TÉCNICO LISBOA

Flight Time and Cost Minimization in Complex Routes

Rafael Alexandre da Silva Marques

Thesis to obtain the Master of Science Degree in

Masters in Aerospace Engineering

Supervisors: Prof. Nuno Roma
Prof. Prof. Luis Russo

Examination Committee

Chairperson: Prof. Name of the Chairperson
Supervisor: Prof. Nuno Roma
Members of the Committee: Prof. Name of First Committee Member
Dr. Name of Second Committee Member
Eng. Name of Third Committee Member

October 2018

Acknowledgments

I would like to thank my parents for their friendship, encouragement and caring over all these years, for always being there for me through thick and thin and without whom this project would not be possible. I would also like to thank my grandparents, aunts, uncles and cousins for their understanding and support throughout all these years.

Quisque facilisis erat a dui. Nam malesuada ornare dolor. Cras gravida, diam sit amet rhoncus ornare, erat elit consectetuer erat, id egestas pede nibh eget odio. Proin tincidunt, velit vel porta elementum, magna diam molestie sapien, non aliquet massa pede eu diam. Aliquam iaculis.

Fusce et ipsum et nulla tristique facilisis. Donec eget sem sit amet ligula viverra gravida. Etiam vehicula urna vel turpis. Suspendisse sagittis ante a urna. Morbi a est quis orci consequat rutrum. Nullam egestas feugiat felis. Integer adipiscing semper ligula. Nunc molestie, nisl sit amet cursus convallis, sapien lectus pretium metus, vitae pretium enim wisi id lectus.

Donec vestibulum. Etiam vel nibh. Nulla facilisi. Mauris pharetra. Donec augue. Fusce ultrices, neque id dignissim ultrices, tellus mauris dictum elit, vel lacinia enim metus eu nunc.

I would also like to acknowledge my dissertation supervisors Prof. Some Name and Prof. Some Other Name for their insight, support and sharing of knowledge that has made this Thesis possible.

Last but not least, to all my friends and colleagues that helped me grow as a person and were always there for me during the good and bad times in my life. Thank you.

To each and every one of you – Thank you.

Abstract

Nulla facilisi. In vel sem. Morbi id urna in diam dignissim feugiat. Proin molestie tortor eu velit. Aliquam erat volutpat. Nullam ultrices, diam tempus vulputate egestas, eros pede varius leo, sed imperdiet lectus est ornare odio. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Proin consectetuer velit in dui. Phasellus wisi purus, interdum vitae, rutrum accumsan, viverra in, velit. Sed enim risus, congue non, tristique in, commodo eu, metus. Aenean tortor mi, imperdiet id, gravida eu, posuere eu, felis. Mauris sollicitudin, turpis in hendrerit sodales, lectus ipsum pellentesque ligula, sit amet scelerisque urna nibh ut arcu. Aliquam in lacus. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Nulla placerat aliquam wisi. Mauris viverra odio. Quisque fermentum pulvinar odio. Proin posuere est vitae ligula. Etiam euismod. Cras a eros.

Keywords

Maecenas tempus dictum libero; Donec non tortor in arcu mollis feugiat;Cras rutrum pulvinar tellus.

Resumo

Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Vestibulum tortor quam, feugiat vitae, ultricies eget, tempor sit amet, ante. Donec eu libero sit amet quam egestas semper. Aenean ultricies mi vitae est. Mauris placerat eleifend leo. Quisque sit amet est et sapien ullamcorper pharetra. Vestibulum erat wisi, condimentum sed, commodo vitae, ornare sit amet, wisi. Aenean fermentum, elit eget tincidunt condimentum, eros ipsum rutrum orci, sagittis tempus lacus enim ac dui. Donec non enim in turpis pulvinar facilisis. Ut felis. Aliquam aliquet, est a ullamcorper condimentum, tellus nulla fringilla elit, a iaculis nulla turpis sed wisi. Fusce volutpat. Etiam sodales ante id nunc. Proin ornare dignissim lacus. Nunc porttitor nunc a sem. Sed sollicitudin velit eu magna. Aliquam erat volutpat. Vivamus ornare est non wisi. Proin vel quam. Vivamus egestas. Nunc tempor diam vehicula mauris. Nullam sapien eros, facilisis vel, eleifend non, auctor dapibus, pede.

Palavras Chave

Colaborativo; Codificaçãoo; Conteúdo Multimédia; Comunicação;

Contents

List of Figures

List of Tables

List of Algorithms

Listings

Acronyms

1

Introduction

Contents

1.1 Motivation

Online Traveling Agencies (OTA) are online applications which sell traveling goods, as, for example, commercial flight tickets. Although consumers retain the option to buy flights directly from airline companies, the majority chooses to use OTA. The main reason for this is that these agencies aggregate flight data from multiple airlines, instead of being limited to a single one, which ultimately increases the options of the consumer. Furthermore, many OTA work as meta-search engines, searching over a variety of websites in order to find the best flights which satisfy the consumer requirements. While OTA usually provide very complete search functionalities for simple flights, the majority fails to offer the same search options for a trip composed of multiple cities.

Consider a trip starting and ending at any given city, which must visit every city specified in a particular list of cities. If there are no constraints associated with the order in which these cities must be visited, this problem is known, in the scientific community, as the Traveling Salesman Problem (TSP). This problem is considered very difficult to solve, since the total number of possible solutions increases in an exponential way, as the number of cities increases.

If commercial flights are the means of transportation between every pair of cities, this problem can no longer be considered the *classic* Traveling Salesman, but rather its *time-dependent* counterpart. This is because some of the major flight characteristics, as its price and duration, cannot be considered constant over time. Rather, they are dependent on the particular flight selected, and the characteristics of that flight may follow no apparent logical rule, at least from the consumers perspective.

Consequently, finding the most efficient set of flights, from the consumer point of view, is a very repetitive and time-consuming task. Faced with this problem, only the most persistent consumer will be able to find the best solution for the problem, and this can only occur for a very small list of cities. As the number of cities increases, even the most persistent consumer cannot verify all possible solutions. This means that, ultimately, the final consumer will pay more than necessary for the requested service.

This work also arises as a response to the public contest called *Traveling Salesman Challenge* [**?**], issued by *Kiwi*, a well established OTA, even though the beginning of this work dates prior to the issue of the challenge. In this challenge, Kiwi recognizes that, in most cases, users do not care about the order in which they visit a given list of cities, and that there exists a market niche interested in this type of services. Kiwi also recognizes that most OTA do not offer these type of services due to the computational complexity associated with the problem.

This work intends to address the problem of solving multi-city trips, by studying it, and by developing the necessary technologies to effectively solve the problem in a time-efficient manner. It also wishes to develop a proof of concept online flight search application, implementing these technologies in order to, ultimately, provide high-quality search for multi-city trips.

1.2 Existing Services

The tourism industry exists since, at least, the 19th century, but it was impacted by some significant chapters of human technology, which lead to an increased and sustained growth of the market size. First, during the 1920's, the development of commercial aviation meant a significant impact on the industry, shifting the transportation focus to the airplane. Much later, during the 90's, the establishment of the internet led to some changes in the market because airlines were able to sell directly to the passengers [**?**]. More recently, the widespread use of mobile phones and lead to new increases in the market size. In 2016, the direct contribution of the tourism industry for the GDP was over 2.3 trillion dollars, while the total contribution was over 7.6 trillion [**?**].

The increase in the market size of the tourism sector is sustained by traveling agencies, whose main function is to serve as an agent, advertising and selling products and services on behalf of other service providers. These services usually include, but are not limited to, transportation, accommodation, insurance, tours and other tourism associated products. There are both online and offline travel agencies, and the focus of this chapter will be given to the online traveling agencies.

Online Traveling Agencies are websites and applications which offer traveling services or reviews, as illustrated in figure **??**. Most OTA function as a metasearch engine, which means that they search multiple independent travel services providers. Furthermore, this is the main differentiator from the search engines provided by OTA's, and those from direct travel suppliers, as are the websites of individual airlines. Direct travel suppliers are limited to display the results of their own services. On the other hand, OTA usually do not own any travel services but serve solely as an intermediary between traveler and travel services provider. In some cases, these metasearch engines may resort to web scraping in order to get real-time data on the flights provided by different airlines. Recent reports show that while OTA are increasing its market share, direct travel suppliers still account for 57% of the total online travel consumption [**?**].

In order to better understand the difference between OTA's and direct travel suppliers, consider the case of a user searching for a simple round flight between two cities. If this user visits an individual airline website, the flight results presented are limited to those offered by this airline. However, there is no guarantee that the airline flies the user defined route. Furthermore, even if there is such a route, it is possible that this route is not flown every day. Note that these two types of problems are very common, especially in low-cost airlines. In contrast, the same round flight search on a metasearch engine of some OTA will produce a variety of results which include several different airlines. Finally, since OTA aggregate data from different airlines and other meta searches, it is less likely for the two problems described above to occur using the OTA search tools. In conclusion, collecting data from multiple sources usually results in a higher variety and quality of results.

Despite the variety of services provided by OTA, this section will focus only on making a review of

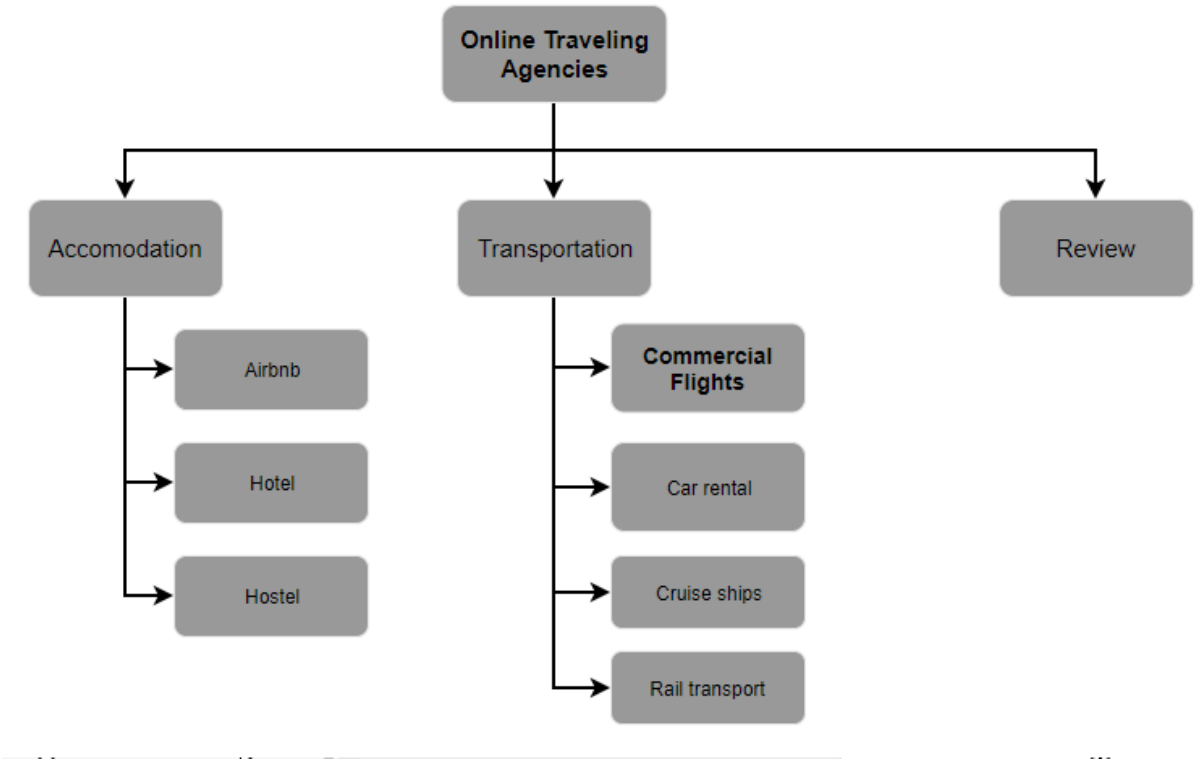

Figure 1.1: The different services provided by Online Traveling Agencies

the existing search tools available for the commercial flight transportation segment. This analysis will be made from both the user and the developer point of view because the offered services often vary according to this.

1.2.1 User Search Tools

Depending on the specific application, online traveling agencies may offer services which include transportation and accommodation. The focus of this section is to create an extensive overview of the search tools available for finding commercial flights. In order to classify the search tools, the results will be grouped according to the *trip type*:

- single/round flight;
- multi-city trip;

This overview will also focus on the different search utilities provided by the search applications. Thus, it will analyze the ability to:

- search over a range of start dates - **DR**;
- present a price overview as a function of time - **PO**;
- present different results for different criteria - **VR**;
- display an interactive map - **Map**;

There are other search utilities, presented by some online search applications, which will not be subject of study. These include:

- price monitoring and alerting service;
- analysis of the probability of price fluctuations;
- information about alternative flights;
- automatic caching for future usage;

This analysis will be considered for the major flight search applications, which include, for example, Google Flights, Momondo, Skyscanner, Kayak, and Kiwi.

Single/Round flights

In order to evaluate the search utilities provided by every flight search application, as well as the actual price displayed, the same search will be performed over all applications. In particular, this search was executed over a month before the actual start date, and corresponds to:

- **S** - single flight - from Lisbon, to Amsterdam, at 8/6/2018;
- **R** - round flight - from Lisbon, to Amsterdam, at 8/6/2018, for a duration of 3 days;

Table **??** displays the results of the search performed over the different applications, together with a checklist specifying which search utilities are provided. It is also worth noting that the price displayed in this table might not correspond to the actual final price. For example, Edreams is well known for charging extra fees only upon the ticket purchase.

Table 1.1: Comparison of the search results, and search utilities, of different online flight search applications, for single and round flights.

company	DR	PO	VR	Map	S	R
Google Flights	✓	✓		✓	65	127
Expedia					81	167
Booking.com / Kayak			✓	✓	67	126
Momondo		✓	✓		62	102
TripAdvisor					56	124
cheapflights.com					75	150
Adioso	✓		✓		69	137
kiwi	✓	✓	✓	✓	76	224
SkyScanner	✓	✓			65	127
Edreams	✓		✓		56	116

From the analysis of table **??**, it is possible to conclude that, for single and round flights:

- different applications provide significantly different results. Single and round flight prices vary up to 44% and 120% respectively;
- half of the applications do not provide date range support;
- half of the applications do not provide price overview as a function of time;
- most applications do not present an interactive map;

Multi-city trips

Following the same methodology as in the previous section, the same multi-city trip search will be performed on the different applications, and the result is presented as **M** in table **??**. This search corresponds to:

- single flight from Lisbon to Amsterdam, at 8/6/2018, for a duration of 3 days;
- single flight from Amsterdam to Berlin, at 11/6/2018, for a duration of 3 days;
- single flight from Berlin to Lisbon, at 14/6/2018;

It is worth noting that all flight search applications which currently perform multi-city trips, do it in the exact same way: a multi-city trip corresponds to a collection of single flights, which must specify the origin and destination, together with a single start date.

This case corresponds to a very inefficient search for a variety of reasons. First of all, it constrains the search to one particular route. Second, it does not allow any kind of flexibility around the start date or duration of stay associated to each city. And finally, changing one parameter of the search might affect the rest of the trip.

Once again, the results of the search, together with the search utilities, are presented in table **??**. Note that MTCS stands for the ability to search for multi-city trips, as those specified above.

Table 1.2: Comparison of the search results, and search utilities, of different online flight search applications, for multi-city trips.

company	MTCS	DR	PO	VR	Map	M
Google Flights	✓					184
Expedia	✓					420
Booking.com / Kayak	✓					191
Momondo	✓			✓		178
TripAdvisor	✓			✓		238
cheapflights.com						
Adioso						
Kiwi	✓		✓			226
SkyScanner	✓			✓		238
Edreams	✓					333

From the analysis of table **??**, it is possible to conclude that, for multi-city trips:

- different applications provide significantly different results, which vary up to 135%;
- most applications provide (limited) multi-city trip search;
- most applications do not present different results for different search criteria;
- no application offers the ability to perform multi-city trips on a range of start dates;
- no applications displays an interactive map;

1.2.2 Developer Search Tools

For a developer that intends to create a flight meta-search engine, a way to access flight data is essential. There are some online traveling agencies which extend their API for public consultation. Amongst the

services that these public API's offer, are the possibility of searching for cached, and sometimes, real-time flight data. In some cases, these API's extend their range of services and include endpoints for the consultation of hotel information, car-rental, cruise ships and even railroad data [**?**, **?**].

Usually, these type of content provider benefits from sharing their API, because it generally increases their traffic and the number of potential clients and bookings. In most cases, metasearch engines do not handle booking directly, but redirect to the original content provider. In this case, the content provider retains the entirety of the booking fees. In the case where the meta-search engine does manage the booking, the content provider retains the majority of the booking fee's, while the second party may retain some share of these revenues [**?**].

Amongst the companies which extend their API for third party are Google, Skyscanner, Expedia and Kiwi. These API usually operate as a *free*, *limited* or *private* service. For example, while Expedia and Kiwi offer free publicly accessible API's, Skyscanner has a *private companies only* [**?**] policy, denying any free public consultation for educational purposes. In its turn, Google has a free, but limited API, in the sense that only 50 free daily queries are available, and anything beyond that has an associated cost [**?**].

Due to the educational purpose of this work, the interest in a flight API is limited to those which are free. Considering the different API's that were consulated and analyzed, Kiwi's free public API stroke as the most useful, efficient and fast. The major search utilities that this API offers are [**?**]:

- access to relevant information about cities, airport data and airline companies;
- access to flight information, with flexible queries;
- the possibility of specifying multiple start dates;
- the possibility of specifying a variable duration associated to a return flight;
- flexibility on the specification of the origin and destination;
- possibility to produce multi-flight queries, by grouping up to a total of 9 simple flight queries.

Note that although Kiwi, and other companies, enable the possibility of quering for multiple flights at once, each flight must specify a particular pair of cities and date. Thus, this is a constrained multi-city search, and does not actually give any information about the best route, schedule or set of flights for a multi-city trip.

1.3 Objectives

This work intends to study the time-dependent traveling salesman problem and develop an algorithm to present high quality solutions in a short period of time. This algorithm is to be used in the development of

an online flight search application capable of finding the best flights for a trip composed of several cities, with no particular routing constraints. The concretization of this search engine requires a user interface to collect trip requests, and both access to real time flight data, and to an optimization algorithm capable of solving the TDTSP instance.

Furthermore, a secondary objective is to develop a search engine capable of solving multi-objective tsp instances. This is believed to be important, because many users care not only about the flight price, but other attributes, as the flight duration.

Finally, the search options of the developed search engine should include the possibility to constrain one particular city to a specific time window. The objective of this is to study the possibility of using the developed algorithm to solve constrained problems, as is the TDTSP with time windows.

1.4 Document structure

Here we describe the document structure

2

Literature review

Contents

This chapter is a review on the literature around Combinatorial Optimization, Traveling Salesman Problem and its time-dependent variation, and the state of the art of the algorithms which address these problems.

2.1 Combinatorial Optimization and Routing Problems

2.1.1 Combinatorial Optimization and Computational Complexity

Lawlers, (1976), [**?**], defined combinatorial analysis as "the mathematical study of the arrangement, grouping, ordering, or selection of discrete objects, usually finite in number". Schrijver (2002), [**?**], following this definition of Lawler, improves it with the important concept of optimal solution, "Combinatorial optimization searches for an optimum object in a finite collection of objects.". This definition is followed by a remark, stating that "typically, the collection (...) grows exponentially in the size of the representation", and concluding that "scanning all objects one by one and selecting the best is not an option".

Following a more consice definition [**?**], a combinatorial optimization problem can be defined as follows:

Definition 1) A combinatorial optimization model $P = (S, \Omega, f)$ consists of:

1. a search space S, defined by a finite set of decision variables, each with a domain;
2. a set Ω of constraints amongst the decision variables;
3. an objective function f: $S \rightarrow \mathbb{R}_0^+$, to be minimized.

The search space is defined by a set of decision variables $X_i, i = (1, ..., n)$, each associated to domain D_i, which specifies the possible value of each decision variable. An instantiation of a variable is an assigment of a value $v_i^j \in D_i$ to a variable X_i. This leads to the definion of a feasible solution $s \in S$, which corresponds to the assignment of a value to each decision variable, according to its domain, in such a way that all constraints in Ω are satisfied. Finally, the objective of the problem is to find a global minimum of P is a solution $s^* \in S$, such that $f(s^*) < f(s) \forall s \in S$, and the set of all set of all global minima is denoted by $S^* \subseteq S$.

When working on a combinatorial optimization problems, it is usefull to have an idea of how difficult the problem is. This characterization is provided by a field called computation complexity. A combinatorial optimization problem Π is said to have worst-case time complexicity $O(g(n))$, if the best algorithm for solving Π finds an optimal solution to any instance of size n of Π, in a computation time upper bounded by $g(n)$.

A problem Π is said to be solvable in polynomial time if the maximum amount of computing time necessary to solve any instance of size n is bounded by a polynomial in n. If k is the largest exponent of such a polynomial, then the combinatorial optimization problem is said to be solvable in $\mathcal{O}(n^k)$ time.

A *polynomial time algorithm* is characterized by a computation time bounded by $\mathcal{O}(p(n))$, for some polynomial function p, where n is the size of the problem instance. If k is the largest exponent of such a polynomial, the problem is said to be solvable in $\mathcal{O}(n^k)$. On the contraty, any algorithm whose computational time can not be bounded by a polynomial function is refered to as an *exponential time algorithm*. Any problem that can be solved in polynomial time is said to be *tractable*, while problems that are not solvable in polynomial time are called *untractable*.

In the field of computational complexity, there is an important concept called *polynomial time reductions*, which transform a problem into another problem, in polynomial time. If the latter problem is solvable in polynomial time, so is the first one. The class of problems which is solvable in polynomial time is called P. On the other hand, there is a class of problems called NP, which stands for *non-deterministic polynomial acceptable problems*, for which given a solution can be *verified* in polynomial time. The class of NP-complete refers to the most difficult problems in NP.

It is worth mentioning that there exists another class, called NP-hard, for which each problem is as hard as the hardest NP-complete problem. More precisely, a problem H is NP-hard when every problem in NP can be reduced to H using a polynomial time transformation. This definition of the class NP-hard leads to the logical conclusion that finding a polynomial time algorithm for *any* problem in NP-hard, would imply the resolution of *all* NP-complete problems in polynomial time. However, up untill now, no polynomial time algorithm was found for any NP-hard problem. Note that the class NP-hard does not necessary belong to the class NP, but this is used following the name convention.

There exists an infamous discussion amongst the cientific community regarding the question "$P = NP?$", since it is one of the major unsolved problems in computer science. We present image (**??**) illustrating the classes according to both possible solutions to the aforementioned question.

2.1.2 The Traveling Salesman Problem

Given a list of cities and the distances between them, the Traveling Salesman is the combinatorial optimization problem of finding a minimum length route which connects every city. By this original definition, proposed by [**?**], the focus of the TSP is to perform optimization on routing problems, as the school bus problem, studied by Merrill Flood in 1942, [**?**], minimizing the total distance of a tour. Some variations of the original formulation allow the adaptation of the problem to suit different optimization goals, [**?**]. For example, instead of distance, the focus may be the minimization of the total cost, travel time, or some other attribute associated to the problem under consideration. It is also possible to search for a route which minimizes two, or more, objective functions at once, [**?**]. In some routing problems, the tour under consideration must satisfy some constraints, [**?**]. Most often, these constraints refer to scheduling conflicts which must be satisfied, [**?**]. A practical application of this is the resolution of routing problems with time windows, [**?**].

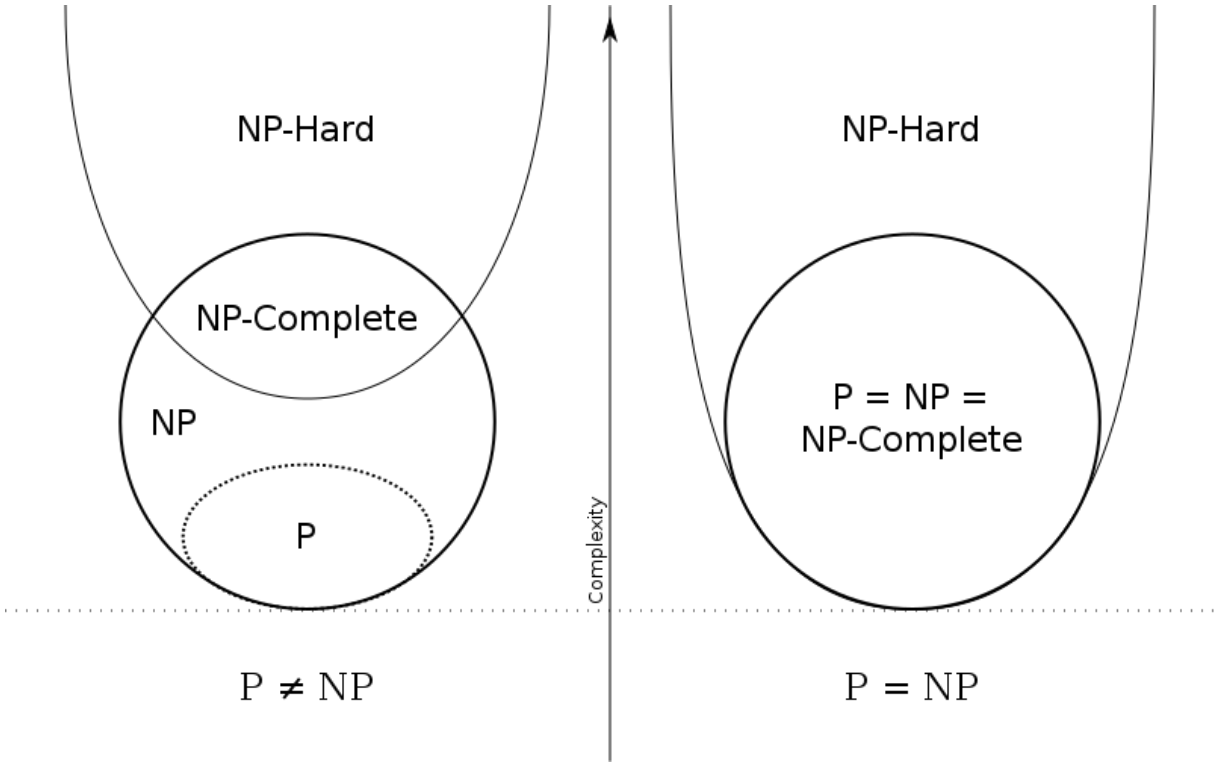

Figure 2.1: Illustration of the classes P, NP, NP-complete and NP-hard

In the field of Graph theory, the TSP is the problem of finding a minimum cost Hamiltonian cycle over a complete, undirected, weighted graph, [**?**]. The problem of finding a minimum cost Hamiltonian cycle was shown to be NP-complete. This implies the NP-hardness of the TSP. [**?**]. In several graph problems, considering a symmetric cost between two points is not suitable. This is known as the asymmetric TSP, and it considers a directed graph instead, [**?**].

In the flight industry, the Traveling Salesman has vast applications. It was applied to airport scheduling, [**?**]. More recently, the TSP and its time dependent variation have been focus of attention in fields related to Unmanned Aerial Vehicle routing, ([**?**], [**?**]). There is also an online website which introduces the Air Traveling Salesman, which introduces the problem of finding layover airports when no direct route is available, [**?**].

In some cases, the classic formulation of the Traveling Salesman does not adequately describe the characteristics of the problem under consideration. To overcome this, different problem formulations are considered. An example of this is the time dependent TSP, [**?**]. In this formulation, the cost of each arc is not constant, but varies as a function of time. In general, this problem is harder to solve, [**?**]. There are several other combinatorial optimization problems which benefit from considering a time dependent approach, [**?**]. The Vehicle Routing Problem is a field which particularly focus on this problem, due to the characteristics of street traffic, [**?**]. It is worth mentioning that the Traveling Salesman Problem is a

special case of the Vehicle Routing Problem, in which the fleet is composed by only one vehicle, the salesman, [**?**]. Because of this, works related to the VRP are also interesting for the resolution of the TSP.

Most people are faced with similar routing problems every day. Consider the problem of walking or driving from point A to point B. This is a graph problem, in which the arcs are the streets, the nodes the streets intersections, [**?**]. In its turn, the weights refer to the distance or travel time, which in its turn may be affected by other parameter, as traffic, [**?**]. If the person is familiar with the graph, they are capable of finding a good route mentally, very fast, [**?**]. If the undertaken route is to visit a set of points exactly once, before returning to the original starting point, this is known as the Traveling Salesman Problem.

2.1.2.A Problem definition

The Traveling Salesman occurs both as combinatorial optimization and as a Graph Problem. In either case, the TSP is defined by a graph $G = (N, A)$, where N is the set of nodes, and A is the set of arcs connecting those nodes. The set of nodes is of size $n = |N|$, while the size of the set of arcs is $m = |N|^2$. Each arc, $a_{ij}, i, j \in N, i \neq j$ has an associated weight, c_{ij}, which represents, for example, the distance between cities. The set of arcs is fully connected, that is, each node is capable of reaching any other node directly, without visiting a third node. When two nodes can not be connected by an arc, the cost of that node is considered as very high. In the classical TSP formulation, the graph is undirected, which implies symmetry in the costs of the arcs, that is: $c_{ij} = c_{ji} \forall i, j \in N$. Because of the characteristics of this TSP formulation, the graph is said to be connected, weighted and undirected.

The objective of the TSP is to find the minimum cost Hamiltonian cycle, that is a path which visits each node exactly once, and returns to the initial node, closing the path. A generic solution to the TSP is any permutation σ over the set of nodes, N. σ is also a set, where $\sigma_i, i \in len(\sigma)$, represents the node in the i'th index of the cycle. The cost of a cycle is given by the sum of the weights of each arc by which it is composed, that is $C(\sigma) = \sum_{i=1}^{n} c_{\sigma_i \sigma_{i+1}}$, where $\sigma_{n+1} = \sigma_1$

The TSP can solve diferent types of problems by optimising different parameters. In the classic formulation, the weight between of an arc connecting two nodes represent the distance between two cities. However, the weight of an arc can represent different things, particularly, travel time or travel cost. Althoug changing the parameter may lead the TSP formulation intact, in some cases, it changes the problem. This occurs for example when considering that costs are time-dependent and this will be approached in section **??**.

2.1.2.B Common TSP variations

The majority of the problems which are variations of the classic TSP, have the problem structure altered by some differences concerning the characteristics of the arcs or arcs costs, as occurs with the assy-

metric and metric TSP. In other cases, the variation of the problem may refer to constraints amongst the variables as occurs in the TSP with time windows described in the following subsection, or in the objective of the optimization, as occurs with the bottleneck TSP. The definitions provided in this section are with respect to those provided by [**?**].

Assymetric TSP

In the assymetrical TSP, the cost matrix is not symmetric. That is, there is no constraint imposing that $c_{ij} = c_{ji}$, as happens with the classical TSP.

The ATSP may be more adequate than the TSP for some specific real world problems. For example, when considering a routing problem over a city, some roads may not be connected in both ways. In this case, the weight of an arc connecting two points is different, depending on the direction of traversal of the arc.

Metric TSP

The metric TSP is a special case of the TSP, in which the arcs cost, in addition to being symmetric, also respect the triangle inequality. That is, $c_{ij} \leq c_{ik} + c_{kj}, \forall\, i, j, k \in N$.

Euclidean TSP

In the Euclidean TSP, the set of nodes is placed in a d-dimensional space, and the weight of each arc is given by the euclidean distance. This distances is calculated based on equation **??**, for two points $x = (x_1, x_2, ..., x_d)$ and $y = (y_1, y_2, ..., y_d)$.

$$\left(\sum_{i=1}^{d} (x_i - y_i)^2 \right)^{1/2} \tag{2.1}$$

The euclidean TSP is a variation which is both symmetric and metric.

Bottleneck TSP

In the Bottleneck TSP, the objective is to find a valid route which minimizes the cost of the highest cost arc of the tour. According to the characteristic of the graph, the Bottleneck TSP may either be symetric, assymetric, metric or time-dependent.

The Messenger Problem

The Messenger problem, also known as the wondering traveling salesman, is the problem of finding a minimum cost hamiltonian path connecting edges u and v of the graph G. It can be seen as a Traveling Salesman Problem in which the tour is not closed, but ends on a specific node, different from the initial one. The Messenger problem can be transformed into the TSP, by considering a cost of $-M$ for the arc (v, u), where M is a large number. If the nodes u and v are not specified, and one wishes to find a minimum cost hamiltonian path in G, this can be achieved by a graph transformation, adding one node and connecting it to all other nodes by arcs of cost $-M$. The optimal solution to the TSP on this modified graph can be used to produce the optimal solution to the original problem.

Generalized TSP

In the Generalized TSP, the set of nodes is partitioned into k clusters $V_1, V_2, ..., V_k$, and the objective is to find a shortest cycle which passes through exactly one node from each cluster V_i for $1 \leq i \leq k$. If the dimension of each cluster is 1, that is, if $|V_i| = 1$ for all i, the problem reduces to the TSP. There exists effective graph transformation techniques which reduce the GTSP into the TSP. The GTSP has interesting applications to the tourism industry. For example, a person may want make a world trip and visit one city in each continent. In this case, the problem can be stated as a GTSP instance, in which the clusters are the continents.

The *m*-salesmen problem

In the *m*-salesmen problem, there are m salesmen positioned in node 1 of G. Each salesman visits a subset X_i of nodes of G exactly once, starting and returning to node 1. The objective is to find a partition X_1, X_2, ..., X_m of $V - \{1\}$, and a route for each salesman such that:

1. $|X_i| \geq 1$ for each i;
2. $\cup_{i=1}^{m} X_i = V - 1$;
3. $X_i \cap X_j = \emptyset$;
4. the total distance travelled by all salesman is minimized.

2.1.2.C Time Dependent TSP

The time-dependent traveling salesman problem is a generalization of the TSP where arc costs depend on their position in the tour, [**?**, **?**]. This section first introduces the TDTSP as a graph problem, followed by the definition as a sequencing problem.

TDTSP as a graph problem

Let $N = 1, 2, ..., n$ and let $N_0 = N \cup \{0\}$. The TDTSP on a complete graph $K(N_0)$ can be modeled as an optimization problem over a layered graph (V, A). V is the set composed by the source node 0, the termination node T, and intermediate nodes (i, t) for $i, t \in N$. In this representation of the intermediate nodes, the first index of (i, t) identifies the vertex i of the graph $K(N)$, and the second index represents the position of vertex i in a path between nodes 0 and T. In its turn, A is the set of arcs connecting each node. This set is composed of initiation, intermediate, and termination arcs. For $i \in N$, $(0, i, 0)$ denotes an initiation arc from node 0 to node $(i, 1)$, and (i, T, n) denotes a termination arc from node (i, n) to node T. Given $i, j \in N$ such that $i \neq j$, and $1 \leq t \leq n - 1$, (i, j, t) denotes an intermediate arc from node (i, n) to node $(j, t + 1)$. The third index of an arc represents its layer, that is, the position in which it occurs in the path, or in other words, the time of the arc traversal, if we consider 1 time unit for each of these traversals.

When working on the TDTSP, it is often convenient to define $G(n)$ as a subgraph of (V, A), induced by $V \backslash \{0, T\}$. This way, $G(n)$ has n^2 nodes $(i, t) : i, t \in N$ and all the $n(n - 1)^2$ intermediate arcs of A.

A path with n vertices in $G(n)$ is of the form $(v_t.t) : v_t \in N, 1 \leq t \leq n$. Since consecutives nodes are in consecutive layers, the path can be described by an ordered array $(v_t : t \in N)$. This path can be extended to a $0 - T$ path of (V, A) by appending node 0 and T to the begining and end of the tour, respectively.

The classical TSP and its time-dependent variation share the same objective, which is to find the minimum cost hamiltionan cycle over graph (V, A). Another property they share is the possibility of working over a symetric or assymetric problem.

TDTSP as one-machine sequencing problem

In operation scheduling, the time dependent problem TSP can also be stated as a one-machine sequencing problem, [**?**].

Consider a set of n jobs, J_1, ..., J_n, to be executed on a single machine. Each job has a setup cost $C^t_{i,j}$, ocurring when job J_i, processed in the t-th time unit, is followed by job J_j, processed in the (t+1)-th time unit. Consider that each job completion takes exactly one time unit. The machine is in some given initial state, denoted by 0, before the job processing beings. As happens with the classical TSP, the machine has to be returned to its original state, after the job processing ends. The problem is to find a sequence, $J_{w(1)}$, ..., $J_{w(n)}$, that minimized the total set-up cost C(w), defined by:

$$C(w) = C^0_{0,w(1)} + \sum_{i=1}^{n-1} C^i_{w(i),w(i+1)} + C^n_{w(n),0} \tag{2.2}$$

It is important to note some important characteristics of this formulation. First, problems with unspecified initial/final state can be formulated in the same way using 0 as the initiation/termination cost, that is $C^0_{0,w(1)} = 0$ and $C^n_{0,w(1)} = 0$. Secondly, the overwriting of the above formulation reduces the defined problem into the classical TSP and the classical Assignment Problem. The first case is achieved by considering that setup costs are not time-dependent, that is, $C^t_{i,j} = C_{i,j}$. The latter is accomplished by considering that setup costs are not dependent on the second/first job, that is, $C^t_{i,j} = C^t_i$.

2.1.2.D TSP with time-windows

The Traveling Salesman Problem with Time Windows (TSPTW) is a generalization of the TSP, in which the objective is to find a minimum cost hamiltionian cycle which visits every city in its requested time window. This problem has important applications in the field of routing and scheduling, and is particularly relevant for the Vehicle Routing Problem, as it may occur as a real world constraint imposed by customers, whose operation hours are limited to a time window, [**?**]. Being a generalization of the TSP, the TSPTW is also NP-complete, [**?**]. This section introduce two definitions, one for the asymetric TSP with time-windows, and a second one for the time-dependent variation of this problem.

Below we present a formal definition of the assymetric traveling salesman problem with time windows

(ATSPTW), based on [**?**]. Consider a complete digraph $G = (V, A)$, where V is the set of $n = |N|$ nodes, and A the set of arcs, each associated with a nonnegative arc cost, $c_{i,j}$, and nonnegative *setup times*, t_{ij} associated to each arc $(i, j) \in A$. The nodes correspond to jobs to be processed (as described in the single-machine sequencing problem), and arcs correspond to job transitions, where the setup times, t_{ij}, defines the changeover time needed to proccess node j right after node i. Each node $i \in A$ has an associated *processing time* $p_i \geq 0$, a *release date* $r_i \geq 0$ and a *deadline* $d_i \geq 0$, where the release date and deadline denote, respectively, the earliest and latest possible starting time for the processing of node i. The *minimal time delay* for processing node j immediatly after node i is given by $v_{ij} = p_i + t_{ij}$. The interval $[r_i, d_i]$ is called the *time window* for node i. The time-window is said to be relaxed if $r_i = 0$ and $d_i \to +\infty$. On the contrary, a time-window is called active if $r_i > 0$ and $d_i < +\infty$. It is possible to reach a node $i \in A$ at a time $t \in \mathcal{Z}^+ \cup \{0\}$, sooner than its release date a_i. In this case, it will be undergo a waiting time $a_i - t$, before leaving node i at time a_i.

When dealing with routing problems with time-windows, it is often necessary to define if the time-windows are *hard* or *soft* constraints. Hard time windows consider that a node $i \in A$ can not be visited after its deadline d_i. On the contrary, considering soft constraints, the node i might be visited after the deadline d_i, but in this case a penalty occurs.

The objective function of the problem under consideration depends on specific definition of the problem, particularly, according to the time-window constraints. Dealing with hard constraints, the objective function is defined by the sum of the costs of each arc belonging to that tour, while using soft constraints, the objective function depends on the specific problem definition and the values associated to the aformentioned penalties.

There are several versions of the TSPTW which introduce time-dependent variations. These variations usually focus on time-dependent arc costs, setup times, or processing time. This generalization may occur, for example, as a result of considering the traffic effects associated to real world routing problems. Below is the formal definition of the ATSPTW with time-dependent travel times and costs (ATSPTW-TDC), as defined in [**?**].

Let $G = (V, A)$ be a simple directed graph, $V = \{v_i{}_{i=0}^{n}\}$ its set of vertices, where v_0 is the depot vertex. Each vertex v_i has an associated time window $[a_i, b_i]$, verifying that $a_i, b_i \in \mathcal{Z}^+ \cup \{0\}$ and $[a_i, b_i] \subseteq [a_0, b_0] \forall i \in \{1, ..., n\}$. Every time window $[a_i, b_i]$ has associated $p_i = b_i - a_i$ instants of time $\{a_i + k - 1\}_{k=1}^{p_i}$. For simplicity, we will denote $t_i^k = a_i + k - 1$, and therefore $t_i^k \in \mathcal{Z}^+ \cup \{0\}$. The time and the cost of traversing an arc $(v_i, v_j) \in A$ depend on the instant of time t_i^k at which the traversing is started. Consider $c_{ij}^t \geq 0$ and ${t_{ij}}^t \in \mathcal{Z}^+ \cup \{0\}$, respectively, the cost and the time of travesing an arc (v_i, v_j) starting at instant t_i^k.

The proposed goal in this formulation of the ATSPTW-TDC is to find a Hamiltionan cycle in G, starting and ending at v_0, starting and ending inside the time window $[a_0, b_0]$, such that:

- Starting the circuit at time $t_i^k \geq a_0$ involves a waiting time cost $cwt_0(t_0^k - a_0) \geq 0$ with $cwt_0(0) = 0$;
- The circuit leaves each vertex $v_i \in V$ during its associated time window;
- If the circuit arrives at vertex $v_i \in V$ at time $t \in \mathcal{Z}^+$, such that $t \leq a_i$, it is allowed a waiting time $a_i - t$ with cost $cwt_i(a_i - t) \geq 0$, with $cwt_i(0) = 0$. In this case the circuit leaves vertex i at time a_i;
- The sum of the costs of traversing arcs and of the waiting time costs is to be minimized.

The authors of the work introduced in [**?**], propose an exact algorithm for the previosuly defined ATSPTW-TDC, using several graph transformations, which sucessively reduce the problem into an assymetric TSP, for which several effecient exact alorithms already exist.

2.1.2.E Multi objective TSP

The multi objective Traveling Salesman problem is a generalization of the classic TSP, and is part of much broader class of problems, the multi objective combinatorial optimization problems, [**?**], and, in particular, multi objective vehicle routing problems, [**?**].

The Multi objective TSP is defined as follows [**?**]:

Given a list of n cities and a set D = $(D_1, D_2, ..., D_k)$ of $n*n$ weight matrix, the objective is to minimize $f(\pi) = (f_1(\pi), f_2(\pi), ..., f_k(\pi))$, with $f_i(\pi) = (\sum_{j=1}^{n-1} d^i_{\pi(j),\pi(j+1)}) + d^i_{\pi(n),\pi(1)}$, where π is a permutation over the set (1, 2, ..., n).

Note that when $D = (D_1)$, this corresponds to single-objective TSP. Note also that the above formulation considers that all objective functions calculate the weight of the hamiltionian cycle, according to the respective weight matrix.

The quality of the results of the multi objective TSP are usually measured according to its performance accross the Pareto criteria, defined as follows [**?**]:

Pareto dominance

A vector $\vec{u} = (u_1, ..., u_k)$ is said to dominate $\vec{v} = (v_1, ..., v_k)$, deonted by $\vec{u} \preceq \vec{v}$, if and only if $\vec{u}$ is partially less than $\vec{v}$, i.e. $\forall i \in \{1, ..., k\} u_i \leq v_i \wedge \exists i \in 1, ..., k : u_i < v_i$.

Pareto Optimality

Pareto optimality is defined asa concept of allocation optimality. An allocation is not pareto optimal if there is at least one alternative allocation which produces improvements

A solution $x \in \Omega$ is said to be Pareto optimal with respect to Ω iff. there is no $x^{'} \in \Omega$ for which $\vec{v} = F(x^{'}) = (f_1(x^{'}), ..., f_k(x^{'}))$ dominates $\vec{u} = F(x) = (f_1(x), ..., f_k(x))$.

Pareto Optimal Set

For a given MOP $F(x)$, the Paretto optimal set $(P*)$ is defined as : $P* = \{x \in \Omega | \nexists x^{'} \in \Omega : F(x^{'}) \preceq F(x)\}$

Althoug the above mentioned problem refers to the multi-objective TSP, without loss of generality, the multi-objective optimization can be performed on a time-dependent TSP. There is very few direct research about multi objective Time dependent TSP, but one can cite [**?**], which proposes a multi-objective tabu search for single machine scheduling problems with sequence-dependent setup times.

2.1.3 Vehicle Routing Problem

The Vehicle Routing Problem is the problem of finding the optimal set of routes for a fleet of vehicles, to serve a given set of customers. The VRP is believed to be introduced by Dantzig, in 1959, in a work with the name of *The Truck Dispatching Problem*, in which it is considered a generalization of the Traveling Salesman, [**?**]. It was latter shown that, being a generalization of the TSP, its computational complexity is also NP-hard, [**?**].

Being an NP-hard problem, the focus of the research usually revolves around heuristic algorithms, although there are some procedures which are known to produce optimal solutions, [**?**], [**?**]. As refered by Donati in [**?**], citing the work of Blum, [**?**], even when an exact procedure is available, it usually requires large computational time, which is not viable in the time-scale of hours, as required by this industry.

Malandraki, [**?**], as early as 1992, stated that the assumption of constant and deterministically known costs, is an approximation of the actual conditions of routing problems, and thus, a time-dependent formulation of the problem should be considered. In 1999, Gambardella and colleagues proposed a multi ant colony system for solving the vehicle routing problems using a meta-heuristic approach, [**?**]. Years later, Gambardella expanded this research to include time-dependent variations, [**?**], as proposed by [**?**]. There are several other works, which propose meta-heuristic solutions to solve the time-dependent VRP, [**?**], including the use of simmulated annealing, [**?**], and genetic algorithms, [**?**].

The rest of this section is structured as follows. The next section presents a formal definition of the Vehicle Routing Problem and its time-dependent variation, as well as the most common objectives of the resolution of this problem. Since the TSP occurs only as a generalization of the non-capacitated vehicle routing problem, the study of the capacited vehicle routing is out of the scope of this work.

2.1.3.A Problem definition

Following the definition proposed by [**?**], let $G = (V, A)$ be a graph, where $V = 1, ..., n$ is a set of vertices, representing nodes/customers/cities, with the depot located at vertex 1, and A is the set of arcs fully connecting the nodes. Each arc (i, j), $i \neq j$, is associated with a non negative weight, c_{ij}. Depending on the context of the work, this weight might represent the distance between nodes, the travel time, or even the travel cost. It is assumed that a fleet of m vehicles is available. The Vehicle Routing Problems consists in finding the set of optimal routes such that:

1. each city in $V\backslash\{1\}$ is visited exactly once, by exacly one vehicle;

2. all routes start and finish at the depot;

3. some constraints must be satisfied;

The most common constraints associated to the 4) include: capacity restrictions associated with each vehicle; limit on the number of nodes that each route might visit; total time restrictions; time-windows in which each node must be visited; precendence relations between nodes.

The goal of the Vehicle Routing Problem usually consists in finding an optimal set of routes, as to minimize the total cost, where the cost depends on the total distance covered, and the fixed costs associated to each vehicle. However, depending on the problem under study, the goal may be different, as to minimize the total travel time, minimize the total number of vehicles, or even both at the same time [**?**].

2.1.3.B Time-dependent VRP

The Vehicle Routing Problem is a very wide class of optimization problems, whose precise problem definition usually depends on the characteristics of the problem under considerations. Thus, introducing time-dependencies on the problem also depend on the specifics of the situation. There are several athors which consider time-dependent travel costs, [**?**], and the objective is to minimize the total costs, while others introduce time-dependent travel times, [**?**], and the objective is to minimize the total travel time. There are also those who consider that the objective function is a function of both travel time and travel costs, and at least one of these (travel time, travel cost) is time-dependent, [**?**]. The definition here proposed follows this last time-dependent variation.

The time-dependent VRP is defined, following the work of [**?**], as follows. Let $G = (V, A)$ be a graph where $A = \{(v_i, v_j) : i \neq j \wedge i, j \in V\}$ is the set of arcs, and $V = (v_0, ..., v_{n+1})$ is the set of vertex Vertices v_0 and v_{n+1} denote the depot at which the vehicles are based. It is considered that each vehicle has an uniform capacity of q_{max}. It is also expected that each vertex in $i \in V$ has an associated demand $q_i \geq 0$, a service time $g_i \geq 0$, and the depot has $q_0 = 0$ and $g_0 = 0$. The set of vertex $C = (v_1, ..., v_n)$ specified the set of n customers. The arrival time of a vehicle at customer i, $i \in C$, is denoted by a_i, and its departure time b_i. Each arc (v_i, v_j) has an associated distance $d_{ij} \geq 0$, and a travel time $t_{ij}(b_i) \geq 0$. Note that the travel time is a function of the departure time from costumer i. The set of available vehicles is denoted by K. Consider that the cost per unit of route *duration* is denoted by c_t, and the cost per unit of route *distance* is denoted by c_d.

In this formulation, there are two goals for the time-dependent VRP. The first corresponds to the minimization of the total number of vehicles used. The second corresponds to the minimization of the total cost, which is a function of both distance and travel time.

There complete definition of the problem follows a mixed integer programming approach, with a total of 11 constraints. These will not be covered in detail here, as the VRP is not the primary object of study of this work, that being the TSP. Thus, it is important to define in which circustances the TDVRP can be transformed into the TDTSP. This is possible by considering only one vehicle, with infinite capacity, and by adapting the objective function according to the problem under consideration.

We conclude this section with the final remark that the above presented definition of the time-dependent VRP corresponds to a static version of the time-dependent case. There is a lot of research around the dynamic case, in which the problem is updated during the execution of the program. This has major applications in the routing industry, and it is often refered to as *real-time* Vehicle Routing. For more information regarding this problem, we refer to [**?**] [**?**].

2.1.3.C Multi-objective VRP

Multi objective optimization corresponds to the resolution of a combinatorial optimization problem in which more than one goals are defined. In the case of the Vehicle Routing Problem, [**?**], the most common objectives include minimizing the fleet size, the total distance traveled, the total time required, the total tour cost, and/or maxizimizing the quality of the service or the profit collected. Note that in most problems, when multiple objectives are identified, the different objectives often conflict with each other.

Multi-objective optimization usually relies on the use of meta-heuristics, [**?**]. There are several works focusing on this problem, and the most promising meta-heuristics for multi-objective optimization include Evolutionary Optimization, [**?**] and Simmulated Annealing, [**?**]. There is also some work considering the Ant Colony Optimization. In particular, a modified ant colony was designed to solve a bi-objective time dependent vehicle routing problem, in which the main goal was the minimization of the fleet size, followed by the minimization of the total cost, [**?**].

2.2 Algorithmic overview

The algorithms which address the Traveling Salesman, and any Combinatorial Optimization problem for that matter, can be classified as exact, heuristic or meta-heuristic. Exact algorithms are thus which always provide an optimal solution to the problem. Although these might seem the first choice, exact algorithms are usually inefficient for solving large problems. In its turn, heuristic algorithms intend to be efficient, abdicating the objective of finding the best solution, and focusing on finding *near-optimal* solutions in a short time. Heuristic algorithms are usually problem specific, while Meta-heuristics algorithms are designed in such a way that they can be used for a variety of Combinatorial Optimization problems, in a fast and efficient way.

2.2.1 Exact algorithms

There are some exact algorithms available for the Traveling Salesman Problem, [**?**], including its time-dependent variations, [**?**], and even with time windows, [**?**]. These algorithms usually require the problem to be formulated as an Integer Linear Programming Instance. In this section we present ILP definitions for both the classical time-dependent TSP. We also present a brienf introduction regarding the Branch and Bound algorithm, which has proven to be very useful for determining exact, or at most, near optimal solutions for the TSP. The software *Concorde*, uses a Branch and Bound algorithm, and was used to solve all 110 instances of the TSPLib, reporting exact solutions in every problem, including a instance with 89.900 nodes, although it required more than 110 CPU years.

2.2.1.A Integer Linear Programming

ILP for the TDTSP

The TSP be formulated as an integer programming problem, (see Laburthe 1998). The decision variables are x_{ij}, which take values of one and zero, followig the following rule.

$$x_{ij} = \begin{cases} 1, \text{ if the tour contains arc (i,j)} \\ 0, \text{ otherwise} \end{cases} \tag{2.3}$$

Let c_{ij} represent the weight of the arc (i, j). The objective of the TSP is described by equation **??**. Equations **??** and **??** represent constrains over the variables, particularly, that a tour must enter and leave each node exactly once. However, this does not completly define the characteristics of a Hamiltonian cycle. To eliminate the possibility of subtours, that is, of having some node more than once in a solution, it is necessary to introduce the sub-tour elimination constraint. This is expressed in equation **??**. Without this constrain, the formulation of the problem reduces to the classical Asignment Problem, that can be solved in polynomial time, $\mathcal{O}(n^3)$.

$$min \sum_{ij} c_{ij} x_{ij} \tag{2.4}$$

$$\forall i, \sum_{j} x_{ij} = 1 \tag{2.5}$$

$$\forall j, \sum_{i} x_{ij} = 1 \tag{2.6}$$

$$\forall S \subset N, S \neq \emptyset, \sum_{i \in S} \sum_{j \notin S} xij \geq 2 \tag{2.7}$$

ILP for the TDTSP

The TDTSP can be formulated as integer linear programming problem, by using bynary decision variables, x_{ijt}. These variables take a value of zero or one, according to the rule of equation **??**.

$$x_{ijt} = \begin{cases} 1, \text{if city j and i are visited in the time period t and t-1, repectively} \\ 0, \text{otherwise} \end{cases} \tag{2.8}$$

The objective function is presented in equation **??**. Equations **??**, **??** and **??** represent constraints over the decision variable. Particularly, they state that each city must be entereded exactly once, left exactly once, and visited in exactly one time period, respectively. As occurs with the classical TSP, the ILP formulation needs to formulate a constraint to eliminate the possible formation of sub tours. This is presented in equation **??**. Finaly, equation **??** guarantees that the decision variable takes binary values.

$$min \quad \sum_{i}\sum_{j}\sum_{t} C_{ijt}x_{ijt} \tag{2.9}$$

$$\sum_{j}\sum_{t} x_{ijt} = 1 \quad i = 1, ..., n \tag{2.10}$$

$$\sum_{i}\sum_{t} x_{ijt} = 1 \quad j = 1, ..., n \tag{2.11}$$

$$\sum_{i}\sum_{j} x_{ijt} = 1 \quad t = 1, ..., n \tag{2.12}$$

$$\sum_{j=1}^{n}\sum_{t=2}^{n} t x_{ijt} - \sum_{j=1}^{n}\sum_{t=1}^{n} t x_{ijt} = 1 \quad i = 1, ..., n \tag{2.13}$$

$$x_{ijt} \in 0, 1 \quad i, j, t \in i, ..., n \tag{2.14}$$

2.2.1.B Branch and Bound

Branch and Bound ($B\&B$) is one of the most used tools to solve large NP-hard combinatorial optimization problems. To be precise, $B\&B$ should be classified as an algorithm paradigm, constitued by 3 main parts, which have to be choosen according to the problem under consideration, and for which many options may exist, [**?**].

The force of the $B\&B$ comes from it being a search algorithm which (indirectly) searches the complete search space of the problem. Since this is not directly feasible, due to the common exponential growth of the solution space, $B\&B$ takes advantage of *bounds*, combined the information about the current best solution, to safely discard certain solutions amongst the search space.

At any point of the algorithm, there is a *current solution*, and a *pool* of unexplored subsets of the

solution space. At the beginning of the algorithm, this pool consists of (only) the root node, and at the end of the algorithm, it will consists of an empty set, meaning that the entire search space was sucessfully explored. The initialisation of the $B\&B$ requires the *incumbent*, which denotes the objective function value of the current solution, to be initialised as ∞. If it possible to generate an initial feasible solution using some heuristic method, this solution is recorded and its objective value is set as incumbent. The process of generating an initial solution has usually a positive impact on the $B\&B$ algorithm. After the initialisation, this algorithm enters in an iterative proccess, untill the pool of unexplored subsets is empty. This processes consists of three main components:

- selection of a node to process;
- bound calculation;
- branching.

Branch and Bound algorithms vary according to the strategies estabilished for each of the three main components of the iterative proccess, aswell as the initial heuristic. In any case, the bounding function selected is the key for any good branch and bounding algorithm, because the selection of a bad function can not be compensated with good choices on the branching and bounding strategies. For example, consider the trivial case where the bounding function is the constanst value of 0. It is obvious that this will always be a lower bound to the problem, but it does not produce any quality information of which solutions to discard. Ideally, the value of the bounding function for a given subproblem should be equal to the value of the beast feasible solution to that problem. This is usually not possible, since subproblems may also be NP-hard. Thus, bounding functions are choosen according to the proximity to the best possible value, and to its time complexity - usually restricted to polynomial time.

To complete this overview, selection strategies for the TSP usually revolve around Best First, Depth First and Breadth First Search. There are several works which discuss the main differences of each selection strategy and report on which strategy might be more adequate according to the problem characteristics. Finally, the branching strategy in the TSP usually consinsts of selecting any node with a degree 3 or higher in the search tree. There are several comments around this strategy, some authors opting for the selection of a node with a low bound, as this theoritically will reduce the number of searches when processing the node with higher bounds.

2.2.2 Heuristic algorithms

In some particular cases, exact algorithms can not be used in the resolution of the TSP problem under question. This usually occurs when dealing with very large instances of the problem, or when there is an urgency in obtaining solutions in a fast way. In these cases, using approximation algorithms may be

a good choice. These algorithms are not guaranteed to produce an optimal solution, however, with a good heuristic, approximation algorithms produce high quality solutions in a reasonable time. Generally, the heuristic may be classified as one of two classes: construction or improvement heuristics, [**?**].

When an optimal solution is not known, measuring the heuristic performance of a method may be difficult. In this cases, the heuristic evaluation can be done by comparison with the Held-Karp lower bound.

Heuristic algorithms are usually specifically designed for a particular optimization problem. For example, the Lin-Kernighan Heuristic was created to solve the symmetric Traveling Salesman Problem, and does not seam to have usefull applications in other combinatorial optimization problems.

2.2.2.A Held-Karp Lower Bound

In some cases, the quality of a heuristic solution can not be directly calculated, as no exact solution for the problem under consideration is known. In this cases, it is important to have a way of evaluating performance. The standart way of doing this is by comparing the heuristic solutions which the solution generated by the Held-Karp (HK) lower bound, [**?**].

The HK lower bound is the solution to the linear programming relaxation of the ILP formulation of the TSP. This solution can be found in polynomial time for moderate instance sizes.. However, for a very large problem, solving the relaxed problem directly is not feasible. In this cases, Held and Karp prose an interative algorithm in order to approximate the solution. This methods involves computing a large number of minimum spanning trees. This iterative version of the algorithm will often keep the solution within 0.01% of the HK lower bound, [**?**].

2.2.2.B Tour construction

A construction algorithm is based on the construction of a valid tour. The construction process stops when a valid tour is found. No improvement over the formulated tour is attempted.

A – Nearest neighbour The nearest neighbour is a very simple heuristic for the TSP. This algorithm starts with the selection of a random node. Then, while there are unvisited nodes, the heuristic always selects the nearest node which was not yet visited. This proccess is repeated while there are unvisited nodes. Finally, when there are none, the construction is complete with the return to the first node.

The computational complexity of the nearest neighbour is $\mathcal{O}(n^2)$. The solutions generated with this heuristic are often within 25% of the optimal solution.

The pseudocode for the NN algorithm is presented below.

1. Select a random city

2. Select the nearest unvisited node

3. If there are unvisited nodes, repeat step (2)

4. Return to first node

B – Greedy heuristic The greedy heuristic is a construction algorithm which creates a valid tour by repeatedly selecting the arc with the lowest weights, and taking into account the problems constraints. In particular, the greedy algorithm rejects an arc which creates a cycle with less than n edges, or which would create a sub tour.

The computational complexity of the greedy heuristic is $\mathcal{O}(n^2 log_2(n))$. The solutions generated by this heuristic are often within the 20% of the optimal solution.

1. Sort all arcs according to its weight

2. Select the lowest weight arc, if it does not violate any constraint

3. If the constructed solution is not complete, repeat (2)

2.2.2.C Tour improvement

Improvement heuristics are algorithms which work over a valid and complete solution in order to improve it. The most commom improvement heuristics are the 2-opt and 3-opt local search algorithms. The Lin-Kernighan algorithm (LKA) is a particular implementation of the above mentioned local searches methods, in which a k-opt local search is employed, where the value of k varies during the algorithm execution. LKA have shown to be very efficient and capable of presenting high quality solutions to the TSP.

A – 2 and 3 opt tours "In optimization, 2-opt is a simple local search algorithm first proposed by Croes in 1958 for solving the traveling salesman problem."

The 2-opt is possibly the most simple local search algorithm. The objective of this method is to find route crossovers, and fold them. When this occurs, the overal cost of the newly constructed tour will decrease.

The 2-opt search works in a recursive way. This search algorithm tries to improve the original tour, by removing two edges from the original cycle, and reconnecting the two paths created. This process is ilustrated in figure **??**. In a 2-opt search, there is only one way of connecting the nodes in a way which will result in a different and valid tour. If the new tour has a lower cost, the cycle restarts, using the new tour as the improvement object. Otherwise, two other edges are selected, and the cycle continues with

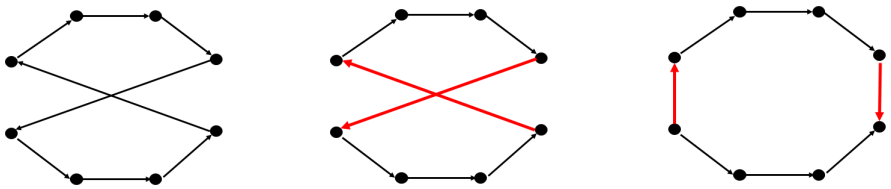

Figure 2.2: The 2-opt local search works by reconnecting two edges, hoping to fold possible crossovers, decreasing the overall tour coust. In the left image, a crossover is identified. In the middle image, a the edges beloning to this crossover are removed, and in the figure to the right, they are reconnected, forming a new valid tour

the original solution. This iterative method continues untill no further improvement is reached over a complete iteration cycle. In this case, the tour is known to be 2-opt.

The 3-opt search is very similar to the 2-opt. Instead of selecting two edges and reconnecting the path, the 3-opt selects 3 edges. In this case, there are two different ways of forming a new valid tour. A 3-opt move can also be seen as two or three 2-opt moves combined in the formation of a new tour. The iterative cycle of the 3-opt search works in the same way as the 2-opt.

B – k opt tour More generally, k-opt local search methods are a way of rearanging a tour, by taking k edges and reconnecting the paths in order to form a new valid tour. Any tour that is known to be $k - opt$ is also $(k-1)opt$. Some particular problems, as "the crossing bridges", figure **??** can only be solved with a 4-opt or higher method.

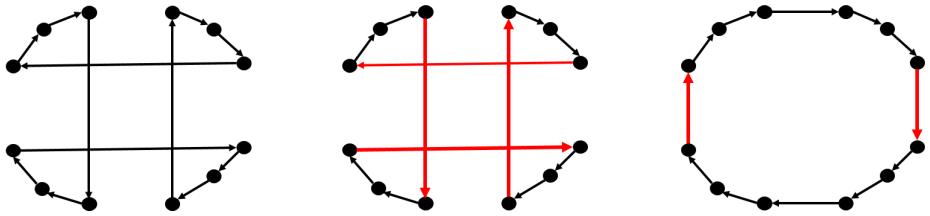

Figure 2.3: The crossing bridges can only be solved by reordering 4 edges. The resolution of this problem with local search is only possible with 4-opt or higher.

C – Lin-Kernighan heuristic The Lin-Kernighan Heuristic (LKH) [**?**], is an algorithm for the symmetric TSP, which was the state of the art for over than 15 years. LKH is known for producing optimal solutions often, for presenting solutions within 2% of the Held-Karp lower bound, and for having a time complexity of approximately $\mathcal{O}^{n^{2.2}}$, [**?**]. This heuristic is constructed for the symmetric, and using it for the asymmetric generalization requires a graph tranformation process, which transforms the asymmetric instance with n nodes, into an equivalent symmetric one with $2n-1$ nodes, [**?**] [**?**]. Thus, for the

same number of nodes, solving an assymetric TSP with the LKH heuristic is usually 4 times harder than solving the symmetric case.

To understand the Lin-Kernighan heuristic, it is necessary to think about the TSP in a slighly different manner. Consider the following way of defining a combinatorial optimization problem: "find from a set S a subset T that satisfies some criterion C and minimizes an objective function f." In the TSP, the objective is to find from the set of all edges (S) of a complete graph, the subset (T) which forms a valid tour (C) and minimizes the objective function (f). Using this formulation it is now possible to explain the LK heuristics.

Given an non-optimal and feasible solution T, it is non-optimal because k elements $\{x_1, ..., x_k\}$ in T are *out of place*. To improve this solution, and make it optimal, one would have to substitue the set of k elements $x_1, ..., x_k$ with the elements $y_1, ..., y_k$ of $S \backslash T$. Because there is no knowledge about how many elements are missplaced, Lin and Kernighan consider that setting the value of k a-priori would seem artificial. Thus, they propose an interative procedure in which the algorithm dynamically estimates the best value for k. In order to do this, the LKH first estimates the most out of place elements, x_1 and y_1. Then, with this values set aside, it tries to repeat this proccess for x_2 and y_2, and so on. It stops this inner loop when no improvement seams plausible, replaces the current solution T with the new solution generated from replacing the now selected elements, and restarts the whole proccess. This proccess is formalized below, as presented by Lin and Kernighan.

This heuristic has not been shown to work for the time-dependent TSP, as it is constructed for a symmetrical $n * n$ cost matrix only. Thus, the overview of this algorithm will not be extensive, as it has no pratical application to the problem under consideration. In any case, being a very relevant heuristic for the classical TSP, it is an algorithm worth mentioning.

2.2.3 Meta-Heuristic algorithms

Meta-Heuristic algorithms are heuristic algorithms which, unlike the classical heuristic, can be applied to a variety of optimization problems. Meta-Heuristic are designed to be applied to combinatorial optimization problem, and not to a specific problem of this class. Meta-Heuristic rose in importante during the 1990's, and have become one of the most important class of algorithms in computer science.

More formally, a meta-heuristic is an iterative generation process, which guides a heuristic by combining intelligently different concepts, for exploring and exploiting the search space, using learning strategies to structure information, as to efficiently find optimal or near-optimal solutions, [**?**].

This subsection will introduce a few of the most relevant meta-heuristics in the resolution of the Traveling Salesman Problem, particularly, the Ant Colony Optimization (ACO) and the Simmulated Annealing procedures (SA). There is a variety of meta-heuristics which are not discussed here, but which have also been succesfully applied to the TSP. Examples of these meta-heuristics are the Tabu-Search, Evo-

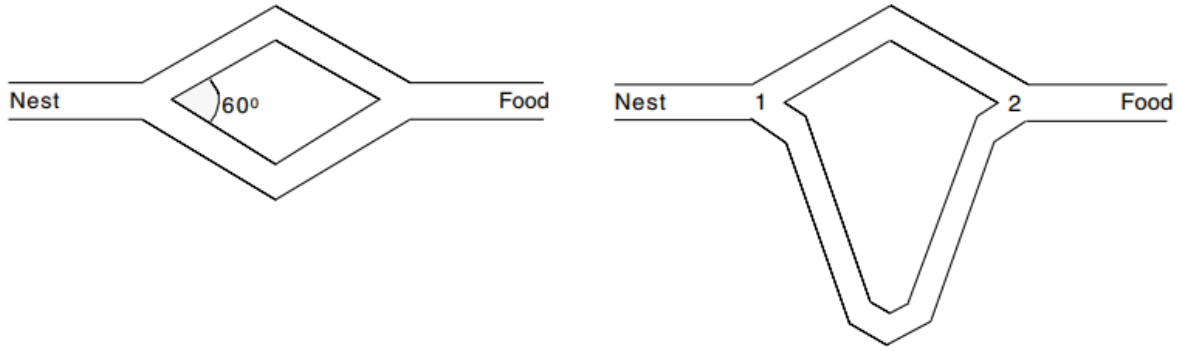

Figure 2.4: The double bridge experiment. On the left, two bridges with the same length. Experimental results show that ants distribute themselves evenly amonst both bridges. On the right, one of the bridges is longer than the other. Experimental results show that ants use the shorter bridge more often.

lutionary Algorithms (EA), in particular the Genetic Algorithm (GA), and many other Swarm Intelligence algorithms, from which the Ant Colony Optimization is the oldest and most widely used.

2.2.3.A Ant Colony Optimization

The Ant Colony Optimization, [**?**, **?**, **?**], is based on the behaviour of real ants, and was developed by M. Dorigo et. al, which were curious about the generelization of the double bridge experiment, [**?**], [**?**], illustrated in (figure **??**). This led to an adaptation of this experiment, substituing the double bridge with a graph, the pheromone trail with artificial pheromone, and the real ants with artificial ants, with presented some extra capabilities intended to facilitate the resolution of more complex problems [**?**].

Using the model of a static combinatorial optimization problem, as defined in section **??**, it is possible to derive a generic pheromone model, that can be exploited by the Ant Colony Optimisation. This means that both the classical and the time-dependent TSP, which can be formulated by the mentioned model, can, potentially, be solved by the ACO algorithm.

The following pseudo-code represents the algorithmic skeleton for the ACO model, and each of its parts will be explained with more detail below.

Reinsert the below hidden SA metaheristic procedure.

The general process of the Ant Colony Optimisation algorithms is as follows. The algorithm starts with a parameter initialization. This is also responsible for setting the pheromones levels to some value, τ_0. This value is usually chosen using a heuristic function. For the TSP case, the heuristic chosen is often the nearest neighbour.

After initialization, and until some specific termination condition is met, the ACO algorithm runs in a loop, which consists of 3 main steps: solution construction, local search (optional), and pheromone update. Each of this steps is detailed below.

The solution construction is a process carried out by each of a specified number of ants. Each ant

starts with an initially empty solution, s_p, and at each iteration step expands its solution with a valid solution component, c_i^j. This construction function differentiates the ACO algorithm for every model. By restricting the construction method to the agents (the ants), the rest of the algorithm does not have to be heavily adapted to the specific model. However, the function that is responsible for selecting the feasible solution components, has to be aware of the model variables and its set of constraints. It has to determine those variables whose addition to the partial solution do not constitute a violation to the set of constraints of the model. This set is represented by $N(s_p)$.

Having the set of all feasible solutions components, $N(s_p)$, it is necessary to choose a single component, c_i^j. This selection is done probabilistically, and the choice takes into account both pheromone (exploitation) and heuristic information (exploration). The algorithmic parameter q_0 is responsible for defining both method's relative importance. The outline execution of the function is presented in equation **??**, and is as follows. A random value, q, is set. If this value is lower than the algorithms parameter q_0, the selection of the node is done with the heuristic rule, equation **??**. Else, the Ant System rule is used, **??**.

$$c_i^j = \begin{cases} \text{heuristic rule}, & if q \leq q_0 \\ \text{ant system rule}, & \text{otherwise} \end{cases} \tag{2.15}$$

$$argmax_{l \in N_i^k} \tau_{il} [\eta_{il}]^\beta \tag{2.16}$$

$$p(c_i^j | s_p) = \frac{\tau_{ij}^\alpha [\eta(c_i^j)]^\beta}{\sum_{c_i^l \in N(s_p)} \tau_{il}^\alpha [\eta(c_{i,l})]^\beta} \tag{2.17}$$

After finalising the construction of valid solutions, the ACO algorithm may implement a local search. Although this step is optional, it has been demonstrated that ACO algorithms reach their best performance when local search is applied. The ant's construction method is biased by the pheromone information, while the pheromones values are biased by the quality of the solutions. Local search, also called Daemon Actions, are techniques which intend to work on the existing solutions, exploring and expanding the search space, and ultimatly improving the quality of the solutions. This will be reflected in the ant's construction method, along the next iterations. The most widely used local search methods are the 2-opt search, the 3-opt search, and the Lin-Kernighan heuristic.

The final step of each loop of the algorithm is the pheromones update. This is responsible for making solution components that belong to good solutions more desirable in the following iterations. To achieve this objective, two methods are implemented. The first is pheromone deposition, which increases the pheromone intensity of the solution components belonging to the most promising solutions. The amount of solutions that are used to deposite pheromone is a parameter of the algorithm. The second method to achieve this step's goal, is pheromone evaporation. While it may seam counter-intuitive to deposit

pheromones and, at the same time, also evaporate them, this step is crucial to avoid a rapid convergence to sub-optimal solutions. pheromone deposition alone is responsible for making good solutions more desirable, while pheromone evaporation reduces both the desirability of bad solutions and the sub optimal convergence of good solutions, favoring the exploration of the search space. The pheromone update is implemented as in equation **??**.

$$\tau_{ij} = (1-\rho)\tau_{ij} + \sum_{s \in S_{upd} | c_i^j \in s} g(s) \tag{2.18}$$

2.2.3.B Simulated Annealing

The SA algorithm was delevoped using an analogy between the phyisical annealing in solids, and finding the minimum cost configuration in combinatorial optimization problems. In the phyisical world, annealing is the process of heating a metal until the melting point, and reducing the temperature in a controlled way. The decrease in temperatur results in a particle rearangement, in which lower energy states are reached. When the heating temperature is very high, and the temperature is decreased very slowly, this will result in the ground state of the solid - its minimum energy state. The analogy between physical world and the combinatorial optimization problems is achieved by considering that the energy of the metal corresponds to the cost of the solution, and the particle rearangement consists in the selection of a neighbourhood solution, [**?**].

The Simulated Annealing algorithms concists in an iterative improvement, which stochastily accepts up-hill moves. More precisely, the procedure starts with the selection of a feasible solution, aswell as the Initialization of some necessary parameters, as the temperature, which will serve as a control variable. After this, the SA enters an iterative proccess. At the core of this iterative proccess is a local search proccess, which is executed a fixed number of times at each iteration. Authors usually estabilish this value as 2 times the number of nodes. After this local search procedure is complete, the temperature is decreased according to its cooling schedule, after which the local search restarts, and the cycle continues, untill either the execution time is reached, or the temperature reaches 0.

Simulated Annealing differs from other iterative improvement algorithms, because it stochastily accepts up-hill moves, which allow it to escape from local minima (as happens in, f.e., the Tabu Search). More precisely, at each stage of the local search procedure, the difference in the energy level, Δ, between the current state and the newly generated state is calculated. If Δ is negative, the new state is better than the current one, and it is (always) accepted. On the contrary, if Δ is positive, the state is accepted if equation **??** is verified, otherwise, it is rejected. This equation is often called the Metropolis criteria. Note that using this criteria, as the temperature approaches zero, less and less bad states are accepted, and at $T = 0$ Simulated Annealing no detoriations will be accepted at all.

$$exp(-\frac{\Delta}{T}) > Random[0,1[\tag{2.19}$$

Time-dependent scheduling problems can also be solved using this meta-heuristic, [**?**], as the algorithm solely relies on the search of a neighourhood set. The pseudo-code describing the SA procedure is presented below.

In the first works published about the Simulated Annealing, it was proven that if the temperature is cooled very slowly, the process will converge to the optimal solution. More precisely, if temperature drops no more quickly than $C/log(n)$, where C is the Boltzman constant, and n is the number of steps taken so far. This result however is not as relevant as it first seems, because this cooling schedule is *very* slow. Some authors refer that it is faster to do exhaustive search than to follow this colling schedule, [**?**].

The Simulated Annealing procedure varies according to: the cooling schedule; the neighbourhood search criteria; the Markov chain length. There are several reports which describe the influence of these modules in the overall performance of the SA procedure. There are other very relevant aspects of this algorithm which may also influence the results as, for example, the initial and the final temperature.

Reinsert the below hidden SA metaheristic procedure.

3

Problem Formulation

Contents

3.1 Flying Tourist Problem

Consider a tourist who wishes to take a trip that visits every node (city) i in the set of nodes V, $|V|$ = N, with no particular order. The start node will be noted as v_0, while the return node as v_{n+1}, and the complete set of nodes is given by V_c = $V \cup \{v_0\} \cup \{v_{n+1}\}$. The trip must start at a time $t \in T_0 = [T_{0m}, T_{0M}]$. Upon visiting a node, the tourist will stay there for a duration of d time-units (days). Consider that for each node to be visited, there is a range for the value d might take, that is $d \in d_i = [d_{im}, d_{iM}]$ and $d_{iM} \geq d_{im} \geq 1$. The complete set of durations associated to each city is given by D, and $|D| = N$. Furthermore, to each city $i \in V$, there is an associated time-window TW, $|TW| = |V| = N$, which defines the set of dates in which the city i may be visited.

By following this definition, the FTP is completely defined by a structure $G = (V_c, A, T_0, D, TW)$, used to create a multipartite graph describing the request. This multipartite graph is divided into k layers, where each layer corresponds to a particular moment in time. Besides this, every node in a layer is connected to all nodes in the subsequent layer. The set of arcs that connects these nodes is given by A. To each arc $a \in A$, it is associated a cost c_a (ticket cost) and a processing time p_a (flight duration), which depend upon the routed nodes, as well as the time in which the arc transition is initiated, that is, $\forall a_{ij}^t \in A$, $c_{ij}^t \geq 0$ and $p_{ij}^t \geq 0$.

A valid solution s to the formulated FTP is a set of arcs (commercial flights) which start from node v_0 during the defined start period, visits every node i in V during its defined time-window $TW(i)$, by considering the staying durations defined by $D(i)$, and finally returns to node v_{n+1}. The set of all valid solutions is given by S. The goal of the FTP is to find the global minimum $s^* \in S$, with respect to the considered objective function.

The objective function associated to this problem depends on the user criteria. While some users might consider the expended cost to be the most important factor, there are others who consider the total flight duration of crucial importance. Thus, a total of three different objective functions shall be herein considered: (i) the expended cost (see eq. **??**), (ii) the flight duration (see eq. **??**), and (iii) the resulting entropy (see eq. **??**), where the latter corresponds to a weighted sum between the former two.

$$F_c(s) = \sum_{n=0}^{N+1} c(s[n]) \tag{3.1}$$

$$F_t(s) = \sum_{n=0}^{N+1} p(s[n]) \tag{3.2}$$

$$F_e(s) = \sum_{n=0}^{N+1} w_c * c(s[n]) + w_p * p(s[n]) \tag{3.3}$$

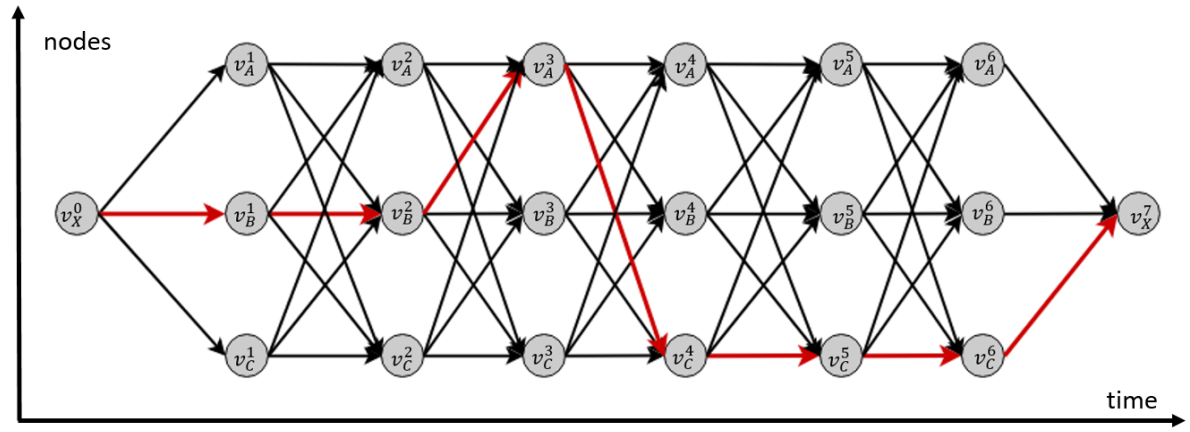

Figure 3.1: Illustration of a Flying Tourist Problem using a multipartite graph. To each node (A,B,C) it is associated a waiting period of respectively (1,2,3) time units. The red arrows represent a possible solution to the problem.

Figure **??** illustrates the multipartite graph associated to a simple instance of the FTP with $v_{n+1} = v_0 = X$, one possible start date ($t = 0$), 3 nodes to visit (A, B, C), with a fixed duration of respectively (1,2,3) time-units, and no constraints relative to the time-window of each city. A possible solution to this problem instance corresponds to the set of arcs $(a^0_{X,B}, a^2_{B,A}, a^3_{A,C}, a^6_{C,X})$.

Despite the apparent complexity of the proposed definition, it can be used to state very simple flight searches, including one-way and round-trip flights. For example, the problem of finding a single flight from A to B at date T can be instantiated as a FTP given by v_0 = A, v_{n+1} = B, $T_0 = T$, and V = D = TW = $\{\}$. In its turn, a round-trop flight involving the same two cities and the same start date, in which the staying period in B is b days, is given by $v_0 = v_{n+1} = A$, $T_0 = X$, $V = \{B\}$, $D = \{b\}$ and TW = $\{\}$. Thus, this definition is adequate either for simple and complex trips, which can be customized according to the user search criteria, by setting either an extended start period, or flexible waiting periods.

3.1.1 Relation to the TSP

As previously stated, the proposed FTP is closely related to the TSP and to its time-dependent variation. Given the following list of constraints:

1. $v_{n+1} = v_0$;
2. $T_0 = 0$;
3. $TW(i) = [0, +\infty[, \forall i \in V$;
4. $D(i) = 1, \forall i \in V$;
5. $c^t_{ij} = c_{ij}, \forall i, j \in V, \forall t$.

constraints (1-4) enable the reduction of the devised FTP to a TDTSP, as proposed by J.C.Picard ([**?**]), and the final constraint (5) reduces the problem to the classical TSP.

Since the FTP occurs as a generalization of the TSP, and given that the latter problem is well-known to be Np-hard complex, than so is the former one.

3.1.2 Graph construction

By considering the presented FTP definition, the total number of layers (k) of the devised multipartite graph represents the total time span between the earliest date at which the trip might start and the latest date in which it should finish. The arcs that connect those nodes are divided into three groups: *initial*, *transition* and *final* arcs.

The *initial* arcs are those which might initiate the trip. Consequently, they must start at node v_0, at a time $t \in T_0 = [T_{0m}, T_{0M}]$, connecting v_0 to every node in V. There are a total of $k_i = T_{0M} - T_{0m} + 1$ layers for the initial arcs.

Conversely, the *final* arcs are those that connect every node in V to the return node, v_{n+1}. There are as many final layers as there are initial layers, and the final layer extends from T_{fm} to T_{fM}, where T_{fm} = $T_{0m} + \sum(D)$ and T_{fM} = $T_{0M} + \sum(D)$, where $\sum(D)$ corresponds to the summation of all entries belonging to D. In the example depicted in Figure **??**, there is a single initial and final layer, since there is only one possible start date.

The *transition* arcs are those which fully connect the N nodes belonging to V. The earliest transition arc occurs at a time no sooner than $t_1 = T_{0m} + min(D)$, where $min(D)$ corresponds to the lowest entry of the set of staying durations. Hence, if the trip starts by transiting an initial arc at time T_{0m}, the first transition arc might only be traversed $min(D)$ time-units later. By following a similar approach, the latest transition arc can occur no latter than $t_2 = T_{0M} + \sum(D) - min(D)$. Thus, there are a total of $k_2 = t_2 - t_1 + 1$ transition layers, and $k_2 * n * (n-1)$ transition arcs.

The union of the initial, transition and final arcs gives the set A of all the arcs, which may be used to construct a solution to the requested trip.

Having the information relative to the multipartite graph associated to the devised FTP, it is now possible to construct a three-dimensional array matrix representing this problem, where each entry of the array corresponds to an arc connecting two nodes, at a particular moment in time. This weight matrix is initialized with a very high cost value (as to reject arcs which may not be part of the solution), and every entry of it is updated according to the information of the multipartite graph and the respective objective function. Finally, this weight matrix may be used as input for the optimization system (see section).

Insert reference

Although it is clear that any arc $a \in A$ corresponds to a particular flight, it should be noted that no specific or limiting assumption was considered up until now. Instead, it was assumed an entirely abstract arc definition, connecting two nodes at a specific moment in time. In order to transform this set of arcs

into a corresponding set of flights, it is necessary to obtain real-world flight data from some external source. This will be further detailed in section

Insert reference

A

Code of Project

Nulla dui purus, eleifend vel, consequat non, dictum porta, nulla. Duis ante mi, laoreet ut, commodo eleifend, cursus nec, lorem. Aenean eu est. Etiam imperdiet turpis. Praesent nec augue. Curabitur ligula quam, rutrum id, tempor sed, consequat ac, dui. Vestibulum accumsan eros nec magna. Vestibulum vitae dui. Vestibulum nec ligula et lorem consequat ullamcorper.

Listing A.1: Example of a XML file.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <StreamInfo version="2.0">
3     <Clip duration="PT01M0.00S">
4         <BaseURL>videos/</BaseURL>
5         <Description>svc_1</Description>
6         <Representation mimeType="video/SVC" codecs="svc" frameRate="30.00" bandwidth="401.90"
7             width="176" height="144" id="L0">
8             <BaseURL>svc_1/</BaseURL>
9             <SegmentInfo from="0" to="11" duration="PT5.00S">
```

```
10              <BaseURL>svc_1-L0-</BaseURL>
11          </SegmentInfo>
12      </Representation>
13      <Representation mimeType="video/SVC" codecs="svc" frameRate="30.00" bandwidth="1322.60"
14          width="352" height="288" id="L1">
15          <BaseURL>svc_1/</BaseURL>
16          <SegmentInfo from="0" to="11" duration="PT5.00S">
17              <BaseURL>svc_1-L1-</BaseURL>
18          </SegmentInfo>
19      </Representation>
20  </Clip>
21 </StreamInfo>
```

Etiam imperdiet turpis. Praesent nec augue. Curabitur ligula quam, rutrum id, tempor sed, consequat ac, dui. Maecenas tincidunt velit quis orci. Sed in dui. Nullam ut mauris eu mi mollis luctus. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Sed cursus cursus velit. Sed a massa. Duis dignissim euismod quam.

Listing A.2: Assembler Main Code.

```
; ********************************************************************
; * Constantes
; ********************************************************************

ON     EQU 1 ; contagem ligada
OFF    EQU 0 ; contagem desligada
INPUT  EQU 8000H ; endereço do porto de entrada
           ;(bit 0 = RTC; bit 1 = botão)
OUTPUT  EQU 8000H ; endereço do porto de saída.


; ********************************************************************
; * Stack
; ********************************************************************

PLACE   1000H
pilha:    TABLE 100H  ; espaço reservado para a pilha
fim_pilha:

; ********************************************************************

PLACE   2000H

; Tabela de vectores de interrupção

tab:    WORD  rot0

; ********************************************************************
; * Programa Principal
; ********************************************************************

PLACE   0

inicio:
  MOV BTE, tab     ; incializa BTE
  MOV R9, INPUT    ; endereço do porto de entrada
  MOV R10, OUTPUT    ; endereço do porto de Ìsada
  MOV SP, fim_pilha
  MOV R5, 1      ; inicializa estado do processo P1
  MOV R6, 1      ; inicializa estado do processo P2
  MOV R4, OFF      ; inicializa controle de RTC
  MOV R8, 0      ; inicializa contador
  MOV R7, OFF      ; inicialmente não permite contagem
  EI0           ; permite interrupções tipo 0
```

```
45    EI            ; activa interrupções
46
47  ciclo:
48    CALL  P1      ; invoca processo P1
49    CALL  P2      ; invoca processo P2
50    JMP   ciclo   ; repete ciclo
51
52  ; ********************************************************************
53  ;* ROTINAS
54  ; ********************************************************************
55
56  P1:
57    CMP R5, 1     ; se estado = 1
58    JZ  P1_1
59    CMP R5, 2     ; se estado = 2
60    JZ  P1_2
61  sai_P1:
62    RET         ; sai do processo.
63
64
65  P1_1:
66    MOVB  R0, [R9]  ; lê porto de entrada
67    BIT R0, 1
68    JZ  sai_P1    ; se botão não carregado, sai do processo
69    MOV R7, ON    ; permite contagem do display
70    MOV R5, 2   ; passa ao estado 2 do P1
71    JMP sai_P1
72
73  P1_2:
74    MOVB  R0, [R9]  ; lê porto de entrada
75    BIT R0, 1
76    JNZ sai_P1    ; se botão continua carregado, sai do processo
77    MOV R7, OFF   ; caso contrário, desliga contagem do display
78    MOV R5, 1   ; passa ao estado 1 do P1
79    JMP sai_P1
```

Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Phasellus eget nisl ut elit porta ullamcorper. Maecenas tincidunt velit quis orci. Sed in dui. Nullam ut mauris eu mi mollis luctus. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos.

This inline MATLAB code `for i=1:3, disp('cool'); end;` uses the `\mcode{}` command.[1]

Nullam ut mauris eu mi mollis luctus. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Sed cursus cursus velit. Sed a massa. Duis dignissim euismod quam. Nullam euismod metus ut orci.

Listing A.3: `Matlab Function`

```
1  for i = 1:3
2      if i >= 5 && a ~= b       % literate programming replacement
3          disp('cool');         % comment with some LaTeXin it:  πx^2
4      end
5      [:,ind] = max(vec);
6      x_last = x(1,end) - 1;
7      v(end);
8      ylabel('Voltage (μV)');
9  end
```

[1] MATLAB Works also in footnotes: `for i=1:3, disp('cool'); end;`

Nullam ut mauris eu mi mollis luctus. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Sed cursus cursus velit. Sed a massa. Duis dignissim euismod quam. Nullam euismod metus ut orci.

Listing A.4: `function.m`

```
%   Copyright 2010 The MathWorks, Inc.
function ObjTrack(position)
% #codegen
% First, setup the figure
numPts = 300;              % Process and plot 300 samples
figure;hold;grid;          % Prepare plot window
% Main loop
for idx = 1: numPts
    z = position(:,idx);   % Get the input data
    y = kalmanfilter(z);   % Call Kalman filter to estimate the position
    plot_trajectory(z,y);  % Plot the results
end
hold;
end   % of the function
```

Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Phasellus eget nisl ut elit porta ullamcorper. Maecenas tincidunt velit quis orci. Sed in dui. Nullam ut mauris eu mi mollis luctus. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Sed cursus cursus velit. Sed a massa. Duis dignissim euismod quam. Nullam euismod metus ut orci. Vestibulum erat libero, scelerisque et, porttitor et, varius a, leo.

Listing A.5: HTML with CSS Code

```
<!DOCTYPE html>
<html>
  <head>
    <title>Listings Style Test</title>
    <meta charset="UTF-8">
    <style>
      /* CSS Test */
      * {
        padding: 0;
        border: 0;
```

```
        margin: 0;
      }
    </style>
    <link rel="stylesheet" href="css/style.css" />
  </head>
  <header> hey </header>
  <article> this is a article </article>
  <body>
    <!-- Paragraphs are fine -->
    <div id="box">
      <p>
        Hello World
      </p>
      <p>Hello World</p>
      <p id="test">Hello World</p>
      <p></p>
    </div>
    <div>Test</div>
    <!-- HTML script is not consistent -->
    <script src="js/benchmark.js"></script>
    <script>
      function createSquare(x, y) {
        // This is a comment.
        var square = document.createElement('div');
        square.style.width = square.style.height = '50px';
        square.style.backgroundColor = 'blue';

        /*
         * This is another comment.
         */
        square.style.position = 'absolute';
        square.style.left = x + 'px';
        square.style.top = y + 'px';

        var body = document.getElementsByTagName('body')[0];
        body.appendChild(square);
      };

```

```
      // Please take a look at +=
      window.addEventListener('mousedown', function(event) {
        // German umlaut test: Berührungspunkt ermitteln
        var x = event.touches[0].pageX;
        var y = event.touches[0].pageY;
        var lookAtThis += 1;
      });
    </script>
  </body>
</html>
```

Nulla dui purus, eleifend vel, consequat non, dictum porta, nulla. Duis ante mi, laoreet ut, commodo eleifend, cursus nec, lorem. Aenean eu est. Etiam imperdiet turpis. Praesent nec augue. Curabitur ligula quam, rutrum id, tempor sed, consequat ac, dui. Vestibulum accumsan eros nec magna. Vestibulum vitae dui. Vestibulum nec ligula et lorem consequat ullamcorper.

Listing A.6: HTML CSS Javascript Code

```

@media only screen and (min-width: 768px) and (max-width: 991px) {

  #main {
    width: 712px;
    padding: 100px 28px 120px;
  }

  /* .mono {
    font-size: 90%;
  } */

  .cssbtn a {
    margin-top: 10px;
    margin-bottom: 10px;
    width: 60px;
    height: 60px;
    font-size: 28px;
    line-height: 62px;
  }
```

Nulla dui purus, eleifend vel, consequat non, dictum porta, nulla. Duis ante mi, laoreet ut, commodo eleifend, cursus nec, lorem. Aenean eu est. Etiam imperdiet turpis. Praesent nec augue. Curabitur ligula quam, rutrum id, tempor sed, consequat ac, dui. Vestibulum accumsan eros nec magna. Vestibulum vitae dui. Vestibulum nec ligula et lorem consequat ullamcorper.

Listing A.7: PYTHON Code

```
class TelgramRequestHandler(object):
    def handle(self):
        addr = self.client_address[0]          # Client IP-adress
        telgram = self.request.recv(1024)      # Recieve telgram
        print "From: %s, Received: %s" % (addr, telgram)
        return
```

B

A Large Table

Aliquam et nisl vel ligula consectetuer suscipit. Morbi euismod enim eget neque. Donec sagittis massa. Vestibulum quis augue sit amet ipsum laoreet pretium. Nulla facilisi. Duis tincidunt, felis et luctus placerat, ipsum libero vestibulum sem, vitae elementum wisi ipsum a metus. Nulla a enim sed dui hendrerit lobortis. Donec lacinia vulputate magna. Vivamus suscipit lectus at quam. In lectus est, viverra a, ultricies ut, pulvinar vitae, tellus. Donec et lectus et sem rutrum sodales. Morbi cursus. Aliquam a odio. Sed tortor velit, convallis eget, porta interdum, convallis sed, tortor. Phasellus ac libero a lorem auctor mattis. Lorem ipsum dolor sit amet, consectetuer adipiscing elit.

Nunc auctor bibendum eros. Maecenas porta accumsan mauris. Etiam enim enim, elementum sed, bibendum quis, rhoncus non, metus. Fusce neque dolor, adipiscing sed, consectetuer et, lacinia sit amet, quam. Suspendisse wisi quam, consectetuer in, blandit sed, suscipit eu, eros. Etiam ligula enim, tempor ut, blandit nec, mollis eu, lectus. Nam cursus. Vivamus iaculis. Aenean risus purus, pharetra in, blandit quis, gravida a, turpis. Donec nisl. Aenean eget mi. Fusce mattis est id diam. Phasellus faucibus interdum sapien. Duis quis nunc. Sed enim. Nunc auctor bibendum eros. Maecenas porta accumsan mauris. Etiam enim enim, elementum sed, bibendum quis, rhoncus non, metus. Fusce neque dolor, adipiscing sed, consectetuer et, lacinia sit amet, quam.

Table B.1: Example table

Benchmark: ANN	**#Layers** (1)	**#Nets** (2)	#Nodes* (3) = 8 · (1) · (2)	Critical path (4) = 4 · (1)	**Latency (T_{iter})** (5)
A1	**3–1501**	1	**24–12008**	**12–6004**	4
A2	501	1	4008	2004	**2–2000**
A3	10	**2–1024**	**160–81920**	40	60†
A4	10	50	4000	40	**80–1200**

Benchmark: FFT	**FFT size‡** (1)	*#Inputs* (2) = $2^{(1)}$	*#Nodes** (3) = 10 · (1) · (2)	*Critical path* (4) = 4 · (1)	**Latency (T_{iter})** (5)
F1	**1–10**	2–1024	**20–102400**	4–40	6–60†
F2	**5**	32	1600	20	**40 – 1500**

Benchmark: Random networks	**#Types** (1)	**#Nodes** (2)	*#Networks* (3)	*Critical path* (4)	**Latency (T_{iter})** (5)
R1	3	10–2000	500	*variable*	(4)
R2	3	50	500	*variable*	(4) × [1; · · · ; 20]

* *Excluding constant nodes.*
† *Value kept proportional to the critical path:* $(5) = (4) * 1.5$.
‡ *A size of x corresponds to a 2^x point FFT.*
Values in bold indicate the parameter being varied.

As **??** shows, the data can be inserted from a file, in the case of a somehow complex structure. Notice the Table footnotes.

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Morbi commodo, ipsum sed pharetra gravida, orci magna rhoncus neque, id pulvinar odio lorem non turpis. Nullam sit amet enim. Suspendisse id velit vitae ligula volutpat condimentum. Aliquam erat volutpat. Sed quis velit. Nulla facilisi. Nulla libero. Vivamus pharetra posuere sapien. Nam consectetuer. Sed aliquam, nunc eget euismod ullamcorper, lectus nunc ullamcorper orci, fermentum bibendum enim nibh eget ipsum. Donec porttitor ligula eu dolor. Maecenas vitae nulla consequat libero cursus venenatis. Nam magna enim, accumsan eu, blandit sed, blandit a, eros.

And now an example (**??**) of a table that extends to more than one page. Notice the repetition of the Caption (with indication that is continued) and of the Header, as well as the continuation text at the bottom.

Table B.2: Example of a very long table spreading in several pages

	Time (s)	**Triple chosen**	**Other feasible triples**
0	(1, 11, 13725)	(1, 12, 10980), (1, 13, 8235), (2, 2, 0), (3, 1, 0)
2745	(1, 12, 10980)	(1, 13, 8235), (2, 2, 0), (2, 3, 0), (3, 1, 0)
5490	(1, 12, 13725)	(2, 2, 2745), (2, 3, 0), (3, 1, 0)
8235	(1, 12, 16470)	(1, 13, 13725), (2, 2, 2745), (2, 3, 0), (3, 1, 0)
10980	(1, 12, 16470)	(1, 13, 13725), (2, 2, 2745), (2, 3, 0), (3, 1, 0)

Continued on next page

Table B.2 – continued from previous page

Time (s)	Triple chosen	Other feasible triples
13725	(1, 12, 16470)	(1, 13, 13725), (2, 2, 2745), (2, 3, 0), (3, 1, 0)
16470	(1, 13, 16470)	(2, 2, 2745), (2, 3, 0), (3, 1, 0)
19215	(1, 12, 16470)	(1, 13, 13725), (2, 2, 2745), (2, 3, 0), (3, 1, 0)
21960	(1, 12, 16470)	(1, 13, 13725), (2, 2, 2745), (2, 3, 0), (3, 1, 0)
24705	(1, 12, 16470)	(1, 13, 13725), (2, 2, 2745), (2, 3, 0), (3, 1, 0)
27450	(1, 12, 16470)	(1, 13, 13725), (2, 2, 2745), (2, 3, 0), (3, 1, 0)
30195	(2, 2, 2745)	(2, 3, 0), (3, 1, 0)
32940	(1, 13, 16470)	(2, 2, 2745), (2, 3, 0), (3, 1, 0)
35685	(1, 13, 13725)	(2, 2, 2745), (2, 3, 0), (3, 1, 0)
38430	(1, 13, 10980)	(2, 2, 2745), (2, 3, 0), (3, 1, 0)
41175	(1, 12, 13725)	(1, 13, 10980), (2, 2, 2745), (2, 3, 0), (3, 1, 0)
43920	(1, 13, 10980)	(2, 2, 2745), (2, 3, 0), (3, 1, 0)
46665	(2, 2, 2745)	(2, 3, 0), (3, 1, 0)
49410	(2, 2, 2745)	(2, 3, 0), (3, 1, 0)
52155	(1, 12, 16470)	(1, 13, 13725), (2, 2, 2745), (2, 3, 0), (3, 1, 0)
54900	(1, 13, 13725)	(2, 2, 2745), (2, 3, 0), (3, 1, 0)
57645	(1, 13, 13725)	(2, 2, 2745), (2, 3, 0), (3, 1, 0)
60390	(1, 12, 13725)	(2, 2, 2745), (2, 3, 0), (3, 1, 0)
63135	(1, 13, 16470)	(2, 2, 2745), (2, 3, 0), (3, 1, 0)
65880	(1, 13, 16470)	(2, 2, 2745), (2, 3, 0), (3, 1, 0)
68625	(2, 2, 2745)	(2, 3, 0), (3, 1, 0)
71370	(1, 13, 13725)	(2, 2, 2745), (2, 3, 0), (3, 1, 0)
74115	(1, 12, 13725)	(2, 2, 2745), (2, 3, 0), (3, 1, 0)
76860	(1, 13, 13725)	(2, 2, 2745), (2, 3, 0), (3, 1, 0)
79605	(1, 13, 13725)	(2, 2, 2745), (2, 3, 0), (3, 1, 0)
82350	(1, 12, 13725)	(2, 2, 2745), (2, 3, 0), (3, 1, 0)
85095	(1, 12, 13725)	(1, 13, 10980), (2, 2, 2745), (2, 3, 0), (3, 1, 0)
87840	(1, 13, 16470)	(2, 2, 2745), (2, 3, 0), (3, 1, 0)
90585	(1, 13, 16470)	(2, 2, 2745), (2, 3, 0), (3, 1, 0)
93330	(1, 13, 13725)	(2, 2, 2745), (2, 3, 0), (3, 1, 0)
96075	(1, 13, 16470)	(2, 2, 2745), (2, 3, 0), (3, 1, 0)
98820	(1, 13, 16470)	(2, 2, 2745), (2, 3, 0), (3, 1, 0)
101565	(1, 13, 13725)	(2, 2, 2745), (2, 3, 0), (3, 1, 0)
104310	(1, 13, 16470)	(2, 2, 2745), (2, 3, 0), (3, 1, 0)
107055	(1, 13, 13725)	(2, 2, 2745), (2, 3, 0), (3, 1, 0)
109800	(1, 13, 13725)	(2, 2, 2745), (2, 3, 0), (3, 1, 0)
112545	(1, 12, 16470)	(1, 13, 13725), (2, 2, 2745), (2, 3, 0), (3, 1, 0)
115290	(1, 13, 16470)	(2, 2, 2745), (2, 3, 0), (3, 1, 0)
118035	(1, 13, 13725)	(2, 2, 2745), (2, 3, 0), (3, 1, 0)
120780	(1, 13, 16470)	(2, 2, 2745), (2, 3, 0), (3, 1, 0)
123525	(1, 13, 13725)	(2, 2, 2745), (2, 3, 0), (3, 1, 0)
126270	(1, 12, 16470)	(1, 13, 13725), (2, 2, 2745), (2, 3, 0), (3, 1, 0)
129015	(2, 2, 2745)	(2, 3, 0), (3, 1, 0)
131760	(2, 2, 2745)	(2, 3, 0), (3, 1, 0)
134505	(1, 13, 16470)	(2, 2, 2745), (2, 3, 0), (3, 1, 0)
137250	(1, 13, 13725)	(2, 2, 2745), (2, 3, 0), (3, 1, 0)
139995	(2, 2, 2745)	(2, 3, 0), (3, 1, 0)
142740	(2, 2, 2745)	(2, 3, 0), (3, 1, 0)
145485	(1, 12, 16470)	(1, 13, 13725), (2, 2, 2745), (2, 3, 0), (3, 1, 0)
148230	(2, 2, 2745)	(2, 3, 0), (3, 1, 0)

Continued on next page

Table B.2 – continued from previous page

	Time (s)	Triple chosen	Other feasible triples
150975	(1, 13, 16470)	(2, 2, 2745), (2, 3, 0), (3, 1, 0)	
153720	(1, 12, 13725)	(2, 2, 2745), (2, 3, 0), (3, 1, 0)	
156465	(1, 13, 13725)	(2, 2, 2745), (2, 3, 0), (3, 1, 0)	
159210	(1, 13, 13725)	(2, 2, 2745), (2, 3, 0), (3, 1, 0)	
161955	(1, 13, 16470)	(2, 2, 2745), (2, 3, 0), (3, 1, 0)	
164700	(1, 13, 13725)	(2, 2, 2745), (2, 3, 0), (3, 1, 0)	