] = range(len(combs))
167 print(combs)     # Display dataframe
168
169 ## Create input QUBO ##
170
171 START_CREATEMODEL = time.time()
172
173 # Copy dataframe with all combinations and prepare it for the next merge
        operation
174 variables_QUBO = combs
175 variables_QUBO['joining_col'] = 1
176 print(variables_QUBO)       # Display dataframe
177
178 # Create dataframe with all the combinations seat1 bookingid1 - seat2 -
        bookingid2
179 full_input = variables_QUBO.merge(variables_QUBO, on = 'joining_col').drop('
        joining_col', axis = 1)
180 variables_QUBO.drop('joining_col', axis = 1, inplace=True)
181 print(full_input)       # Display dataframe
182
183 # Rename columns
184 full_input.columns = ['seat1', 'row1', 'column1', 'bookingID1', 'n_passengers1',
        'id1', 'seat2', 'row2', 'column2', 'bookingID2', 'n_passengers2', 'id2']
185
186 # Filter rows in order to have an upper triangular QUBO matrix
187 full_input = full_input.loc[full_input['id2']>=full_input['id1']]
188
189 ## Create QUBO ##
190
191 START_APPLY = time.time()
192
193 # QUBO = full_input.apply(calculate_matrix, axis = 1)
194 QUBO = parallelize_dataframe(full_input, apply_calculate_matrix)
195
196 END_APPLY = time.time()
197
198 print(QUBO)
199
```

```
200 psize = len(variables_QUBO)
201 Q = np.zeros((psize, psize))
202
203 for q in QUBO:
204     for (k,v) in q[0]:
205         try:
206             Q[k, v] = q[0][(k,v)]
207         except:
208             print(k)
209             continue
210
211 print(Q)
212
213 END_CREATEMODEL = time.time()
214
215 np.savetxt("../matrix/" + filenameQ, Q)                          # Store Q
216 variables_QUBO.to_csv("../variables/" + filenameVariablesQUBO)    # Save
        variables_QUBO in a CSV file
217
218 plt.figure(figsize=(10, 10))
219 plt.imshow(Q)
220
221 START_SOLVE = time.time()
222
223 out = {}
224 for i in np.arange(len(Q)):
225     for j in np.arange(i,len(Q)):
226         out[(i,j)] = Q[i,j]
227
228 response = QBSolv().sample_qubo(out)
229 output = list(response.samples())[0]
230 result = []
231
232 for i in np.arange(len(Q)):
233     result.append(float(output[i]))
234
235 END_SOLVE = time.time()
236
237 energy = response.to_pandas_dataframe()['energy'][0]
238
239 ## Output and checks ##
240 output = variables_QUBO.copy()
241 output['result'] = result
242 output = output.loc[(output['result']==1)]
243 print(output)       # Display output dataframe
244
245 # Check A
246 term_a_feas = output.groupby(['booking_id', 'n_passengers']).agg({'result': 'sum
        '}).reset_index()
```

```
247 print(term_a_feas)
248
249
250 # Check B
251 term_b_feas = output.groupby('seat_id').agg({'result': 'sum'}).reset_index()
252 print(term_b_feas)
253
254
255 # Check C - LEFT / RIGHT
256 count_infeasibilites_c = 0
257 by_column = output.groupby('column')              # Group by column
258 for col, frame in by_column:
259     frame = frame.sort_values(by=['row'])         # Order by row
260     prev_row = -1
261     prev_booking_id = ""
262     for index, entry in frame.iterrows():
263         curr_row = entry['row']
264         curr_booking_id = entry['booking_id']
265         if (curr_row == prev_row + 1) & (curr_booking_id != prev_booking_id):
266             count_infeasibilites_c += 1
267         prev_row = curr_row
268         prev_booking_id = curr_booking_id
269
270
271 # Check D - IN FRONT / BEHIND
272 count_infeasibilites_d = 0
273 by_row = output.groupby('row')                  # Group by row
274 for row, frame in by_row:
275     frame = frame.sort_values(by=['column'])    # Order by column
276     prev_col = -1
277     prev_booking_id = ""
278     for index, entry in frame.iterrows():
279         curr_col = entry['column']
280         curr_booking_id = entry['booking_id']
281         if (curr_col == prev_col + 1) & (curr_booking_id != prev_booking_id):
282             count_infeasibilites_d += 1
283         prev_col = curr_col
284         prev_booking_id = curr_booking_id
285
286
287 # Check E
288 # Create a matrix that will be used both for the term E value computation
289 # and the graphical representation of the wagon
290
291 # Define the dimensions of the matrix
292 max_row = max(seats['row'])
293 max_col = max(seats['column'])
294
295 # Create the max_row x max_col matrix
```

```
296 wagon = np.zeros(shape=(max_row, max_col))
297
298 # Fill the matrix
299 rows_and_cols_solution = output[['row','column', 'booking_id']]
300 for index, line in rows_and_cols_solution.iterrows():
301     r = line['row']
302     c = line['column']
303     b_id = line['booking_id']
304     id_num = b_id.replace("BOOKING", "")
305     wagon[r-1, c-1] = int(id_num)
306
307 # Print matrix
308 # for i in range(max_row):
309 #     for j in range(max_col):
310 #         print(wagon[i][j])
311
312 count_e = 0
313 for i in range(max_row):
314     for j in range(max_col):
315         count_e += check_booking_ids_of_neighbors(wagon, max_row, max_col, i, j,
    wagon[i][j])
316
317 # Feasibility check results
318 print('Energy: ' + str(energy))
319 df = bookings[~bookings['booking_id'].isin(term_a_feas['booking_id'])]
320 print('Number of infeasibilites due to term A: ' + str(len(df)))   # Nb. of
    unused booking IDs
321 print('Number of infeasibilites due to term B: ' + str(len(term_b_feas.loc[
    term_b_feas['result']>1])))
322 print('Number of infeasibilites due to term C: ' + str(count_infeasibilites_c))
323 print('Number of infeasibilites due to term D: ' + str(count_infeasibilites_d))
324 print('Number of passengers with a seat: {}'.format(str(int(term_a_feas['result
    '].values.sum()))))
325 print('Number of passengers with no seat: ' + str(len(term_a_feas.loc[term_a_feas
    ['result']!=term_a_feas['n_passengers']])))
326 print('Number of passengers with same booking ID assigned to nearest seats: ' +
    str(count_e))
327
328
329 ## Graphical representations of the wagon ##
330
331 # Free / Occupied seats
332 wagon_FO = np.zeros(shape=(max_row, max_col))
333 rows_and_cols_solution_FO = output[['row','column']]
334 for index, line in rows_and_cols_solution_FO.iterrows():
335     r = line['row']
336     c = line['column']
337     wagon_FO[r-1, c-1] = 1
338
```

```
339 plt.figure(figsize=(11, 4))
340 cmap = matplotlib.colors.ListedColormap(matplotlib.cm.get_cmap("Set1").colors
        [:3])
341
342 plt.imshow(wagon_FO, alpha=0.8, cmap=cmap.reversed())
343 plt.xticks(np.arange(max_col))
344 plt.yticks(np.arange(max_row))
345 plt.xlabel('Column')
346 plt.ylabel('Row')
347 plt.title('Seating arrangement in the current wagon')
348
349 legend_elements = [Line2D([0], [0], marker='o', color='w', label='Occupied seat',
350                           markerfacecolor=cmap.colors[0], markersize=15),
351                    Line2D([0], [0], marker='o', color='w', label='Free seat',
352                           markerfacecolor=cmap.colors[2], markersize=15)]
353 plt.legend(handles=legend_elements, bbox_to_anchor=(1.05, 1), loc='upper left')
354
355
356 # Occupied seats by BOOKING IDs
357 plt.figure(figsize=(10, 3))
358 cmap = matplotlib.colors.ListedColormap(matplotlib.cm.get_cmap("tab20").colors)
359
360 plt.pcolor(wagon, edgecolors='k', linewidths=2, cmap=cmap)
361 plt.gca().invert_yaxis()
362 plt.xticks(np.arange(max_col))
363 plt.yticks(np.arange(max_row))
364 plt.xlabel('Column')
365 plt.ylabel('Row')
366 plt.title('Seating arrangement in the current wagon')
367
368 legend_elements = [Line2D([0], [0], marker='o', color='w', label='Free seat',
369                           markerfacecolor=cmap.colors[0], markersize=15)]
370 plt.legend(handles=legend_elements,  bbox_to_anchor=(1.05, 1), loc='upper left')
371
372 plt.savefig("../plots/" + filenamePlotWagon, bbox_inches='tight')
373
374 END_PROGRAM = time.time()
375
376 ## Performance results ##
377 print("Number of variables: {}".format(len(variables_QUBO)))
378 print("Time for APPLY function: {} seconds".format(str(END_APPLY - START_APPLY)))
379 print("Time for CREATE MODEL: {} seconds".format(str(END_CREATEMODEL -
        START_CREATEMODEL)))
380 print("Time for QBSolv function: {} seconds".format(str(END_SOLVE - START_SOLVE))
        )
381 print("Total time to run the program: {} seconds".format(str(END_PROGRAM -
        START_PROGRAM)))
```

Bibliography

[1] Agarwal R., *"Make your Pandas apply functions faster using Parallel Processing"*, Towards Data Science, https://towardsdatascience.com/make-your-own-super-pandas-using-multiproc-1c04f41944a1

[2] Armstrong, A. & Meissner, J. *"Railway Revenue Management: Overview and Models (Operations Research)"*, Department of Management Science, Lancaster University, Working Papers (2010)

[3] Badeer P., *"The Easiest Way to Get Fake Data"*, Towards Data Science, https://towardsdatascience.com/dont-create-or-scrape-fake-data-53b02f16adfb

[4] Booth, M., & Reinhardt, S.P. *"Partitioning Optimization Problems for Hybrid Classical / Quantum Execution Technical Report"*, Semantic Scholar (2017)

[5] Boothby, K., Bunyk, P., Raymond, J., & Roy, A. *"Next-Generation Topology of D-Wave Quantum Processors"*, arXiv: Quantum Physics (2020), arXiv:2003.00133 [quant-ph]

[6] Bunyk, P., Hoskinson, E., Johnson, M.W., Tolkacheva, E., Altomare, F., Berkley, A., Harris, R., Hilton, J.P., Lanting, T., Przybysz, A., & Whittaker, J. *"Architectural Considerations in the Design of a Superconducting Quantum Annealing Processor"*, IEEE Transactions on Applied Superconductivity (2014), 24, 1-10, doi: 10.1109/TASC.2014.2318294

[7] D-Wave Hybrid Solvers, D-Wave Documentation, https://docs.ocean.dwavesys.com/en/stable/overview/hybrid.html#dwave-hybrid-solvers

[8] D-Wave Leap's Hybrid Solvers' Timing Information, D-Wave Documentation, https://docs.dwavesys.com/docs/latest/timing_hybrid.html#timing-hybrid

[9] D-Wave Ocean SDK Documentation, https://docs.ocean.dwavesys.com/en/stable/getting_started.html

[10] D-Wave QBSolv function, docs: https://docs.ocean.dwavesys.com/projects/qbsolv/en/latest/index.html, source code: https://github.com/dwavesystems/qbsolv

[11] D-Wave QPU Architecture: Topologies, D-Wave Documentation, https://docs.dwavesys.com/docs/latest/c_gs_4.html

[12] D-Wave Quantum Solvers, D-Wave Documentation, https://docs.ocean.dwavesys.com/en/stable/overview/qpu.html

[13] D-Wave Solver Computation Time, D-Wave Documentation, https://docs.dwavesys.com/docs/latest/doc_timing.html

[14] Feld S., Roch C., Gabor T., Seidel C., Neukart F., Galter I., Mauerer W., Linnhoff-Popien

C. *"A Hybrid Solution Method for the Capacitated Vehicle Routing Problem Using a Quantum Annealer"*, Frontiers in ICT (2019), vol.6, pag.13, doi:10.3389/fict.2019.00013

[15] Glover, F.W., & Kochenberger, G. *"A Tutorial on Formulating QUBO Models"*, ArXiv abs/1811.11538 (2018), arXiv:1811.11538 [cs.DS]

[16] Goh, S.T., Gopalakrishnan, S., Bo, J., & Lau, H.C. *"A Hybrid Framework Using a QUBO Solver For Permutation-Based Combinatorial Optimization"*, arXiv: Optimization and Control (2020), arXiv:2009.12767 [math.OC]

[17] Gurobi MIP starts, Gurobi Documentation, `https://www.gurobi.com/documentation/9.1/examples/mip_starts.html`

[18] Gurobi TimeLimit, Gurobi Documentation, `https://www.gurobi.com/documentation/9.1/refman/timelimit.html`

[19] Gurobi Tuplelist, Gurobi Documentation, `https://www.gurobi.com/documentation/9.1/refman/py_tuplelist.html`

[20] Kochenberger, G., Hao, JK., Glover, F. et al. *"The unconstrained binary quadratic programming problem: A survey"*, Journal of Combinatorial Optimization (2014), 28, doi:10.1007/s10878-014-9734-0

[21] Lewis, M.W., & Glover, F.W. *"Quadratic unconstrained binary optimization problem preprocessing: Theory and empirical analysis"*, Networks 70 (2017): 79-97, arXiv:1705.09844 [cs.AI]

[22] Linee guida trasporto pubblico e trasporto scolastico dedicato, Ministero delle infrastrutture e della mobilità sostenibili, `https://www.mit.gov.it/comunicazione/news/linee-guida-trasporto-pubblico-e-trasporto-scolastico-dedicato`

[23] Lucas A. *"Ising formulations of many NP problems"*, Frontiers in Physics (2014), vol.2, p.5, doi: 10.3389/fphy.2014.00005

[24] Marchenkova A., *"What's the difference between quantum annealing and universal gate quantum computers?"*, `https://medium.com/quantum-bits/what-s-the-difference-between-quantum-annealing-and-universal-gate-quantum-computers-c5e5099175a1`

[25] Nam, Y., Chen, J., Pisenti, N.C., Wright, K., Delaney, C., Maslov, D., Brown, K., Allen, S., Amini, J., Apisdorf, J., Beck, K., Blinov, A., Chaplin, V., Chmielewski, M., Collins, C., Debnath, S., Ducore, A.M., Hudek, K., Keesan, M., Kreikemeier, S., Mizrahi, J., Solomon, P., Williams, M., Wong-Campos, J.D., Monroe, C., & Kim, J. *"Ground-state energy estimation of the water molecule on a trapped-ion quantum computer"*, npj Quantum Information (2019), vol.6, pag.1-6

[26] Neukart F., Compostella G., Seidel C., von Dollen D., Yarkoni S., Parney B. *"Traffic Flow Optimization Using a Quantum Annealer"*, Frontiers in ICT (2017), vol.4, doi:10.3389/fict.2017.00029

[27] O'Gorman, B., Rieffel, E., Do, M.B., Venturelli, D., & Frank, J. *Compiling Planning into Quantum Optimization Problems: A Comparative Study*, Semantic Scholar (2015)

[28] Peng-Sheng Y. *"An efficient computational approach for railway booking problems"*, European Journal of Operational Research (2008), vol.185, pag.811-824, doi: https://doi.org/10.1016/j.ejor.2006.12.049

[29] QPU Topology, Ocean Documentation, `https://docs.ocean.dwavesys.com/en/stable/concepts/topology.html`

[30] Resch, Salonik and Ulya R. Karpuzcu. *"Quantum Computing: An Overview Across the System Stack"*, arXiv: Quantum Physics (2019), arXiv:1905.07240 [quant-ph]

[31] Schuld, M., Sinayskiy, I., & Petruccione, F. *"An introduction to quantum machine learning"*, Contemporary Physics (2014), vol.56, pag.172 - 185

[32] Shor, P. *"Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer"*, SIAM J. Comput. (1999), vol.26, pag.1484-1509

[33] Three Truths and the Advent of Hybrid Quantum Computing, `https://medium.com/d-wave/three-truths-and-the-advent-of-hybrid-quantum-computing-1941ba46ff8c`

[34] Travelling by train during Phase 2 of the COVID-19 health emergency, Ferrovie dello Stato Italiane, `https://fsitaliane.it/content/fsitaliane/en/media/news/2020/5/4/travelling-by-train-during-phase-2-of-the-covid-19-health-emerge.html`

[35] Venegas-Andraca, S. E., William Cruz-Santos, C. McGeoch and M. Lanzagorta. *"A cross-disciplinary introduction to quantum annealing-based algorithms"*, Contemporary Physics 59 (2018): 174 - 197, arXiv:1803.03372 [quant-ph]

[36] Venturelli, D., Marchand, D.J., & Rojo, G. *"Quantum Annealing Implementation of Job-Shop Scheduling"*, arXiv: Quantum Physics (2015), arXiv:1506.08479 [quant-ph]

[37] Welcome to D-Wave, D-Wave Documentation, `https://docs.dwavesys.com/docs/latest/c_gs_1.html#network-gs`

[38] Yuan, W., & Nie, L. *"Optimization of seat allocation with fixed prices: An application of railway revenue management in China"*, PloS one (2020), 15(4), e0231706. https://doi.org/10.1371/journal.pone.0231706