Models and heuristics for hard routing and knapsack problems

Pierotti, J.

DOI
10.4233/uuid:bad12fb8-ef0a-41ba-b0ae-f5e5ce2fa5fe

Publication date
2022

Document Version
Final published version

Citation (APA)
Pierotti, J. (2022). *Models and heuristics for hard routing and knapsack problems*. [Dissertation (TU Delft), Delft University of Technology]. https://doi.org/10.4233/uuid:bad12fb8-ef0a-41ba-b0ae-f5e5ce2fa5fe

MODELS AND HEURISTICS FOR HARD ROUTING AND KNAPSACK PROBLEMS

MODELS AND HEURISTICS FOR HARD ROUTING AND KNAPSACK PROBLEMS

Proefschrift

ter verkrijging van de graad van doctor
aan de Technische Universiteit Delft,
op gezag van de Rector Magnificus prof. dr ir. T.H.J.J. van der Hagen,
voorzitter van het College voor Promoties,
in het openbaar te verdedigen op dinsdag 26 april 2022 om 12:30 uur

door

Jacopo PIEROTTI

Master of Science in Automation and Control Engineering,
Politecnico di Milano, Milaan, Italië,
geboren te Bergamo, Italië.

Dit proefschrift is goedgekeurd door de

promotor: prof. dr. ir. K.I. Aardal
copromotor: dr. ir. J.T. van Essen

Samenstelling promotiecommissie:

Rector Magnificus,	voorzitter
Prof. dr. ir. K.I. Aardal,	Technische Universiteit Delft
Dr. ir. J.T. van Essen,	Technische Universiteit Delft

Onafhankelijke leden:

Prof. dr. Helena Ramalhinho Lourenço,	University Pompeu Fabra
Prof.dr. Bart van Arem,	Technische Universiteit Delft
Prof.dr. Paola Festa,	University of Naples Federico II
dr. Valentina Morandi,	Free University of Bolzano
Prof. dr. Hai Xiang Lin,	Technische Universiteit Delft, reservelid

Dr. Lorenzo Ferretti, Prof. Dr. Laura Pozzi, Dr. Lina Simeonova, Msc. Maximilian Khronmuller en Dr. W. Böhmer hebben in belangrijke mate aan de totstandkoming van het proefschrift bijgedragen.

Dit onderzoek is deels gefinancierd door het DIAMANT (Discrete, Interactive and Algorithmic Mathematics, Algebra and Number Theory) cluster.

Keywords: Metaheuristics, Logistics, Routing, Integer Linear Programming, Balanced Traveling Salesman Problem, Special Education Needs School Bus Routing Problem, Reinforcement Learning, Knapsack Problem

Printed by: GVO Drukkers en Vermogen B.V

Front & Back: GVO Drukkers en Vermogen B.V.

An electronic version of this dissertation is available at
http://repository.tudelft.nl/.

Je veux dédier ce poème
a toutes les femmes qu'on aime
pendant quelques instants secrets

Antoine Pol

CONTENTS

SUMMARY

One of the world's biggest challenges is that living beings have to share a limited amount of resources. As people of science, we strive to find innovative ways to better use these resources, to reach and positively affect more and more people. In the field of optimization, we aim at finding an optimal allocation of limited sets of resources to maximize a certain objective. Some of these problems can be solved in polynomial time; others are more difficult to be solved. Current state-of-the-art methods can solve NP-hard problems (a class of optimization problems) in exponential time, in the worst case. To give an idea, for input size $n = 100$ and parameter $k = 2$: polynomial time $n^k = 100^2 = 10,000$; exponential time $k^n = 2^{100} = 1,267,650,600,228,229,401,496,703,205,376$. Yet, many relevant and practical problems are NP-hard and have to be solved in a short amount of time. Our research focuses on formulating and solving four of these problems. Among those, three are vehicle routing problems (VRP, Chapters 2, 3 and 4).

VRPs are problems where vehicles have to perform routes in order to minimize an objective function (for example, minimize routing costs) while being subjected to constraints (for example, each location has to be visited). Routing costs have a significant impact on society and on the cost of products (the transportation sector makes up 13.2% of the EU's GDP (Joint Research Centre, 2021)). Although VRPs have been thoroughly studied for over half a century, new technologies (autonomous driving, real-time information, etc.) and new customers' demands (increase in online shopping, a more competitive delivery market, etc.) create variants of the standard VRP that are more and more complex to formulate and solve. VRPs are well-known for being NP-hard and difficult to approximate, and hence solve. We formulated three novel VRPs and solve those both exactly via branch-and-bound (Chapters 3 and 4, the latter also uses valid inequalities) and metaheuristicly (Chapters 2 and 4). To increase generalizability, we introduced an almost non-parametric algorithm that encompasses all the most famous heuristic operators for VRP (Chapter 4). To increase performance, we proposed an adaptive, i.e., self-tuning, algorithm (Chapter 2) that can *detect* problem's features and steer its decisions to achieve better solutions.

Lastly, Chapter 5 focuses on what we believe will be the most radical transformation in the metaheuristic field in coming years: machine learning for combinatorial optimization. Machine learning established its fundamental importance in many fields and it is currently paving its way into combinatorial optimization. We developed a self-attention based deep reinforcement learning algorithm without any problem-specific knowledge to solve one of the most studied combinatorial optimization problems. Our results suggest that machine learning can (and we conjecture that it will) tackle combinatorial optimization on its own, without problem-specific knowledge and will be a fundamental element in future state-of-the-art heuristics for combinatorial optimization.

SAMENVATTING

Een van 's werelds grootste uitdagingen is dat levende wezens een zeer beperkte hoeveelheid middelen moeten delen. Als wetenschapper zoeken we naar innovatieve manieren om deze middelen beter te gebruiken, om zo steeds meer mensen te bereiken en hen positief te beïnvloeden. Bij wiskundige optimalisatie richten we ons op het vinden van de optimale toewijzing van beperkte sets van middelen om een doelstelling te maximaliseren. Soms kunnen we deze problemen in polynomiale tijd oplossen; soms zijn problemen moeilijker op te lossen. Huidige geavanceerde methoden kunnen in het ergste geval NP-moeilijke problemen (een klasse van optimalisatieproblemen) in exponentiële tijd oplossen. Om een idee te geven: polynomiale tijd $n^k = 100^2 = 10.000$; exponentiële tijd $k^n = 2^{100} = 1.267.650.600.228.229.401.496.703.205.376$. Veel relevante en praktische problemen zijn echter NP-hard en moeten in korte tijd worden opgelost. Ons onderzoek richt zich op het formuleren en oplossen van vier van deze problemen. Drie daarvan zijn problemen beschouwen de routebepaling van voertuigen (VRP, hoofdstukken 2, 3 en 4).

VRP's zijn problemen waarbij een aantal voertuigen een aantal routes moet uitvoeren om een doel functie te minimaliseren (bijvoorbeeld de vervoerskosten minimaliseren), terwijl ze aan een aantal beperkingen voldoen (bijvoorbeeld elke locatie moet worden bezocht). Vervoerskosten hebben een aanzienlijke impact op de samenleving en op de kosten van producten (de transportsector vertegenwoordigt 13.2% van het BBP van de EU (Joint Research Centre, 2021)). Hoewel VRP's al meer dan een halve eeuw grondig worden bestudeerd, creëren nieuwe technologieën (autonoom rijden, realtime informatie, enz.) en nieuwe eisen van klanten (toename van online winkelen, een meer concurrerende bezorgmarkt, enz.) variaties op de standaard VRP die steeds complexer worden om te formuleren en op te lossen. VRP's zijn berucht als NP-moeilijke problemen en zijn moeilijk te benaderen (en dus op te lossen). We formuleren drie nieuwe VRP's en lossen die op zowel vanuit een optimaal oogpunt via branch-and-bound (hoofdstukken 3 en 4, de laatste gebruikt ook geldige ongelijkheden) als vanuit een metaheuristisch oogpunt (hoofdstukken 2 en 4). Om dit verder te generaliseren, hebben we een bijna aparametrisch algoritme geïntroduceerd dat alle meest bekende heuristische operatoren voor VRP omvat (hoofdstuk 4). Daarnaast hebben we om de prestaties te verbeteren, een adaptief, d.w.z. zelfafstemmend, algoritme voorgesteld (hoofdstuk 2) dat de kenmerken van het probleem *begrijpt* en beslissingen kan sturen om de winst te maximaliseren.

Ten slotte richt hoofdstuk 5 zich op wat volgens ons de meest radicale transformatie in het metaheuristische veld in de komende jaren zal zijn: machine learning voor combinatorische optimalisatie. Machine learning heeft zijn fundamentele belang op veel gebieden bewezen en baant zich momenteel een weg naar combinatorische optimalisatie. We hebben een op self-attention gebaseerd deep reinforcement learning algoritme ontwikkeld zonder enige probleemspecifieke kennis om een van de meest bestudeerde combinatorische optimalisatieproblemen op te lossen. Onze resultaten suggereren dat machine learning de combinatorische optimalisatie op zichzelf kan (en we vermoeden

zal) aanpakken, zonder probleemspecifieke kennis, en een fundamenteel element zal zijn in toekomstige geavanceerde heuristieken voor combinatorische optimalisatie.

1

INTRODUCTION

This dissertation treats a wide variety of topics. In order to guide the reader, Figure 1.1 displays the relationships between the chapters of this dissertation and their research fields (combinatorial optimization and artificial intelligence), their real-life applications (vehicle routing problem and knapsack problem), their complexity (NP-hard) and the used solution methods (heuristic and optimal). All the aforementioned terminology is explained in detail in the remainder of this chapter.

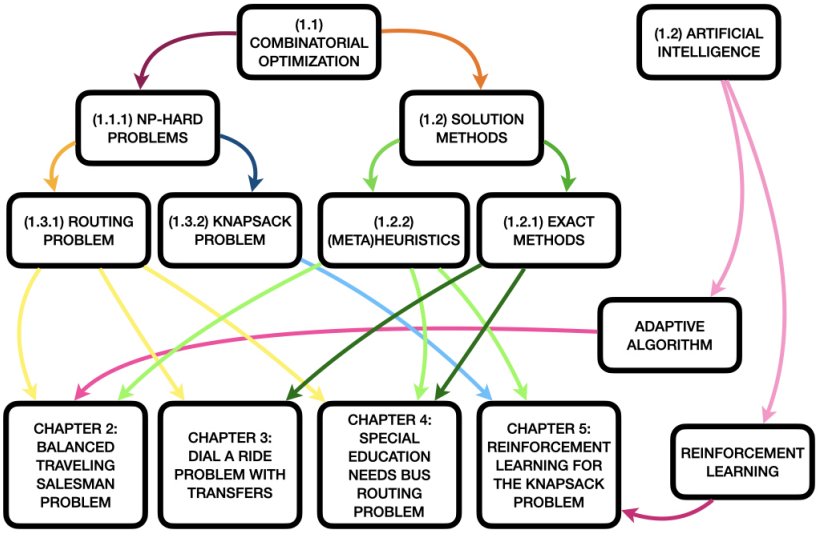


Figure 1.1: Connections among the main topics treated in this dissertation and its chapters. The numbers in the boxes indicate the sections where these topics are discussed.

1

1.1. COMBINATORIAL OPTIMIZATION

Combinatorial optimization (CO) is hidden in uncountable many aspects of everyday life: where a business should locate its facilities, how to take the shortest path from home to work, how to assign weapons to targets, how to schedule doctors and nurses in a hospital, etc. All these problems and many other problems can be formulated (and solved) as combinatorial optimization problems. In general, we can say that CO involves finding an optimal object within a finite set of objects (Schrijver, 2003). CO problems can be formulated naturally in terms of graphs and/or as (integer) linear programs (see Section 1.1.2) and solving them can be a very complex task (see Section 1.1.1).

1.1.1. COMPLEXITY

There is a lot of open debate about the complexity of CO problems. When we talk about the complexity of a CO problems, we are talking about the number of steps needed in order to solve them. CO problems can be divided in *decision* and *optimization* problems. A combinatorial optimization problem deals with finding the optimal solution among a finite set of possible solutions while a decision problem concerns answering a *yes* or *no* question. For instance, 'what is the minimum distance to travel to visit a set of cities?' is an optimization problem while 'is there a path which visits a set of cities?' is a decision problem. Optimization problems can be transformed into decision problems by introducing a threshold, say t. For instance, 'can a set of cities be visited within a travelling distance of t?' is a decision problem.

Among combinatorial optimization decision problems, the two most studied complexity classes are the P and the NP class. The P class contains the decision problems whose solutions can be surely found in a polynomial (with respect to the input size) number of steps. The NP class is an extension of the P-class and it also contains the decision problems for which we are uncertain about the minimum number of steps needed to obtain an answer in the worst case (but the number of steps is upperbounded by an exponential). However, a yes-solution to any decision problem in NP (and thus also in P) can be verified in polynomial time.

Within NP, there is a class of decision problems called NP-complete. If one NP-complete problem could be solved in polynomial time even in its worst case, then, all NP-complete problems could also be solved in polynomial time (Goldreich, 2010). Yet, there exists no known method to ensure that a NP-complete can be solved in polynomial time and, so far, most of the scientific community believes that a solution to an NP-complete problem can be found even in its worst case only in an exponential number of steps. To explain the difference between a problem in P and a NP-complete problem, let us give an example with an instance size n of 100. Depending on the problem, an instance of size 100 can be a reasonably sized instance. For example, in Chapters 2, 4 and 5 we solve instances up to size 3000, 168 and 100, respectively. For a constant coefficient $k = 2$, a polynomial number of steps would result in $n^k = 100^2 = 10,000$ steps, while an exponential number of steps would results in $k^n = 2^{100} = 1,267,650,600,228,229,401,496,703,205,376$ steps[1]. This means that, even-

[1] This example is just indicative. The number of steps needed could be scaled by a constant scaling factor. Moreover, given the same instance size, the k coefficients for the P and the NP-complete problem could be different. For the sake of the example, we assume the k coefficients to be the same and we assume the scaling

tually, as the instances' sizes become bigger and bigger, they will require too much time to be solved to optimality. Even if we were able to use the computational power of all the hardware in the world, a real-world instance could easily still require years or more to be solved to optimality.

All problems treated in this dissertation are NP-hard optimization problems. An optimization problem is NP-hard when the associated decision problem is NP-complete. If a problem is NP-hard, it means that it is, at least, as complex as any NP-complete problem. Moreover, there is no known method to surely verify the optimality of solutions to NP-hard optimization problems in less than exponential time. There are still many open questions regarding NP complexity and this subject is of such paramount importance that it earned a position in the seven mathematical millennium prize problems[2].

1.1.2. LINEAR PROGRAMMING MODELS

Many real-life optimization problems can be formulated as linear programs (LP). Linear programs are models where a linear objective function is maximised (or, equivalently, minimised) while being subjected to some linear constraints (Korte et al., 2011). For example, one can be sitting hungry at a restaurant while trying to minimize expenses. The optimization problem could be to eat enough to achieve at least a minimum number of calories while spending the minimum amount possible. This can be easily formulated as a linear program. In this case, the variables would be the quantities of the items on the menu and the objective would be the sum of the prices of the menu's items times their quantity (i.e., the total price of the final bill). In this little example, the only constraints would be the following. The sum of the quantities of the items times their caloric intake should be greater than or equal to a certain threshold and the quantities should be greater than or equal to zero.

If we solve this problem to optimality, we would find a solution in polynomial time; yet, this solution may indicate to order $\frac{2}{3}$ of a meal. In fact, linear problems belong to the P-class but, when integrality constraints are added, the problems *might* become NP-hard. Integrality constraints are special constraints that state that one or more variables have to assume only integer values instead of being able to assume continuous values. This is indeed necessary in many applications, as, for instance, one can only order an integer number of meals. When a problem allows only for integer variables, we call that problem an integer linear programming (ILP) problem; if, instead, some variables are integer and some are continuous, the problem is called a mixed integer linear programming (MILP) problem. In general, ILP and MILP belong to the same class of complexity (NP-hard). When the variables of an ILP can assume only a finite number of integer values, the optimization problem is to find the optimal solution among a finite set of objects, hence, it is a combinatorial optimization problem.

We stated that a linear program *might* become NP-hard after introducing integrality constraints. There are some notable cases, such as the shortest path problem, where problems stay in the P class even after introducing integrality. In the remainder of this dissertation, we only deal with NP-hard problems.

factors to be one.

[2] The seven millennium prize problems are considered some of the most fundamentals and groundbreaking problems in current days mathematics (https://en.wikipedia.org/wiki/Millennium_Prize_Problems).

1

1.2. SOLUTION METHODS

All the problems treated in this dissertation are NP-hard and, to solve them, we either rely on classical exact techniques (which return one optimal solution, see Section 1.2.1, or a family of optimal solutions, see Section 1.2.4) or we design (meta)heuristics (algorithms that quickly return not-necessarily-optimal solutions), both using classical techniques (see Section 1.2.2)) or with newer artificial intelligent techniques (see Section 1.2.3).

1.2.1. EXACT SOLUTION METHOD

Linear problems can be solved in polynomial time and there are many ways to solve them. Internal point methods (Kojima et al., 1989) can solve LPs in polynomial time while the simplex method (Dantzig et al., 1955) solves LPs, in the worst case, in exponential time. The commercial software used in this dissertation to solve LPs (Gurobi Optimization, LLC, 2021), uses a combination of the simplex method and the barrier method (Lange, 1994). Also to solve (M)ILPs, algorithms for solving LPs play an important role. There exists a close relationship between MILPs and LPs; in fact, whenever we remove the integrality constraints from a MILP, we obtain an LP. This process of removing integrality constraints is called *relaxation*. A linear relaxation of a (M)ILP is an LP with the same variables, objective and constraints as the original (M)ILP, except the integrality constraints. Without loss of generality, we assume to solve an ILP whose objective function is a maximization one. Note that the linear relaxation always exhibits a (non-strictly) higher objective function value than its integer counterpart. This is due to the fact that the linear relaxation can reach, at the very least, all the solutions of the ILP.

Firstly, to solve an ILP, we solve its linear relaxation. This can be done in polynomial time since the relaxation is an LP. Once the linear relaxation is solved, we check if all the variables have integer values. If that is not the case, we can branch the problem. In this case, to branch means to generate two subproblems from the original one. For instance, considering the previous example, the solution of the linear relaxation could be to order $\frac{4}{3}$ dishes of pasta. We can then branch the problem into two different subproblems. On one side, we impose the additional constraint to order *at most* one dish of pasta; on the other side, we impose to order *at least* two dishes of pasta. Then, we solve again the two subproblems (which are still LPs). Doing so, we do not allow the algorithm to choose again $\frac{4}{3}$ dishes of pasta. Note that, due to the extra constraint(s), the objective function value of a subproblem cannot be greater than the objective function value of its original maximization problem.

This process of branching is iterated on subproblems that return a non-integer solution until the optimal integer solution is found. However, as soon as a subproblem returns an integer solution, we can prune unpromising branches. In fact, each returned integer solution might or might not be the optimal solution, but we know for sure that the optimal solution presents a greater or equal objective function value. So, if the relaxation of a subproblem presents an objective function value which is smaller than the objective function value of the best known integer solution, we can discard that branch and prune it.

All in all, we have a directed rooted tree[3] of LPs, with relaxed objective function values and a best known integer objective function value. The process terminates when the objective function values of the leaves are not strictly greater than the objective value of the best known integer solution and we return the best known integer solution as the optimal solution.

The method described above is a basic version of the branch-and-bound (B&B, Land and Doig, 1960). B&B is the most used method to optimally solve NP-hard problems and it is the method used in this dissertation to solve MILPs. In addition, there exist plenty of speed-ups one can apply to the B&B (on which variable to branch, in which order one should solve the subproblems, etc.). These speed-ups are already implemented in the commercial solver used in this dissertation (Gurobi Optimization, LLC, 2021).

1.2.2. HEURISTICS

Although there is no unanimous consensus in the optimization community, in this dissertation, we name a heuristic any algorithm that aims to quickly (meaning in polynomial time with respect to the number of variables of a problem) return a solution. More specifically, a heuristic can either return a solution in polynomial time or cannot return a solution at all. In case a solution is not returned, the heuristic could be bad designed or even finding a feasible solution could require a number of steps which is more than polynomial. Given an instance and a fixed random seed, a heuristic always returns the same solution, independent of how many times the heuristic has been run. We name a metaheuristic an algorithm that returns a solution and the returned solution depends on the computational runtime of the metaheuristic itself (the maximum computational runtime is often a design parameter). The solution returned by a metaheuristic depends on its runtime because metaheuristics are designed such that, given *enough time*, they can escape local minima. This means that: if we let a metaheuristic run for t_0 computational time, it might return solution s_0, while, if we let it run for t_1 computational time, it might return solution $s_1 \neq s_0$. Usually, a metaheuristic quickly finds a solution (just as a heuristic) and later it keeps trying to iteratively improve it. The improvement is usually time-dependent, with more computation time leading to a better improvement.

Some heuristic algorithms, called *approximation algorithms*, can return solutions within some performance guarantee; for instance, an approximation algorithm could return a solution which is, at most, twice as bad as the optimal solution. Even one of the most standard combinatorial (routing) problems, the traveling salesman problem (definition in Section 1.3.1), in its most general version, does not have any known polynomial-time approximation algorithm: in fact, even finding a feasible solution is proven to be NP-hard (Hartmanis, 1982). Only if we introduce assumptions on the graph (triangular inequality, fully connected graph, undirected and non-degenerative edges), then there exist approximation algorithms (Edmonds, 1965, Christofides, 1976). However, approximation algorithms do not yield competitive results in practice (Toth and Vigo, 2002).

[3] A directed rooted tree is a mathematical object where a root (in our case, the relaxation of the original problem) is branched into nodes. The edges are oriented from the parent to the children, i.e. from the root to the nodes. The nodes can also be branched into other nodes and different parents cannot direct to the same child. The nodes which are not branched, i.e., they have no outgoing edges, are called leaves (Bollobás, 1998)

Despite the existence of hundreds of metaheuristics, they all have a general common structure. Metaheuristics work iteratively, starting from one (or more) solutions and trying to improve it *locally*. By locally, we mean that, generally, only *small* parts of a solution can be changed and improved. We limit ourselves to changing *small* parts because of computational efficiency. Once a locally optimum solution has been found and it cannot be further improved by local moves, the solution is *broadly* changed. Doing so, we move to a different part of the solution space where local moves are potentially effective. At the end of the process, the overall best solution found is returned.

Generally speaking, if, in infinite time, a metaheuristic can return the optimal solution, it means that it can navigate the whole solution space. This means that, independently from where the algorithm starts, it can reach any possible solution. Usually, to allow the algorithm to navigate the whole solution space we introduce *random* moves. In fact, although random moves may decrease the quality of a solution (since they do not take quality into account and they are usually applied to locally optimum solutions), they allow to move away from an already well-explored area of the solution space.

1.2.3. ARTIFICIAL INTELLIGENCE

Artificial intelligence (AI) is the field that studies any system that perceives its environment and takes actions that maximize its chance of achieving its goals (Poole et al., 1998). Within AI, machine learning (ML) is the study of computer algorithms that can improve their performance automatically through experience and by the use of data (Mitchell, 1997). In recent years, ML has shown state-of-the-art capabilities in speech recognition, language translation, image classification, etc. (O'Shea and Nash, 2015, Bontemps et al., 2016, Vaswani et al., 2017). Lately, more and more combinatorial optimization problems have been studied under the lens of machine learning (Bengio et al., 2021).

On one hand, ML has been applied to the branch-and-bound process, assisting in making smart decisions on how to branch to reduce the computation time drastically (Lodi and Zarpellon, 2017, Balcan et al., 2018, Prouvost et al., 2020). Nonetheless, even these state-of-the-art fine-tuned exact methods take too much time to determine the optimal solution for medium to big size instances. On the other hand, in recent years, heuristics have been hybridized with more and more elements of machine learning (Mayer et al., 2018, Nazari et al., 2018).

When ML is used to quickly generate a single solution from an input instance, it is often called an *end-to-end* method. Note that, since always the same solution is returned, this method classifies as a heuristic but not as a metaheuristic. Among the many machine learning algorithms (supervised, unsupervised,...), in this dissertation (Chapter 5), we focus on one specific type of ML algorithm: an end-to-end reinforcement learning (Sutton and Barto, 2018) algorithm for a CO problem. With reinforcement learning (especially 'end-to-end' reinforcement learning), an agent learns on its own while interacting with the environment. In our case, the environment is the collection of instances, constraints, objective function and such related to the CO problem we are analysing. The reinforcement learning algorithm can act on and observe the environment but cannot change its dynamics. For instance, the algorithm can read an instance (i.e., observe) propose the variables' values for a solution (i.e., act) and analyze the objective function of the proposed solution (i.e., observe again). However, it cannot modify any constraint or

the objective function. In theory, since the algorithm is exploring and creating a decision policy on its own and it is not imitating good known solutions, it could learn a strategy that is better than any possible strategy humans could apply on their own. This is the difference between algorithms performing at *human expert* levels and algorithms breaking those barriers. In practice, this has already happened multiple times, for example with AlphaGO[4] and AlphaZero (Silver et al., 2016, Silver et al., 2017).

In addition to ML, there are other forms of artificial intelligence that have been used in heuristics. One example is adaptiveness. Adaptiveness is a feature by which an algorithm can modify the probability to select some operator based on their performance (Lourenço et al., 2003). As stated before, Poole et al., 1998 defines *intelligent* any system that perceives its environment and takes actions that maximizes its chance of achieving its goals. In Chapter 2, we describe an adaptive algorithm. An adaptive algorithm is an algorithm that uses heuristics (called *operators*) as subroutines. The parameters of the adaptive algorithm (for example, the parameters to decide which operator to use) can be modified in time. Indeed, our algorithm has to iteratively choose among many possible operators and based on their outcomes (i.e., perceiving the environment), the algorithm modifies their successive selection probabilities (i.e., takes actions) in order to increase the odds of choosing the most performing operators (i.e., to maximize its chance of achieving its goals). So, our adaptive algorithm is to be considered as an artificially intelligent algorithm.

1.2.4. PARETO FRONTIER

So far we assumed that the problems to be solved were single objective, i.e., the objective function value is a scalar. However, in this dissertation, we also treat multi-objective problems (see Chapter 4). As single-objective problems, also multi-objective problems can be solved both optimally or heuristically. To solve a multi-objective problem means to identify the set of solutions that are non-dominated (i.e. the Pareto front or Pareto frontier, Censor, 1977). A solution, say s, is *dominated* if there exists another solution, say s', with the same or *better* objectives than s and at least one being strictly *better*. In the previous sentence, *better* means greater or smaller, depending on whether we are maximizing or minimizing that objective. Clearly, this adds complexity to the problem because, instead of looking for an optimal solution, we are now looking for a set of optimal solutions.

1.3. PROBLEMS

In this dissertation, we treat two types of CO problems: routing problems and the knapsack problem. Section 1.3.1 illustrates the family of routing problems and Section 1.3.2 portrays the knapsack problem.

1.3.1. ROUTING PROBLEMS

Food delivery, international shipping of goods from production to consumption countries, same-day delivery, ride hailing and freight transportation are just a few of the many

[4] Free documentary available at: https://www.youtube.com/watch?v=WXuK6gekU1Y, website accessed on September 2020

problems falling under the umbrella of routing problems (Toth and Vigo, 2002). These problems are tremendously relevant for transportation, shipping and logistic companies as well as for any other businesses that rely on some sort of delivery.

Basically all routing problems (with a few exceptions, e.g. the shortest path problem (Dijkstra et al., 1959)) are NP-hard and, on top of that, they are also difficult to approximate. In our opinion, routing problems are just as complicated as they are interesting. In fact, their NP-hardness and complexity in being solved make them notably suitable for being benchmarks for (meta)heuristics testing. Moreover, in our global, connected, consumerist and fast-paced world, we often take for granted how beautifully complex it can be to manage such a large and unpredictable environment. Plenty of requests are placed every second, tons of goods have to be shipped every hour, deadlines are getting tighter and tighter, customers' satisfaction standards are steadily increasing and many other factors are the reasons why the logistic sector generates more and more complex problems to be solved. These new and "always closer to reality" problems are often referred to as *rich* vehicle routing problems (rich VRP or RVRP). In addition, these problems are of decisive impact for companies since transportation costs are usually a significant component of the cost of a product (about 10%, Rodrigue et al., 2016).

In this dissertation, we analyze and solve three different VRP problems, all of which are described in the following paragraphs.

THE BALANCED TRAVELING SALESMAN PROBLEM

The balanced traveling salesman problem (BTSP) is a CO problem that was first introduced at the Metaheuristics Summer School 2018[5] under the format of a competition. The BTSP is a variation of the traveling salesman problem (TSP, Lawler et al., 1985) which, in turn, is a special case of VRP and it is likely to be the most studied combinatorial optimization problem ever (Applegate et al., 2011). In the TSP, a salesman has to travel among a set of cities while making a Hamiltonian tour (i.e. a cycle which visits each node exactly once). The objective of the salesman is to minimize the travelled distance. Given its main application to logistics, distances are all strictly positive. In Figure 1.2, we show an example of a graph and some different ways a salesman can travel it.

The BTSP is a variation of the TSP where arcs are associated with a cost, that can either be positive or negative. The objective is to find an Hamiltonian cycle whose cost is as close as possible to zero. This problem can find applications in, for instance, on-route charging, where an electric vehicle can travel specific segments that can recharge its battery without having to stop. In this way, some arcs have a negative cost (energy spent to travel), others have a positive cost (energy recharged) and the objective is to return the vehicle at its starting point with a battery -more or less- as charged as when it left.

THE DIAL-A-RIDE PROBLEM WITH TRANSFERS

Thirty years ago, virtually nobody could expect the arrival of ride-hailing companies such as Uber, Lyft, Curb or such. Now, some of those appear in Fortune 500[6]. Similarly, it was difficult to forecast the role that ride-sharing and car-pooling would have

[5] MESS2018 https://www.ants-lab.it/mess2018/, website accessed: August 2021

[6] Fortune 500 is the list of the most successful U.S. businesses, https://fortune.com/company/uber-technologies/fortune500/ website accessed in August 2021

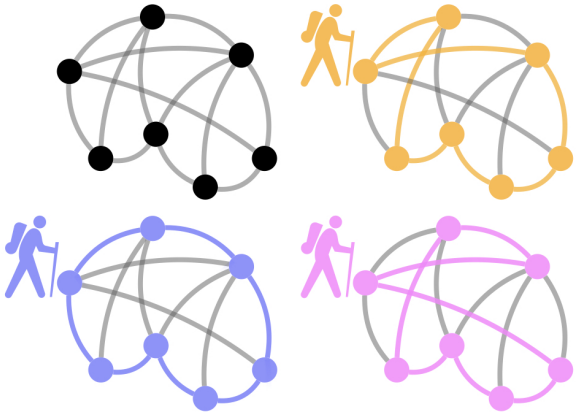


Figure 1.2: From top left, clockwise, the graph and three possible Hamiltonian tours. In grey, the non-travelled edges, in yellow, blue and pink, the travelled ones.

assumed in today's society. The next transportation revolution is difficult to forecast, but it will, most likely, involve self-driving vehicles. These autonomous vehicles have the potential to transform personal mobility from private assets to on-demand services. In addition, due to real-time information technologies, these vehicles can be coordinated via an intelligent system aimed at optimizing the system optimum rather than the users' optimum.

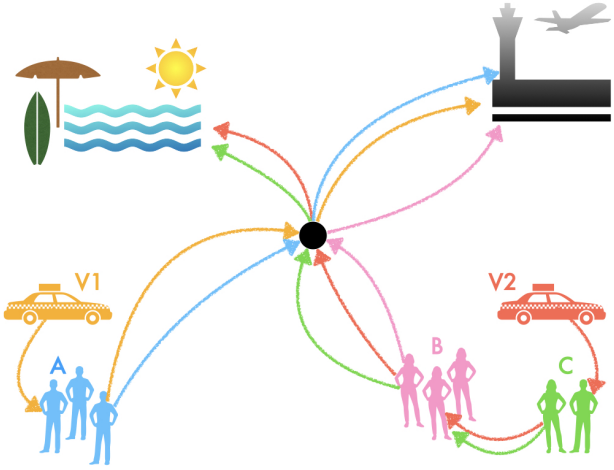


Figure 1.3: Example of transfer. On the top, two different destinations; on the bottom, vehicles (in yellow and red) and passengers (in blue, pink and green). The colour of each arrow represents the vehicle or passenger it refers to.

Another feature that the next transportation revolution might embrace is the pos-

sibility of inter-vehicles transfers. We explain this via an example (see Figure 1.3). Let us suppose to have three groups of people (A, B and C) requesting a ride. Imagine that vehicle V_1 is parked close to group A while group B and C are located fairly close to vehicle V_2. Finally, assume that groups A and B want to travel to the airport while group C wants to go to the beach. A possible solution involving transfers is the following. Vehicle V_1, carrying group A, meets vehicle V_2, carrying groups B and C, in a convenient spot where group B transfers from V_2 to V_1. Afterwards, one vehicle goes to the airport while the other goes to the beach.

SPECIAL EDUCATION NEEDS SCHOOL BUS ROUTING PROBLEM

While delivering goods, minimizing costs is the main goal in mind; yet, this might not be of such paramount importance in other contexts. For instance, when dealing with people transportation, offering a good quality service may be more relevant than saving a (small) portion of the costs. Clearly, there exists a trade-off between saving costs and the quality of the service. How much one should reduce the quality of the service to save transportation cost is a difficult question to answer.

Quality of service becomes even more important when transporting students with disabilities (such as metachromatic leukodystrophy, attention deficit hyperactivity disorder, obsessive-compulsive disorder, etc.). since their well-being can easily be altered. On one hand, transporting students with disabilities is quite expensive because special vehicles and dedicated staff are required. On the other hand, blindly trying to minimize costs leads to the design of routes that do not take into consideration the personal preferences of the students who could be distressed when riding with people they are unfamiliar with. A graphical example can be found in Figure 1.4.

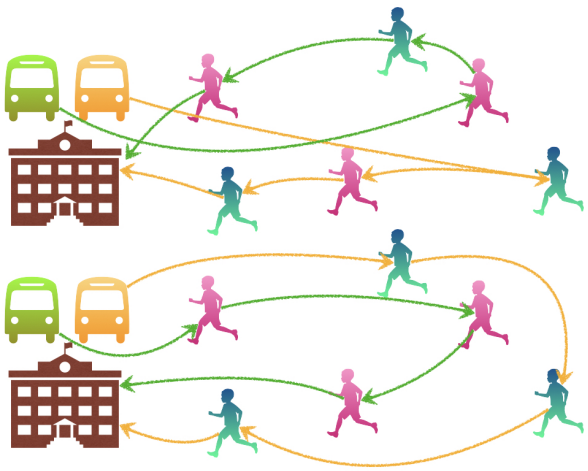


Figure 1.4: Example of two possible routings, on the left, the school and the busses. On the right, the students. Students that are comfortable with each other are depicted in the same colour. On the top, a routing aimed at minimizing travel costs; on the bottom, a routing aimed at maximizing quality of the service.

Up to now, the problems treated were single-objective problems; in this case instead,

we are solving a multi-objective (bi-objective) problem since we are optimizing at the same time for minimizing costs and maximizing the quality of the service.

1.3.2. KNAPSACK PROBLEM

Chapter 5 investigates a new take on one of the oldest NP-hard combinatorial problems, the knapsack problem (Salkin and De Kluyver, 1975). The anecdote around the knapsack problem is the following (see Figure 1.5 for a graphical representation).

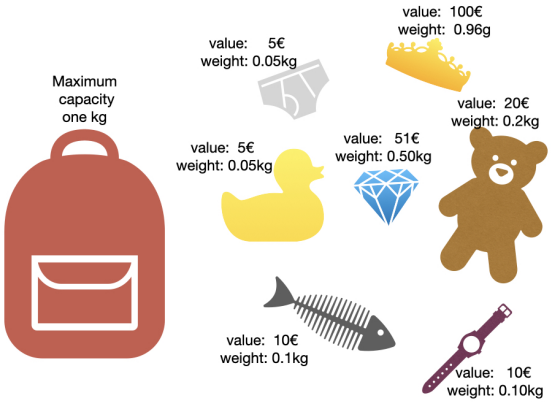


Figure 1.5: Example of a possible instance of the knapsack problem. On the left, a knapsack of maximum capacity one kilogram, on the right, a set of selectable objects, each one with its own weight and value.

Imagine you are a thief in a room full of objects, each having a value and a weight. You have only one knapsack where to store the stolen objects. Unfortunately, your knapsack is fragile and can hold only to a certain amount of weight. How do you choose which objects to pick to maximize your reward without violating the capacity constraint? Would you pick the most valuable objects first and fill the remaining room in the knapsack with small objects? Would you pick objects based on their ratio of value divided by weight? None of these greedy techniques can guarantee an optimal solution; however, heuristics can return excellent results in polynomial time.

1.4. DISSERTATION OUTLINE

The remainder of this chapter explains which problems are treated in each chapter and how we dealt with them (see also Figure 1.1).

Chapter 2 discusses how to find good solutions for the balanced traveling salesman problem with an adaptive metaheuristic. The BTSP was first introduced as a competition, to which we participated obtaining valid results. In fact, our algorithm achieved the best known solution in many instances and ranked fifth in the overall competition ranking.

In Chapter 3, we investigate how to precisely model and formulate the Dial-a-Ride problem with transfers. Given its real-life application, we decided to design a quite com-

plex and articulated model to capture many of the dynamics involved in people transportation. Afterwards, we found, via a commercial solver, optimal routes for small instances of this problem based on real-data from the region of South-Holland. Also, we examine how much the possibility of transfers affects the quality of the optimal solutions.

In Chapter 4, we explore the special education needs school bus routing problem (both from an optimal and a metaheuristic point of view), focusing especially on how to find the family of optimal solutions. We tested our metaheuristic on small instances of which we were able to retrieve some of the points belonging to the Pareto front via exact techniques. Once we assessed the performance of the metaheuristic on small instances and thanks to the opportunity of working with a special education needs school in the county of Kent (South-West England), we tested the metaheuristic on a real-life case study, obtaining various different solution points.

Finally, inspired by the results obtained in other fields by machine learning, we decided to apply end-to-end deep reinforcement learning with little to no human knowledge input, to one of the most classical combinatorial optimization problems, the knapsack problem. Our result came extremely close (within 1%) to optimality suggesting that RL can very well be an efficient tool to tackle combinatorial optimization problems.

2

Adaptive Iterated Local Search with Random Restarts for the Balanced Travelling Salesman Problem

Jacopo Pierotti, Lorenzo Ferretti, Laura Pozzi, Theresia van Essen

Metaheuristics have been widely used to solve NP-hard problems with excellent results. Among all NP-hard problems, the Travelling Salesman Problem (TSP) is potentially the most studied one. In this chapter, a variation of the TSP is considered; the main differences being, edges may have positive or negative costs and the objective is to return a Hamiltonian cycle with cost as close as possible to zero. This variation is called the balanced TSP (BTSP). To tackle this new problem, we present an adaptive variant of the iterated local search metaheuristic featuring also random restart. This algorithm was tested on the MESS2018 metaheuristic competition and achieved notable results, scoring the 5th position overall. In this chapter, we detail all the components of the algorithm itself and present the best solutions identified. Even though this metaheuristic was tailored for the BTSP, with small modifications its structure can be applied to virtually any NP-hard problem. In particular, we introduce the uneven reward-and-punishment rule which is a powerful tool, applicable in many contexts where fast responses to dynamic changes are crucial.

Parts of this chapter have been published in "Metaheuristics for Combinatorial Optimization", 2021 doi: 10.1007978-3-030-68520-1

2

2.1. INTRODUCTION

In the Travelling Salesman Problem (TSP), a salesman has to visit a given set of cities and, after travelling along all of them, has to return to the one he started from, hence, completing a cycle. Given a set of cities V and a cost matrix D with entries $d_{i,j}$ $(i, j \in V)$ –in the *standard* TSP, costs represent the distance between cities–, the goal of the traveller is to find the cycle of minimum cost covering all the cities. The TSP is an NP-hard problem and, therefore, there are no known techniques able to provide an optimal solution efficiently. In this work, we consider a variation of the *standard* TSP. In particular, we address the *balanced* TSP (BTSP). In this variant, the entries of the cost matrix can be negative and the goal of the problem is to find the cycle, visiting all the cities, with cost as close as possible to zero. Despite the complexity of these problems, the TSP and its variants are widely studied and used in many different contexts. Typical applications are: clustering (Lenstra, 1974), vehicle routing (Lenstra and Kan, 1981), computer wiring (Lenstra and Kan, 1975), wallpaper cutting (Hahsler and Hornik, 2007), job-scheduling (Whitley et al., 1989), DNA sequencing (Caserta and Voß, 2014) and pattern-allocation (Madsen, 1988) problems. To cope with these problems and their complexity, many approaches have been proposed in the literature. The main discriminant among them is their ability to obtain an optimal solution or to rapidly discover a near-optimal one. In the latter, there is a trade off between the time required to identify the solution and the quality of the solution provided.

Among the different approaches proposed in literature, some of the most relevant are: Genetic Algorithms (GA), Ant Colony Optimisation (ACO), Tabu Search (TS), Adaptive Large Neighbourhood Search (ALNS), Simulating Annealing (SA), Local Search (LS) and Iterated Local Search (ILS). The approach proposed by Juneja et al., 2019 exploits the ability of population-based heuristics to search for multiple solutions in each iteration of the algorithm and, by using various combinations of selection, crossover and mutation techniques, to continuously improve the quality of current solutions. Dorigo and Gambardella, 1997 were the first to introduce the possibility to use ACO, a heuristic algorithm which navigates the solution space by mimicking ants finding food as a group, as a viable strategy to solve the TSP. More recently, Escario et al., 2015 have refined this approach introducing different types of agents –specialised ants– and population dynamics to organise the ants' movements. The TS approach proposed by Toth and Vigo, 2003 is based on the use of restricted neighbourhoods, allowing to reduce the solution space and leading to a more efficient implementation of candidate strategies proposed for tabu search algorithms. The ALNS heuristic, proposed by Ribeiro and Laporte, 2012, is based on the algorithm initially devised by Ropke and Pisinger, 2006 and extends the large neighbourhood search of Shaw, 1997 by using destroy and repair methods within the same search process. Differently from the previous metaheuristics, SA techniques mimic the metal annealing process by considering probabilistic moves depending on a temperature parameter. The SA methodologies proposed by Geng et al., 2011 and Malek et al., 1989 have been adopted for the TSP resulting in a viable alternative to the above mentioned metaheuristics. Lastly, LS and ILS techniques have been vastly used to solve

the TSP, see Johnson, 1990, Voudouris and Tsang, 1999 and Paquete and Stützle, 2003. As described in Lourenço et al., 2003, by using LS, a sequence of viable solutions is iteratively generated within the embedded heuristic.

In this work, we propose a variation of the ILS technique. ILS, differently from LS techniques, alternates the search phase with the perturbation phase in order to escape tenacious local optima. Given a graph, our approach searches for a Hamiltonian cycle within it and, by iterating over a sequence of actions operating on the current solution, navigates the neighbourhood of the current solution while searching for local optima. Perturbations are then applied in order to escape local optima, hence searching different regions of the solution space. The main contribution of this work consists in the introduction of an *uneven reward-and-punishment* adaptation rule (see Section 2.3.2), which in turn leads to a more reactive response to the current solution.

The chapter continues as follows: Sect. 2.2 presents the problem formulation, Sect. 2.3 describes the advantages of ILS techniques and introduces as proposed methodology an Adaptive Iterated Local Search with Random Restarts (AILS-RR). Sect. 2.4 shows experimental evidence for the effectiveness of the AILS-RR technique introduced and finally, Sect. 2.5 concludes the chapter and presents final remarks.

2.2. PROBLEM FORMULATION

The goal of the TSP is to minimise the cost of the cycle connecting a set of given cities that a salesman has to visit exactly once. The cycle of the salesman can be defined as a sub-graph of graph $G(V,E)$ where V is the set of vertices –cities– and E is the set of edges of the graph representing the connections between cities. Given two cities $i, j \in V$, we define d_{ij} as the cost of travelling from city i to city j. In addition, we define a binary variable x_{ij} which specifies if the cycle of the salesman includes the edge from city i to city j as follows:

$$x_{ij} = \begin{cases} 1, & \text{if edge } ij \in E \text{ is in the cycle of the salesman} \\ 0, & \text{otherwise.} \end{cases} \tag{2.1}$$

Then, we define the total cost of the cycle as:

$$C_{total} = \sum_{ij \in E} d_{ij} x_{ij}. \tag{2.2}$$

In the *classic* TSP, all costs are strictly positive and the objective is to find the Hamiltonian cycle of minimum cost. In the BTSP, costs can be either positive or negative and the objective is to find the Hamiltonian cycle of cost as close as possible to zero. A Hamiltonian cycle is a connected sequence of edges that joins a sequence of vertices, such that each vertex in V is visited exactly once and the edge sequence is closed. A closed sequence starts from a vertex i and, through an ordered sequence of vertices connected by edges, returns to the original vertex i. Given an ordered sequence, we name adjacent vertices of a generic vertex its previous and its following vertex. In general, graphs are not fully connected, i.e. not all vertices $i, j \in V$ are connected by a direct edge. Edges are undirected and each of them is associated with a cost which can be either positive or negative. We define the degree of a vertex as the number of its outgoing edges.

The differences between the BTSP and the TSP translate the model of the *classic* TSP in a similar one, hereby reported in (2.3) - (2.8):

$$\min \quad C_{total} \tag{2.3}$$

$$C_{total} \geq \sum_{ij \in E} d_{ij} x_{ij} \tag{2.4}$$

$$C_{total} \geq -\sum_{ij \in E} d_{ij} x_{ij} \tag{2.5}$$

$$\sum_{j \in V} x_{ij} = 2 \qquad \forall i \in V \tag{2.6}$$

$$\sum_{i,j \in T, i \neq j} x_{ij} \leq |T| - 1 \qquad \forall T \subset V, T \neq \emptyset \tag{2.7}$$

$$x_{ij} \in \{0,1\} \qquad \forall ij \in E \tag{2.8}$$

$$C_{total} \in \mathbb{R}^{+} \tag{2.9}$$

Objective function (2.3) and Constraints (2.6), (2.7) and (2.8) are standard TSP constraints. In particular, Constraints (2.6) impose the cycle to visit each vertex exactly once –one incoming, one outgoing edge– while Constraints (2.7) avoid the presence of subtours T in the cycle. Considering that is a minimisation problem, Constraints (2.4), (2.5) and (2.9) impose variable C_{tot} to assume the absolute value of the cost of the cycle. Lastly, constraints (2.8) force variables x_{ij} to be binary and Objective function (2.3) aims at minimising the absolute cost of the cycle. By minimising the absolute value, we force the costs to be as close to zero as possible.

To solve this problem, we used AILS-RR, which is variant of ILS. We define as a feasible solution any Hamiltonian cycle and as objective function the cost of the cycle itself. Thus, we aim at identifying the best candidate in the neighbourhood of the current solution by applying modifications to the solution structure.

2.3. METHODOLOGY

The Iterated Local Search methodology repeatedly builds a sequence of solutions generated by a local search heuristic embedded in a framework. As defined by Lourenço et al., 2003, given a current solution s, ILS generates an intermediate solution $\hat{s}$ by applying changes on s. Then, the embedded local search heuristic is applied to $\hat{s}$, which leads to a new solution s'. If this solution improves s, it becomes the new current solution in our sequence and, thus, we keep navigating the solution space modifying s'. Otherwise, if s' does not introduce an improvement with respect to s, ILS continues to apply modification to s. Starting from a feasible solution, ILS explores its neighbourhood and determines the best solution within it. Thus, this metaheuristic exploits the possibility to search for a solution in a reduced space defined by the output of the local search heuristic, instead of searching over the entire solution space.

The methodology proposed in this chapter, i.e. AILS-RR, relies on ILS by searching in the neighbourhood of an already existing solution. From a graph $G(V,E)$ (representing a BTSP instance), our methodology starts by generating an initial solution. Once a solution has been identified, this is given as input to AILS-RR, which iteratively searches

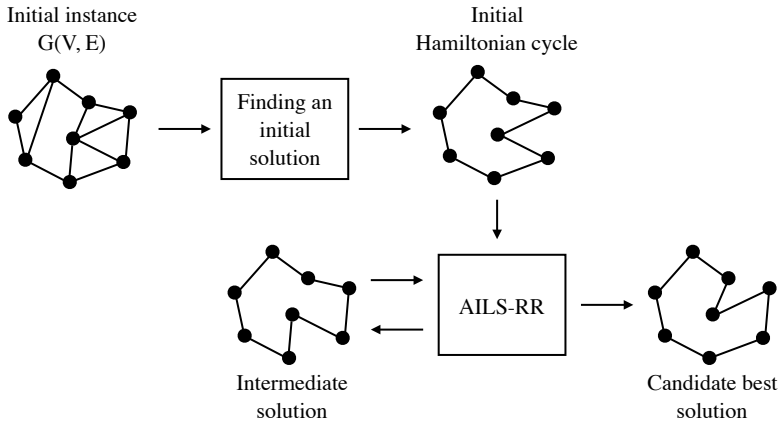


Figure 2.1: The AILS-RR framework and its phases.

for improving solutions until a certain stopping criterion is reached. Figure 2.1 shows an overview of the steps of our methodology.

Details about the strategy adopted to identify an initial solution are presented in Section 2.3.1, while the AILS-RR procedure is described in Section 2.3.2.

2.3.1. FINDING AN INITIAL SOLUTION

Determining whether there exists a Hamiltonian cycle in a not fully connected graph is an NP-complete problem (Garey et al., 1976). In order to find a Hamiltonian cycle, we used the Snakes-and-Ladders Heuristics (SLH) described in Baniasadi et al., 2014. The SLH is a polynomial time algorithm inspired by the k-opt heuristic. In SLH, vertices are ordered on a circle where edges between adjacent vertices represent the arcs of the circle while all other edges are considered as chords of the circle. The heuristic attempts to place all edges of a Hamiltonian cycle on the circle by seeking changes in the arrangement of vertices of the graph, with the goal of maximising the number of edges on the circle.

For this work, we have relied on the online implementation of SLH provided by Baniasadi et al., 2014 [1]. However, the online implementation accepts a maximum of 2000 vertices. To generate a cycle in the instances with more than 2000 vertices, we randomly partitioned all the vertices in equally sized subsets, with size smaller than 2000. For each of them, we independently found a Hamiltonian cycle; then, considering only edges from one subset to another, we selected the k vertices with highest degree. Finally, for each of the k vertices and its adjacent ones in the Hamiltonian cycle, namely A and B, we checked if they were connected to any couple of adjacent vertices in the other subset, namely C and D. If so, it means that the two disjoint cycles can be united as $A-$*first cycle*$-B-C-$*second cycle*$-D-A$. When none of the k vertices could be used to join the two cycles, the whole procedure was repeated by randomly re-partitioning the nodes. Figure 2.2 shows a flowchart of the process used to identify an initial solution. For all the eval-

[1] "http://www.flinders.edu.au/science_engineering/csem/research/programs/flinders-hamiltonian-cycle-project/slhweb-interface.cfm", website accessed 01-August-2018

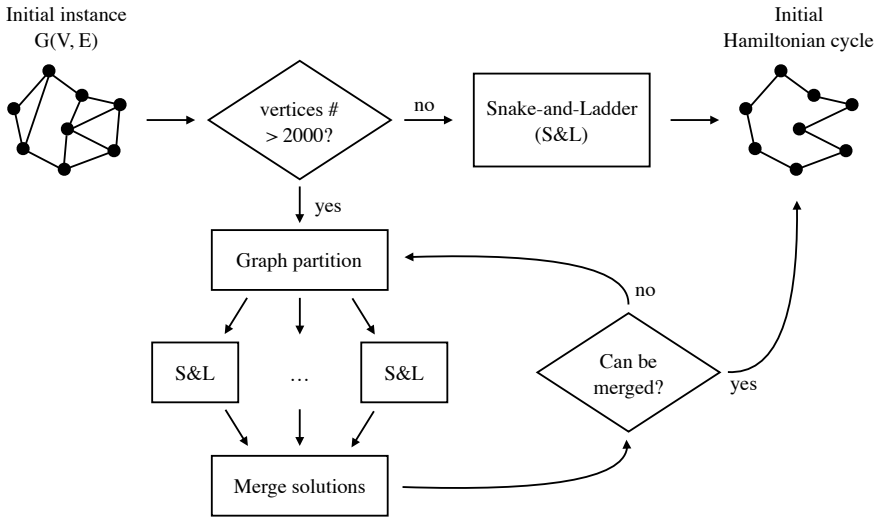


Figure 2.2: Flowchart of the process to generate the initial solution.

uated instances, we were able to find an initial solution without having to re-partition more than two times.

2.3.2. ADAPTIVE ITERATED LOCAL SEARCH WITH RANDOM RESTARTS

Once an initial feasible solution is found, we used AILS-RR to improve its objective value. The AILS-RR procedure proposed in this work relies on four main phases: *local search*, *update*, *perturbation* and *random restart*.

The procedure starts from the initial solution s provided by the method described in Section 2.3.1. Then, a *local search* is performed on s and a solution s' is returned. This solution can be: a) a new solution different from s or b) the same solution s in case no improving solution has been identified. Thus, the *update* phase amends the local search heuristic according to the resulting solution generated. In this *update* phase, the adaptive part of the algorithm takes place.

Once solution s' has been generated and the *update* has been performed, we replace s_{best} – the best solution found so far – with s' if the latter strictly improves it. In case no improved solution, with respect to s, was found after $MaxIteration$ consecutive iterations, a *perturbation* to s is applied and a new solution is generated. Then, the next AILS-RR iteration starts from the perturbed solution and the counter for the not improving solution is reset. Additionally, if for *too many* consecutive iterations, no solution has improved s_{best}, a *random restart* from a *good* known solution is performed. In particular, details on the number of consecutive iterations needed and on what is considered a *good* solution are presented in Section 2.3.2. Random restart is applied to avoid extensively searching unpromising regions of the solution space, hence moving to a more fruitful one. Finally, the AILS-RR terminates whenever the stopping criterion is met. Further details on the stopping criterion are given in Section 2.3.2.

A pseudo-code of the approach is illustrated in Algorithm 1. In the algorithm, there

are five main functions: a) *LocalSearch*, b) *Update*, c) *Perturbation*, d) *RandomRestart* and e) *StoppingCriterion*. Our algorithm differentiates from the *standard* ILS in terms of the update phase as well as the introduction of the random restart from a known solution. In the following paragraphs, we explain each component of the algorithm in detail.

Algorithm 1 Adaptive Iterated Local Search with Random Restart

```
1: procedure AILS (Input: Graph G, Hamiltonian cycle s; Output: Hamiltonian cycle s_best)
2:     s_best = s;
3:     notImproving = 0;
4:     while not StopCriterion() do
5:         i = 0;
6:         while i < MaxIterations do
7:             s'=LocalSearch(s);
8:             Update(LocalSearch());
9:             if |Cost(s')| < |Cost(s)| then
10:                s = s';
11:                i = 0;
12:                if |Cost(s')| < |Cost(s_best)| then
13:                    s_best = s';
14:                    notImproving = 0;
15:                else
16:                    notImproving++;
17:                end if
18:            else
19:                i++; notImproving++;
20:                if notImproving > RestartFactor then
21:                    s = RandomRestart();
22:                    notImproving = 0;
23:                end if
24:            end if
25:        end while
26:        s=Perturbation(s);
27:    end while
28:    return s_best
29: end procedure
```

LOCAL SEARCH

Starting from a current solution s, the local search aims at finding an improved solution s'. It consists of applying modifications, which are dictated by different operators, to the solution structure. An operator is a function that, given cycle s, applies modifications on its structure to generate multiple cycles which are variations of cycle s. The resulting cycles define a neighbourhood of s. Every solution in the neighbourhood is evaluated and only the solution which most improves s is accepted. If no solution improves

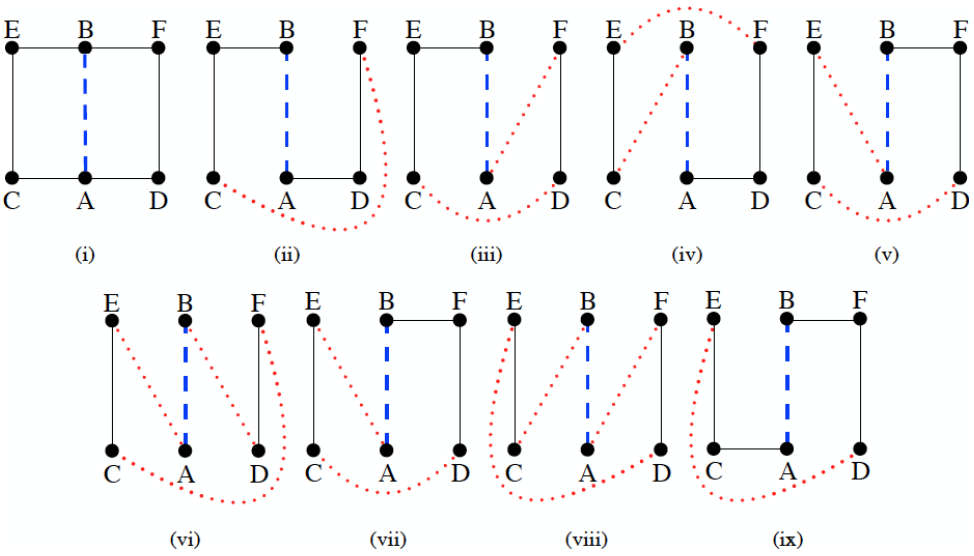


Figure 2.3: Examples of single edge insertion. Dashed blue lines show the edge to be inserted; dotted red lines indicate the edges that may or may not exist, while black lines show the edges belonging to the original cycle. Figure (i) shows the original cycle, while figures (ii)–(ix) show the possible insertions.

s, s itself is returned. During the local search, the algorithm uses – with probability depending on their weights – one of these three operators: *one edge insertion*, *two edges insertion* and *cycle modification*. Each operator selects at least one edge to be inserted in s. The insertion of an edge divides the original cycle in two subtours. Selecting the edge to be inserted determines which subtours will be created. Dually, identifying a desired subtour lets us establish which edge is to be inserted. Given the particular structure of the instances, we have chosen to determine the edges first. More on this is presented in Section 2.4.1. The following paragraphs introduce the operators adopted.

One edge insertion. Given a cycle s, the first operator selects, at random, one edge $\overline{AB}$ which is not in s. The extreme points of the edge, A and B, are adjacent to two vertices each in s –C and D for A, E and F for B. For the time being, we assume every vertex to be different with respect to each other; straightforward modifications can be applied if this is not the case. There is only a limited number of possibilities to insert edge $\overline{AB}$ in the existing solution; at most, there are eight possible outcomes. The cost, Equation (2.3), of all possible outcomes is evaluated and the best one is chosen. Since the graph is not complete, in general not all the combinations exist. Indeed, naming p the probability that an edge exists and assuming they are all independent, we can analyse quantitatively the probability for each combination to exist. Figure 2.3 shows all possible outcomes; the edge to be inserted is represented in blue, the edges that may or may not exist are shown in red, while black indicates the edges belonging to the original cycle. By construction, we know the existence of edges $\overline{AB}$, $\overline{AC}$, $\overline{AD}$, $\overline{BE}$ and $\overline{BF}$. Hence, we deduce there are two combinations with probability p, four combinations

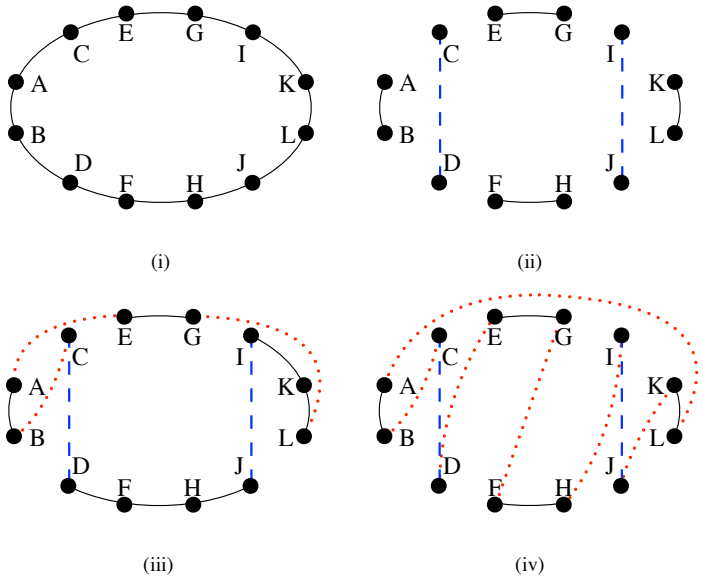


Figure 2.4: Example of two edge modification. (i) Initial cycle. (ii) Insertion of two edges and subtours generated. (iii) Reconstructed cycle with probability p^3. (iv) Reconstructed cycle with probability p^6. The original edges are shown in black while dashed blue lines depict the inserted edges and dotted red lines depict the edges existing with probability p.

with probability p^2 and two combinations with probability p^3.

Two edges insertion. Similarly to the previous operator, this process chooses two edges not yet in the current solution and tries to insert them. If the four extreme vertices of the two selected edges are all different, isolating them divides the cycle in four subtours. Hence, the solution is now decomposed in six subtours – four from the original cycle and two from the two inserted edges, that can be considered subtours as well, see Figure 2.4.

There are $10!!$[2] possible ways to combine the four subtours and the two edges. This number comes from $(2\cdot(t-1))!!$, where t is the number of subtours – six, in our case – and -1 because a degree of freedom is lost for the intrinsic symmetry of cycles. A multiplicative factor of 2 is added since each subtour can be linked to the next one through two different endpoints. Having t subtours implies having, as their endpoints, $2t$ vertices. Intuitively, the double factorial follows because a vertex can be connected to $2t-2$ other vertices, every vertex but itself and the other endpoint of its subtour. Once connected, the following vertex can be connected to $2t-4$ others. This includes all the vertices but itself, the other endpoint of its subtour and the endpoints of the subtour to which it is already linked to. Recursively, we can see how this develops, for the remaining vertices, in a double factorial structure. Among these combinations, only the ones with at least probability p^3 to exist are considered by our methodology.

Generally speaking, these first two operators can be viewed as modified versions

[2] $!!$ is double factorial, i.e. $f!! = f\cdot(f-2)\cdot(f-4)\ldots$
In our case, $10!! = 3840$

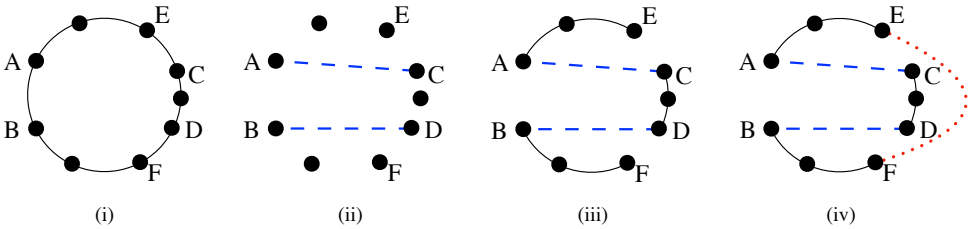


Figure 2.5: Example of cycle modification. (i) Original cycle. (ii) Selection of the edges to be inserted. (iii) Construction of the existing path. (iv) Closing the cycle with edge $\overline{EF}$ which exists with probability p. Original edges are shown in black, dashed blue lines depict the inserted edges and dotted red lines indicate the edges that exist with probability p.

of k-opt. Figure 2.4 shows an example of two edges insertion. Starting from an initial cycle – Figure 2.4i – two edges are inserted. The new edges divide the cycle into four different subtours – Figure 2.4ii. Finally, Figure 2.4iii and Figure 2.4iv show an example of a reconstructed cycle with probability p^3 and p^6, respectively.

Cycle modification. The two edge insertion generates multiple intermediate solutions, but it is computationally more expensive with respect to the one edge insertion operator. To compensate the computational requirements of the two edge insertion operator, we introduce the cycle modification operator.

This operator selects, at random, an edge in the existing solution s. We name A and B its extreme vertices, which are consecutive in the original solution s. Then, we select at random one edge, not in solution s, which is outgoing A and is entering, without loss of generality, in C. At the same time, we select at random one edge, not in solution s, which is outgoing B and is entering, without loss of generality, in $D \neq C$, see Figure 2.5ii. Subsequently, we consider the path, in the original cycle, from C to D, that passes through A and B. In that cycle, we name E and F the follower of C and the predecessor of D, respectively. By construction, there exist paths $\overline{EA}, \overline{CD}, \overline{BF}$ and edges $\overline{AC}, \overline{BD}$. Hence, there exists a path connecting $\overline{EA} - \overline{AC} - \overline{CD} - \overline{DB} - \overline{BF}$, see Figure 2.5iii. Finally, if edge $\overline{EF}$ exists, we obtain a feasible cycle, see Figure 2.5iv. In general, edge $\overline{EF}$ exists with probability p. To increase the size of the neighbourhood, this procedure is repeated for all outgoing edges of B. It is not, however, repeated for all combinations of outgoing edges of A and outgoing edges of B, because this would be computationally too expensive.

UPDATE

Each operator of the local search is applied with a probability proportional to its associated weight. These weights are constrained to be greater than a parameter $MinWeight$ and their sum is forced to a value lower that the upperbound parameter $MaxWeights$. Whenever an operator returns a solution which does not improve the input solution, we subtract f –in this work, f has value 1– from its associated weight. In case an operator returns a better solution than the solution given as input, its associated weight is increased by 10% and rounded up to the nearest integer. In addition, if the returned solution is even better than the best known solution –s_{best}– the weight of the operator leading to

the improved solution receives an extra reward of $10f$ in addition to the normal reward obtained for improving the previous solution. We call this discrepancy among a constant decrease and a proportional increase an *uneven reward-and-punishment* adaptation rule. In our opinion, an even reward-and-punishment adaptation rule is more suited to grasp stable characteristics, such as the ones related to the structure of the $graph$ itself, while an uneven rule is more keen to quickly adapt to variations, such as the ones in the changing structure of the $solution$.

PERTURBATION

The *perturbation* is applied when we are not able to improve a local solution for a significant number of iterations –$MaxIterations$. To perform a perturbation, the algorithm uses the same operators as the local search. The main differences, with respect to the local search, is that every change is accepted –not only an improving one– and it is performed only once. There is no evidence that more perturbations results in better solutions. In fact, more perturbations cause the current solution to drift too much away from a promising part of the solution space. In addition, since the costs of the edges are neither Euclidean nor we were able to find any pattern within them, even a slight modification of a few edges can lead to dramatic changes in the objective function.

RANDOM RESTART

Perturbations allow to explore different regions of the solution space; nonetheless, some of those regions could be unpromising. To avoid exploring inadequate regions of the solution space, it is useful to restart the search from a region where *good* solutions are known to exist. If, after *too many* consecutive iterations, no solution improved the best known objective function, then a *random restart* from a *good* known solution is performed. In particular, we define as $History$ an array storing the $HistorySize$ best solutions and their number of occurrences. If, after $RestartFactor$ consecutive not improving iterations, s_{best} was not improved, we perform a random restart from any of the solutions stored in $History$.

We define $RestartFactor$ as:

$$RestartFactor = cMax + \frac{MostVistitedSolution(History)}{HistoryStep}, \tag{2.10}$$

where $MostVistitedSolution(History)$ assumes the value of the number of visits to the most visited solution in $History$, while $cMax$ and $HistoryStep$ are parameters. $cMax$ indicates the minimum number of iterations the algorithm has to perform before a *random restart* can happen, while $HistoryStep$ is a scaling factor. While we perform the random restart to avoid going too far away from a region of the solution space where good solutions exist, we reduce the frequency of restarts when the same solution is visited more and more times to escape that tenacious local minimum. In fact, it could happen that too frequent restarts drives the local search to the same local minima. In addition, restarting from any of the solutions stored in $History$ helps to maintain a certain degree of diversity.

STOPPING CRITERION

AILS-RR has no memory of all the solutions discovered since it started and in general there is no guarantee of optimality. Hence, without a stopping criterion, it would indefinitely search for improving solutions. The stopping criterion we implemented terminates the execution of the algorithm if any of the following conditions is met: a) the solution cost is zero and, thus, we have reached the optimal solution, b) a user-defined time limit was exceeded, c) the algorithm returned for more than $MaxIterationHistory$ times the same solution.

If condition a) is met, then, it is not possible to further improve the solution identified. Condition b) offers a knob for setting a reasonable usage of resources required to search for improving solutions and condition c) is useful to avoid expensive explorations of particularly tenacious local optima from where the algorithm cannot escape even with its *perturbation* move.

2.4. EXPERIMENTS

In this section, we explain the experiments setup and the performance of our algorithm. In Section 2.4.1, we describe the instances tested, in Section 2.4.2, the parameters used and, lastly, in Section 2.4.3, the performance of our algorithm.

2.4.1. INSTANCES

The algorithm was tested on 27 given instances[3] , which vary in size from 10 vertices and 40 edges, to 3,000 vertices and 12,000 edges. Hence, on average, each vertex has degree 4. This motivates the analysis on the probability of existence of an edge in the AILS-RR. The absolute value of the costs of every edge can be written as $k_1 \cdot 100{,}000 + k_2$, where k_1 and k_2 are integers in the range $[0,99]$. We run the algorithm twice per instance. The first time, we used as input the instances considering as cost only k_1. In the following, we refer to this change in the cost associated with the edges as a cost modification. With this data, the algorithm was able to find a solution of cost zero for all instances. Then, the algorithm was run a second time, starting from the previously found solution, with the real costs of the arcs.

2.4.2. PARAMETERS TUNING

The local search procedure is repeated until for $MaxIterations = 100$ consecutive iterations no improving solution is found. $HistorySize$, the number of how many *good* solutions were stored in array $History$, is set to 100. Parameters $cMax$ and $HistoryStep$, which are used to determine when to restart from a random solution in $History$, are set to 1,000 and 100, respectively. Parameter $MaxIterationHistory$ determines how many times a solution can be visited before the stopping criterion is met and is set to 1,000,000. This means that a random restart can happen as often as after 1,000 consecutive not improving iterations, or as rarely as after 10,999 consecutive not improving iterations. In Paragraph 2.3.2, we explained how often the random restart happens depending on how many times the most inspected solution is visited. We may have a restart after 10,999 consecutive not improving iterations and not after 11,000 times as ex-

[3] available at "https://195.201.24.233/mess2018/home.html", Accessed 01-August-2018.

Parameter name	Value	Description
$MaxIterations$	100	maximum not improving cycles of LS
$HistorySize$	100	dimension of array $History$
$cMax$	1,000	minimum number of cycles for a random restart
$HistoryStep$	100	random restart parameter
$MaxIterationHistory$	1,000,000	stopping condition parameter
$MinWeight$	1	minimum weight for each operator
$MaxWeights$	1,000	maximum value of the sum of the weights of all operators
$MaxTime$	2h	maximum time per instance - modified costs
$MaxTime$	12h	maximum time per instance - original costs

Table 2.1: Parameters tuning

pected if $MostVisitedSolution(History)$ assumes value $MaxIterationHistory$. In fact, $MostVisitedSolution(History)$ cannot assume value $MaxIterationHistory$ in Equation (2.10) because, if so, the stopping criterion is met and the execution of the whole algorithm is terminated. Every single operator weight is initially set to 333 and restricted to integer values above $MinWeight = 1$ and such that their sum does not exceed $MaxWeights = 1{,}000$. If the sum of the weights exceeds $MaxWeights$, the weight of every parameter is decreased by one, unless this violates the lower bound $MinWeight$, until the threshold is respected. For the tests with the modified cost, the maximum running time for each instance was set to two hours while, for the tests with the original cost, the maximum running time for each instance was set to twelve hours. Table 2.1 summarises all the parameters used in the algorithm.

2.4.3. PERFORMANCE

Instances were run overnight on different machines. In particular, instances up to 100 nodes were run on an Intel Core i7-6600U CPU @2.60GHz 2.80GHz with 8 GB RAM and instances from 150 to 400 nodes on a Intel Core i7 @2.9 GHz with 8 GB RAM. Bigger instances (500 to 3,000 nodes) were run on a 32 core machine with Intel Xeon E5-4650L CPU @2.60GHz 3.1GHz with 500 GB of physical memory. Since the BTSP is a new problem, introduced for the MESS2018 solver challenge, no comparison with the state of the art is possible. In general, optimal solutions are not known but they cannot have a better objective function than zero. By executing the AILS-RR on the instances with modified costs of the edges, as explained in Section 2.4.1, we know that the optimal solutions for all these modified instances have cost zero. In general, a zero cost solution for the modified instances does not translate into an optimal solution for the original instances; nonetheless, it is a *good* initial when solving the instance with original costs. In the following paragraphs, unless explicitly mentioned, results refer to the modified costs. For the tests with the modified costs, we set a time limit of two hours and a limit of 10,000 iterations by counting how many times *LocalSearch* is called. In order to perform a qualitative analysis of the obtained results and of the operator effectiveness, we (re-)tested the

instances on a 32 core machine with Intel Xeon E5-4650L CPU @2.60GHz 3.1GHz with 500 GB of physical memory. Each test was run 10 times to obtain average results. In the following paragraphs, we introduce in detail a meaningful instance case and, afterwards, we present results for all instances.

INSTANCE 3,000 VERTICES

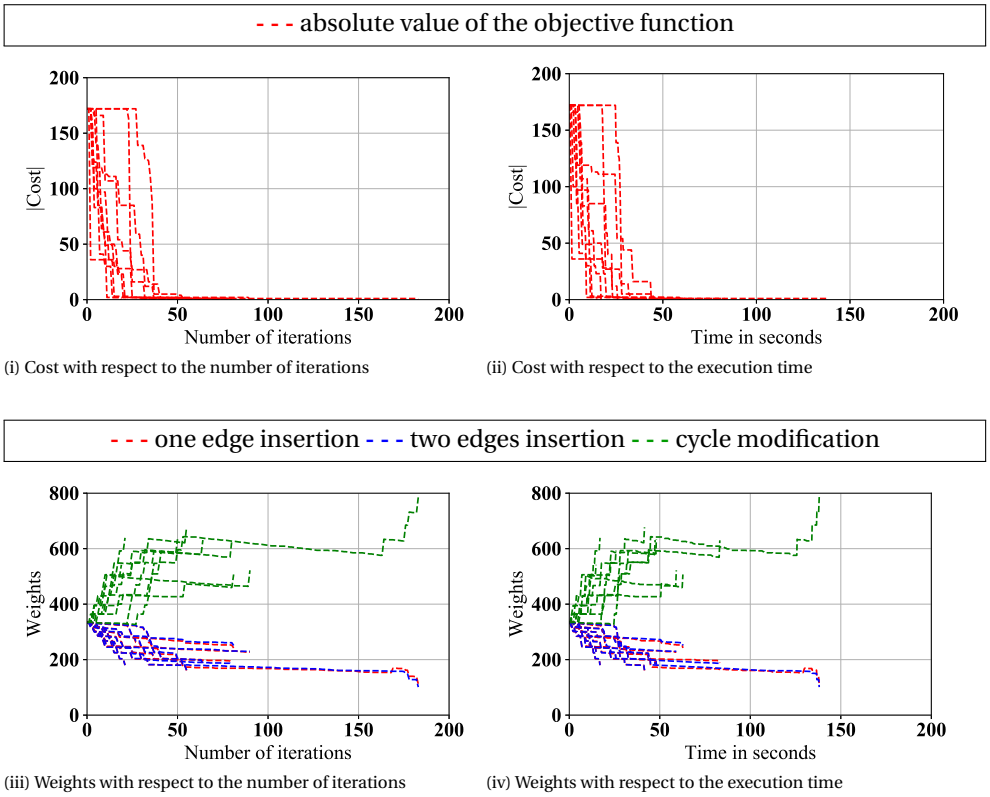
The instance presented in this section is representative of the entire set. In fact, Figure 2.6i and Figure 2.6ii display the trends of the objective function, for the same executions, with respect to iterations and time while Figure 2.6iii and Figure 2.6iv show the evolution of the weights during the ten tests of the algorithm, with respect to iterations and time. Since all the tests are plotted simultaneously, these figures give some idea on the variance of the trends and how many tests terminated their execution in just a few iterations. First of all, plotting results with respect to iterations or with respect to time only slightly modifies the overall shape of the figures. This is due to the fact that comparable amounts of time are needed for each operator to perform its local search. In Figure 2.6iii and Figure 2.6iv, sharp peaks with slow decline are visible. This is exactly the effect of the *uneven reward-and-punishment adaptation rule*; since increases are proportional while decreases are constant, rapid changes in the weights of the operators are visible. In this case, and basically also for all other instances, it is clear that operator *cycle modification* performs better than the others. Secondly, Figure 2.6i and Figure 2.6ii show the absolute value of the best solution found so far by the algorithm. Even though this instance is the biggest one, even for the worst of the ten tests, our algorithm was able to find an optimal solution in roughly two minutes.

RESULTS FOR ALL INSTANCES

Small instances were solved in a few iterations with no particularly interesting trend; because of that, in this paragraph, we consider only instances of size strictly greater than one hundred vertices. Since no particular difference arises from plotting with respect to the number of iterations or with respect to time, the figures in this paragraph refer to the iterations. Furthermore, for the sake of readability, we decided to plot average results instead of all the 10 trends. Averaging the results highlights the trends but smooths peaks which instead are visible in Figure 2.6iii and Figure 2.6iv. For instances with more than one hundred nodes, trends of the weights of the operators and of the solution developments are shown in Figure 2.7 and Figure 2.8, respectively. These trends show the average results for the ten tests. Figure 2.7 shows that, for all simulations, all tests over all the instances, but one, returned the optimal solution within few iterations –resulting in few minutes of execution time–, way before encountering the time or the iteration limit. In our opinion, this is a powerful indicator of the effectiveness of our algorithm. Similarly, Figure 2.8, shows how among all the instances, the *cycle modification* is the most effective operator. Nonetheless, it is worth noticing that, while for the medium-sized instances, weights are *almost* equivalently distributed among operators, the bigger the instance, the greater the probability of *cycle modification* to be chosen.

FINAL RESULTS

In this section, for the sake of further comparison, we display the results submitted to the competition. All the results proposed in this section are computed with the original



(i) Cost with respect to the number of iterations

(ii) Cost with respect to the execution time

(iii) Weights with respect to the number of iterations

(iv) Weights with respect to the execution time

Figure 2.6: Evolution of absolute cost of the solutions and operator weights with respect to number of iterations and execution time. Results are shown for the instance with 3,000 vertices.

costs. In particular, Table 2.2 portraits: in the first column, the instance size –expressed in the number of vertices– and, in the second column, the absolute value of the solutions.

2.5. CONCLUSION

This chapter illustrates the AILS-RR methodology applied to the balances travelling salesman problem. With slight modifications of the local search operators, we believe that the same metaheuristic can obtain significant results in many operational research problems. The proposed metaheuristic is a variant of ILS and it features the adaptive use of the local search operators and restart moves. Key advantages of the AILS-RR are: its effectiveness in navigating the solution space, as shown in the achieved ranking in the MESS2018 Metaheuristics Competition, its easiness to implement and its ability to quickly obtain near optimal solutions. Additional motivations and a detailed description of our algorithm are presented in Section 2.3, which presents the algorithm structure focusing on the different phases of the metaheuristic. In particular, the description details the main contribution of the proposed methodology which lays in the newly intro-

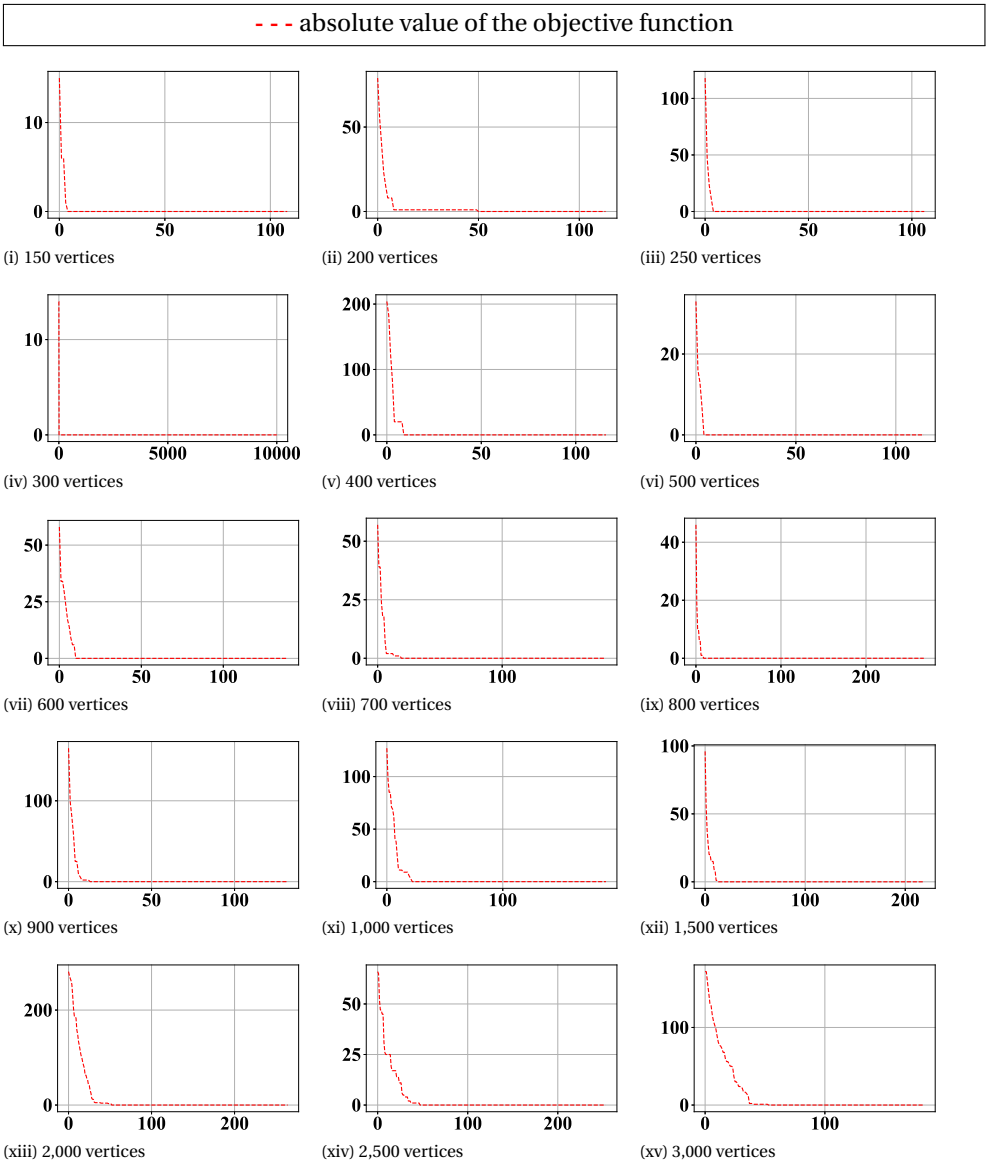


Figure 2.7: Evolution of the objective function over different instances. Number of iterations on the x-axis, objective function value on the y-axis.

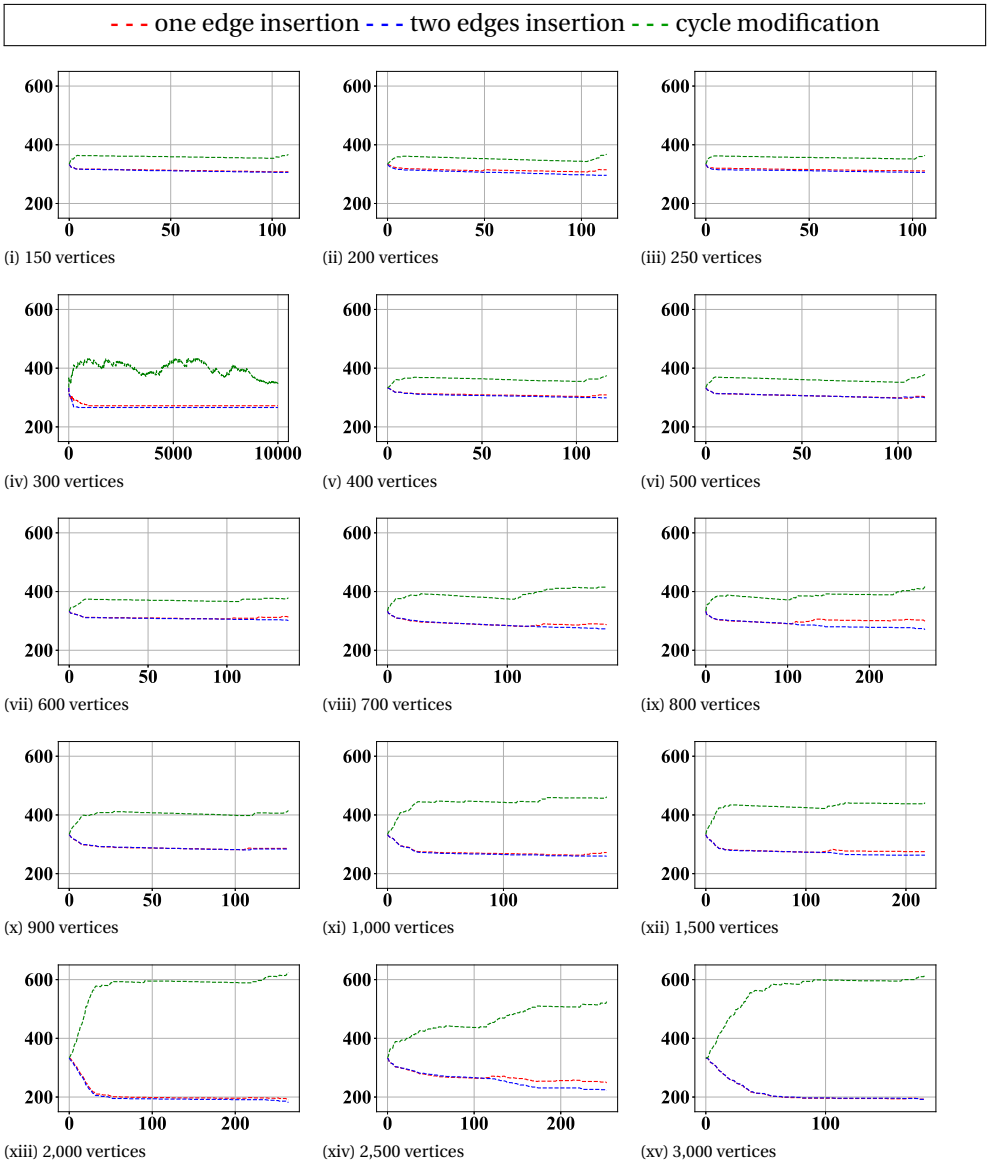


Figure 2.8: Evolution of the weights assigned to the different operators over different instances. Number of iterations on the x-axis, operators' weights on the y-axis.

# vertices	|Solution cost|
10	105
15	271
20	296
25	375
30	433
40	473
50	717
60	918
70	1,056
80	929
90	1,098
100	1,245
150	2,035
200	2,657
250	3,811
300	4,846
400	6,509
500	8,418
600	9,784
700	17,989
800	18,233
900	20,596
1,000	22,597
1,500	37,662
2,000	49,882
2,500	36,607
3,000	24,423

Table 2.2: Results.

duced *uneven reward-and-punishment adaptation* rule. To the best of our knowledge, this is the first time that such a strategy is used. Section 2.4 proves that our AILS-RR achieves notable results, scoring remarkable positions in almost every instance ranking and achieving the 5th position in the competition.

3

MILP MODELS FOR THE DIAL-A-RIDE PROBLEM WITH TRANSFERS

Jacopo PIEROTTI, Theresia VAN ESSEN

Automated vehicles (AVs) are becoming a reality. Expectations are that AVs will ultimately transform personal mobility from privately owned assets to on-demand services. This transformation will enhance the possibility of sharing trips, leading to shared AVs (SAVs). The preeminent aim of this chapter is to lay foundations for fast and efficient algorithms to be used in such new driving conditions. These algorithms must be able to solve Dial-a-Ride problems with transfers (DARPT). Hence, they should efficiently assign passengers to vehicles and routes while also: administering vehicles dispatch, determining convenient parking for idling vehicles and managing vehicle routing in real-time. In this chapter, we develop two integer linear programming models (one in continuous time and one in discrete time) and their extensions to solve the DARPT. Our models take into account routing, service times, constraints on maximum route time-span, unserved requests, preferred arrival and departure time, nonconstant travel times, convenient parking while optimizing routing costs and quality of the service. The models are tested on instances based on Google Maps data by solving them with a commercial solver. The results of these tests are the starting point for validating the performance of forthcoming, ad hoc metaheuristics to be used in real-life sized scenarios.

Parts of this chapter have been published in EURO Journal on Transportation and Logistics , doi: https://doi.org/10.1016/j.ejtl.2021.100037

3.1. INTRODUCTION

Automated vehicles (AVs) will reshape our transportation system. The opportunities and potential they offer will lead to the most significant transportation revolution since the introduction of the internal combustion engine (Spieser et al., 2014). In fact, nowadays, cities are facing an increasing personal transportation demand combined with a growing population, while spatial resources remain static. Traditional solutions to congestion (roads expansion, added bus services, new subway lines, etc.) cannot mitigate the traffic escalation. In order to meet this expanding mobility demand, a new transportation mentality involving shared autonomous vehicles (SAVs) is likely to become the dominant mindset in the coming years (Fagnant and Kockelman, 2014, International Transport Forum, 2015). Although this SAVs system has some disadvantages, such as high initial cost barriers (Fagnant and Kockelman, 2015, Conceição et al., 2017), loss of vehicles' status symbol (Correia and van Arem, 2016) and distrust from the public, this new mentality is accelerated by the growing number of environmentally conscious citizens and the rising popularity of on-demand ridesharing services, especially among young adults. In addition, thanks to the possible interactivity among SAVs and to a smart managing system on top, lower and lower congestion levels will be achieved (Spieser et al., 2014, International Transport Forum, 2015, Liang et al., 2017).

As pointed out by Spieser et al., 2014, while automated vehicle technology continues to move forward, less attention has been devoted to the logistics of effectively managing a fleet of potentially thousands of such vehicles. Accordingly, our work aims to fill this gap to a certain degree by presenting exact formulations to solve the routing of SAVs. In particular, we demonstrate the benefits of transfers and ridesharing for small instances. The performance of forthcoming metaheuristics can be compared against these results. A more formal definition of 'ridesharing' and 'transfer' is presented in Section 3.2. Since organizing transfers is a meticulous and precise action that requires information on the position of all vehicles, we assume vehicles to be autonomous and to be managed by a centralized controller. Although these autonomous vehicles will most likely be electric, we do not take battery consumption (A. Zhang et al., 2017) into account in this research, since this will further complicate an already complex problem.

To simulate on-demand ridesharing systems, almost every aforementioned author (Fagnant and Kockelman, 2014, Martinez et al., 2015, International Transport Forum, 2015,Fagnant and Kockelman, 2015, Liu et al., 2017) used agent based simulation. In this chapter, instead, we use exact optimization techniques to achieve optimal solutions; however, this limits the ability to solve real-life sized instances due to high computational times.

The problem we aim to solve shares many characteristics with two traditional problems in the vehicle routing problem (VRP) literature: the (Splittable) Pick Up and Delivery Problem (PDP) and the Dial-A-Ride Problem (DARP). Since these are infamously known to be NP-Hard and difficult to approximate (Masson et al., 2014), many authors focused on heuristic solution methods.

PDP shares most of the routing structure with our problem. The main difference lies in the fact that goods (typically letters or small parcels[1]) are carried instead of people.

[1] since small packages are moved, the problem is often treated as capacity unconstrained

Hence, in the PDP, the quality of the time spent travelling is not considered. In addition, goods facilities may have time windows while, in general, there is no preferred arrival or departure time. Also, transfers are considered, but only in a few predetermined hubs. To solve the PDP with transfers, Rais et al., 2014 introduce an integer programming model and use standard branch-and-bound while Cortés et al., 2010 adopt branch-and-cut techniques. Also, Peng et al., 2019 developed a MILP formulation for the (selective) PDP with transfers and compared its solution with a particle swarm optimization metaheuristic. Given the complexity of the problem, Peng et al., 2019, Cortés et al., 2010 and Rais et al., 2014 solve instances with five, six and seven requests to optimality, respectively. Thangiah et al., 2007 combined a constructive heuristic with local optimization to quickly (under 5 seconds) solve the online version of the PDP. Danloup et al., 2018 tested two metaheuristics, namely large neighborhood search (LNS) and genetic algorithms (GA), showing a very performing implementation of the latter. Masson et al., 2013 developed an adaptive large neighborhood search (ALNS) algorithm with different destroy and repair methods, showing its performance on real-life data from the area of Nantes. Also, Petersen and Ropke, 2011 solve the pickup and delivery problem with cross-docking opportunity (a variant of the PDP) using LNS. In addition, their algorithm was tested on real-life instances with sizes ranging from 500 to 1000 requests (but only one possible transfer node). All the aforementioned authors who solve the PDP considered only few nodes (hubs) as possible transfer nodes, while we consider all nodes as possible transfer nodes.

On the other hand, the DARP consists of designing routes for vehicles and schedules for users who specify pickup and delivery requests between origins and destinations. DARPs are a well studied class of problems and extensive reviews can be found in Agatz et al., 2012, Molenbruch et al., 2017 and Ho et al., 2018. In general, ridesharing is allowed but transfers are not considered (Cordeau and Laporte, 2003). Since people are transported, the quality of the travel time has to be considered. Most authors state that minimizing the routing costs or minimizing the time of the routes implies minimizing the loss of quality of the service. Although reasonable, we prefer a more explicit approach, as detailed in Section 3.3.2 and Section 3.4.1. Even though standard DARP problems do not take transfers into account, some papers do consider DARP with transfers (DARPT). Masson et al., 2014 used an adaptive strategy combined with a ruin and repair mechanism while Deleplanque and Quilliot, 2013 developed a general insertion scheme. In these papers, fast heuristic conditions were introduced to check if a repaired route was feasible or not. Hou et al., 2016 developed a MILP formulation and a greedy heuristic to compare electric taxi usage between the nontransferable taxi-sharing and the transferable one. They show that, during rush hours, transfers could improve the number of served passengers and the shared travel distance by 22% and 37%, respectively. Nevertheless, transfers can only happen at recharging stations.

Moving outside the conventional boundaries of the DARP with transfers, Reinhardt et al., 2013 and Posada et al., 2017 developed multi modal frameworks. In these papers, transfers happen between different transport modes within an airport or between the public and the private transportation, respectively.

Our contribution is the development of two MILPs -one in continuous time and one in discrete time- to solve the DARP while considering:

- the possibility to transfer in all nodes of the network,
- routes with cycles such that a vehicle or a request can visit the same node several times,
- the absence of a depot location, or equivalently, all nodes are considered to be depots,
- non-constant travel times,
- focus on quality of time: requests' preferred arrival and departure time and quality loss due to transfers and waiting times,
- attention to convenient parking.

As shown by van den Berg and van Essen, 2019 and confirmed by our results (Section 3.5.3), either the continuous time models or the discrete one can perform better depending on the situation. In order to tie continuous and discrete variables in the continuous time model, we introduce the concept of 'move' (Section 3.3.1). To the best of our knowledge, cycles have been handled by creating dummy copies of nodes, but they have not been explicitly modeled in the literature. Also, cycles are very common in practical applications of people transportation, for instance, a taxi may visit the airport or the city center multiple times during a work shift. As shown in this chapter, these features increase the complexity of the standard DARP but allow for more flexibility.

The remainder of this chapter is organized as follows: the problem, our assumptions and the notation used are described in detail in Section 3.2. Section 3.3 and Section 3.4 depict the model in continuous time and in discrete time, respectively. Section 3.5 details performance and computational results and, finally, Section 3.6 concludes the chapter.

3.2. PROBLEM FORMULATION

In this chapter, we describe how to solve the Dial-a-Ride problem with transfers to optimality. The problem we consider is the following: given a set of requests R and a set of vehicles V, minimize the generalized cost Z (routing and quality of service), allowing transfers and ride sharing while considering time and vehicle limitations. Allowing transfers means that each passenger may be picked up by a vehicle, taken to a certain location, dropped off and picked up by another vehicle. This procedure may be repeated multiple times. Without loss of generality, we suppose the initial time to be zero.

The road network is modelled as a graph with node set N connected by a set of arcs A. For now, we assume that each vehicle can idle and each request can wait at every node which, in turn, are all possible transfer nodes. In Appendix 3.B, we describe how the set of transfer nodes can be limited.

We consider the length l_{ij} of arc $(i, j) \in A$ with $i, j \in N$ to be known. The travel time of each arc is a function of the arc $(i, j) \in A$ and the time t at which a vehicle starts travelling on it, i.e. $F(ijt)$. In this chapter, we consider two cases: the case where $F(ijt)$ returns a parameter dependent only on arc $(i, j) \in A$ and the case where the travel time is depending on both arc $(i, j) \in A$ and time t. In Section 3.3.3 and Section 3.4, we describe how to derive a linear formulation even for non-constant travel times (travel times depending on t).

Each request $r \in R$ is characterized by nine parameters. The first one, e^r, determines its earliest possible departure time; so, every request can be picked up only after time e^r.

The second parameter, l^r, determines the time instant before which each request must reach its destination; otherwise, it is considered unserved. Every request $r \in R$ is also characterized by parameters pd^r and pa^r. These parameters state the preferred time instants for departure and arrival. It must hold that $e^r \leq pd^r \leq pa^r \leq l^r$. Parameters e^r and l^r define a hard time window (on the request, not on the node) while pd^r and pa^r define a soft one. This newly introduced type of time limitation differs from standard hard time windows, which are imposed on a node and not on a client, as well as from maximum ride time, which are imposed on a client but do not consider departure and arrival time *per se* but only their difference. The maximum number of transfers allowed for request $r \in R$ is given by b^r and the party size of the same request is given by q^r. The origin and destination of request $r \in R$ is given by $o^r \in N$ and $d^r \in N$, respectively.

Each vehicle $v \in V$ is described by an initial position $o^v \in N$ and a capacity q^v. $\alpha, \beta, \gamma_1, \gamma_2, \gamma_3, \gamma_4, \eta$ and E are cost parameters detailed in Section 3.5.2, while B is a sufficiently large number. We name ϵ the minimum travel time among any arc $(i, j) \in A$ (i.e. $\min_{ij \in A, t \geq 0} F(ijt)$) and we let T_{Max} be equal to the maximum latest arrival time l^r over all requests $r \in R$ (i.e. $\max_{r \in R} l^r$).

3.3. CONTINUOUS TIME MODEL

In this section, we detail the concept of a 'move' (Section 3.3.1), the core continuous time model (Section 3.3.2) and its extension (Section 3.3.3). The extended model fulfills the same aim as the core model (i.e. solving the DARPT), but also includes additional features such as service times and non-constant travel times. These features are added to have a closer resemblance to real-life. Both the continuous time and discrete time model rely on the idea of tracking requests' flows (from their origin to their destinations) and forcing vehicles' flows to overlap and be paired with them (more details in Section 3.3.2 and Section 3.4.1). We refer to the formulations based on this idea as *flow formulations*.

3.3.1. MOVES

While designing routes involves discrete variables, adopting a continuous time model implies having variables in continuous time. Since discrete and continuous variables influence each other and have to be considered simultaneously, we introduce the 'move' concept. A 'move' refers to the act of travelling an arc. In fact, we associate each request $r \in R$ with a set $M^r = \{1, 2, 3, .., \mathscr{M}^r\}$. The cardinality $\mathscr{M}^r$ of this set is bigger than the maximum number of arcs request $r \in R$ can travel given its time limits (from e^r to l^r). In particular, given the time limitations on each request and a non degenerative graph (i.e. each travel time is strictly greater than zero), there is a bound on the maximum number of arcs a request can possibly travel. Equivalently, there is a bound on the maximum number of arcs that a vehicle can travel; so, each vehicle $v \in V$ is also associated with a set of moves $M^v = \{1, 2, 3, .., \mathscr{M}^v\}$. The cardinalities of sets M^r and M^v are given by $\mathscr{M}^r = \lfloor \frac{l^r - e^r}{\epsilon} \rfloor$ and $\mathscr{M}^v = \lfloor \frac{T_{Max}}{\epsilon} \rfloor$, respectively. Figure 3.1 graphically shows how these moves are counted. The text next to each arc indicates who is travelling that arc and at which move. The figure clearly shows that vehicles and requests can travel the same arc at different moves.

All the parameters described in Section 3.2 and in this section are summarized in

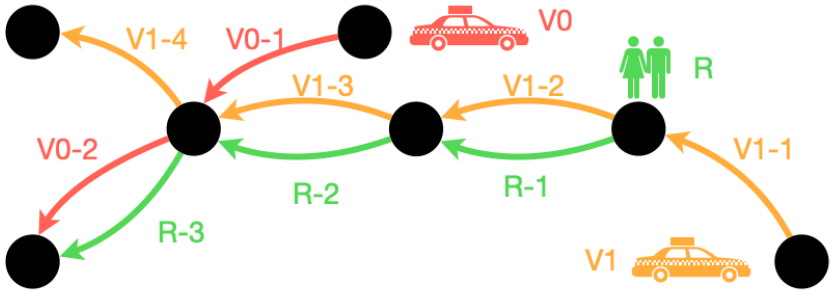


Figure 3.1: Example of moves. Colours refer to who is travelling that arc (red for vehicle $V0$, yellow for vehicle $V1$ and green for request R). The text next to each arc indicates who is travelling that arc and at which move.

Table 3.1.

R	set of requests
e^r	earliest departure time for request $r \in R$
pd^r	preferred departure time for request $r \in R$
pa^r	preferred arrival time for request $r \in R$
l^r	latest arrival time for request $r \in R$
b^r	maximum number of transfers allowed per request r
q^r	party size of request $r \in R$ (how many people in request $r \in R$)
o^r	origin of request $r \in R$
d^r	destination of request $r \in R$
M^r	set of all the possible moves for request $r \in R$
$\mathscr{M}^r$	cardinality of set M^r for request $r \in R$
V	set of vehicles
o^v	origin of vehicle $v \in V$
q^v	capacity of vehicle $v \in V$
M^v	set of all the possible moves for vehicle $v \in V$
$\mathscr{M}^v$	cardinality of set M^v, $v \in V$
N	set of nodes
A	set of all arcs
l_{ij}	length of arc $(i, j) \in A$
$F(ijt)$	function that returns the travel time along arc $(i, j) \in A$ at time t
T_{Max}	time instant after which no requests can be delivered, i.e. $T_{Max} \geq l^r, \forall r \in R$
ϵ	shortest travel time in the network, i.e. $\min_{(i,j)\in A, t\geq 0} F(ijt)$
$\alpha, \beta, \gamma_1, \gamma_2, \mu_1, \mu_2, \mu_3, \mu_4, \eta, E$	cost coefficients.
B	parameter with high value
Z	generalized cost

Table 3.1: Sets, parameters and function

3.3.2. CORE MODEL

After introducing the concept of 'move', we dedicate this section to the description of the core continuous model. This core model takes into account routing, timing, pairing, capacity constraints, constraints on the maximum number of possible transfers and the possibility of unserved requests while minimizing a trade off between routing costs and the loss of the quality of the service.

VARIABLES

In this section, we describe the variables used in the model. To model the routing, we employ binary variables x^r_{ijm} which assume value one when request $r \in R$ travels arc $(i,j) \in A$ in move $m \in M^r$ and zero otherwise. If considered in the order of the moves, these variables describe the route of request $r \in R$. Next to the routing, variables describing the chronological framework are needed; for this, we introduce continuous variables t^r_m and w^r_m for request $r \in R$ and move $m \in \{0\} \cup M^r$. Since m defines the move from one node to another, t^r_m assumes the value of the time instant at which request $r \in R$ arrives in a node after move $m \in M^r$, while w^r_m assumes the value of how much time request $r \in R$ waits after move $m \in M^r$. In the timing and waiting variables (t^r_m and w^r_m), the additional move $\{0\}$ is considered to determine the initial departure time which occurs before the first move. Figure 3.2 depicts the relation between x^r_{ijm}, t^r_m and w^r_m. In this figure, we analyze the first two moves of the route travelled by request $r \in R$. We suppose the origin o^r to be in node $i \in N$. The black arrows illustrate the moves; in particular, the first move is from node $i \in N$ to node $j \in N$ while the second one is from node $j \in N$ to node $k \in N$. The red arrows are associated with the timing variables t^r_m and indicate which node the arrival time values refers to. The green boxes relate to the waiting times w^r_m and are depicted next to the nodes they are assigned to.

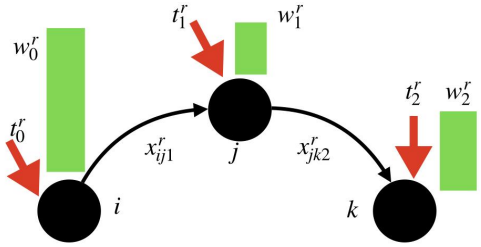


Figure 3.2: Example of routing (black arrows), timing (red arrows) and waiting variables (green boxes) for the first two moves of request $r \in R$.

Equivalent variables for the vehicles are needed. For routing purposes, we introduce binary variables y^v_{ijm} which are one when vehicle $v \in V$ travels arc $(i,j) \in A$ at move $m \in M^v$ and zero otherwise. In addition, continuous variables t^v_m and w^v_m are used to characterize the timing of vehicle $v \in V$ for move $m \in \{0\} \cup M^v$.

Naming c_r the actual arrival time of request $r \in R$ to its destination d^r, we define continuous variables c^+_r and c^-_r such that $c^+_r = \max(c_r, pa^r)$ and $c^-_r = \min(c_r, pa^r)$. Similar considerations apply for continuous variables d^+_r and d^-_r which, in turn, are constrained such that $d^+_r = \max(z^r, pd^r)$ and $d^-_r = \min(z^r, pd^r)$, where z^r is the actual departure

time of request $r \in R$ from its origin o^r. This is useful to penalize late or early arrival and departure times, as described in Section 3.3.2.

Binary variables a_{rv} are one when request $r \in R$ is paired with vehicle $v \in V$ during its route and zero otherwise. In addition, binary variables $p_{rm_rvm_v}$ are one when request $r \in R$ during move $m_r \in M^r$ is paired with vehicle $v \in V$ during move $m_v \in M^v$ and zero otherwise. The behaviour of these variables is explained in detail in Section 3.3.2. Finally, we introduce binary variable u^r which assumes value one when request $r \in R$ is unserved and zero otherwise. Table 4.3.2 reports all the variables previously described.

t^r_m	time variable indicating when request $r \in R$ arrives at a node after move $m \in \{0\} \cup M^r$
t^v_m	time variable indicating when vehicle $v \in V$ arrives at a node after move $m \in \{0\} \cup M^v$
w^r_m	waiting variable indicating how much time request $r \in R$ waits after move $m \in \{0\} \cup M^r$
w^v_m	waiting variable indicating how much time vehicle $v \in V$ waits after move $m \in \{0\} \cup M^v$
x^r_{ijm}	binary variable which is one if request $r \in R$ travels arc $(i,j) \in A$ at move $m \in M^r$, zero otherwise
y^v_{ijm}	binary variable which is one if vehicle $v \in V$ travels arc $(i,j) \in A$ at move $m \in M^v$, zero otherwise
c^+_r	variable indicating late arrival for request $r \in R$
c^-_r	variable indicating early arrival for request $r \in R$
d^+_r	variable indicating late departure for request $r \in R$
d^-_r	variable indicating early departure for request $r \in R$
a_{rv}	binary variable which is one if request $r \in R$ is paired with vehicle $v \in V$, zero otherwise
$p_{rm_rvm_v}$	binary variable which is one if request $r \in R$ at move $m_r \in M^r$ is paired with vehicle $v \in V$ at move $m_v \in M^v$, zero otherwise
u^r	binary variable which is one if request $r \in R$ is unserved, zero otherwise

Table 3.2: Variables continuous time model

OBJECTIVE FUNCTION

The objective function aims at minimizing the generalized cost Z. This cost is set equal to the sum of the following eight terms. The first term indicates the travel cost of a solution which is given by:

$$\alpha \sum_{v \in V} \sum_{(i,j) \in A} \sum_{m \in M^v} y^v_{ijm} l_{ij}.$$

To penalize the time travelled by each passenger inside a vehicle (which is computed as arrival time minus departure time and waiting time), we introduce the second term:

$$\beta \sum_{r \in R} q^r (t^r_{\mathcal{M}^r} - \sum_{m \in \{0\} \cup M^r} w^r_m - t^r_0).$$

In this second term, $t^r_{\mathcal{M}^r}$ assumes the value of the time instant at which a request arrives at its destination. This is explained in detail in Section 3.3.2. Please note that the previous term penalizes long trips more than short ones.

The third term handles early and late departure while the fourth term determines the penalty for early and late arrival:

$$\mu_1 \sum_{r\in R}(pd^r - d_r^-) + \mu_2 \sum_{r\in R}(d_r^+ - pd^r),$$

$$\mu_3 \sum_{r\in R}(c_r^+ - pa^r) + \mu_4 \sum_{r\in R}(pa^r - c_r^-).$$

We assign a penalty related to the loss of quality every time there is a transfer. This is taken into account by the fifth term:

$$\eta \sum_{r\in R}\sum_{v\in V} a_{rv} q^r.$$

In order to penalize how much time passengers wait at transfer nodes, we add the following sixth term:

$$\gamma_1 \sum_{r\in R}\sum_{m\in M^r} q^r w^r_m.$$

In this sixth term, the first move (move number zero) is excluded because, for a request, waiting at its origin node is already penalized as an early or late departure.

The seventh term determines parking costs, which are considered proportional to the parking time:

$$\gamma_2 \sum_{v\in V}\sum_{m\in\{0\}\cup M^v} w^v_m.$$

Finally, the last term penalizes the unserved requests:

$$+E \sum_{r\in R} u^r q^r.$$

Hence, the generalized cost Z is given by:

$$\begin{aligned} Z = {} & \alpha \sum_{v\in V}\sum_{(i,j)\in A}\sum_{m\in M^v} y^v_{ijm} l_{ij} + \beta \sum_{r\in R} q^r (t^r_{\mathcal{M}^r} - \sum_{m\in\{0\}\cup M^r} w^r_m - t^r_0) + \\ & \mu_1 \sum_{r\in R}(pd^r - d_r^-) + \mu_2 \sum_{r\in R}(d_r^+ - pd^r) + \mu_3 \sum_{r\in R}(c_r^+ - pa^r) + \mu_4 \sum_{r\in R}(pa^r - c_r^-) + \\ & \eta \sum_{r\in R}\sum_{v\in V} a_{rv} q^r + \gamma_1 \sum_{r\in R}\sum_{m\in M^r} q^r w^r_m + \gamma_2 \sum_{v\in V}\sum_{m\in\{0\}\cup M^v} w^v_m + E \sum_{r\in R} u^r q^r. \end{aligned} \tag{3.1}$$

Although this objective function is quite elaborate, it approximates real-life.

ROUTING CONSTRAINTS

Firstly, to model the routing, we have to impose that, at each move, at most one arc can be chosen. This is provided by constraints (3.2).

$$\sum_{(i,j)\in A} x^r_{ijm} \le 1, \forall r \in R, \forall m \in M^r \tag{3.2}$$

Also, we have to ensure that each request is either unserved (i.e. $u^r = 1$) or starts at its origin, possibly travel through some nodes and finally ends at its destination. This is ensured by constraints (3.3)-(3.5). In particular, constrains (3.3) enforce that, if a request moves (i.e. is served), it has to start from its origin. Constraints (3.4) establish flow conservation in each node while constraints (3.5) impose that request $r \in R$ has either to arrive at its destination or it is unserved.

$$\sum_{(o^r,j)\in A} x^r_{o^r j1} \geq \sum_{(i,j)\in A} x^r_{ij1}, \forall r \in R, \tag{3.3}$$

$$\sum_{(i,j)\in A} x^r_{ijm} = \sum_{(j,k)\in A} x^r_{jk(m+1)}, \forall r \in R, \forall m \in M^r \setminus \{\mathscr{M}^r\}, j \neq d^r, \tag{3.4}$$

$$u^r + \sum_{m_r \in M^r} \sum_{(i,d^r)\in A} x^r_{id^r m_r} = 1, \forall r \in R \tag{3.5}$$

Similar considerations hold for the vehicle flow. Constraints (3.6) restrict each vehicle to select at most one arc per move while constraints (3.7) force a moving vehicle to start from its origin. Constraints (3.8) ensures flow conservation in all nodes. In fact, this set of constraints imposes that, if a vehicle uses exactly $\hat{m} \leq \mathscr{M}^v$ arcs to reach its destination, it holds that for each $m \leq \hat{m}$, $\sum_{(i,j)\in A} y^v_{ijm} = 1$ and that after the $\hat{m}_{th}$ move $\sum_{(i,j)\in A} y^v_{ijm} = 0$. The last move is not considered because it has no following move $(m+1)$. Vehicles have to obey constraints similar to the ones of the requests, but have no defined destination node; hence, there is no associated constraint.

$$\sum_{(i,j)\in A} y^v_{ijm} \leq 1, \forall v \in V, \forall m \in M^v \tag{3.6}$$

$$\sum_{(o^v,j)\in A} y^v_{o^v j1} \geq \sum_{(i,j)\in A} y^v_{ij1}, \forall v \in V \tag{3.7}$$

$$\sum_{(i,j)\in A} y^v_{ijm} \geq \sum_{(j,k)\in A} y^v_{jk(m+1)}, \forall v \in V, \forall m \in M^v \setminus \{\mathscr{M}^v\}, \forall j \in N \tag{3.8}$$

TIMING CONSTRAINTS

Without loss of generality, we assume time to start at zero. Firstly, we ensure, through constraints (3.9) and (3.10), timing and waiting variables to be positive.

$$t^r_m \geq 0, \forall r \in R, \forall m \in \{0\} \cup M^r \tag{3.9}$$

$$w^r_m \geq 0, \forall r \in R, \forall m \in \{0\} \cup M^r \tag{3.10}$$

Then, we impose chronological timing via the so-called precedence constraints. With chronological timing, we mean that the arrival time at any node (but the first) is the arrival time at the previous node plus the waiting time at the previous node plus the travel time. This is ensured by constraints (3.11). Clearly, if $F(ijt)$ is a time independent parameter, constraints (3.11) are linear. Nonetheless, in Section 3.3.3, we derive a linear

formulation for the case where the travel time depends on the departure time. If no arc is chosen, then $\sum_{(i,j)\in A} x^r_{ijm}$ is zero, hence $t^r_{m+1} = t^r_m + w^r_m$.

$$t^r_{m+1} = t^r_m + w^r_m + \sum_{(i,j)\in A} x^r_{ijm} F(ij(t^r_m + w^r_m)), \forall r \in R, \forall m \in \{0\} \cup M^r \setminus \{\mathscr{M}^r\} \tag{3.11}$$

Also, we force the initial time of request $r \in R$ to be its earliest time instant (constraints (3.12)) and we force its last time instant to be smaller than its latest arrival time (constraints (3.13)).

$$t^r_0 = e^r, \forall r \in R \tag{3.12}$$

$$t^r_{\mathscr{M}^r} \leq l^r, \forall r \in R \tag{3.13}$$

To not have conveniently large waiting times at the end of the route to better fit time preferences, we impose constraints (3.14). In fact, when no arc is chosen for a certain move, i.e. request $r \in R$ has reached its destination, we have no waiting time. This means that $t^r_m = t^r_{\hat{m}}$ for $m \geq \hat{m}$, with $\hat{m}$ the move with which request $r \in R$ has reached its destination.

$$w^r_m \leq B \sum_{(i,j)\in A} x^r_{ijm}, \forall r \in R, \forall m \in M^r \tag{3.14}$$

Constraints (3.13) do not imply that we have to reach the destination node at the last move. Rather, they say that if a request reaches its destination at move $\hat{m}$, then $t^r_m = t^r_{\hat{m}}$ for $\hat{m} \leq m \leq \mathscr{M}^r$. This is ensured by the absence of waiting times once the destination is reached (constraints 3.14).

For vehicles, we duplicate the equivalent of the timing constraints with small adjustments; in particular, we impose:

$$t^v_m \geq 0, \forall v \in V, \forall m \in \{0\} \cup M^v, \tag{3.15}$$

$$w^v_m \geq 0, \forall v \in V, \forall m \in \{0\} \cup M^v, \tag{3.16}$$

$$t^v_{m+1} = t^v_m + w^v_m + \sum_{(i,j)\in A} y^v_{ijm} F(ij(t^v_m + w^v_m)), \forall v \in V, \forall m \in \{0\} \cup M^v \setminus \{\mathscr{M}^v\}, \tag{3.17}$$

$$t^v_0 = 0, \forall v \in V, \tag{3.18}$$

$$t^v_{\mathscr{M}^v} \leq T_{Max}, \forall v \in V, \tag{3.19}$$

$$w^v_m \leq B \sum_{(i,j)\in A} y^v_{ijm}, \forall v \in V, \forall m \in M^r. \tag{3.20}$$

DEPARTURE AND ARRIVAL TIMES CONSTRAINTS

We want variable d^+_r to assume the maximum of the preferred departure time pd^r and the actual departure time $t^r_0 + w^r_0$ of request $r \in R$ and d^-_r to assume the minimum of these two. Hence, we adopt the following constraints:

$$d^+_r \geq pd^r, \forall r \in R \tag{3.21}$$

$$d^+_r \geq t^r_0 + w^r_0, \forall r \in R \tag{3.22}$$

$$d_r^- \leq pd^r, \forall r \in R \tag{3.23}$$

$$d_r^- \leq t_0^r + w_0^r, \forall r \in R \tag{3.24}$$

Similar constraints hold for the arrival time.

$$c_r^+ \geq pa^r, \forall r \in R \tag{3.25}$$

$$c_r^+ \geq t^r_{\mathcal{M}^r}, \forall r \in R \tag{3.26}$$

$$c_r^- \leq pa^r, \forall r \in R \tag{3.27}$$

$$c_r^- \leq t^r_{\mathcal{M}^r}, \forall r \in R \tag{3.28}$$

Given the previous constraints and the composition of the objective function, the departure and arrival time will be forced to be as close as possible to pd^r and pa^r.

PAIRING

In this section, we explain how to pair vehicles and requests. We use moves to discretize routes even though the timing is expressed in continuous variables. We use binary variable $p_{rm_rvm_v}$ to pair couple [request $r \in R$ - request move $m_r \in M^r$], with couple [vehicle $v \in V$- vehicle move $m_v \in M^v$]. On one hand, we impose that each request can be paired to at most one vehicle per move in constraints (3.29); on the other hand, a vehicle can be paired simultaneously with multiple requests as long as this does not violate its capacity, as expressed in constraints (3.30).

$$\sum_{v \in V} \sum_{m_v \in M^v} p_{rm_rvm_v} \leq 1, \forall r \in R, m_r \in M^r \tag{3.29}$$

$$\sum_{r \in R} \sum_{m_r \in M^r} p_{rm_rvm_v} q^r \leq q^v, \forall v \in V, m_v \in M^v \tag{3.30}$$

As long as a request and a vehicle are paired, they have to travel the same arcs (constraints (3.31) and (3.32)) at the same time (constraints (3.33) and (3.34)). Constraints (3.31)-(3.34) are the linearization of:

$$\sum_{(i,j) \in A} x^r_{ijm_r} y^v_{ijm_v} \geq p_{rm_rvm_v}, \forall r \in R, \forall m_r \in M^r, \forall v \in V, \forall m_v \in M^v$$

and

$$p_{rm_rvm_v}(t^v_{m_v} - t^r_{m_r}) = 0, \forall v \in V, \forall r \in R, \forall m_r \in M^r, \forall m_v \in M^v$$

which are always true when $p_{rm_rvm_v} = 0$. The inequality in the first nonlinear set of constraints is due to the fact that, by chance, a vehicle and a request may travel the same arc while not being paired together.

$$x^r_{ijm_r} \leq y^v_{ijm_v} + (1 - p_{rm_rvm_v}), \forall v \in V, \forall r \in R, \forall (i,j) \in A, \forall m_r \in M^r, \forall m_v \in M^v \tag{3.31}$$

$$x^r_{ijm_r} \geq y^v_{ijm_v} - (1 - p_{rm_rvm_v}), \forall v \in V, \forall r \in R, \forall (i,j) \in A, \forall m_r \in M^r, \forall m_v \in M^v \tag{3.32}$$

$$t^r_{m_r} \leq t^v_{m_v} + (1 - p_{rm_rvm_v})B, \forall v \in V, \forall r \in R, \forall m_r \in M^r, \forall m_v \in M^v \tag{3.33}$$

$$t^r_{m_r} \geq t^v_{m_v} - (1 - p_{r m_r v m_v})B, \forall v \in V, \forall r \in R, \forall m_r \in M^r, \forall m_v \in M^v \tag{3.34}$$

Constraints (3.33) and (3.34) (in combination with the timing constraints (3.17)) impose that, if a vehicle is paired to more than one request in $i \in N$, the arrival time in $j \in N$ of the vehicle and of all the paired requests is equal to the arrival time of the request that arrived the latest in $i \in N$ plus its waiting time plus the travel time through $(i, j) \in A$.

In addition, if a request is not paired with any vehicle, it cannot move (constraints (3.35)).

$$\sum_{(i,j)\in A} x^r_{ijm_r} \leq \sum_{v\in V} \sum_{m_v \in M^v} p_{r m_r v m_v}, \forall r \in R, \forall m_r \in M^r \tag{3.35}$$

Finally, we impose an upperbound on the maximum number of transfers a request can experience (constraints (3.36) and (3.37)). Variable a_{rv} captures if request $r \in R$ has ever been paired to vehicle $v \in V$. The number of transfers a request can encounter is the number of vehicles the request has been paired with minus the first one.

$$B a_{rv} \geq \sum_{m_r \in M^r} \sum_{m_v \in M^v} p_{r m_r v m_v}, \forall v \in V, \forall r \in R \tag{3.36}$$

$$\sum_{v\in V} a_{rv} - 1 \leq b^r, \forall r \in R \tag{3.37}$$

DOUBLE PICK UP

There is an extremely unlikely case where the model does not see a transfer. This eventuality never happens in our instances and rarely can happen in practice; nevertheless, for the sake of completeness, we explain the possible problem throughout an example (depicted in Figure 3.3). Suppose that we have 5 time instants in chronological order, $t_1 < t_2 < t_3 < t_4 < t_5$ and that vehicle $v \in V$ and request $r \in R$ are paired at time t_1. Then, at time t_2 the request is dropped off at node $i \in N$ while the vehicle continues its route reaching node $j \in N$ at time t_3. Finally, the same vehicle comes back to node i at time t_4 to pick up again request $r \in R$ and brings it to node $k \in N$ at time t_5. This second pick up is not counted in the model. In general, we can say that if a particular vehicle, for multiple times, picks up and drops off the very same request consecutively, not all the resulting transfers are properly counted. Please note that if two different vehicles sequentially pick up the same request, the transfer is correctly accounted for by the model (as we show in the example depicted in Fig. 3.8ii). This double pick up by the same vehicle can happen only when there are large time windows and tight capacity constraints at the same time. In fact, the complete route of a new request should fit inside the route of an already existing one. Time limitations make this possibility unlikely. Given the extreme rarity of this situation, we do not consider this eventuality.

3.3.3. MODEL EXTENSION

In this section, we show how to extend the model such that it considers people dependent service times (time for get-in operations). As motivated by Ichoua et al., 2003, Donati et al., 2008 and Schmid and Doerner, 2010, we also consider nonconstant travel times. Both features are added to have a closer resemblance to real-life.

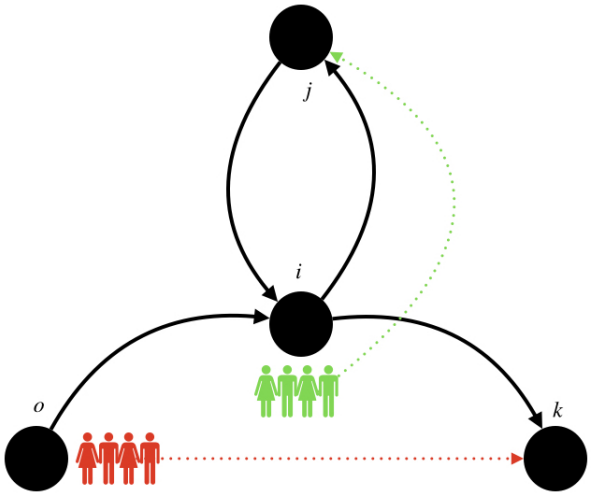


Figure 3.3: Example of a double pick up by the same vehicle. The black arrows represent the route of the vehicle (o, i, j, i, k). The red request has origin o and destination k while the green request has origin i and destination j. The combined party sizes of the two requests exceeds the capacity of the vehicle.

PEOPLE DEPENDENT SERVICE TIME

In this section, we explain how to consider different service times for different requests (people dependent service time). As service time, we consider the time needed to get in a vehicle. Also, we assume each request $r \in R$ to be picked up by a particular vehicle $v \in V$ at most once. Each request is considered to have a known service time named s^r. We introduce binary variable g^r_m which assumes value one if request $r \in R$ experiences a get in operation at move $m \in M^r$ and zero otherwise. To ensure this, we impose the following three constraints: constraints (3.38) for the first move of a request, constraints (3.39) for the first move of a vehicle and constraints (3.40) for all other moves of requests and vehicles.

$$g^r_1 \geq \sum_{v \in V} \sum_{m_v \in M^v} p_{r1vm_v}, \forall r \in R \tag{3.38}$$

$$g^r_{m_r} \geq \sum_{v \in V} p_{rm_rv1}, \forall r \in R, \forall m_r \in M^r \tag{3.39}$$

$$g^r_{m_r} \geq p_{rm_rvm_v} - p_{rm_r-1vm_v-1}, \forall v \in V, \forall m_v \in M^v \setminus \{1\}, \forall r \in R, \forall m_r \in M^r \setminus \{1\} \tag{3.40}$$

Constraints (3.40) exploit that, when a request is picked up, the vehicle and the request were not paired at the previous move (hence, $p_{rm_r-1vm_v-1} = 0$) while, at the current move, they are paired (hence, $p_{rm_rvm_v} = 1$). Thus, the difference between the two values is one. The corresponding timing constraints (3.11) are then modified to:

$$t^r_{m+1} = t^r_m + w^r_m + g^r_m s^r + \sum_{(i,j) \in A} x^r_{ijm} F(ij(t^r_m + w^r_m)), \forall r \in R, \forall m \in \{0\} \cup M^r \setminus \{\mathscr{M}^r\}. \tag{3.41}$$

We assume that, when vehicle $v \in V$ picks up two requests simultaneously from the same node, only the largest service time is considered. In Appendix 3.A.1, we show a different formulation for the case when service times add up.

To model this for vehicles, we introduce binary variables h_{vmr} which assumes value one if and only if two conditions apply. The first condition is that vehicle $v \in V$ at move $m \in M^v$ picks up request $r \in R$ (and maybe some others). The second condition is that, among all the requests picked up at move $m \in M^v$ by vehicle $v \in V$, request $r \in R$ has the highest service time. To ensure that h_{vmr} can be one for only one request at move $m \in M^v$ of vehicle $v \in V$, we impose constraints (3.42).

$$\sum_{r \in R} h_{vmr} \leq 1, \forall v \in V, \forall m \in M^v \tag{3.42}$$

To ensure the selection of the request with the highest service time, we impose the following three constraints: constraints (3.43) for the first move of a vehicle, constraints (3.44) for the first move of a request and constraints (3.45) for all other moves of requests and vehicles.

$$\sum_{r \in R} h_{v1r} s^r \geq p_{r m_r v 1} s^r, \forall v \in V, \forall r \in R, \forall m_r \in M^r. \tag{3.43}$$

$$\sum_{r \in R} h_{v m_v r} s^r \geq p_{r 1 v m_v} s^r, \forall v \in V, \forall m_v \in M^v, \forall r \in R. \tag{3.44}$$

$$\sum_{r \in R} h_{v m_v r} s^r \geq (p_{r m_r v m_v} - p_{r m_r - 1 v m_v - 1}) s^r, \forall v \in V, \forall m_v \in M^v \setminus \{1\}, \forall r \in R, \forall m_r \in M^r \setminus \{1\}. \tag{3.45}$$

For vehicle $v \in V$ at move $m_v \in M^v$, constraints (3.43)-(3.45) force binary variables h_{vmr} to be one for request $r \in R$ such that its related service time s^r is greater than or equal to the service times that the vehicle is experiencing. Then, we impose through constraints (3.46) and (3.47) a tight upper bound. In fact, constraints (3.46) allow variable h_{vmr} to assume value one only if request $r \in R$ was not paired to vehicle $v \in V$ at the previous move. Constraints (3.47) force h_{vmr} to be zero if request $r \in R$ is not paired with vehicle $v \in V$ at move $m \in M^v$.

$$h_{v m_v r} \leq 1 - \sum_{m_r \in M^r} p_{r m_r v m_v - 1}, \forall v \in V, \forall m_v \in M^v \setminus \{1\}, \forall r \in R \tag{3.46}$$

$$h_{v m_v r} \leq \sum_{m_r \in M^r} p_{r m_r v m_v}, \forall v \in V, \forall m_v \in M^v, \forall r \in R \tag{3.47}$$

Finally, we change constraints (3.17) into:

$$t^v_{m+1} = t^v_m + w^v_m + \sum_{r \in R} s^r h_{vmr} + \sum_{(i,j) \in A} y^v_{ijm} F(ij(t^v_m + w^v_m)), \forall v \in V, \forall m \in \{0\} \cup M^v \setminus \{\mathscr{M}^v\}. \tag{3.48}$$

NONCONSTANT TRAVEL TIME

So far, we have considered the generic function $F(ijt)$. In this section, we show how to linearize this function. A similar approach to deal with nonconstant travel times can be found in Malandraki and Daskin, 1992. The main difference is that Malandraki and Daskin, 1992 developed a method for the TSP; hence, for each node, exactly one incoming and one outgoing arc is travelled. With our formulation instead, for each node, zero, one, or more incoming and outgoing arcs can be travelled.

Firstly, when nonconstant travel times are taken into account, we need to ensure the same starting time for each pairing[2]. Hence, we add constraints (3.49) and (3.50). These constraints explicitly force the starting time of a pairing to be the same.

$$t^r_{m_r-1} + w^r_{m_r-1} \leq t^v_{m_v-1} + w^v_{m_v-1} + (1 - p_{r m_r v m_v})B, \forall v \in V, \forall r \in R, \forall m_r \in M^r, \forall m_v \in M^v \tag{3.49}$$

$$t^r_{m_r-1} + w^r_{m_r-1} \geq t^v_{m_v-1} + w^v_{m_v-1} - (1 - p_{r m_r v m_v})B, \forall v \in V, \forall r \in R, \forall m_r \in M^r, \forall m_v \in M^v \tag{3.50}$$

Since variables t^r_m and w^r_m are defined for $m \in \{0\} \cup M^r$, constraints (3.49) can range for all $m_r \in M^r$ even though they refer to time and waiting variable at the previous move. Similar considerations hold for constraints (3.50) and variables t^v_m and w^v_m. Secondly, we define TS as the set of time slots in which we can have different travel times (e.g. three time slots in Figure 3.4).

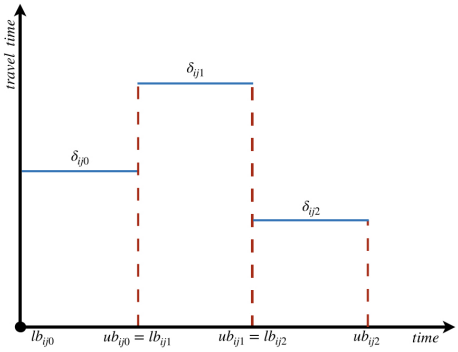


Figure 3.4: Example of three time slots

In order to have a finite number of time slots, the function $F(ijt)$ must be piece-wise constant. If not, it is still possible to approximate $F(ijt)$ to a piece-wise discrete function and use its approximation. We introduce binary variables $TT^r_{ijm_rk}$ and $TT^v_{ijm_vk}$ which are one when arc $(i, j) \in A$ is chosen for move $m_r \in M^r$ of request $r \in R$ and for move $m_v \in M^v$ of vehicle $v \in V$, respectively, in time slot $k \in TS$ and zero otherwise. Parameters δ_{ijk} represent the amount of time needed to travel arc $(i, j) \in A$ in the time slot $k \in TS$. Each time slot $k \in TS$ is defined by a lower and an upper bound - namely, lb_{ijk} and ub_{ijk}, respectively - such that $ub_{ijk-1} = lb_{ijk}$. Subsequently, we need to constrain variables TT^r_{ijmk} and TT^v_{ijmk} to assume value one if and only if, at move $m \in M^v$ or $m \in M^r$, respectively, arc $(i, j) \in A$ is chosen at time t which falls into time slot $k \in TS$. To ensure this, we impose constraints (3.51), (3.52) and (3.53) for the requests and constraints (3.54), (3.55) and (3.56) for the vehicles.

$$\sum_{k \in TS} TT^r_{ijmk} = x^r_{ijm}, \forall r \in R, \forall m \in M^r, \forall (i, j) \in A \tag{3.51}$$

[2] For example, given time-dependent travel time T(t) such that T(t_1) = 3,T(t_2) = 2 and $t_2 = t_1 + 1$, then different starting times can lead to the same arrival time.

$$\sum_{(i,j)\in A} TT^r_{ijmk} lb_{ijk} \leq t^r_m + w^r_m + g^r_m s^r, \forall r \in R, \forall m \in M^r, \forall k \in TS \tag{3.52}$$

$$t^r_m + w^r_m + g^r_m s^r \leq \sum_{(i,j)\in A} TT^r_{ijmk} ub_{ijk} + B(1 - \sum_{(i,j)\in A} TT^r_{ijmk}), \forall r \in R, \forall m \in M^r, \forall k \in TS \tag{3.53}$$

$$\sum_{k\in TS} TT^v_{ijmk} = y^v_{ijm}, \forall v \in V, \forall m \in M^v, \forall (i,j) \in A \tag{3.54}$$

$$\sum_{(i,j)\in A} TT^v_{ijmk} lb_{ijk} \leq t^v_m + w^v_m + \sum_{r\in R} s^r h_{vmr}, \forall v \in V, \forall m \in M^v, \forall k \in TS \tag{3.55}$$

$$t^v_m + w^v_m + \sum_{r\in R} s^r h_{vmr} \leq \sum_{(i,j)\in A} TT^v_{ijmk} ub_{ijk} + B(1 - \sum_{(i,j)\in A} TT^v_{ijmk}), \forall v \in V, \forall m \in M^v, \forall k \in TS \tag{3.56}$$

Variables $TT^r_{ijm_rk}$ and $TT^v_{ijm_vk}$ can fully substitute variables $x^r_{ijm_r}$ and $y^v_{ijm_v}$, as we can clearly see from the equality constraints (3.51) and (3.54).

Finally, we change constraints (3.41) to:

$$t^r_{m+1} = t^r_m + w^r_m + g^r_m s^r + \sum_{k\in TS} \sum_{(i,j)\in A} TT^r_{ijmk} \delta_{ijk}, \forall r \in R, \forall m \in M^r \setminus \{\mathscr{M}^r\}, \tag{3.57}$$

and constraints (3.48) to:

$$t^v_{m+1} = t^v_m + w^v_m + \sum_{r\in R} s^r h_{vmr} + \sum_{k\in TS} \sum_{(i,j)\in A} TT^v_{ijmk} \delta_{ijk}, \forall v \in V, \forall m \in M^v \setminus \{\mathscr{M}^v\}. \tag{3.58}$$

In this way, we obtain a linear formulation which is able to consider nonconstant travel times.

3.4. DISCRETE TIME MODEL

In this section, we describe the core discrete time model (Section 3.4.1) and its extension (Section 3.4.2). We assume the discretization step to be one time unit; if not, every time related equation should be multiplied by a scaling factor. Given the introduction of the discretization step, we define T as the set of all time instants, from the very first time instant zero until the last possible time instant T_{Max}. Also, for every request $r \in R$, we define T^r as the set of all time instants in $[e^r, l^r - 1]$. For each request $r \in R$, the set T^r defines the time instants at which r could start traveling an arc without violating its latest time instant.

For this model, we use a space-time network, which means that every node is duplicated for each considered time instant. Also, for every arc $(i,j) \in A$, i.e. the physical network, there exist many arcs in the space-time network. In general, there exist one for each time instant. Arcs are modelled as follows: in the space-time network, each node $i \in N$ at time $t \in T$ is connected to each node $j \in N$ at time $t_2 \in T$ such that t_2 is equal to t plus the travel time from i to j at time t, i.e. δ_{ijt}. Each node $i \in N$ at time t is also connected to node $i \in N$ at time $t+1$. These last arcs are used to model parking or passengers waiting for a(nother) vehicle. Figure 3.5 displays a simple example of how to transform

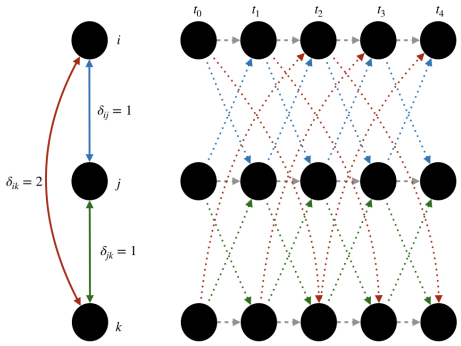


Figure 3.5: Example of space-time network. On the left, the original network; on the right, its associated space-time network.

a standard network into a space-time network. The structure of the space-time network allows to directly consider arcs with travel times depending on the departure time. We denote the set of all arcs in the space-time network by A^*. Since the travel times are embedded in the space-time formulation, routing variables embed timing causality. In fact, we introduce, both as routing and timing variables, binary variables x^r_{ijt} and y^v_{ijt}. The first variable assumes value one if request $r \in R$ travels arc $(i, j, t) \in A^*$ and zero otherwise; identically, the second variable assumes value one if vehicle $v \in V$ travels arc $(i, j, t) \in A^*$ and zero otherwise. In addition, we introduce binary variables a_{rv} and a_{rvt}. These variables are used to model request-vehicle pairing. In fact, a_{rvt} assumes value one if request $r \in R$ is carried by vehicle $v \in V$ at time $t \in T$ and a_{rv} assumes value one if request $r \in R$ was carried by vehicle $v \in V$ at any time $t \in T$. Finally, we introduce variable u^r for request $r \in R$ which assumes value one if the request is unserved, zero otherwise. These sets, parameters and variables are summarized in Table 3.3 and Table 3.4.

T	set of all time instants
T^r	set of time instants $t \in T$ such that $t \in [e^r, l^r - 1], \forall r \in R$
A^*	set of arcs in the space-time network
δ_{ijt}	travel time of arc $(i, j, t) \in A^*$

Table 3.3: Sets and parameters for the discrete time model

Although they may appear similar, our formulation strongly differs from time-index formulations (van den Bergh et al., 2016) because, in our formulation, vehicles are allowed to travel the same arc multiple times. Moreover, some authors (Cortés et al., 2010, Masson et al., 2014) modelled transfer nodes as a couple of dummy nodes (one node for the drop-off and one for the pick-up) connected by a direct arc with zero travel time. The main advantage of duplicating transfer nodes is that, in doing so, the chronological order within the transfer is respected by construction. In general, this is a useful property, but it is redundant in our case. In fact, in the discrete time case, the chronological order

x^r_{ijt}	binary variable which is one if request $r \in R$ travels arc $(i, j, t) \in A^*$, zero otherwise
y^v_{ijt}	binary variable which is one if vehicle $v \in V$ travels arc $(i, j, t) \in A^*$, zero otherwise
a_{rvt}	binary variable which is one if vehicle $v \in V$ carries request $r \in R$ at time $t \in T$, zero otherwise
a_{rv}	binary variable which is one if vehicle $v \in V$ carries request $r \in R$, zero otherwise
u^r	binary variable which is one if request $r \in R$ is unserved, zero otherwise

Table 3.4: Variables for the discrete time model

within the transfer is also respected by construction in the space-time network. In the continuous case instead, it is enforced due to explicit timing variables and constraints (Section 3.3.2). Introducing duplicates for the transfer nodes would not exclude the need for timing variables and constraints since we also aim at minimizing travel times, waiting times at transfer nodes and premature/late arrival and departure, all of which still has to be explicitly modelled via the timing variables and constraints that we introduce.

3.4.1. CORE MODEL

Before introducing the mathematical formulation, we explain how we model waiting and parking. We model parking of vehicle $v \in V$ at node $i \in N$ at time $t \in T$, by moving from node $i \in N$ at time $t \in T$ to node $i \in N$ at time $t+1 \in T$; hence, variable y^v_{iit} assumes value one. The same holds for requests when they are waiting.

We employ a similar approach to determine early or late arrival and departure. For example, to know at what time request $r \in R$ arrives at its destination $d^r \in N$, we consider the values of the flows $x^r_{d^r d^r t}$ from $t = l^r$ backwards. The time instant $t \in T$ such that $x^r_{d^r d^r t-1} = 0$ and $x^r_{d^r d^r t} = 1$ is the time at which request $r \in R$ arrived at its destination $d^r \in N$. Additionally, if request $r \in R$ has preferred arrival time $pa^r \in T$ for destination $d^r \in N$, we compute $\sum_{t \geq pa^r}(1 - x^r_{d^r d^r t})$ to determine the number of time instants it was late. Similar ideas are used to determine early or late arrival, early or late departure and also to establish waiting times at transfer nodes.

OBJECTIVE FUNCTION

The objective function minimizes the generalized cost Z (i.e. routing cost and costs related to a loss of quality of the service) and it is composed of eight terms. Each term penalizes one of the following: travel costs, passenger time spent in the vehicle, early or late arrival and departure times, number of transfers, passenger time spent waiting at transfer nodes, parking costs and unserved requests.

The first term determines the travel costs:

$$\alpha \sum_{v \in V} \sum_{(i,j,t) \in A^*} y^v_{ijt} l_{ij}.$$

The time spent by passengers in the vehicles is penalized in the second term:

$$\beta \sum_{r \in R} \sum_{(i,j,t) \in A^*, i \neq j} x^r_{ijt} \delta_{ijt} q^r.$$

The third and fourth terms are composed of two components each and determine the incurred penalty for departing and arriving early or late. To define the penalty for departing early, we use

$$\mu_1 \sum_{r\in R} \sum_{t<pd^r \in T^r} (1 - x^r_{o^r o^r t}) q^r$$

while the penalty if a client departs too late is regulated by

$$\mu_2 \sum_{r\in R} \sum_{t\geq pd^r \in T^r} x^r_{o^r o^r t} q^r.$$

The penalty for arriving early is given by

$$\mu_3 \sum_{r\in R} \sum_{t<pa^r \in T^r} x^r_{d^r d^r t} q^r$$

while the penalty for arriving late is defined by:

$$\mu_4 \sum_{r\in R} \sum_{t\geq pa^r \in T^r} (1 - x^r_{d^r d^r t}) q^r.$$

The fixed penalty for each transfer is given by the fourth term:

$$\eta \sum_{r\in R} \sum_{v\in V} a_{rv} q^r.$$

To consider the reduction of the quality of the service due to waiting at transfer nodes, we introduce

$$\gamma_1 \sum_{r\in R} \sum_{(i,i,t)\in A^*} x^r_{iit} q^r$$

with $i \neq o^r$ and $i \neq d^r$. To determine the parking cost, we add:

$$\gamma_2 \sum_{v\in V} \sum_{(i,i,t)\in A^*} y^v_{iit}.$$

Finally, to penalize the unserved requests we add:

$$E \sum_{r\in R} u^r q^r.$$

Hence, the generalized cost Z is given by:

$$\begin{aligned} Z = {} & \alpha \sum_{v\in V} \sum_{(i,j,t)\in A^*} y^v_{ijt} l_{ij} + \beta \sum_{r\in R} \sum_{(i,j,t)\in A^*, i\neq j} x^r_{ijt} \delta_{ijt} q^r + \mu_1 \sum_{r\in R} \sum_{t<pd^r \in T^r} (1 - x^r_{o^r o^r t}) q^r + \\ & \mu_2 \sum_{r\in R} \sum_{t\geq pd^r \in T^r} x^r_{o^r o^r t} q^r + \mu_3 \sum_{r\in R} \sum_{t<pa^r \in T^r} x^r_{d^r d^r t} q^r + \mu_4 \sum_{r\in R} \sum_{t\geq pa^r \in T^r} (1 - x^r_{d^r d^r t}) q^r + \\ & \eta \sum_{r\in R} \sum_{v\in V} a_{rv} q^r + \gamma_1 \sum_{r\in R} \sum_{(i,i,t)\in A^*} x^r_{iit} q^r + \gamma_2 \sum_{v\in V} \sum_{(i,i,t)\in A^*} y^v_{iit} + E \sum_{r\in R} u^r q^r. \end{aligned} \tag{3.59}$$

ROUTING AND TIMING CONSTRAINTS

Because of the structure of the space-time network, routing constraints directly translate to timing constraints. For each request $r \in R$, we enforce its flow to start from its origin at $t = e^r$ (constraints (3.60) and (3.61)) and either to end at $t = l^r$ or to be unserved (constraints (3.62)).

$$\sum_{i,j \in N, i \neq o^r} x^r_{ije^r} = 0, \forall r \in R \tag{3.60}$$

$$\sum_{j \in N} x^r_{o^r j e^r} = 1, \forall r \in R \tag{3.61}$$

$$\sum_{i \in N} x^r_{id^r t_2} + u^r = 1, \forall r \in R, t_2 | t_2 + \delta_{ijt_2} = l^r \tag{3.62}$$

For each vehicle, we impose that its route starts from its origin at time $t = 0$ (constraint (3.63) and (3.64)) and ends at $t = T_{Max}$ (constraint (3.65)).

$$\sum_{i,j \in N, i \neq o^v} y^v_{ij0} = 0, \forall v \in V \tag{3.63}$$

$$\sum_{j \in N} y^v_{o^v j 0} = 1, \forall v \in V, \tag{3.64}$$

$$\sum_{i \in N} \sum_{j \in N} y^v_{ijt} = 1, \forall v \in V, \forall t \in T | t + \delta_{ijt} = T_{Max} \tag{3.65}$$

We establish flow conservation through constraints (3.66) and (3.67).

$$\sum_{i \in N} x^r_{ijt_2} = \sum_{i \in N} x^r_{jit}, \forall r \in R, \forall j \in N, \forall t \in T^r \tag{3.66}$$

$$\sum_{i \in N} y^v_{ijt_2} = \sum_{i \in N} y^v_{jit}, \forall v \in V, \forall j \in N, \forall t \in T / \{T^{max}\} \tag{3.67}$$

where $t_2 + \delta_{ijt_2} = t$ and $t_2 \in T$.

PAIRING CONSTRAINTS

In this section, we explain how to pair vehicles and requests. Firstly, we have to impose that each request at any time instant can be paired to at most one vehicle (constraints (3.68)). A request can be paired with no vehicle for some time; for example, when it is waiting.

$$\sum_{v \in V} a_{rvt} \leq 1, \forall r \in R, \forall t \in T^r \tag{3.68}$$

Differently, a vehicle can be paired with more than one request, as long as this does not violate its capacity constraint, which is given by:

$$\sum_{r \in R} a_{rvt} q^r \leq q^v, \forall v \in V, \forall t \in T. \tag{3.69}$$

Also we have to impose that, as long as a vehicle and a request are paired, they have to travel the same route, i.e. if $a_{rvt} = 1$, then $x^r_{ijt} = y^v_{ijt}, \forall (i, j, t) \in A^*$. We do so by imposing

$$x^r_{ijt} \leq y^v_{ijt} + (1 - a_{rvt}), \forall r \in R, \forall v \in V, \forall (i,j,t) \in A^* | t \in T^r \tag{3.70}$$

and

$$x^r_{ijt} \geq y^v_{ijt} - (1 - a_{rvt}), \forall r \in R, \forall v \in V, \forall (i,j,t) \in A^* | t \in T^r. \tag{3.71}$$

Yet, constraints (3.70) and (3.71) can be equivalently rewritten as:

$$\sum_{j \in N} x^r_{ijt} \leq \sum_{j \in N} y^v_{ijt} + (1 - a_{rvt}), \forall r \in R, \forall v \in V, \forall i \in N, \forall t \in T^r, \tag{3.72}$$

$$\sum_{j \in N} x^r_{ijt} \geq \sum_{j \in N} y^v_{ijt} - (1 - a_{rvt}), \forall r \in R, \forall v \in V, \forall i \in N, \forall t \in T^r, \tag{3.73}$$

$$\sum_{i \in N} x^r_{ijt} \leq \sum_{i \in N} y^v_{ijt} + (1 - a_{rvt}), \forall r \in R, \forall v \in V, \forall j \in N, t \in T^r, \tag{3.74}$$

$$\sum_{i \in N} x^r_{ijt} \geq \sum_{i \in N} y^v_{ijt} - (1 - a_{rvt}), \forall r \in R, \forall v \in V, \forall j \in N, \forall t \in T^r. \tag{3.75}$$

Constraints (3.70)-(3.75) are always true when $a_{rvt} = 0$. The first formulation (3.70) and (3.71) is composed by $2 \cdot |N|^2 \cdot |T^r| \cdot |R| \cdot |V|$ constraints, while the second formulation (3.72)-(3.75) requires only $2/|N|$ of those constraints. Even though fewer constraints do not necessary mean that the problem is easier to solve (valid inequalities are an example), in this case it does.

Additionally, to enforce that whenever a request is not paired to any vehicle, it has to wait, we impose constraints (3.76).

$$\sum_{i \in N} x^r_{iit} \geq 1 - \sum_{v \in V} a_{rvt}, \forall r \in R, \forall t \in T^r \tag{3.76}$$

Finally, to model that each request may have a maximum of b^r transfers, we introduce constraints (3.77) and (3.78):

$$B a_{rv} \geq \sum_{t \in T} a_{rvt}, \forall v \in V, \forall r \in R, \tag{3.77}$$

$$\sum_{v \in V} a_{rv} - 1 \leq b^r, \forall r \in R. \tag{3.78}$$

3.4.2. MODEL EXTENSION

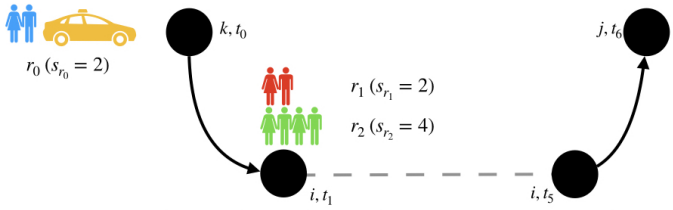
As for the continuous time case (Section 3.3.3), we show how to extend the model in order to consider people dependent service time (time for get-in operations). Differently with respect to the continuous time model, nonconstant travel times are directly embedded in the structure of the space-time network.

PEOPLE DEPENDENT SERVICE TIMES

In this section, we present how to introduce people dependent service times. We can set any finite integer number of time steps (s^r) as service times through constraints (3.79).

$$\sum_{i \in N} x^r_{iit} \geq a_{rvt} - a_{rvt-s^r}, \forall t \in T^r | t - s^r \in T^r, \forall r \in R, \forall v \in V \tag{3.79}$$

If, at the same time, more than one request are picked up, only the longest one is considered. How to model the sum of all service times instead of the longest service time is shown in Appendix 3.A.2. Figure 3.6 shows an example of how constraints (3.79) work. In this example, there are three requests; request r_0 is already in the taxi at time t_0 and it dropped off at time t_1. The other two requests, r_1 and r_2, can be picked up at t_1 the earliest (the time the taxi arrives in $i \in N$). Nevertheless, the service time of r_2 ($s^{r_2} = 4$) constraints the vehicles to move at t_5 the earliest. Hence, while the pick up process of r_2 starts immediately, the boarding process of r_1 (service time $s^{r_1} = 2$) starts at time t_3.



	t_0	t_1	t_2	t_3	t_4	t_5	t_6
a_{r_0vt}	1	0	0	0	0	0	0
$a_{r_0vt} - a_{r_0vt-s_{r_0}}$	0	0	-1	0	0	0	0
a_{r_1vt}	0	0	0	1	1	1	1
$a_{r_1vt} - a_{r_1vt-s_{r_1}}$	0	0	0	1	1	0	0
a_{r_2vt}	0	1	1	1	1	1	1
$a_{r_2vt} - a_{r_2vt-s_{r_2}}$	0	1	1	1	1	0	0

Figure 3.6: Example of service times. Request r_0 is already on vehicle v at time t_0 and drops off at time t_1. Request r_1 has a service time $s^{r_1} = 2$ time instant and leaves node i at time t_5. Request r_2 has a service time $s^{r_2} = 4$ time instant and leaves node i at time t_4.

Also, introducing service times leads to the following modifications of the two terms related to departure in the objective function (3.59):

$$\sum_{r \in R} \sum_{t < (pd^r + s^r) | t \in T^r} \mu_1 (1 - x^r_{o^r o^r t}) q^r$$

and

$$\sum_{r \in R} \sum_{t > (pd^r + s^r) | t \in T^r} \mu_2 x^r_{o^r o^r t} q^r .$$

In fact, the service time causes the departure time and the time at which the node is left to differ by s^r time instants. Given the different approach in modeling early and late departure, these modifications are not needed in the continuous time model.

3.5. COMPUTATIONAL EXPERIMENTS

Even though the DARPT is an NP-hard problem, the number of variables and constraints in both the continuous time and discrete time formulations are bounded by polynomial functions in the size of the problem. Table (3.5) summarizes variables in the continuous and in the discrete model.

Continuous time model	Discrete time model
t^r_m	–
t^v_m	–
w^r_m	–
w^v_m	–
x^r_{ijm}	x^r_{ijt}
y^v_{ijm}	y^v_{ijt}
c^+_r	–
c^-_r	–
d^+_r	–
d^-_r	–
a_{rv}	a_{rv}
$p_{rm_rvm_v}$	a_{rvt}
$g^v_{m_v}$	–
$g^r_{m_r}$	–
TT^v_{ijmk}	–
TT^r_{ijmk}	–
u^r	u^r

Table 3.5: Variables in continuous and discrete time model

In order to compare the models, we assume the discrete time step to be ϵ; then, it holds that $|M^v| = |T|$ and $|M^r| = |T^r|$. It follows that the variables in the continuous time model are more than the ones in the discrete time model. In fact, the continuous time model is composed of $3\cdot|R|\cdot|T^r|+3\cdot|V|\cdot|T|+|R|\cdot|A|\cdot|T^r|+|V|\cdot|A|\cdot|T|+5\cdot|R|+|R|\cdot|V|+|R|\cdot|T^r|\cdot|A|\cdot|K|+|V|\cdot|T|\cdot|A|\cdot|K|+|R|\cdot|T^r|\cdot|V|\cdot|T|$ variables (of which $2|R|\cdot|T^r|+2|V|\cdot|T|+4|R|$ are continuous), while the discrete time model has $|R|\cdot|A|\cdot|T^r|+|V|\cdot|A|\cdot|T|+|R|\cdot|V|\cdot|T^r|+|R|$ variables (all binary). Although these are two (almost) equivalent models, the continuous time one has $2\cdot|R|\cdot|T^r|+2\cdot|V|\cdot|T|+4\cdot|R|$ continuous variables more and $|R|\cdot|V|\cdot|T|(|T^r|-1)+|V|\cdot|T^v|+|R|\cdot|T^r|+|V|\cdot|T^v|\cdot|A|\cdot|K|+|R|\cdot|T^r|\cdot|A|\cdot|K|$ binary variables more.

Table 3.6 shows the number of constraints, divided by type, in both models. Also in this case, the discrete time model has less constraints with respect to its continuous counterpart.

3.5.1. BENCHMARK

To test the models, we apply the minor changes explained in Appendix 3.C; then, we create the following benchmark, based on real-life data. In general, in order to be use-

Type of constraints	Continuous time model	Discrete time model
Routing	$2\|R\|\cdot\|T^r\|+2\|R\|+2\|V\|\cdot\|T\|+\|V\|$	$3\|R\|+\|R\|\cdot\|N\|\cdot\|T^r\|$ $+3\|V\|+\|V\|\cdot\|N\|\cdot\|T\|$
Timing	$4\|R\|\cdot\|T^r\|+2\|R\|+4\|V\|\cdot\|T\|+2\|V\|$	–
Departure and arrival	$8\|R\|$	–
Pairing	$2\|R\|\cdot\|T^r\|+\|V\|\cdot\|T\|+\|R\|\cdot\|V\|$ $+2\|A\|\cdot\|R\|\cdot\|T^r\|\cdot\|V\|\cdot\|T\|$ $+2\cdot\|R\|\cdot\|T^r\|\cdot\|V\|\cdot\|T\|+\|R\|$	$2\|R\|\cdot\|T^r\|+\|V\|\cdot\|T\|$ $+4\|R\|\cdot\|V\|\cdot\|N\|\cdot\|T^r\|$ $+\|R\|\cdot\|V\|+\|R\|$
People depending service time	$\|R\|+\|R\|\cdot\|T^r\|+\|V\|\cdot\|T\|$ $+\|R\|\cdot\|T^r\|\cdot\|V\|+3\|R\|\cdot\|V\|\cdot\|T\|$ $+\|R\|\cdot\|T^r\|\cdot\|V\|\cdot\|T\|$	$\|R\|\cdot\|V\|\cdot\|T^r\|$
Nonconstant travel time	$2\|R\|\cdot\|T^r\|\cdot\|V\|\cdot\|T\|+\|R\|\cdot\|T^r\|\cdot\|A\|$ $+2\|R\|\cdot\|T^r\|\cdot\|TS\|+2\|V\|\cdot\|T\|\cdot\|TS\|$ $+\|V\|\cdot\|T\|\cdot\|A\|+\|R\|\cdot\|T^r\|+\|V\|\cdot\|T\|$	–

Table 3.6: Constraints in continuous and discrete time model

ful, transfers ask the length of the trips to be longer than the deviation and the stop of the vehicle. Hence, we choose to test our models with interurban trips. We select the twenty most populated cities (the central station of each city has been considered as the exact coordinate) in the most densely populated province of the Netherlands, i.e. South Holland. In particular, we choose: Rotterdam, Delft, Capelle aan den IJssel, Schiedam, Gouda, The Hague, Rijswijk, Voorburg, Leiden, Zoetermeer, Dordrecht, Zwijndrecht, Gorinchem, Spijkenisse, Vlaardingen, Barendrecht, Maassluis, Alphen aan den Rijn, Ridderkerk and Papendrecht. We connect these cities as shown in Figure 3.7.

Figure 3.7: South Holland region with our network.

We collected travel data, i.e. length and time of each arc, from Google Maps. On July 17, 2019 we acquired (expected) travel times for July 18, 2019 from 8:00AM to 9:45AM with time intervals of 15 minutes each. The chosen day is an average commuting day (Thursday) where no major road blocks were present. In our tests, we used a time step of one minute; hence, we set travel times for the time instants in between two collected data points to the value of the earliest data point. On this map, we test ten times each of the following cases: V2R3, V2R4, V2R5, V3R4, V3R5, V3R6, V3R7, V4R5, V4R6, V4R7, V4R8, V4R9. By ViRj, we mean that there are i vehicles and j requests in the instance.

Vehicles are initially positioned to give a good coverage of the area. Better coverage configurations could exist (Jagtenberg et al., 2015); however, this falls outside the scope of this paper. We position the first vehicle in Dordrecht, the second one in Vlaardingen, the third one (if any) in Alphen aan den Rijn and the last one (if any) in Rotterdam. The positions of the vehicles are indicated by the black nodes in Figure 3.8. The capacity q^v of each vehicle is set to 6.

We set the destination d^r of each request to The Hague (the green node in Figure 3.8) and the latest arrival time l^r to 10:00. This increases the chances that the last parts of some requests' routes overlap in space and are close in time. This makes transfers more likely to happen.

All requests have randomly chosen parameters such that the origin o^r is different from The Hague and such that the time preferences are coherent. In fact, we set the earliest time e^r to be earlier than the preferred departure time pd^r which in turn should be earlier than the preferred arrival time pa^r which also has to be earlier than the latest arrival time l^r. In addition, we set the preferred arrival time pa^r to be later than the preferred departure time pd^r plus the service time s_r. In formulas, this translates to: $e^r \leq pd^r \leq pd^r + s_r \leq pa^r \leq l^r$. These random choices are repeated until $1.2SPP \leq l^r - e^r \leq 1.5SPP$, where SPP is the shortest possible time to go from origin o^r to destination d^r (in the constant travel time configuration). For each request, the maximum number of transfers b^r and the party size q^r are uniformly randomly chosen among {1,2,3}. Service times s_r are set to q^r minutes.

All the above described instances and their logs with detailed solutions are available online[3]. The instances are tested in continuous and discrete time, both in the core and enlarged configurations. For comparison reasons, we also test the same instances in a no transfer setting by fixing the maximum number of transfers b^r to zero.

3.5.2. TUNING PARAMETERS

To tune the parameters, we refer mainly to Correia and van Arem, 2016 which, in turn, estimated the cost parameters based on a semi-real case study of private and public transportation in the Netherlands. The main differences and additions are the values of the travel costs α, the cost of a transfer η and the cost of waiting at a transfer node γ_1. The value of the travel cost α was set to €1 per km, instead of €0.1 per km, because this is closer to the prices for Dutch taxis (note that Correia and van Arem, 2016 referred to privately owned vehicles, not an on-demand service). Also, we set the cost of waiting at a transfer node γ_1 equal to €1.106 per minute, in between the cost of the time spent inside a vehicle β = €0.806 per minute and the cost of premature departure μ_1 = €1.306

[3] http://doi.org/10.4121/uuid:1ad27269-cdcf-43ed-a639-8fea09c48449

per minute. In addition, we set the cost of late arrival μ_4 to the cost of premature departure $\mu_1$and the costs of late departure μ_2 and premature arrival μ_3 to €0.306 per minute. Finally, the cost of each transfer η is set to €1 and the penalty E for each unserved request to €999. Table 3.7 displays all the values of the parameters.

Name	Value	Description
α	1 € per km	Travel cost per kilometer
β	0.806 € per min	Cost of time spent inside a vehicle per minute per person
γ_1	1.106 € per min	Cost of waiting at transfer nodes per minute per person
γ_2	1.81 € per $hour$	Cost of parking per minute
μ_1	1.306 € per min	Cost of premature departure per minute per person
μ_2	0.306 € per min	Cost of late departure per minute per person
μ_3	0.306 € per min	Cost of premature arrival per minute per person
μ_4	1.306 € per min	Cost of late arrival per minute per person
η	1 €	Cost of transfer
E	999 €	Penalty per person per unserved request

Table 3.7: Values of the parameters

3.5.3. TESTS

All tests are run on a Linux machine with the following architecture: x86_64, 4 CPUs (Intel(R) Core(TM) i5 CPU 660 @ 3.33GHz). It ran code written in Python 2.7 through the Spyder interface and adopting Gurobi 7.5.2 as the MILP solver. A time limit of one hour was set for the solution of each MILP.

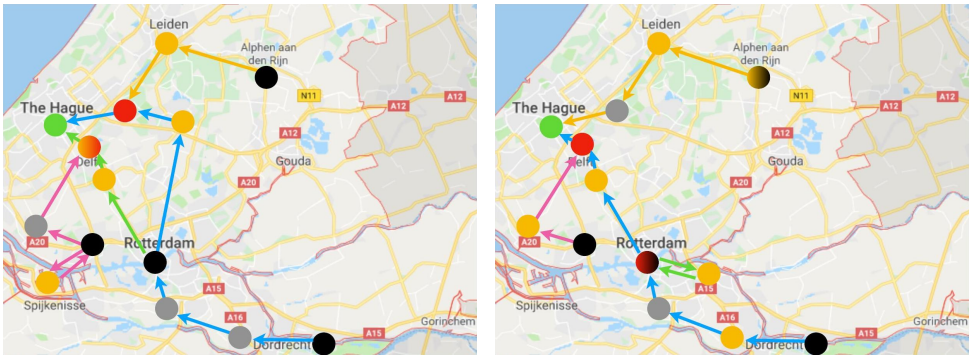
EXAMPLES

Figure 3.8 shows the results for the third and eighth instance of $V4R7$. In both cases, 4 vehicles are used to serve 5 requests (two requests are unserved). The grey nodes represent nodes that are simply passed by the vehicles (no pick up or transfers); the colored arcs indicate the path of each vehicle (one colour per vehicle) while their origin nodes are denoted by the black nodes. The red nodes represent nodes where a transfer happens, the origin nodes of the requests are depicted by yellow nodes while the green node indicates their final destination. The three nodes with double colors represent nodes where multiple actions happen. In particular in Figure 3.8i, the yellow and red node denotes the origin node of a request as well as a transfer node; in Figure 3.8ii, the yellow and black node denotes the origin node of a request as well as the origin node of a vehicle while the red and black node describes the origin node of a vehicle as well as a transfer node.

Figure 3.8i shows a transfer between the green and the pink vehicle and a transfer between the yellow and the blue vehicle. Figure 3.8ii shows a sequence of two transfers, from the green and the pink vehicle to the blue vehicle.

AVERAGE RESULTS

Table 3.8 shows the average results of the conducted tests. We remind that, for each scenario, 10 different instances were generated. The rows illustrate the features of the



(i) Two unrelated transfers

(ii) Two sequential transfers

Figure 3.8: Examples of transfers. The colored arcs indicate the path of each vehicle and their origin nodes are denoted by the black nodes. The origin nodes of the requests are depicted by yellow nodes while the green node indicates their final destination. The red nodes represent nodes where a transfer happens and the grey nodes represent nodes that are simply passed by the vehicles.

solutions. The first row indicates the number of vehicles $|V|$ and requests $|R|$ in the instances. The following eight rows specify how many instances were solved to optimality within the time limit for the various cases. The cases differ in continuous (C) and discrete (D) time model, core (Core) and enlarged (Enl) model and transfers (T) and no transfers (No-T) case.

Then, the following four rows indicate how many instances present at least one transfer. The number in brackets indicates how many of the instances not solved to optimality present at least one transfer in their incumbent solution. After showing how many instances have transfers, we display the total number of transfers happening over all instances per case. Similarly as before, the number in brackets indicate how many transfers happen in the incumbent solution of the instances not solved to optimality.

Succeeding the rows on transfers, we present features regarding the computation time and the objective function for the continuous time and discrete time models where transfers are allowed. The presented values of the objective functions and computation times are the averages over the values of the instances that both models could solve to optimality. In some cases, one of the two models could not solve any instance. Only in these cases, the average objective values and computation times of the model that could solve some instances are calculated on all these instances solved to optimality.

The following four rows indicate the gap percentage in the objective function and computation time between the continuous time and discrete time models. Each parameter is computed as $\frac{V_D - V_C}{V_D}$, where V_D can assume either the value of the objective function or the computational time of the discrete time model and V_C assumes either the value of the objective function or the computational time of the continuous time model.

Finally, the last eight rows assess the benefits (in the objective functions) of introducing transfers. Since setting $b^r = 0$ is not the best way to model the standard DARP without transfers, we do not present the computational time of the no transfers case. In

these rows, each parameter is computed as $\frac{V_{No-T} - V_T}{V_T}$, where V_T assumes the value of the objective function in the models where transfers are allowed while V_{No-T} assumes the value of the objective function of the models where transfers are not allowed.

For all models, we notice that when an empty vehicle has to move, it tries to do so during a traffic jam. This can easily be explained by looking at the objective function. Since for empty vehicles travelling costs are only related to travel distance and not to travel time, vehicles try to maximize their commuting time to minimize their parking fees. This does not hold when a request is being served by the vehicle, because a time-related penalty has to be paid.

Continuous time instances may achieve lower objective function values because of the quality loss implied in discretization itself. For example, if the best departure time for a request is generic t, in discrete time, it would need to rely on t's closest time instant. A time step of one minute is small enough to curtail the error embedded in the discretization itself. In fact, the gap in the objective function is at most 0.39%.

Analyzing the results, we can state that, in general, discretizing the time results in slightly worse solutions in terms of objective function value but better results in terms of computational times. Although most of the discretized instances were solved in considerably less time than their continuous counterpart, this seems not to hold when the size of the problems increases. Indeed, the continuous time model solved more instances to optimality (and often in less time) in every scenario where 4 vehicles are present. This confirms the conclusions of van den Berg and van Essen, 2019 which states that for different conditions, the continuous or discrete model could yield better results in terms of computation time.

Since more instances are solved within the time limit in the core models compared to the enlarged models, we can deduce that including service times and variable travel times increases the level of difficulty in solving the problem.

Although transfers are thought to save travel costs, this does not always show in randomly created small instances. Nevertheless, transfers improve the average best solution by 12.24% in $V3R7$ (for the enlarged discrete case). In this case, one instance can serve all requests when transfers are allowed while it can serve all requests but one when transfers are not allowed. This creates the difference in the objective function values. It may seem that, in some cases, allowing transfers makes the problem easier (for instance, in continuous enlarged $V4R6$, more instances were solved when allowing transfers). Since the models and instances used (both for the transfer and the no transfers case) are designed to include and favour transfers, this comparison would be unfair.

The number of transfers (considering also transfers in the incumbent solutions) seems to increase steadily as the instance size grows. This is a very promising feature of this problem. We conjecture that transfers can lead to considerable savings in bigger instances.

3.6. CONCLUSIONS

In this chapter, we described and tested two mixed integer linear models and their extensions, for the DARPT where cycles are allowed. Additionally, we introduced the 'move' concept. This is useful when modelling loops and relating continuous variables (timing

size	**V2R3**	**V2R4**	**V2R5**	**V3R4**	**V3R5**	**V3R6**	**V3R7**	**V4R5**	**V4R6**	**V4R7**	**V4R8**	**V4R9**
# Instances Opt C T Core	9/10	7/10	6/10	10/10	8/10	8/10	3/10	9/10	9/10	6/10	3/10	2/10
# Instances Opt D T Core	10/10	10/10	10/10	10/10	10/10	9/10	9/10	5/10	5/10	5/10	2/10	0/10
# Instances Opt C T Enl	9/10	3/10	4/10	8/10	3/10	1/10	1/10	7/10	5/10	4/10	1/10	1/10
# Instances Opt D T Enl	10/10	10/10	10/10	10/10	10/10	8/10	10/10	2/10	1/10	0/10	0/10	0/10
# Instances Opt C No-T Core	10/10	8/10	7/10	10/10	9/10	8/10	3/10	9/10	7/10	2/10	2/10	0/10
# Instances Opt D No-T Core	10/10	10/10	10/10	10/10	10/10	10/10	10/10	8/10	8/10	7/10	6/10	4/10
# Instances Opt C No-T Enl	8/10	5/10	3/10	8/10	5/10	0/10	1/10	2/10	1/10	0/10	0/10	0/10
# Instances Opt D No-T Enl	10/10	10/10	10/10	10/10	10/10	10/10	10/10	7/10	8/10	3/10	1/10	2/10
# Instances Trans C T Core	1 (0)	0 (1)	1 (0)	1 (0)	2 (0)	1 (0)	0 (0)	4 (0)	3 (1)	3 (4)	2 (3)	1 (6)
# Instances Trans D T Core	1 (0)	1 (0)	1 (0)	1 (0)	2 (0)	1 (0)	1 (0)	2 (2)	2 (2)	4 (2)	1 (4)	0 (4)
# Instances Trans C T Enl	0 (0)	0 (1)	2 (0)	0 (0)	0 (1)	1 (0)	0 (2)	3 (0)	2 (2)	3 (6)	1 (3)	0 (6)
# Instances Trans D T Enl	0 (0)	0 (0)	1 (0)	0 (0)	1 (0)	1 (0)	1 (0)	0 (3)	0 (4)	0 (5)	0 (4)	0 (4)
# Trans C T Core	1 (0)	0 (2)	1 (0)	1 (0)	2 (0)	1 (0)	0 (0)	4 (0)	3 (1)	3 (6)	2 (3)	1 (8)
# Trans D T Core	1 (0)	2 (0)	1 (0)	1 (0)	2 (0)	1 (0)	1 (0)	2 (2)	2 (2)	6 (2)	1 (8)	0 (4)
# Trans C T Enl	0 (0)	0 (2)	2 (0)	0 (0)	0 (1)	1 (0)	0 (2)	3 (0)	2 (2)	3 (8)	1 (3)	0 (8)
# Trans D T Enl	0 (0)	0 (0)	1 (0)	0 (0)	1 (0)	1 (0)	1 (0)	0 (3)	0 (4)	0 (7)	0 (8)	0 (7)
Time C T Core	112.92	1442.01	988.76	490.19	1001.09	1036.39	466.98	1265.61	967.57	2691.73	1425.74	2905.04
Time D T Core	9.99	41.67	413.77	231.87	186.61	826.32	190.60	1733.12	1446.40	2646.50	1739.84	–
Time C T Enl	787.35	481.26	1346.05	1888.43	1300.43	3348.60	2329.74	2671.78	1890.01	3254.65	2213.50	3187.35
Time D T Enl	13.80	41.69	202.08	339.98	223.91	335.19	507.51	121.36	2450.13	–	–	–
Obj C T Core	640.67	1227.71	2495.72	593.31	830.24	1081.63	760.88	819.11	102.77	1157.01	1032.88	1312.12
Obj D T Core	640.67	1227.71	2495.72	593.31	830.24	1081.63	760.88	819.11	104.06	1159.16	1036.92	–
Obj C T Enl	868.60	519.04	2418.89	582.16	679.04	958.48	861.21	455.01	1073.34	1219.85	1259.39	1353.13
Obj D T Enl	871.19	521.86	2420.95	582.63	679.13	960.59	861.46	455.51	1075.59	–	–	–
Gap C/D T Obj Core (%)	0	0	0	0	0	0	0	0	0.13	0.19	0.39	–
Gap C/D T Time Core (%)	-1030.74	-3360.56	-138.97	-111.41	-436.46	-25.42	-145.00	26.98	33.10	-1.71	18.05	–
Gap C/D T Obj Enl (%)	0.30	0.54	0.09	0.08	0.01	0.22	0.03	0.11	0.21	–	–	–
Gap C/D T Time Enl (%)	-5605.46	-1054.25	-566.10	-455.46	-480.77	-899.00	-359.06	-2101.47	22.86	–	–	–
Gap T/No-T C Obj Core (%)	0.06	0.00	2.22	0.04	0.08	0.04	0.00	0.44	0.25	0.11	0.02	–
Gap T/No-T C Obj Enl (%)	0.00	0.00	0.02	0.00	0.00	–	0.00	0.00	0.00	–	–	–
Gap T/No-T D Obj Core (%)	0.04	0.01	1.40	0.04	0.06	0.03	6.31	0.54	0.12	0.26	0.23	–
Gap T/No-T D Obj Enl (%)	0.00	0.00	1.21	0.00	11.23	0.02	12.24	0.00	0.00	–	–	–

Table 3.8: Computational results

variables) to binary ones (routing and causality variables). We showed how much transfers increase the complexity of the problem (more instances were solved to optimality in the no transfers case compared to the case where transfers were allowed) and how much cost savings (in the objective functions) they can lead to. Also, we illustrated a method to create instances based on Google Maps data. Clearly, this method can easily scale up and create real-life sized instances which would be useful in practice for testing (meta)heuristics.

All the models proposed in this chapter allow for a great deal of flexibility. In fact, requests and vehicles can have different dimensions and, in general, there is no need for parameters $\alpha, \beta, \gamma_i, \mu_i$ to be constant or equal for every passenger or vehicle. In practical applications, it would be possible to dynamically tune these parameters depending on the history of a customer. In such a way, it is possible to ensure an equally distributed quality of service among clients.

Interestingly, transfers are not explicitly modelled. In fact, transfers and their related complexity are embedded in the *flow formulation* of the vehicles and of the requests. This formulation easily allows for sequences of transfers and transfers with multiple vehicles. Also, with this *flow formulation*, the number of variables is not affected by how many transfers are allowed. However, since the resulting models have many equality constraints, they may be computationally more challenging than formulations where only inequality constraints are used.

Some of the small instances were not solved to optimality within the given time limit. This was expected since we faced a more complex variant of an infamous NP-hard problem. Nevertheless, the sizes of the solved instances are in the range of the ones solved in the literature for similar problems.

This work lays the foundations for other interesting developments such as the offline large-scale DARPT and its online counterpart. Most likely, these problems can be solved only with heuristic methods.

APPENDIX

3.A. MODIFICATIONS ABOUT PEOPLE DEPENDENT SERVICE TIMES

In Sections 3.3.3 and 3.4.2, we explained how to include people dependent service times under the assumption that, if multiple requests are picked up simultaneously, only the largest service time is considered. In this appendix, we consider the same problem under the assumption that, if multiple requests are picked up simultaneously, their sum is considered.

3.A.1. CONTINUOUS TIME MODEL

The constraints related to the requests ((3.38) - (3.41)) hold also in this case while the ones related to the vehicles ((3.42) - (3.47)) are modified as follows.

To ensure that h_{vm_vr} assumes value one whenever request $r \in R$ is picked up by vehicle $v \in V$ at move $m_v \in M^v$, we impose the following three constraints: constraints (3.80) for the first move of a vehicle, constraints (3.81) for the first move of a request and constraints (3.82) for all other moves of requests and vehicles.

$$h_{v1r} \geq \sum_{m_r \in M^r} p_{rm_rv1}, \forall v \in V, \forall r \in R \tag{3.80}$$

$$h_{vm_vr} \geq p_{r1vm_v}, \forall v \in V, \forall m_v \in M^v, \forall r \in R \tag{3.81}$$

$$h_{vm_vr} \geq (p_{rm_rvm_v} - p_{rm_r-1vm_v-1}), \forall v \in V, \forall m_v \in M^v \setminus \{1\}, \forall r \in R, \forall m_r \in M^r \setminus \{1\} \tag{3.82}$$

To ensure a tight upper bound, we impose constraint (3.83). Since $\sum_{v \in V} \sum_{r \in R} a_{vr} s_r$ is exactly the sum of all service times that actually took place, constraint (3.83) prohibits h_{vmr} to assume value one just to have conveniently large waiting times to better fit time preferences.

$$\sum_{v \in V} \sum_{m_v \in M^v} \sum_{r \in R} h_{vm_vr} s_r \leq \sum_{v \in V} \sum_{r \in R} a_{vr} s_r \tag{3.83}$$

The modified timing constraints (3.48) remain unchanged.

3.A.2. DISCRETE TIME MODEL

For the discrete time model, we introduce binary variable h_{rvt} which assumes value one if request $r \in R$ is being picked up by vehicle $v \in V$ at time $t \in T$. Hence, we substitute constraints (3.79) by:

$$h_{rvt} \geq a_{rvt} - a_{rvt-s_r}, \forall r \in R, \forall v \in V, \forall t \in T^r | t - s_r \in T^r \tag{3.84}$$

and

$$\sum_{r \in R} h_{rvt} \leq 1, \forall v \in V, \forall t \in T. \tag{3.85}$$

This allows to consider one request at a time, hence the total service time experienced becomes the sum of the single service times. To ensure a tight upper bound, we impose constraint (3.86).

$$\sum_{v \in V} \sum_{t \in T^r} \sum_{r \in R} h_{rvt} s_r \leq \sum_{v \in V} \sum_{r \in R} a_{vr} s_r \tag{3.86}$$

3.B. IDLING, PARKING, TRANSIT AND TRANSFER NODES

In this appendix, we define constraints to model the nodes with restricted accessibility. This is useful, for instance, to model nodes situated near historical attractions. Requests can easily be directed there but parking is not allowed. For the sake of ease, we define:

- an idling node as a node where a request can wait but a vehicle cannot park.
- a parking node as a node where a request cannot wait but a vehicle can park.
- a transit node as a node where a request cannot wait and a vehicle cannot park.
- a transfer node as a node where a request can wait and a vehicle can park.

3.B.1. CONTINUOUS TIME MODEL

To model this in the continuous time model, we impose:

- $w^v = 0$ for every idling and transit node.
- $w^r = 0$ for every parking and transit node.

Given a disjoint partition N_1 of idling nodes, N_2 of parking nodes, N_3 of transit nodes and N_4 of transfer nodes, we impose the following set of constraints:

$$w^r_m \leq \sum_{(i,j) \in A | j \in N_1 \cup N_4} x^r_{ijm} B, \forall r \in R, m \in \{0\} \cup M^r. \tag{3.87}$$

We impose a similar set of constraints for parking, with the addition that a vehicle should still be able to stop to pick a request up at any idling node.

$$w^v_m \leq \sum_{(i,j) \in A | j \in N_2 \cup N_4} y^v_{ijm} B + \sum_{r \in R} h_{vmr} s_r, \forall v \in V, \forall m \in \{0\} \cup M^v \tag{3.88}$$

3.B.2. DISCRETE TIME MODEL

To model idling, parking, transit and transfer nodes in the discrete time model, we impose:

- $y^v_{iit} = 0$ for every idling and transit node.
- $x^r_{iit} = 0$ for every parking and transit node.

Given a disjoint partition N_1 of idling nodes, N_2 of parking nodes, N_3 of transit nodes and N_4 of transfer nodes, we impose the following sets of constraints:

$$\sum_{r\in R}\sum_{(i,i,t)\in A^*|i\in N_2\cup N_3, t\in T^r} x^r_{iit} = 0 \tag{3.89}$$

and

$$\sum_{t\in T} y^v_{iit} \leq \sum_{r\in R} h_{rvt}, \forall i \in N_1 \cup N_3, \forall v \in V, \forall t \in T. \tag{3.90}$$

3.C. EQUIVALENT MODELS

Even though the continuous and discrete time model share the same foundations, sometimes their objective functions differ despite having the same routing and timing solution. This appendix explains how to modify the models such that, for the same solution, they return the same objective function value. These modifications do not affect the values of the routing and timing variables, which are already equivalent for the two models.

3.C.1. UNSERVED REQUESTS AND LATE ARRIVAL

When a request is not served, a penalty has to be paid. In the continuous time model, this is in addition to the (presumed) late departure and arrival term. Since the penalty for late departure is irrelevant with respect to the one for unserved requests, this does not affect the general routing. In order to obtain from both models the same objective function value, we modify constraints (3.26) into:

$$c^+_r \geq t^r_{\mathcal{M}_v} - Bu_r, \forall r \in R. \tag{3.91}$$

Similarly, we modify constraints (3.24) into:

$$d^-_r \leq t^r_0 + w^r_0 + Bu_r, \forall r \in R. \tag{3.92}$$

3.C.2. OBJECTIVE FUNCTIONS

Only in the discrete time model, the service time is considered as waiting time; therefore, vehicles have to pay parking costs during the service times. To balance this difference, we add

$$+\gamma_2 \sum_{r\in R}\sum_{v\in V}\sum_{m\in M^v} h_{rvm}s_r$$

to the objective function of the continuous time model, where γ_2 represents the parking costs per minute.

Finally, we add the following term to the continuous time objective function:

$$+\gamma_1 \sum_{r\in R}\sum_{v\in V}\sum_{m\in M^v} h_{rvm}s_r.$$

By doing so, the service time of a request is equally penalized by a γ_1 factor in both models when the request is waiting at a transfer node. Recall that γ_1 represents the cost of waiting at a transfer node per minute per person.

3.D. COMPLETE MODELS

In this appendix, we present the complete models. This appendix does not add or modify any constraints with respect to the ones previously introduced.

3.D.1. CONTINUOUS TIME MODEL - CORE

$$\begin{aligned}\text{minimize } Z = {} & \alpha \sum_{v\in V}\sum_{(i,j)\in A}\sum_{m\in M^v} y^v_{ijm} l_{ij} + \beta \sum_{r\in R} q^r (t^r_{\mathscr{M}_r} - \sum_{m\in\{0\}\cup M^r} w^r_m - t^r_0) + \\ & \mu_1 \sum_{r\in R}(pd^r - d^-_r) + \mu_2 \sum_{r\in R}(d^+_r - pd^r) + \mu_3 \sum_{r\in R}(c^+_r - pa^r) + \mu_4 \sum_{r\in R}(pa^r - c^-_r) + \\ & \eta \sum_{r\in R}\sum_{v\in V} a_{rv} q^r + \gamma_1 \sum_{r\in R}\sum_{m\in M^r} q^r w^r_m + \gamma_2 \sum_{v\in V}\sum_{m\in\{0\}\cup M^v} w^v_m + E \sum_{r\in R} u^r q^r \end{aligned} \tag{3.1}$$

$$\text{such that } \sum_{(i,j)\in A} x^r_{ijm} \le 1 \qquad \forall r\in R, \forall m\in M^r \tag{3.2}$$

$$\sum_{(o^r,j)\in A} x^r_{o^r j1} \ge \sum_{(i,j)\in A} x^r_{ij1} \qquad \forall r\in R \tag{3.3}$$

$$\sum_{(i,j)\in A} x^r_{ijm} = \sum_{(j,k)\in A} x^r_{jk(m+1)} \qquad \forall r\in R, \forall m\in M^r\setminus\{\mathscr{M}^r\}, j\neq d^r \tag{3.4}$$

$$u^r + \sum_{m\in M^r}\sum_{(i,d^r)\in A} x^r_{id^r m} = 1 \qquad \forall r\in R \tag{3.5}$$

$$\sum_{(i,j)\in A} y^v_{ijm} \le 1 \qquad \forall v\in V, \forall m\in M^v \tag{3.6}$$

$$\sum_{(o^v,j)\in A} y^v_{o^v j1} \ge \sum_{(i,j)\in A} y^v_{ij1} \qquad \forall v\in V \tag{3.7}$$

$$\sum_{(i,j)\in A} y^v_{ijm} \ge \sum_{(j,k)\in A} y^v_{jk(m+1)} \qquad \forall v\in V, \forall m\in M^v\setminus\{\mathscr{M}_v\}, \forall j\in N \tag{3.8}$$

$$t^r_m \ge 0 \qquad \forall r\in R, \forall m\in\{0\}\cup M^r \tag{3.9}$$

$$w^r_m \ge 0 \qquad \forall r\in R, \forall m\in\{0\}\cup M^r \tag{3.10}$$

$$t^r_{m+1} = t^r_m + w^r_m + \sum_{(i,j)\in A} x^r_{ijm}\delta_{ij} \qquad \forall r\in R, \forall m\in\{0\}\cup M^r\setminus\{\mathscr{M}_r\} \tag{3.11}$$

$$t^r_0 = e^r \qquad \forall r\in R \tag{3.12}$$

$$t^r_{\mathscr{M}_r} \le l^r \qquad \forall r\in R \tag{3.13}$$

$$w^r_m \le B \sum_{(i,j)\in A} x^r_{ijm} \qquad \forall r\in R, \forall m\in M^r \tag{3.14}$$

$$t^v_m \ge 0 \qquad \forall v\in V, \forall m\in\{0\}\cup M^v \tag{3.15}$$

$$w^v_m \ge 0 \qquad \forall v\in V, \forall m\in\{0\}\cup M^v \tag{3.16}$$

$$t^v_{m+1} = t^v_m + w^v_m + \sum_{(i,j)\in A} y^v_{ijm}\delta_{ij} \qquad \forall v\in V, \forall m\in\{0\}\cup M^v\setminus\{\mathscr{M}_v\} \tag{3.17}$$

$$t_0^v = 0 \qquad \forall v \in V \quad (3.18)$$

$$t_{\mathcal{M}_v}^v \leq T_{Max} \qquad \forall v \in V \quad (3.19)$$

$$w_m^v \leq B \sum_{(i,j)\in A} y_{ijm}^v \qquad \forall v \in V, \forall m \in M^r \quad (3.20)$$

$$d_r^+ \geq pd^r \qquad \forall r \in R \quad (3.21)$$

$$d_r^+ \geq t_0^r + w_0^r \qquad \forall r \in R \quad (3.22)$$

$$d_r^- \leq pd^r \qquad \forall r \in R \quad (3.23)$$

$$d_r^- \leq t_0^r + w_0^r \qquad \forall r \in R \quad (3.24)$$

$$c_r^+ \geq pa^r \qquad \forall r \in R \quad (3.25)$$

$$c_r^+ \geq t_{\mathcal{M}_r}^r \qquad \forall r \in R \quad (3.26)$$

$$c_r^- \leq pa^r \qquad \forall r \in R \quad (3.27)$$

$$c_r^- \leq t_{\mathcal{M}_r}^r \qquad \forall r \in R \quad (3.28)$$

$$\sum_{v\in V} \sum_{m_v \in M^v} p_{rm_r vm_v} \leq 1 \qquad \forall r \in R, m_r \in M^r \quad (3.29)$$

$$\sum_{r\in R} \sum_{m_r \in M^r} p_{rm_r vm_v} q^r \leq q^v \qquad \forall v \in V, m_v \in M^v \quad (3.30)$$

$$x_{ijm_r}^r \leq y_{ijm_v}^v + (1 - p_{rm_r vm_v}) \qquad \forall v \in V, \forall r \in R, \forall (i,j) \in A, \forall m_r \in M^r, \forall m_v \in M^v \quad (3.31)$$

$$x_{ijm_r}^r \geq y_{ijm_v}^v - (1 - p_{rm_r vm_v}) \qquad \forall v \in V, \forall r \in R, \forall (i,j) \in A, \forall m_r \in M^r, \forall m_v \in M^v \quad (3.32)$$

$$t_{m_r}^r \leq t_{m_v}^v + (1 - p_{rm_r vm_v})B \qquad \forall v \in V, \forall r \in R, \forall m_r \in M^r, \forall m_v \in M^v \quad (3.33)$$

$$t_{m_r}^r \geq t_{m_v}^v - (1 - p_{rm_r vm_v})B \qquad \forall v \in V, \forall r \in R, \forall m_r \in M^r, \forall m_v \in M^v \quad (3.34)$$

$$\sum_{(i,j)\in A} x_{ijm_r}^r \leq \sum_{v\in V} \sum_{m_v \in M^v} p_{rm_r vm_v} \qquad \forall r \in R, \forall m_r \in M^r \quad (3.35)$$

$$Ba_{rv} \geq \sum_{m_r \in M^r} \sum_{m_v \in M^v} p_{rm_r vm_v} \qquad \forall v \in V, \forall r \in R \quad (3.36)$$

$$\sum_{v\in V} a_{rv} - 1 \leq d^r \qquad \forall r \in R \quad (3.37)$$

$$x_{ijm}^r \in \{0,1\} \qquad \forall r \in R, \forall (i,j) \in A, \forall m \in M^r \quad (3.93)$$

$$y_{ijm}^v \in \{0,1\} \qquad \forall v \in V, \forall (i,j) \in A, \forall m \in M^v \quad (3.94)$$

$$a_{rv} \in \{0,1\} \qquad \forall r \in R, \forall v \in V \quad (3.95)$$

$$p_{rm_r vm_v} \in \{0,1\} \qquad \forall r \in R, \forall m_r \in M^r, \forall v \in V, \forall m_v \in M^v \quad (3.96)$$

3.D.2. CONTINUOUS TIME MODEL - EXTENSION

$$\begin{aligned}\text{minimize } Z = {} & \alpha \sum_{v\in V}\sum_{(i,j)\in A}\sum_{m\in M^v} y^v_{ijm} l_{ij} + \beta \sum_{r\in R} q^r (t^r_{\mathscr{M}_r} - \sum_{m\in\{0\}\cup M^r} w^r_m - t^r_0) + \\ & \mu_1 \sum_{r\in R}(pd^r - d^-_r) + \mu_2 \sum_{r\in R}(d^+_r - pd^r) + \mu_3 \sum_{r\in R}(c^+_r - pa^r) + \mu_4 \sum_{r\in R}(pa^r - c^-_r) + \\ & \eta \sum_{r\in R}\sum_{v\in V} a_{rv} q^r + \gamma_1 \sum_{r\in R}\sum_{m\in M^r} q^r w^r_m + \gamma_2 \sum_{v\in V}\sum_{m\in\{0\}\cup M^v} w^v_m + E \sum_{r\in R} u^r q^r \end{aligned} \tag{3.1}$$

such that

$$\sum_{(i,j)\in A} x^r_{ijm} \le 1 \qquad \forall r\in R, \forall m\in M^r \tag{3.2}$$

$$\sum_{(o^r,j)\in A} x^r_{o^r j1} \ge \sum_{(i,j)\in A} x^r_{ij1} \qquad \forall r\in R \tag{3.3}$$

$$\sum_{(i,j)\in A} x^r_{ijm} = \sum_{(j,k)\in A} x^r_{jk(m+1)} \qquad \forall r\in R, \forall m\in M^r, j\ne d^r \tag{3.4}$$

$$u^r + \sum_{m\in M^r}\sum_{(i,d^r)\in A} x^r_{id^r m} = 1 \qquad \forall r\in R \tag{3.5}$$

$$\sum_{(i,j)\in A} y^v_{ijm} \le 1 \qquad \forall v\in V, \forall m\in M^v \tag{3.6}$$

$$\sum_{(o^v,j)\in A} y^v_{o^v j1} \ge \sum_{(i,j)\in A} y^v_{ij1} \qquad \forall v\in V \tag{3.7}$$

$$\sum_{(i,j)\in A} y^v_{ijm} \ge \sum_{(j,k)\in A} y^v_{jk(m+1)} \qquad \forall v\in V, \forall m\in M^v\setminus\{\mathscr{M}_v\}, \forall j\in N \tag{3.8}$$

$$t^r_m \ge 0 \qquad \forall r\in R, \forall m\in\{0\}\cup M^r \tag{3.9}$$

$$w^r_m \ge 0 \qquad \forall r\in R, \forall m\in\{0\}\cup M^r \tag{3.10}$$

$$t^r_0 = e^r \qquad \forall r\in R \tag{3.12}$$

$$t^r_{\mathscr{M}_r} \le l^r \qquad \forall r\in R \tag{3.13}$$

$$w^r_m \le B \sum_{(i,j)\in A} x^r_{ijm} \qquad \forall r\in R, \forall m\in M^r \tag{3.14}$$

$$t^v_m \ge 0 \qquad \forall v\in V, \forall m\in\{0\}\cup M^v \tag{3.15}$$

$$w^v_m \ge 0 \qquad \forall v\in V, \forall m\in\{0\}\cup M^v \tag{3.16}$$

$$t^v_0 = 0 \qquad \forall v\in V \tag{3.18}$$

$$t^v_{\mathscr{M}_v} \le T_{Max} \qquad \forall v\in V \tag{3.19}$$

$$w^v_m \le B \sum_{(i,j)\in A} y^v_{ijm} \qquad \forall v\in V, \forall m\in M^r \tag{3.20}$$

$$d^+_r \ge pd^r \qquad \forall r\in R \tag{3.21}$$

$$d^+_r \ge t^r_0 + w^r_0 \qquad \forall r\in R \tag{3.22}$$

$$d^-_r \le pd^r \qquad \forall r\in R \tag{3.23}$$

$$d^-_r \le t^r_0 + w^r_0 \qquad \forall r\in R \tag{3.24}$$

$$c_r^+ \geq pa^r \qquad \forall r \in R \quad (3.25)$$

$$c_r^+ \geq t^r_{\mathcal{M}_r} \qquad \forall r \in R \quad (3.26)$$

$$c_r^- \leq pa^r \qquad \forall r \in R \quad (3.27)$$

$$c_r^- \leq t^r_{\mathcal{M}_r} \qquad \forall r \in R \quad (3.28)$$

$$\sum_{v \in V} \sum_{m_v \in M^v} p_{rm_rvm_v} \leq 1 \qquad \forall r \in R, m_r \in M^r \quad (3.29)$$

$$\sum_{r \in R} \sum_{m_r \in M^r} p_{rm_rvm_v} q^r \leq q^v \qquad \forall v \in V, m_v \in M^v \quad (3.30)$$

$$x^r_{ijm_r} \leq y^v_{ijm_v} + (1 - p_{rm_rvm_v}) \quad \forall v \in V, \forall r \in R, \forall (i,j) \in A, \forall m_r \in M^r, \forall m_v \in M^v \quad (3.31)$$

$$x^r_{ijm_r} \geq y^v_{ijm_v} - (1 - p_{rm_rvm_v}) \quad \forall v \in V, \forall r \in R, \forall (i,j) \in A, \forall m_r \in M^r, \forall m_v \in M^v \quad (3.32)$$

$$t^r_{m_r} \leq t^v_{m_v} + (1 - p_{rm_rvm_v})B \qquad \forall v \in V, \forall r \in R, \forall m_r \in M^r, \forall m_v \in M^v \quad (3.33)$$

$$t^r_{m_r} \geq t^v_{m_v} - (1 - p_{rm_rvm_v})B \qquad \forall v \in V, \forall r \in R \forall m_r \in M^r, \forall m_v \in M^v \quad (3.34)$$

$$\sum_{(i,j) \in A} x^r_{ijm_r} \leq \sum_{v \in V} \sum_{m_v \in M^v} p_{rm_rvm_v} \qquad \forall r \in R, \forall m_r \in M^r \quad (3.35)$$

$$Ba_{rv} \geq \sum_{m_r \in M^r} \sum_{m_v \in M^v} p_{rm_rvm_v} \qquad \forall v \in V, \forall r \in R \quad (3.36)$$

$$\sum_{v \in V} a_{rv} - 1 \leq d^r \qquad \forall r \in R \quad (3.37)$$

$$g^r_1 \geq \sum_{v \in V} \sum_{m_v \in M^v} p_{r1vm_v} \qquad \forall r \in R \quad (3.38)$$

$$g^r_{m_r} \geq \sum_{v \in V} p_{rm_rv1} \qquad \forall r \in R, \forall m_r \in M^r \quad (3.39)$$

$$g^r_{m_r} \geq p_{rm_rvm_v} - p_{rm_r-1vm_v-1} \quad \forall v \in V, \forall m_v \in M^v \setminus \{1\}, \forall r \in R, \forall m_r \in M^r \setminus \{1\} \quad (3.40)$$

$$\sum_{r \in R} h_{vmr} \leq 1 \qquad \forall v \in V, \forall m \in M^v \quad (3.42)$$

$$\sum_{r \in R} h_{v1r} s_r \geq p_{rm_rv1} s_r \qquad \forall v \in V, \forall r \in R, \forall m_r \in M^r \quad (3.43)$$

$$\sum_{r \in R} h_{vm_vr} s_r \geq p_{r1vm_v} s_r \qquad \forall v \in V, \forall m_v \in M^v, \forall r \in R \quad (3.44)$$

$$\sum_{r \in R} h_{vm_vr} s_r \geq (p_{rm_rvm_v} - p_{rm_r-1vm_v-1}) s_r \quad \forall v \in V, \forall m_v \in M^v \setminus \{1\}, \forall r \in R, \forall m_r \in M^r \setminus \{1\} \quad (3.45)$$

$$h_{vm_vr} \leq 1 - \sum_{m_r \in M^r} p_{rm_rvm_v-1} \qquad \forall v \in V, \forall m_v \in M^v \setminus \{1\}, \forall r \in R \quad (3.46)$$

$$h_{vm_vr} \leq \sum_{m_r \in M^r} p_{rm_rvm_v} \qquad \forall v \in V, \forall m_v \in M^v, \forall r \in R \quad (3.47)$$

$$t^r_{m_r-1} + w^r_{m_r-1} \leq t^v_{m_v-1} + w^v_{m_v-1} + (1 - p_{rm_rvm_v})B \quad \forall v \in V, \forall r \in R, \forall m_r \in M^r, \forall m_v \in M^v \tag{3.49}$$

$$t^r_{m_r-1} + w^r_{m_r-1} \geq t^v_{m_v-1} + w^v_{m_v-1} - (1 - p_{rm_rvm_v})B \quad \forall v \in V, \forall r \in R, \forall m_r \in M^r, \forall m_v \in M^v \tag{3.50}$$

$$\sum_{k \in TS} TT^r_{ijmk} = x^r_{ijm} \qquad \forall r \in R, \forall m \in M^r, \forall (i,j) \in A \tag{3.51}$$

$$\sum_{(i,j) \in A} TT^r_{ijmk} lb_{ijk} \leq t^r_m + w^r_m + g^r_m s^r \qquad \forall r \in R, \forall m \in M^r, \forall k \in TS \tag{3.52}$$

$$t^r_m + w^r_m + g^r_m s^r \leq \sum_{(i,j) \in A} TT^r_{ijmk} ub_{ijk} + B(1 - \sum_{(i,j) \in A} TT^r_{ijmk}) \quad \forall r \in R, \forall m \in M^r, \forall k \in TS \tag{3.53}$$

$$\sum_{k \in TS} TT^v_{ijmk} = y^v_{ijm} \qquad \forall v \in V, \forall m \in M^v, \forall (i,j) \in A \tag{3.54}$$

$$\sum_{(i,j) \in A} TT^v_{ijmk} lb_{ijk} \leq t^v_m + w^v_m + \sum_{r \in R} s^r h_{vmr} \qquad \forall v \in V, \forall m \in M^v, \forall k \in TS \tag{3.55}$$

$$t^v_m + w^v_m + \sum_{r \in R} s^r h_{vmr} \leq \sum_{(i,j) \in A} TT^v_{ijmk} ub_{ijk} + B(1 - \sum_{(i,j) \in A} TT^v_{ijmk}) \quad \forall v \in V, \forall m \in M^v, \forall k \in TS \tag{3.56}$$

$$t^r_{m+1} = t^r_m + w^r_m + g^r_m s^r + \sum_{k \in TS} \sum_{(i,j) \in A} TT^r_{ijmk} \delta_{ijk} \qquad \forall r \in R, \forall m \in M^r \setminus \{\mathscr{M}^r\} \tag{3.57}$$

$$t^v_{m+1} = t^v_m + w^v_m + \sum_{r \in R} s^r h_{vmr} + \sum_{k \in TS} \sum_{(i,j) \in A} TT^v_{ijmk} \delta_{ijk} \quad \forall v \in V, \forall m \in M^v \setminus \{\mathscr{M}^v\} \tag{3.58}$$

$$x^r_{ijm} \in \{0,1\} \qquad \forall r \in R, \forall (i,j) \in A, \forall m \in M^r \tag{3.93}$$

$$y^v_{ijm} \in \{0,1\} \qquad \forall v \in V, \forall (i,j) \in A, \forall m \in M^v \tag{3.94}$$

$$a_{rv} \in \{0,1\} \qquad \forall r \in R, \forall v \in V \tag{3.95}$$

$$p_{rm_rvm_v} \in \{0,1\} \qquad \forall r \in R, \forall m_r \in M^r, \forall v \in V, \forall m_v \in M^v \tag{3.96}$$

$$g^r_m \in \{0,1\} \qquad \forall r \in R, \forall m \in M^r \tag{3.97}$$

$$h_{vmr} \in \{0,1\} \qquad \forall r \in R, \forall v \in V, \forall m \in M^v \tag{3.98}$$

3.D.3. DISCRETE TIME MODEL - CORE

$$\begin{aligned}\text{minimize } Z = {} & \alpha \sum_{v\in V}\sum_{(i,j,t)\in A^*} y^v_{ijt} l_{ij} + \beta \sum_{r\in R}\sum_{(i,j,t)\in A^*, i\neq j} x^r_{ijt}\delta_{ijt} q^r + \\ & \mu_1 \sum_{r\in R}\sum_{t<pd^r\in T^r} (1 - x^r_{o^r o^r t}) q^r + \mu_2 \sum_{r\in R}\sum_{t\geq pd^r\in T^r} x^r_{o^r o^r t} q^r + \\ & \mu_3 \sum_{r\in R}\sum_{t<pa^r\in T^r} x^r_{d^r d^r t} q^r + \mu_4 \sum_{r\in R}\sum_{t\geq pa^r\in T^r} (1 - x^r_{d^r d^r t}) q^r + \\ & \eta \sum_{r\in R}\sum_{v\in V} a_{rv} q^r + \gamma_1 \sum_{r\in R}\sum_{(i,i,t)\in A^*} x^r_{iit} q^r + \\ & \gamma_2 \sum_{v\in V}\sum_{(i,i,t)\in A^*} y^v_{iit} + E \sum_{r\in R} u^r q^r \end{aligned} \tag{3.59}$$

such that:

$$\sum_{i,j\in N, i\neq o^r} x^r_{ije^r} = 0 \qquad \forall r \in R \tag{3.60}$$

$$\sum_{j\in N} x^r_{o^r je^r} = 1 \qquad \forall r \in R \tag{3.61}$$

$$\sum_{i\in N} x^r_{id^r t_2} + u^r = 1 \qquad \forall r \in R, t_2 \in T^r | t_2 + \delta_{id^r t_2} = l^r \tag{3.62}$$

$$\sum_{i,j\in N, i\neq o^v} y^v_{ij0} = 0 \qquad \forall v \in V \tag{3.63}$$

$$\sum_{j\in N} y^v_{o^v j0} = 1 \qquad \forall v \in V \tag{3.64}$$

$$\sum_{i\in N}\sum_{j\in N} y^v_{ijt} = 1 \qquad \forall v \in V, \forall t \in T | t + \delta_{ijt} = T_{Max} \tag{3.65}$$

$$\sum_{i\in N} x^r_{ijt_2} = \sum_{i\in N} x^r_{jit} \qquad \forall r \in R, \forall t \in T^r, t_2 \in T^r | t_2 + \delta_{ijt_2} = t, \forall j \in N \tag{3.66}$$

$$\sum_{i\in N} y^v_{ijt_2} = \sum_{i\in N} y^v_{jit} \qquad \forall v \in V, \forall j \in N, \forall t \in T/\{T^{max}\}, t_2 \in T | t_2 + \delta_{ijt_2} = t \tag{3.67}$$

$$\sum_{v\in V} a_{rvt} \leq 1 \qquad \forall r \in R, \forall t \in T^r \tag{3.68}$$

$$\sum_{r\in R} a_{rvt} q^r \leq q^v \qquad \forall v \in V, \forall t \in T \tag{3.69}$$

$$\sum_{j\in N} x^r_{ijt} \leq \sum_{j\in N} y^v_{ijt} + (1 - a_{rvt}) \qquad \forall r \in R, \forall v \in V, \forall i \in N, \forall t \in T^r \tag{3.72}$$

$$\sum_{j\in N} x^r_{ijt} \geq \sum_{j\in N} y^v_{ijt} - (1 - a_{rvt}) \qquad \forall r \in R, \forall v \in V, \forall i \in N, \forall t \in T^r \tag{3.73}$$

$$\sum_{i\in N} x^r_{ijt} \leq \sum_{i\in N} y^v_{ijt} + (1 - a_{rvt}) \qquad \forall r \in R, \forall v \in V, \forall j \in N, \forall t \in T^r \tag{3.74}$$

$$\sum_{i\in N} x^r_{ijt} \geq \sum_{i\in N} y^v_{ijt} - (1 - a_{rvt}) \qquad \forall r \in R, \forall v \in V, \forall j \in N, \forall t \in T^r \tag{3.75}$$

$$\sum_{i\in N} x^r_{iit} \geq 1 - \sum_{v\in V} a_{rvt} \qquad \forall r \in R, \forall t \in T^r \tag{3.76}$$

$$B a_{rv} \geq \sum_{t\in T} a_{rvt} \qquad \forall v \in V, \forall r \in R \tag{3.77}$$

$$\sum_{v \in V} a_{rv} - 1 \leq d^r \qquad \forall r \in R \tag{3.78}$$

$$x^r_{ijt} \in \{0,1\} \qquad \forall r \in R, \forall (i,j,t) \in A^* \tag{3.99}$$

$$y^v_{ijt} \in \{0,1\} \qquad \forall v \in V, \forall (i,j,t) \in A^* \tag{3.100}$$

$$a_{rv} \in \{0,1\} \qquad \forall r \in R, \forall v \in V \tag{3.101}$$

$$a_{rvt} \in \{0,1\} \qquad \forall r \in R, \forall v \in V, \forall t \in T^r \tag{3.102}$$

3.D.4. DISCRETE TIME MODEL - EXTENSION

$$\begin{aligned}\text{minimize } Z = {} & \alpha \sum_{v\in V}\sum_{(i,j,t)\in A^*} y^v_{ijt} l_{ij} + \beta \sum_{r\in R}\sum_{(i,j,t)\in A^*, i\neq j} x^r_{ijt}\delta_{ijt} q^r + \\ & \mu_1 \sum_{r\in R}\sum_{t<(pd^r+s^r)\in T^r} (1-x^r_{o^r o^r t}) q^r + \mu_2 \sum_{r\in R}\sum_{t\geq(pd^r+s^r)\in T^r} x^r_{o^r o^r t} q^r + \\ & \mu_3 \sum_{r\in R}\sum_{t<pa^r\in T^r} x^r_{d^r d^r t} q^r + \mu_4 \sum_{r\in R}\sum_{t\geq pa^r\in T^r} (1-x^r_{d^r d^r t}) q^r + \\ & \eta \sum_{r\in R}\sum_{v\in V} a_{rv} q^r + \gamma_1 \sum_{r\in R}\sum_{(i,i,t)\in A^*} x^r_{iit} q^r + \\ & \gamma_2 \sum_{v\in V}\sum_{(i,i,t)\in A^*} y^v_{iit} + E\sum_{r\in R} u^r q^r \end{aligned} \tag{3.59*}$$

such that:

$$\sum_{i,j\in N, i\neq o^r} x^r_{ije^r} = 0 \qquad \forall r\in R \tag{3.60}$$

$$\sum_{j\in N} x^r_{o^r je^r} = 1 \qquad \forall r\in R \tag{3.61}$$

$$\sum_{i\in N} x^r_{id^r t_2} + u^r = 1 \qquad \forall r\in R, t_2\in T^r \,|\, t_2+\delta_{id^r t_2} = l^r \tag{3.62}$$

$$\sum_{i,j\in N, i\neq o^v} y^v_{ij0} = 0 \qquad \forall v\in V \tag{3.63}$$

$$\sum_{j\in N} y^v_{o^v j0} = 1 \qquad \forall v\in V \tag{3.64}$$

$$\sum_{i\in N}\sum_{j\in N} y^v_{ijt} = 1 \qquad \forall v\in V, \forall t\in T \,|\, t+\delta_{ijt} = T_{Max} \tag{3.65}$$

$$\sum_{i\in N} x^r_{ijt_2} = \sum_{i\in N} x^r_{jit} \qquad \forall r\in R, \forall t\in T^r, t_2\in T^r \,|\, t_2+\delta_{ijt_2} = t, \forall j\in N \tag{3.66}$$

$$\sum_{i\in N} y^v_{ijt_2} = \sum_{i\in N} y^v_{jit} \qquad \forall v\in V, \forall j\in N, \forall t\in T/\{T^{max}\}, t_2\in T \,|\, t_2+\delta_{ijt_2} = t \tag{3.67}$$

$$\sum_{v\in V} a_{rvt} \leq 1 \qquad \forall r\in R, \forall t\in T^r \tag{3.68}$$

$$\sum_{r\in R} a_{rvt} q^r \leq q^v \qquad \forall v\in V, \forall t\in T \tag{3.69}$$

$$\sum_{j\in N} x^r_{ijt} \leq \sum_{j\in N} y^v_{ijt} + (1-a_{rvt}) \qquad \forall r\in R, \forall v\in V, \forall i\in N, \forall t\in T^r \tag{3.72}$$

$$\sum_{j\in N} x^r_{ijt} \geq \sum_{j\in N} y^v_{ijt} - (1-a_{rvt}) \qquad \forall r\in R, \forall v\in V, \forall i\in N, \forall t\in T^r \tag{3.73}$$

$$\sum_{i\in N} x^r_{ijt} \leq \sum_{i\in N} y^v_{ijt} + (1-a_{rvt}) \qquad \forall r\in R, \forall v\in V, \forall j\in N, \forall t\in T^r \tag{3.74}$$

$$\sum_{i\in N} x^r_{ijt} \geq \sum_{i\in N} y^v_{ijt} - (1-a_{rvt}) \qquad \forall r\in R, \forall v\in V, \forall j\in N, \forall t\in T^r \tag{3.75}$$

$$\sum_{i\in N} x^r_{iit} \geq 1 - \sum_{v\in V} a_{rvt} \qquad \forall r\in R, \forall t\in T^r \tag{3.76}$$

$$Ba_{rv} \geq \sum_{t\in T} a_{rvt} \qquad \forall v\in V, \forall r\in R \tag{3.77}$$

$$\sum_{v \in V} a_{rv} - 1 \leq d^r \qquad \forall r \in R \tag{3.78}$$

$$\sum_{i \in N} x^r_{iit} \geq a_{rvt} - a_{rvt-s^r} \qquad \forall t \in T^r | t - s^r \in T^r, \forall r \in R, \forall v \in V \tag{3.79}$$

$$x^r_{ijt} \in \{0,1\} \qquad \forall r \in R, \forall (i,j,t) \in A^* \tag{3.99}$$

$$y^v_{ijt} \in \{0,1\} \qquad \forall v \in V, \forall (i,j,t) \in A^* \tag{3.100}$$

$$a_{rv} \in \{0,1\} \qquad \forall r \in R, \forall v \in V \tag{3.101}$$

$$a_{rvt} \in \{0,1\} \qquad \forall r \in R, \forall v \in V, \forall t \in T^r \tag{3.102}$$

4

SPECIAL EDUCATION NEEDS SCHOOL BUS ROUTING PROBLEM

Jacopo PIEROTTI, Lina SIMEONOVA, Theresia VAN ESSEN

Schools for special education needs (SEN) students in the UK often organise the transport of students from their homes to the school. This service significantly affects schools' budgets because, in addition to having dedicated vehicles and drivers, attenders, i.e., specialised staff, are also needed to accompany the students for health and safety reasons. Given the behavioural and learning needs of the SEN students, travelling to school can be perceived as stressful and can cause anxiety or episodes of aggression. This often occurs when SEN students travel to school with people they are unfamiliar with. A sufficient level of familiarity with other students and staff on the bus can give the students a sense of safety and comfort, which would make the journey smoother and more enjoyable. We consider socio-economic trade-offs of two conflicting objectives, namely minimizing the total distance travelled and maximizing student familiarity. Our problem is motivated by a real-life case in the county of Kent in the United Kingdom and we use real data from the area to design our problem instances. We firstly model the special education needs bus routing problem (SENBRP) as an integer linear program and solve small instances to optimality via a commercial solver. In order to solve real-life instances, a metaheuristic approach is described, implemented and tested on the case study.

4.1. INTRODUCTION

The school bus routing problem (SBRP) is an important problem which impacts not only transportation companies, but also public policy makers, local schools and millions of families. Our research falls in a relatively under-researched area, namely School Bus

Routing Problem for students with Special Educational Needs (SENBRP) and it is motivated by the transport practices of a local school in the South East of England. According to the classification of the Kent County Council, our research focuses on students with Behavioural and Learning needs (Bradley et al., 2002), which includes social, emotional and mental health (SEMH) needs, Metachromatic Leukodystrophy (MLD), Attention Deficit Hyperactivity Disorder (ADHD), Obsessive-Compulsive Disorder (OCD) and others.

Most of the research in the SBRP domain focuses predominantly on economic gains through cost (Parvasi et al., 2017) and time (Li and Fu, 2002) optimisation, and there are a few papers which also focus on sustainable operations from an environmental perspective (Chalkia et al., 2016). However, to the best of our knowledge, there is no research which includes the social aspect of school bus routing, and more specifically for such a vulnerable demographic. While having a robust and cost efficient routing schedule is very important due to the limited school and council budgets, it is even more important to plan the school bus routes in a considerate manner in order to minimize the potential negative impact the journey to school can have on the students. In this paper, one of the most important considerations when creating bus routes is to ensure that every student boards a bus where students have a good level of familiarity with each other. Familiarity with the school bus attendant and fellow students is a very important issue because it provides stability, re-assurance and a feeling of safety and security. A positive experience during the journey can provide a good foundation for the school day ahead and increase the general productivity of the students. According to local transport organisers, if the journey is not pleasant, it can have a more serious impact on students with special needs. This is also supported by research around the effect of travel on students with disabilities and special needs and how important it is to address the social aspect of bus routing (Buliung et al., 2021). Therefore, we formulate and solve our problem in a bi-objective fashion, where we aim to minimize the cost of travel and maximise the level of student familiarity.

Route familiarity has gained attention in the Operations Research (OR) community as a key aspect for implementation and efficiency in routing. So far, it has mainly been researched from the perspective of the drivers, ensuring they are familiar with their daily routes (Intini et al., 2019) and they are consistent in satisfying customer demands at approximately the same times (Goeke et al., 2019). The research around optimising social aspects of routing is fairly limited, in comparison to the environmental and economic aspects. In the school bus routing domain even more so, as, to our best knowledge, there is only one paper which explicitly incorporates special students needs. Caceres et al., 2019 considered those students who require special equipment, such as ramps and wheelchairs. Therefore, the problem focuses on the partitioning of buses, which can accommodate a mix of wheelchairs and regular seats.

In many cases, in practice, schools outsource by hiring private cars to fill in any gaps when there are insufficient spaces on buses (Parvasi et al., 2017). This leads to highly inefficient and expensive routing and this is one of the main reasons why SBRP is an important optimization problem, which can lead to significant savings. On top of that, our problem originates from schools with learning and behavioural needs, where students are highly vulnerable. Therefore, when planning the school bus routing for students with

special needs, it is very important to consider efficiency and fitting into tight budgets, but it is just as critical to ensure students' health and safety and that their needs are properly met.

The remainder of the paper is organized as follows. Section 4.2 illustrates an overview of the state-of-the-art research on SBRP. In Section 4.3, we present a mathematical formulation of the SENBRP and Section 4.4 explains the exact and heuristic methods we use to solve the SBRP. We test these methods on a case study and computational results are discussed in Section 4.5. The paper is concluded with some considerations in Section 5.4. Additional sensitivity analysis, Pareto optimal solutions (Marler and Arora, 2010) and trade-offs are discussed in Appendix 4.A and Appendix 4.B.

4.2. LITERATURE REVIEW

The School Bus Routing Problem falls within the broad field of Vehicle Routing Problems (VRP, Toth and Vigo, 2002). VRPs are an extensively studied class of optimization problems due to their practical relevance. VRPs deal with managing a fleet of vehicles under a set of constraints (usually capacity, routing and timing constraints). The SBRP was firstly introduced by Newton and Thomas, 1969, but it has received much less attention in the literature than the classical VRP for freight applications. There is no dominant approach to studying the SBRP and most of the solution methods proposed are very problem specific, typically motivated by different school practices and school transport contexts. A lot of the research in this area examines North American contexts, where the school bus routing is significantly different to the UK, which is the focus of our research. For instance, in North America, private schools and larger buses are more common. The SBRP review by Ellegood et al., 2020 shows that there is no research to date in a UK setting.

Similar to the VRP, the most common objective functions are minimizing the total distance, bus travel time or student travel time. When the objective of the SBRP is to mimimize student walking distance, which is assumed to be the distance between the home locations and the bus stops, the problem is called bus stop location problem (Schittekat et al., 2013). Due to health and safety constraints, in our problem, each student is picked up at their home address; thus, the walking distance is already at its minimum.

In the SBRP, it is common to have contrasting objectives; thus, the problem is often formulated as multi-objective, usually bi-objective (Zajac and Huber, 2021). To solve a multi-objective problem means to retrieve the solution points which are not dominated. In a bi-objective maximization problem, a solution s in solution space S is not dominated if there does not exist a solution $s' \in S$ such that the objective function values of s' are larger (and at least one objective function value strictly larger) than the objective function values of s. The set of non-dominated solutions is called the Pareto front or the Pareto frontier (Marler and Arora, 2010).

The SBRP finds applications in an urban (Parvasi et al., 2017) or rural setting (Souza Lima et al., 2016) and various applications for single (Park et al., 2009, Caceres et al., 2019) or multiple schools (Li and Fu, 2002, Shafahi et al., 2018). The transportation of students from different schools allows for consolidation of pick up locations and fewer trips. Having a homogeneous fleet is much more common and that is also the assumption in our case, but there are a few applications of heterogeneous bus fleets. Souza Lima

et al., 2016 addressed for the first time an SBRP which combines a heterogeneous fleet and a mixed load, where students from multiple schools can ride the same bus. This is a very interesting problem, which occurs in a rural setting in Brazil, where multi-grade students attend the same classes. An application by Prah et al., 2018 also considers a rural setting and incorporates the characteristics of the terrain using three-dimensional geographical data.

The SBRP with school bell adjustment is also a popular problem in the literature, which considers bus routing to multiple schools. In the SBRP with school bell adjustment, schools' start and end times need to be considered such that schools' bus fleets can be better utilised and serve multiple schools. Miranda et al., 2021 study the school bell adjustment problem using a memetic algorithm and iterated local search. Wang and Haghani, 2020 address a stochastic version of the problem with an algorithm which incorporates aspects of Column Generation (Desaulniers et al., 2006), Simulated Annealing (Malek et al., 1989), Insertion Algorithm (Artigues and Roubellat, 2000) and Greedy Randomized Adaptive Search Procedure (GRASP, Feo and Resende, 1995). There are a few applications of the SBRP where authors consider multiple objectives. For instance, Pacheco et al., 2013 aim to minimize the travel cost and minimize the longest route, which is a cost-service trade-off. The authors use Tabu Search within a multi-objective adaptive memory programming framework.

The SBRP literature also contains some sustainable applications. There are papers that consider the environmental aspect of SBRP and some touch on the social aspect of vehicle routing. Chalkia et al., 2016 considered student safety, both on transport and student pedestrian routes, Ahmed et al., 2020 studied air pollution exposure reduction and Mokhtari and Ghezavati, 2018 examined the multi-objective SBRP whose objectives are minimizing cost and students riding time.

Given the typical schools' sizes, the SBRP is usually solved via heuristic methods. Park et al., 2009 solved the mixed load SBRP by a combination of Sweep, Hungarian method and a post-improvement procedure. Shafahi et al., 2018 approached the SBRP in an integrated manner solving the problem with a two-step heuristic with minimum cost matching and post improvement. Lewis and Smith-Miles, 2018 considered a real life SBRP, which includes aspects of bin packing, routing and set covering. The authors researched aspects of merging and splitting routes and bus dwell times using a local search in combination with a kick operator. Schittekat et al., 2013 also examined real-life situations, this time in Belgian schools. In the Flemish region of Belgium, students living within certain distances of their school have access to free transportation systems to and from school which is organized by the Flemish transportation company. The authors implemented a column generation approach hybridised with an iterative metaheuristic based on GRASP and Variable Neighbourhood Descend (VND, Hansen et al., 2010), to find solutions to real life instances. Instead of relying on classical heuristic techniques, Köksal Ahmed et al., 2020 introduced a reinforcement learning structure to minimize students waiting time as well as bus operators costs. Most of the solution methodologies to solve multi-objective routing problems are hybrid, population-based or multi-stage (a detailed review can be found in Zajac and Huber, 2021) which makes them inherently complex. In contrast, we propose to design a simple and general heuristic.

The contribution of our paper is three-fold.

- Our problem is bi-objective, where we aim to maximize route familiarity (homogeneity) whilst minimizing the total distance travelled. To the best of our knowledge, there is no research to date, which solves SBRP with a bi-objective function balancing socio-economic gains.
- There is very little research in the area of SBRP with special needs and there is potential for contribution and savings. Just in the county of Kent in the UK there are 22 schools with students with special needs, operating under very limited budgets.
- We propose an exact formulation (with efficient valid inequalities) for the SENBRP and test its performance. Moreover, we develop an almost parameter-free metaheuristic able to tackle large size case studies.

4.3. PROBLEM FORMULATION

In this section, we describe how to model the special education needs bus routing problem. In this work, we consider only trips from the students home locations to the school. Trips from the school to the students home locations can be found via the same methods explained in this work with few modifications. In accordance to the real-life case we are examining, we assume the vehicles to be homogeneous. We consider the following problem: given a depot location d, a school location s, a set of student home locations H and a set of homogeneous buses B, we aim to design feasible routes visiting all home locations while minimizing the travel cost Z_t (routing and attendants costs) and maximizing the familiarity of the passengers Z_q. For a route to be feasible, it has to start at the depot d, end in the school node s and respect time constraints, bus capacity and attendant safety limitations.

We model the road network as a graph with nodes N connected by a set of arcs A. The set of nodes N is composed of the depot d, the school s and the set of all the student home locations H. In the case study we are analysing, the school s and the depot d are situated at the same node; nonetheless, we model those as two different nodes to adopt a generic formulation. In cases where two or more students have the same home location, they are modelled with multiple distinct dummy nodes. Doing so, each student is uniquely associated with a node in H. From now on, depending on the context, when referring to $i \in H$, we either refer to the student or to his/her home location. For each student $i \in H$, we assume a maximum ride time R. In order to ensure that students needs are met safely and professionally, at least one attendant needs to be present on the bus. Therefore, we consider a set of attendants K, situated at depot d. Multiple attendants can be present on the same bus in order to guarantee more safety and a higher quality of the service. For an efficient scheduling of the personnel and to avoid overfilling buses, the number of attendants per bus is limited to a maximum of A_{max}. The buses given by set B are homogeneous with capacity Q. This capacity resource is gradually consumed whenever a student $i \in H$ or an attendant $k \in K$ is riding on bus $b \in B$.

We consider the graph to be complete and that the triangular inequality holds. Each arc $(i, j) \in A$ is associated with a known travel time $\delta_{(i,j)}$ and a travelling cost $c_{(i,j)}$. In order to model the familiarity among students, we define a subset of arcs $A_f \subseteq A$. A_f contains all and only the arcs among students which are familiar with each other. Moreover, we assume that each attendant is familiar with every student; thus, the quality of

Table 4.3.1: Sets and parameters

d	depot node, i.e. the starting point of the buses and attendants
s	school node, i.e. the final destination
H	set of the students or of their home locations
N	$H \cup \{d, s\}$
A	set of all arcs
A_f	set of arcs connecting students who are familiar with each other
$\delta_{(i,j)}$	travel time for arc $(i, j) \in A$
$c_{(i,j)}$	travel cost for arc $(i, j) \in A$
B	set of homogeneous buses
Q	capacity of each bus $b \in B$
K	set of attendants
A_{max}	maximum number of attendants per bus
R	maximum ride time for each student $i \in H$
α	cost of each attendant
γ	weight coefficient to tune the trade-off between the objectives
f_{min}	minimum familiarity level to be guaranteed for each student

the service increases as more attendants are placed in a bus. In the following, we refer to the familiarity level as the quality of the service. To avoid solutions with an unevenly spread quality of the service (e.g. a solution where all students but one experience a high quality of service and one student experiences an extremely low quality of service), a minimum familiarity level of f_{min} has to be guaranteed for all the students. For example, a solution where 4 students experience a familiarity of 3 is to be preferred to a solution where 3 students experience a familiarity of 4 and one student experience a familiarity of zero. Table 4.3.1 summarises the used notation.

4.3.1. DECISION VARIABLES

In this section, we describe the variables used in the model. To model the routing, we employ binary variables $y^b_{(i,j)}$, assuming value one if bus $b \in B$ travels through arc $(i, j) \in A$ and zero otherwise. To avoid cycles, integer variables u_i ranging from one to $Q-1$ for $i \in H$ are introduced. These variables indicate the order in which students are picked up and have upper bound $Q-1$ because at least one of the Q seats in each bus is reserved for an attendant. To associate attendants with buses, we introduce integer variables a^b which assume the value of the number of attendants in bus $b \in B$.

To measure the familiarity, we utilise binary variables $p^b_{(i,j)}$ which assume value one if and only if students i and $j \in H$ are familiar (i.e. $(i, j) \in A_f$) and sit on bus $b \in B$. This means that variable $p^b_{(i,j)}$ exists only if $(i, j) \in A_f$. Then, we adopt variables f^b_i to measure the familiarity of student $i \in H$ if (s)he would sit on bus $b \in B$. These variables are intermediate variables needed to compute the integer variables f_i which assume the familiarity value that student $i \in H$ is actually experiencing.

Table 4.3.2 summarises the used variables.

Table 4.3.2: Variables

$y^b_{(i,j)}$	binary	one if bus $b \in B$ travels arc $(i,j) \in A$, zero otherwise
u_i	integer	variables to forbid cycling (see Constraints 4.9)
a^b	integer	number of attendants on bus $b \in B$
$p^b_{(i,j)}$	binary	one if bus $b \in B$ picks up students i and $j \in H$ and $(i,j) \in A_f$
f^b_i	integer	familiarity value of student $i \in H$ if (s)he sits on bus $b \in B$
f_i	integer	familiarity value of student $i \in H$

4.3.2. OBJECTIVE FUNCTION

The model aims to minimize the travel costs Z_t while maximizing the familiarity value Z_q. Since these are in general conflicting objectives, we are solving a multi-objective optimization problem. In order to achieve optimal solutions, we temporally reduce the problem to a single objective optimization problem by defining generalized cost Z.

We define as routing costs:

$$Z_t = \sum_{b \in B} \sum_{(i,j) \in A} c_{(i,j)} y^b_{(i,j)} + \alpha \sum_{b \in B} a^b, \tag{4.1}$$

where α is the cost of hiring an extra attendant for one trip.

To consider the familiarity level, we maximize the sum of the familiarity levels. Hence, we set:

$$Z_q = \sum_{i \in H} f_i. \tag{4.2}$$

Thus, we define the generalized cost as:

$$Z = \gamma Z_t - (1-\gamma) Z_q, \tag{4.3}$$

where γ is a tuning parameter to modify the trade-off among the two objectives (see Section 4.4). Finally, the objective function of our problem is:

$$\min Z \tag{4.4}$$

4.3.3. CONSTRAINTS

In this section, we discuss the constraints of our model. Firstly, we ensure that every bus starts from the depot (Constraints 4.5) and ends at the school (Constraints 4.6).

$$\sum_{(i,j) \in A} y^b_{(i,j)} \leq Q \sum_{j \in H} y^b_{(d,j)}, \forall b \in B \tag{4.5}$$

$$\sum_{j \in H} y^b_{(d,j)} = \sum_{i \in H} y^b_{(i,s)}, \forall b \in B \tag{4.6}$$

Constraints (4.5) does not force the solution to use all the available buses. In fact, given the capacity limitation, each bus can pick up at most $Q-1$ students (one seat is reserved for the attendant), which results in a total of Q travelled arcs ($Q-1$ to pick up students plus one to go reach the trip's destination, i.e. the school).

Secondly, we have to ensure flow conservation (Constraints 4.7) and to visit all the home locations of the students (Constraints 4.8) .

$$\sum_{i \in N} y^b_{(i,j)} - \sum_{k \in N} y^b_{(j,k)} = 0, \forall j \in H, \forall b \in B \tag{4.7}$$

$$\sum_{b \in B} \sum_{j \in N} y^b_{(i,j)} = 1, \forall i \in H \tag{4.8}$$

Constraints (4.5)-(4.8) alone do not suffice to avoid cycles, hence, we introduce ordering Constraints (4.9).

$$u_i - u_j + (Q+1) \sum_{b \in B} y^b_{(i,j)} \leq Q, \forall i, j \in H, i \neq j \tag{4.9}$$

This set of constraints forces u_j to assume a value greater than or equal to $u_i + 1$, if any bus travels arc $(i, j) \in A$. This prevents cycling. Note that, in general, variables u_i have to be in strictly increasing order, but they do not have to be increasing by one. Hence, for a bus picking up students $1, 2, 3$ (in that order), $u_1 = 2, u_2 = 5, u_3 = 8$ can be an acceptable solution.

The capacity limitations are modelled via Constraints (4.10).

$$\sum_{(i,j) \in A | i \neq d} y^b_{(i,j)} + a^b \leq Q, \forall b \in B \tag{4.10}$$

In order to enforce the maximum ride time, we use the fact that for each bus, the student who is picked up first, travels the most. This corresponds to the whole bus route without the first arc (the one from the depot). Hence, we set:

$$\sum_{(i,j) \in A | i \neq d} y^b_{(i,j)} \delta_{(i,j)} \leq R, \forall b \in B \tag{4.11}$$

When students with special needs ride in a bus, for safety reasons, there should be at least one attendant. By imposing

$$\sum_{i \in H} y^b_{(d,i)} \leq a^b, \forall b \in B, \tag{4.12}$$

we ensure that if the bus is not empty, at least one attendant is present.

In addition, we have to ensure to use at most $|K|$ attendants, thus we enforce:

$$\sum_{b \in B} a^b \leq |K|. \tag{4.13}$$

To bound variables $p^b_{(i,j)}$, to assume the value of one when students $i \in H$ and $j \in H$ are familiar with each other and sit on bus $b \in B$, we impose Constraints (4.14).

$$p^b_{(i,j)} \leq 0.5(\sum_{k \in N} y^b_{(i,k)} + \sum_{k \in N} y^b_{(j,k)}), \forall b \in B, \forall (i,j) \in A_f \tag{4.14}$$

Once variables $p^b_{(i,j)}$ are bounded, we determine the value of variables f^b_i through Constraints (4.15) and (4.16).

$$f^b_i \leq \sum_{(i,j)\in A_f} p^b_{(i,j)} + a^b, \forall b \in B, \forall i \in H \tag{4.15}$$

$$f^b_i \leq (Q-1) \sum_{j\in N} y^b_{(i,j)}, \forall b \in B, \forall i \in H \tag{4.16}$$

On one hand, Constraints (4.15) impose an upperbound on variable f^b_i to a maximum composed by the sum of the students on bus b with whom student $i \in H$ is familiar with plus the number of attendants on the same bus. On the other hand, Constraints (4.16) force variables f^b_i to zero if bus $b \in B$ does not visit student $i \in H$. We determine the value of the actual familiarity level of student $i \in H$, i.e. f_i, via Constraints (4.17).

$$f_i \leq \sum_{b\in B} f^b_i, \forall b \in B, \forall i \in H \tag{4.17}$$

For each $i \in H$, Constraints (4.17) let variable f_i assume the value of the only non-zero variable among f^b_i. To ascertain a minimum level of familiarity for each student, we impose Constraints (4.18).

$$f_i \geq f_{min}, \forall i \in H \tag{4.18}$$

Finally, we impose the following binary and integer constraints.

$$y^b_{(i,j)} \in \{0,1\}, \forall (i,j) \in A \tag{4.19}$$

$$u_i \in \{0,\cdots,Q-1\}, \forall i \in H \tag{4.20}$$

$$a^b \in \{0,\cdots,A_{max}\}, \forall b \in B \tag{4.21}$$

$$p^b_{(i,j)} \in \{0,1\}, \forall b \in B, \forall (i,j) \in A_f \tag{4.22}$$

$$f^b_i \in \{0,\cdots,Q-1\}, \forall b \in B, \forall i \in H \tag{4.23}$$

$$f_i \in \{0,\cdots,Q-1\}, \forall i \in H \tag{4.24}$$

4.4. METHODS

To find optimal solutions belonging to the Pareto frontier, we use the weighted sum method (Marler and Arora, 2010). This technique consists of iteratively solving the same instance of a problem while modifying the weight coefficient γ (see Eq. 4.3) of the single objective function. By doing so, we explore different trade-offs between the two objective functions.

4.4.1. VALID INEQUALITIES

In order to reduce the computation time, we introduce valid inequalities. By definition, valid inequalities are constraints that do not reduce the solution space of an ILP but they reduce (and simplify) its relaxation. Instead, we allow valid inequalities to remove some optimal solutions from the solution space, provided that at least one optimal solution is left. This is helpful to reduce the computational time needed to solve a problem with many symmetric solutions (for example, a solution where bus b_1 serves only student h_1 and bus b_2 serves only student h_2 is identical to a solution where bus b_1 serves only student h_2 and bus b_2 serves only student h_1). To implement such valid inequalities, we impose an ordering of the buses. We write $b_1 < b_2$ to state that b_1 precedes b_2.

Note that, given the capacity constraints and $|H|$ students to visit, at least $\left\lceil \frac{|H|}{Q-1} \right\rceil$ buses have to be routed. Hence, we set the outgoing flow from the depot of the first $\left\lceil \frac{|H|}{Q-1} \right\rceil$ buses (set $B^{'}$) to one and we set their number of attendants to be greater than or equal to one via Constraints (4.25) and (4.26), respectively.

$$\sum_{(d,j)\in A} y^b_{(d,j)} = 1, \forall b \in B^{'} \tag{4.25}$$

$$a^b \geq 1, \forall b \in B^{'} \tag{4.26}$$

In addition, we impose that the buses have to carry a decreasing number of students.

$$\sum_{(i,j)\in A} y^{b_1}_{(i,j)} \geq \sum_{(i,j)\in A} y^{b_2}_{(i,j)}, \forall b_1, b_2 \in B, b_1 < b_2 \tag{4.27}$$

To start with a tighter bound on Z_q, we consider that it is composed of two terms. The first term is related to the familiarity among the students while the second term is related to the increase in familiarity due to the presence of attendants. The familiarity among the students can be upper-bounded by considering the arcs in A_f. Hence, we define $\tilde{F} = \sum_{i\in H} \min(Q-2, \sum_{(i,j)\in A_f} 1)$. The minus two comes from the fact that, among the Q possible seats, one is reserved for the student him/herself and at least another one is reserved for the attendant.

Adding an attendant to a bus $b \in B$ with s^b students, would increase Z_q by s^b (because s^b students would increase their familiarity level by one). More in general, the increase in familiarity given by the attendants is $\sum_{b\in B} a^b s^b$, where a^b and s^b are the number of attendants and students on bus $b \in B$, respectively. Hence, we determine an upper bound on the familiarity due to the presence of attendants by solving the following integer quadratic problem (IQP).

$$\max \sum_{b \in B} a^b s^b \tag{4.28}$$

$$\text{s.t.} \sum_{b \in B} a^b \leq |K| \tag{4.29}$$

$$\sum_{b \in B} s^b = |H| \tag{4.30}$$

$$s^b + a^b \leq Q \qquad \forall b \in B \tag{4.31}$$

$$(Q-1)a^b \geq s^b \qquad \forall b \in B \tag{4.32}$$

$$a^b \in \{0, 1, .., A_{max}\} \qquad \forall b \in B \tag{4.33}$$

$$s^b \in \{0, 1, .., |H|\} \qquad \forall b \in B \tag{4.34}$$

$\sum_{b \in B} a^b s^b$ is the increase in familiarity due to introducing attendants and we name this quantity $\tilde{A}$. Constraints (4.29) and (4.30) impose to use at most $|K|$ attendants and to visit all $|H|$ students, respectively. Constraints (4.31) act as capacity constraints and Constraints (4.32) impose to have at least one attendant on bus b if it is not empty. Constraints (4.33) and (4.34) are integrality constraints and bound the maximum values of variables a^b to A_{max} and s^b to $|H|$. Although this is an IQP, it can be solved within a fraction of a second.

Resulting from this, we impose the valid inequality that variable Z_q can be at most the sum of $\tilde{A}$ and $\tilde{F}$.

$$Z_q \leq \tilde{A} + \tilde{F} \tag{4.35}$$

4.4.2. METAHEURISTIC METHOD

When solving large sized SENBRP instances in reasonable time (see Section 4.5), (meta)heuristics are needed. As discussed in Section 4.2, multi-objective bus routing problems are heuristically predominantly solved with evolutionary algorithms (EA) or a hybrid approach with an evolutionary element. The multi-objective (MO) solution approaches are typically rather complex structures such as multi-component or multi-phase heuristics, hybrid metaheuristics or matheuristics. In this paper instead, we propose a simple randomised metaheuristic approach, where only a generalised local search operator (GLSO) is used as a solution generation mechanism. The idea of using a single and relatively simple operator stems from the solution method adopted by Geiger and Graf, 2019 to solve the VeRoLog Solver challenge 2019[1]. In the solution method of Geiger and Graf, 2019, a single random operator has been applied to a single objective VRP problem and it showed very strong performance both in terms of solution quality and computation time. We conjecture that a single effective operator can yield very good results also in a multi-objective setting.

Our ambition is to implement and test a solution approach, which is in line with the fundamental characteristics of metaheuristics, namely: simplicity, generalisability, flexibility and efficiency. In addition, multi-objective metaheuristic methods usually make use of various empirically-tuned parameters, which are related to the search intensity,

[1] https://verolog2019.ortec.com/, website accessed in November 2021.

perturbation, goodness of fit, etc. In contrast, we aim to limit the problem specific parameters in order to maintain simplicity and robustness.

METAHEURISTIC OVERVIEW

Our metaheuristic iteratively generates new solutions and collects the non-dominated ones in set S. Set S is initialised with the solution returned via the Sweep method (Section 4.4.2). Then, the algorithm iteratively explores the solutions in S and, given a solution $s \in S$, it generates a new solution s' by applying the GLSO and 2-opt (Section 4.4.2). The new solution s' is accepted if it satisfies the acceptance criterion (Section 4.4.2) and it is added to S if it is not dominated. For each solution $s \in S$, this process is repeated for λ_{max} consecutive non-improving iterations before moving to the following solution in S. Once every solution in S has been considered, the set S is reduced only to the non-dominated solutions and the process is repeated until a stopping criterion (based on a time limit) is met. We need to remove non-dominated solutions from S because an intermediate solution s could be accepted, only to later discover solution s' which dominates it. Possible speedups are discussed in Section 4.4.2. Algorithm 2 summarises the overall structure.

Algorithm 2 Algorithm overview

```
S = {Sweep(instance)}
while stopping criterion do
    for s ∈ S do
        λ = 0
        while λ ≤ λ_max do
            s' ← GLSO & 2 − opt of s
            if acceptance criterion s' then
                s ← s'
                if s' is non-dominated then
                    S ← S ∪ {s'}
                    λ = 0
                end if
            else
                λ ← λ + 1
            end if
        end while
    end for
    S ← only non-dominated solutions S
end while
return S
```

INITIAL SOLUTION

To generate an initial solution, we use the Sweep algorithm (Gillett and Miller, 1974). We compute the relative angles between the depot and each student location and sort the students in ascending order. Starting with an empty route, we assign it a random

number of attendants from one to A_{max}. Then, we collect students in order until either a capacity or a time violation occurs. When a violation occurs, another vehicle is added. This is repeated until all students are assigned to a bus.

GLSO

While current VRP literature uses various shift, swap, insert and eject operators (Toth and Vigo, 2002), each with their advantages and disadvantages, we propose to use one single versatile operator which encompasses all possible inter-route operations among two routes. We call such an operator a generalised local search operator (GLSO). Given a solution s, the GLSO selects two routes r_1 and r_2 at random. From each route, a chain (i.e. a sequence of consecutive nodes) is isolated, where an empty chain is also allowed. The starting point of the chain is chosen uniformly at random among the student home locations within a route, the length of the chain is a random integer number between zero and the number of student home locations after the starting point (extremes included). The two chains are then swapped, creating the new routes r_1' and r_2'. The GLSO can perform swaps or insertions (if one of the chains is empty) of virtually any dimension, which makes the operator universal. All other routes in the solution remain unchanged.

After swapping the chains, we check if both routes are feasible. We compute the minimum number of attendants needed (multiple attendants could be needed to satisfy the minimum familiarity level requirement, see Constraints (4.18)) and we assign them to the routes. Not all GLSO operations return a feasible solution. A solution is infeasible (and will not be accepted) if any of the following applies: there are not enough available attendants, we cannot fit the needed attendants in the vehicles or any route is not feasible from a capacity or a timing point of view. Otherwise, if both routes are feasible and more attendants are available, an extra attendant can be assigned to the first route uniformly at random (i.e. with 50% probability). The same applies for the second route.

Since the GLSO is an inter-route operator, it means that intra-route improvements are not explored. Therefore, to find better sequencing between the nodes, we perform 2-opt on the new routes r_1' and r_2' and we obtain the new solution s'. We chose to apply 2-opt because it is powerful and computationally cheap (Barma et al., 2019). Figure 4.4.1 shows two possible GLSO (and 2-opt) operations, the first being a swap and the second being an insertion (meaning that one of the randomly selected chains is empty).

ACCEPTANCE CRITERION

A new solution s' is accepted, but not yet added to S, if at least one of the following two conditions applies. The first condition is that solution s' weakly dominates solution s, i.e. either the routing cost of s' is strictly lower than the routing cost of s, or the total familiarity of s' is strictly higher than the one of s. The second condition states that solution s' can still be accepted (even if it does not weakly dominate solution s) with a fixed probability p. While the first condition ensures us to not discard promising solutions, the second condition allows us to introduce some randomness to encourage exploration and escaping local minima. Moreover, if s' is not dominated by any other solution in S, s' is added to set S. The overall algorithm contains merely three parameters, namely the maximum computation time, the maximum number of non-improving iterations (λ_{max}) and the probability of accepting non-improving solutions (p).

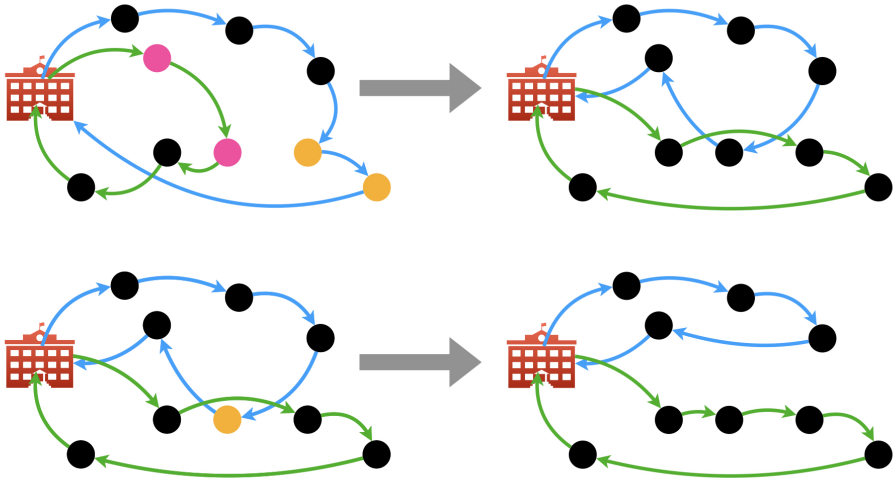


Figure 4.4.1: Two examples of possible GLSO operations. On the top, two nonempty chains are swapped; on the bottom, one chain is empty. Green and blue arrows show the two routes, in red there is the depot (which coincides with the school), and pink and yellow nodes show the selected chains.

SPEEDUP

Although this work was motivated by the search for an efficient, simple and robust metaheuristic, problem-specific speedups can increase the performance in solving a specific problem (yet they usually decrease the performance for other problems). Given a specific type of problem, most likely, there exist a number C_{max} such that, if we *sometimes* impose the algorithm to choose chains of at most length C_{max}, the metaheuristic performance increases. This is due to the fact that exchanging large chains can lead to transformations which are too radical and do not let the metaheuristic explore enough of the neighbourhood close to the current solutions.

Introducing this simple speedup of limiting the maximum length of a chain leads to an increase in performance (see Appendix 4.B); however, it introduces at least two extra parameters. The first one is the probability of choosing this speedup instead of the completely random approach (i.e. without limiting the dimension of the chain). The second parameter is the maximum chain length C_{max}. In the following, we call *guided* the search done while limiting the maximum dimension of the chain and *random* the one done without limitations. The introduction of new parameters makes the algorithm a bit more complex and slightly less robust (as these parameters have to be handpicked and are problem specific).

4.5. COMPUTATIONAL RESULTS

The dataset is generated based on real data purchased from a third-party company that specialises in travel time estimation in the UK, for Heavy Goods Vehicles (HGVs), smaller lorries, vans and cars. The exact locations of the students are not used for data protection

reasons, but approximate locations are generated using a reduced postcode method. The postal codes in the UK typically have 7 digits, for instance CT18 5EQ. We used the reduced version CT18 5** to generate the locations of students. Based on the information given to us by the SEN school, there are 10 classes for a total of 168 students. Then, we used the time and distance matrix dataset from the third-party data provider for 168 students' locations. We partitioned the dataset into 14 instances of 12 students, where each student belongs to exactly one instance. Each of the 12-students instances has been randomly created such that each student knows at least two other students (i.e. $\sum_{j \in H | (i,j) \in A_f} 1 \geq 2, \forall i \in H$) and that the average student knows three other students (i.e $|A_f| \geq 3|H|$). In each instance, six attendants and three buses with a capacity of nine seats are present. The cost coefficient α (attendants' cost) is set to twelve, the maximum ride time R is set to one hour and the minimum familiarity level f_{min} for each student is set to two. The school has advised us that there can be up to two attendants per bus due to efficient human resource management; so, we have designed our method in accordance with this (i.e. $A_{max} = 2$).

We also generated larger instances with 50, 75 and 100 students to test the metaheuristic (see Appendix 4.B). The larger sized problems have some overlap, where some students belong to more than one instance. According to the local school transport and teaching team, there are three types of familiarity between students and these are the rules by which we have generated the student familiarity:

- Students from the same class mostly get along with each other, but not always. Therefore, our approximation is that 80% of students in each class are familiar with each other and can be comfortably transported together on the same bus.
- Students from the same level (primary, secondary) are familiar with approximately 40% of the students from that level, because they attend the same assembly, may live in the same neighbourhood/town or by other means.
- Students between levels are familiar with approximately 10% of the other levels, because they may be from the same neighbourhood/town or by other means.

4.5.1. EXACT SOLUTIONS

The ILP model (4.1)-(4.27) is useful to obtain optimal solutions, but its solution process is computationally challenging. Hence, it can be tested in reasonable time only on the 12-students instances. The solutions of these smaller instances are used to compare if and how fast the metaheuristic can achieve the same results.

The 12-students instances are solved with the following coefficient γ values (see Eq. 4.3): 0.99, 0.9, 0.8, 0.7, 0.6, 0.5, 0.4, 0.3, 0.2, 0.1 and 0.01. We did not test for $\gamma \in \{0, 1\}$ to avoid extreme solutions (such as, for example, solutions with non-optimal routing because transportation costs do not appear in the objective function).

We solve the ILPs iteratively in the order of the values of γ we presented and update the bounds; so, for example, the transportation cost for the solution for γ equal to 0.7 will be a lower bound for the transportation cost of the problem with gamma equal to 0.6, since we minimize the transportation cost and the weight is reduced, and the quality of the service for the solution for γ equal to 0.7 will be a lower bound for the quality of the

service of the problem with gamma equal to 0.6, since we maximize the quality of service and the weight is increased. The optimal solution of a weight coefficient value is fed as the initial solution for the following coefficient value. All tests are run on an Intel(R) Core(TM)2 Quad CPU Q8400 @2.66GHz machine using Gurobi Optimizer version 9.0.1 as the MILP solver (Gurobi Optimization, LLC, 2021).

As detailed in Marler and Arora, 2010, using a weighted sum for multi-objective optimization problems lets us discover some points belonging to the Pareto frontier but, in general, not all of them. In fact, if the region is non-convex, we might not find some of the points in the Pareto frontier. Nonetheless, in our experiments, even using the weighted sum method, we have found good solution diversity, which was the purpose of the ILP.

Figure 4.5.1 shows the observed Pareto frontier of the 13th instance. Clearly, we can see the trade-off between minimizing the travel costs and maximizing the quality of travel. We can also observe the logarithmic-like shape of the trade-off curve which is typical for bi-objective optimization problems.

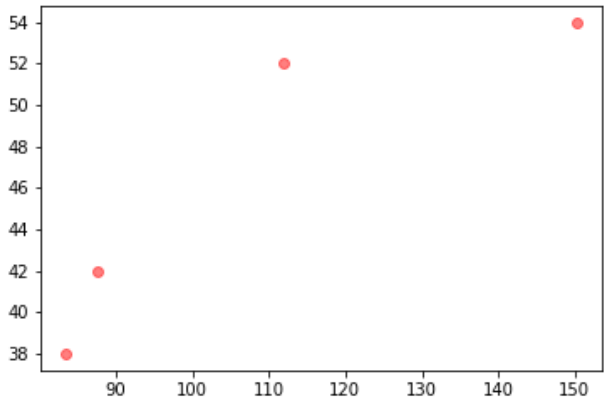


Figure 4.5.1: Observed Pareto frontier of the 12-students instance number 13. On the x-axis the routing cost; on the y-axis, the quality of the service.

EXACT METHOD SOLUTION TIMES

To analyse the speed-up due to valid inequalities (Section 4.4.1), we test the 12-students instances with and without the valid inequalities. In this section, Constraints (4.25), (4.26) and (4.27) are called *symmetry breaking inequalities* and Constraint (4.35) is called *familiarity valid inequality*.

In Figure 4.5.2, we display the barplot of the total average computation times. As we can see, combining both sets of valid inequalities yields the best result (meaning the least computation time). When considering the sets of valid inequalities independently, the symmetry breaking inequalities are the best performing, reducing the computation time by 22.20%. Surprisingly, adding the familiarity valid inequality alone decreases the solution speed; however, when combined with the symmetry breaking constraints, it

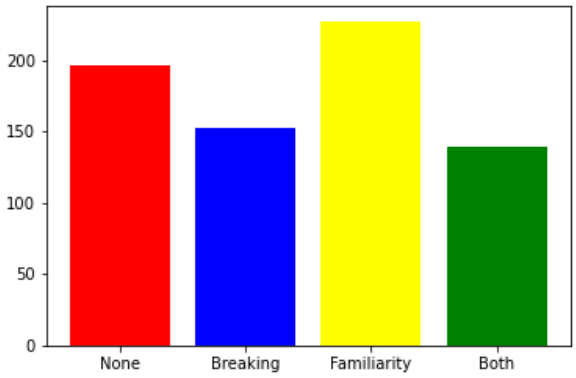


Figure 4.5.2: Barplot of the average computation times in seconds (y-axis) with respect to valid inequalities (x-axis). In red, no valid inequality is applied; in blue, only the symmetry breaking constraints are applied; in yellow, only the familiarity constraints are applied; in green, both of the previous are applied.

boosts the solution speed. This can be explained by considering that, in general, adding constraints increases the computation time needed to solve a problem. Valid inequalities are an exception to the previous statement; however, their effectiveness depends on the polytope. It might be the case that, with respect to the original polytope (the ones where no valid inequalities have been applied), the familiarity inequality is not effective. However, it becomes effective when applied to the polytope obtained by adding the symmetry breaking constraints. In fact, when both symmetry breaking and familiarity inequalities are introduced, they sped up the solution process with the average speed-up being 29.18% of the computation time.

In Figure 4.5.3, we show the boxplots of the computation times with respect to the values of γ. For readability reasons, in Figure 4.5.3, we plot tests' results when both sets of valid inequalities or none of the sets were considered. As we can see, including valid inequalities reduces the median computation time for 9 over 11 values of γ.

We noticed that increasing the number of students per instance rapidly increases the solution times. In addition to the obvious reasons related to NP-hardness, we want to mention that valid inequalities fail to reduce the solution space when the number of students is increased. This is mainly due to the fact that $\tilde{F}$ (which represent the familiarity due to the presence of the students themselves, defined as $\sum_{i\in H} \min(Q-2, \sum_{(i,j)\in A_f} 1)$, see Section 4.4.1) becomes too optimistic. When the number of students is increased, the quantity $\sum_{(i,j)\in A_f} 1$ increases as well. Since this happens for all students, the corresponding value for each student saturates at $Q-2$ due to the min operator. Hence, the growth of their sum eventually makes the upperbound too large, making the valid inequality not binding.

OBJECTIVE VALUES

Table 4.5.1 shows the average results for the conducted tests. The first column (names) presents the name of the instance and the second column (points) shows the number

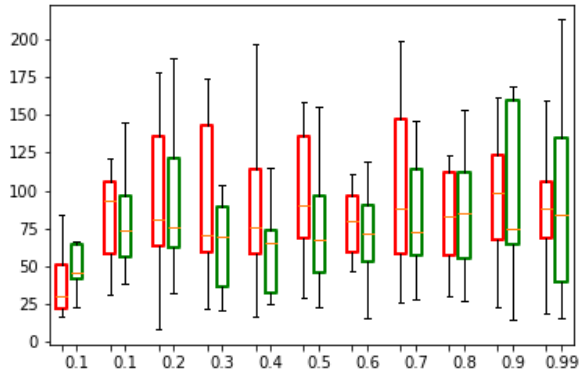


Figure 4.5.3: Boxplot of the computational times in seconds (y-axis) with respect to the γ values (x-axis). In red, results when no valid inequalities are applied; in green, results when both sets of valid inequalities are applied.

of observed distinct points which determine the Pareto frontier. Subsequently, columns min VRP, avg VRP and max VRP report the minimum, the average and maximum value for the transportation costs, respectively. Similarly, columns min Qual, avg Qual and max Qual display the minimum, the average and maximum value for the quality of the service, respectively.

Table 4.5.1: Average results for the 12 students instances

names	points	min VRP	avg VRP	max VRP	min Qual	avg Qual	max Qual
s12_1	5	121.49	140.87	183.948	30	39.91	58
s12_2	2	85.69	94.41	109.68	38	43.09	52
s12_3	3	93.85	102.66	148.36	38	42.91	52
s12_4	4	119.20	138.13	177.02	26	35.27	52
s12_5	3	113.74	132.14	172.28	28	35.82	52
s12_6	5	126.11	138.79	165.92	38	44.91	56
s12_7	3	131.04	146.30	183.25	28	33.36	46
s12_8	4	71.38	81.90	110.08	36	40.27	50
s12_9	4	123.70	133.33	158.68	32	35.64	44
s12_10	3	81.096	88.73	105.10	40	43.91	52
s12_11	5	74.34	84.03	104.11	32	42.09	52
s12_12	4	111.42	121.58	140.04	32	38.36	48
s12_13	6	82.728	96.09	151.76	36	43.18	54
s12_14	4	83.49	95.44	150.17	38	42.73	54

4.5.2. METAHEURISTIC RESULTS

To test the efficiency of the metaheuristic, we first run it on the 12-students instances and compare the results of the metaheuristic with the results of the exact algorithm. All the optimal solutions found by the exact method were found by the metaheuristic in, on average, 8 seconds (with 10 out of 14 instances solved under one second). The metaheuristic was able to find also other non-dominated solutions which were not found by the exact algorithm due to non-convexity. In particular, over the 14 instances, the exact method found a total of 55 optimal solutions, while the metaheuristic found 101 solutions belonging to the estimated Pareto frontiers (the metaheuristic found the 55 solutions also found by the exact method and 46 extra ones). In Appendix 4.A, we graphically show the plots of these results.

Since the metaheuristic was able to identify all the solutions found by the exact algorithm in significantly less time, we tested the metaheuristic on bigger and bigger instances. Instances of size 50, 75 and 100 were tested and extensive results can be found in Appendix 4.B.

Finally, we tested the metaheuristic on the case study (168 students). For this test, we implemented the speedup technique described in Section 4.4.2, setting the probability of limiting the length of the chain to 50% and C_{max}, the maximum length of the chain, to one. In Appendix 4.B, we present a sensitivity analyses regarding the probability of limiting the chain and the length of the chain. The metaheuristic was run for one hour on the case study and it ultimately estimated a Pareto frontier composed of 116 different points. Figure 4.5.4 shows the dynamics of the solution points estimated by the metaheuristic. For readability reasons, we display only the estimated Pareto front after 5, 10, 15, 30 and 60 minutes.

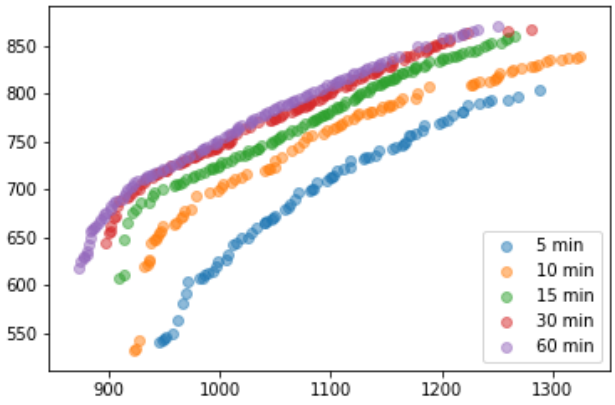


Figure 4.5.4: Observed Pareto frontier on the case study (168 students). On the x-axis, the routing cost; on the y-axis, the quality of service.

4.6. CONCLUSIONS

In this paper, we propose, model and analyse a new bi-objective optimization problem: the school bus routing problem for students with special educational needs (SENBRP). Our research was informed by a real-life case study, a school for students with special needs in the region of Kent (UK). In a typical VRP setting, costs are proportional to the distance travelled, but when transporting students with special needs, their comfort and well-being is equally important. Travelling with unfamiliar people could cause them distress and discomfort; hence, the most efficient route distance-wise might not be the best solution. Similarly, focusing only on the students' comfort could lead to economically unacceptable solutions, which is also not feasible, as schools and local councils operate on very tight budgets. Therefore, we model the SENBRP as a bi-objective problem and we solved it via the weighted sum method. By doing so, we are able to identify some of the non-dominated points which constitute the Pareto frontier. Given its complexity, we were able to retrieve points of the Pareto front only for small instances via this exact method. In order to address our case study with a size of 168 students, we implemented a robust and versatile metaheuristic, with very few parameters. Our metaheuristic is based on the idea of having one universal randomised local search operator. To test the metaheuristic effectiveness, we compare the metaheuristic solutions with the results obtained by the exact method on the small instances. Our metaheuristic was able to find all the known Pareto frontier points in roughly one second for almost all the small instances. We then tested its performance on larger instances ranging from 50 - 168 students, which can be found in Section 4.5 and Appendix 4.B. For each instance, we recorded the time needed to reach a stable estimated Pareto front, which shows a clear trade-off between travel cost and the quality of service. Finally, we tested the metaheuristic on the case study returning a set of diverse possible solutions.

Our method is simplistic and almost parameter-free and it is able to find very diverse sets of solutions for all instances. Therefore, we believe it to be effective and robust, and it could be generalised to other similar problems with multiple and even conflicting objectives.

Whilst reviewing the literature on SBRP, we found that there are very few applications on sustainable school bus travel from both environmental and social perspective. Transportation to school for students with special needs is an important problem, which needs to receive more attention, because it is often an alternative to individual private hire transportation. The SENBRP provides a balanced approach to the triple bottom line, where schools need to carefully balance tight budgets and ensuring the safety of a vulnerable student population.

APPENDIX

4.A. GRAPHICAL RESULTS FOR THE 12-STUDENTS INSTANCES

Figure 4.A.1 displays the observed Pareto frontiers of the 14 instances with 12 students. In green, we display the solutions found by both the exact algorithm and the metaheuristic; in violet, we present the solutions found only by the metaheuristic.

4.B. SENSITIVITY ANALYSIS

Figures 4.B.1, 4.B.2 and 4.B.3 display results for instances of size 50, 75 and 100, respectively. In all the tests, we can see the trade-off between the quality of service and routing costs. As expected, the number of points belonging to the Pareto frontier increases as the size of the instance grows. It is also very interesting to notice different instances' time dynamics. Most 50-students instances converge (meaning the metaheuristic does not find any new solution for 5 minutes) to a stable frontier after 5 or 10 minutes, while no bigger instance converges within 15 minutes. However, after 15 minutes, most of the instances seem to have found a fairly stable Pareto front.

In this appendix, we empirically show why, in the case study, we set the probability of limiting the chain length to 50% with a maximum chain length C_{max} of 2. In this appendix, we exhibit empirical proof of our choice.

Figure 4.B.4 shows different estimated Pareto frontiers for the case study (168 students) depending on the probability of limiting the maximum chain length (guided search) or not (random search). For readability reasons, we display only a completely random approach (100% random), a fairly guided approach (20% random and 80% guided) and a 50%-50% approach, which is the one used in the case study.

Figure 4.B.5 shows the different estimated Pareto frontiers obtained by the algorithm for different C_{max} values (tests are run for one hour on the case study, i.e. 168 students locations). Considering all the solution points of the estimated Pareto frontiers of the tests at the same time, the algorithm with the value of $C_{max} = 2$ is the one that returns the most non-dominated solutions (101 out of 116).

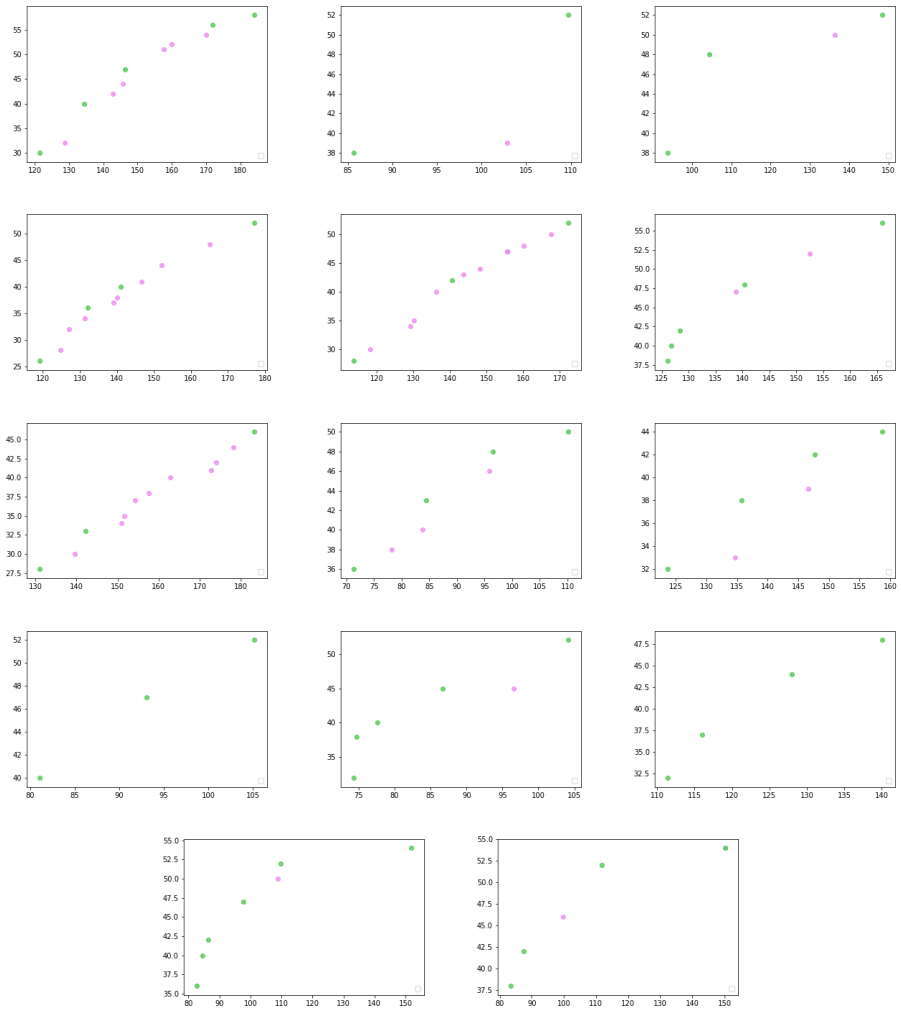


Figure 4.A.1: Observed Pareto frontiers of the instances with 12 students. On the x-axis, the routing costs; on the y-axis, the quality of service. In green, points found solving the ILP and by the metaheuristic; in violet, points found only by the metaheuristic.

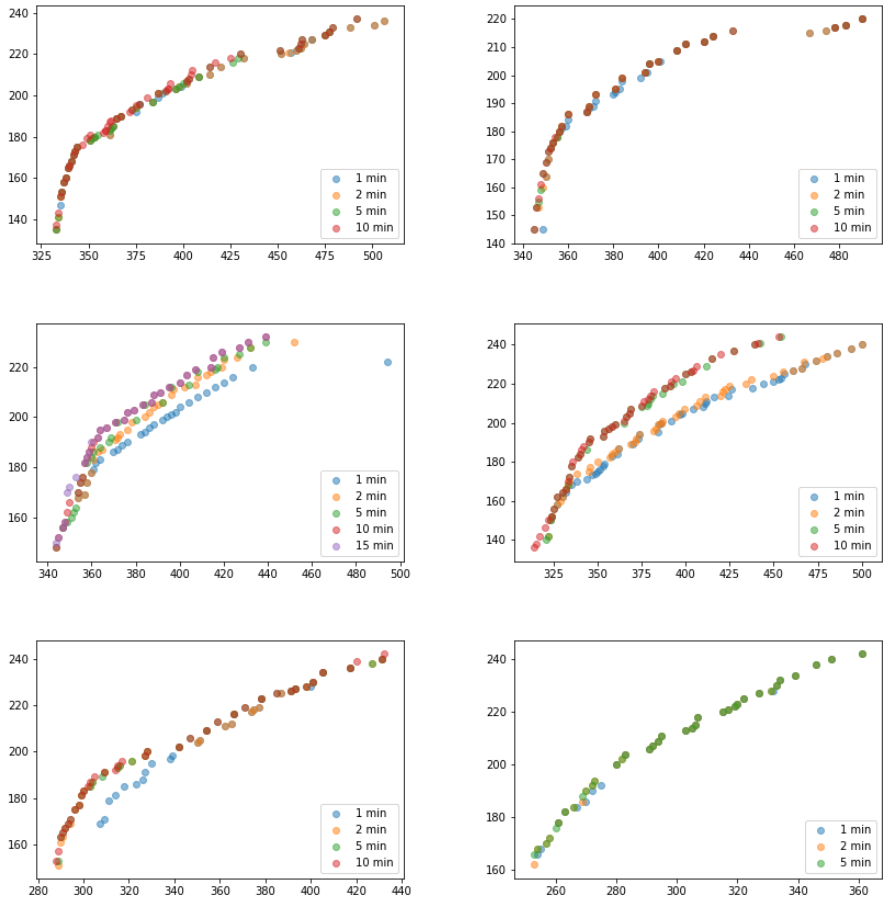


Figure 4.B.1: Estimated Pareto frontiers via the metaheuristic at different time instants for instances of size 50 students.

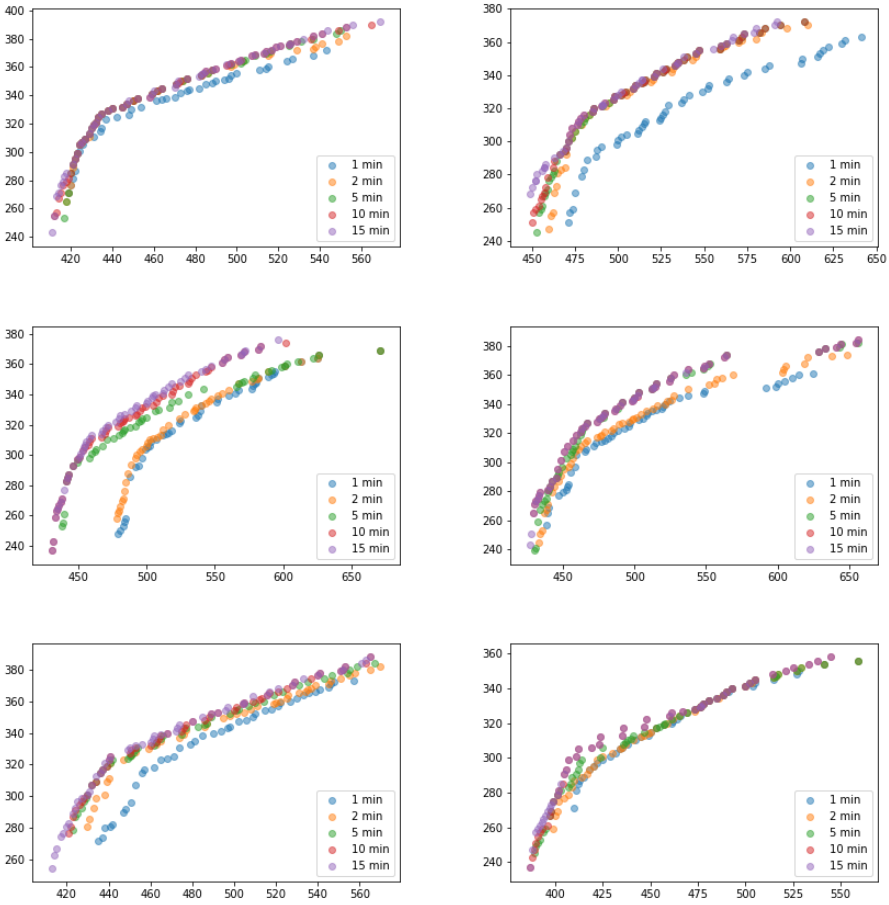


Figure 4.B.2: Estimated Pareto frontiers via the metaheuristic at different time instants for instances of size 75 students.

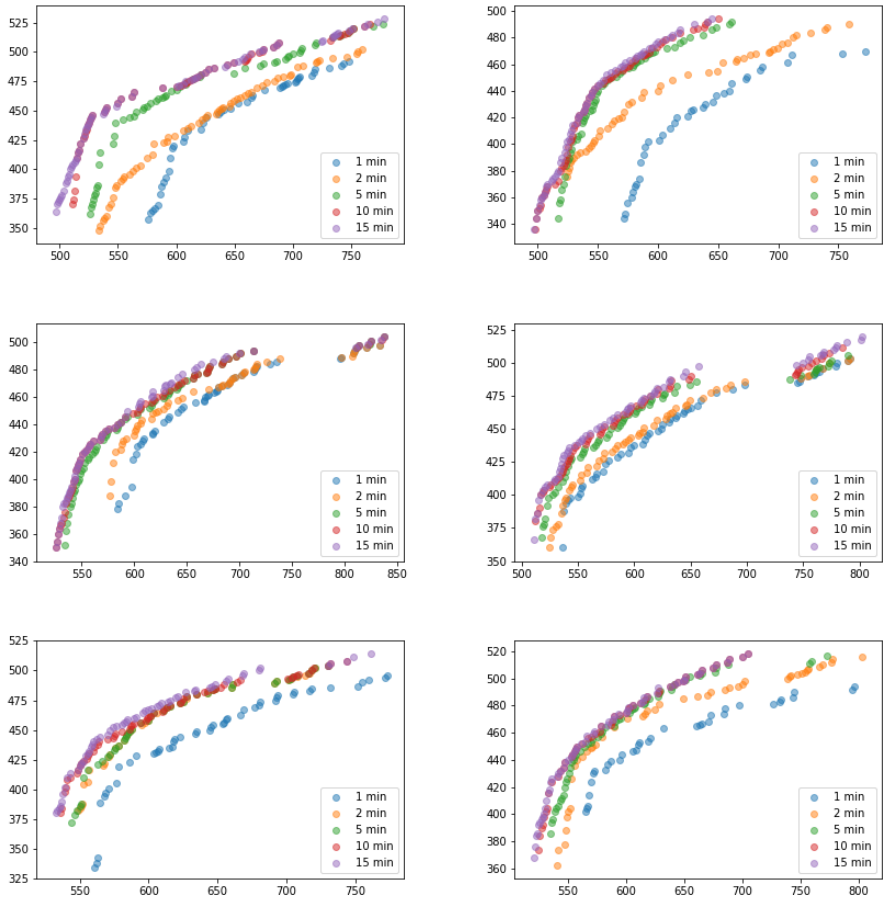


Figure 4.B.3: Estimated Pareto frontiers via the metaheuristic at different time instants for instances of size 100 students.

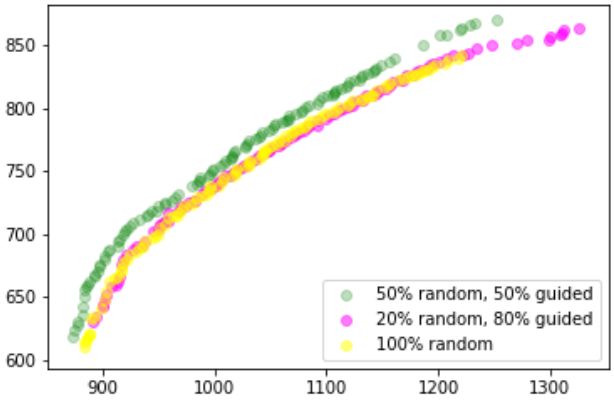


Figure 4.B.4: Different estimated Pareto frontiers depending on the probability of selecting a guided search. On the x-axis, the routing cost; on the y-axis, the quality of service.

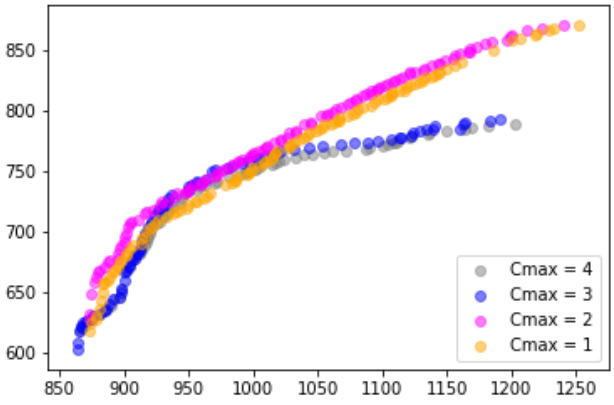


Figure 4.B.5: Estimated Pareto frontiers for different C_{max} values. On the x-axis, the routing cost; on the y-axis, the quality of service.

5

REINFORCEMENT LEARNING FOR THE KNAPSACK PROBLEM

Jacopo PIEROTTI, Maximilian KRONMÜLLER, Javier ALONSO-MORA, Theresia VAN ESSEN, Wendelin BÖHMER

Combinatorial optimization (CO) problems are at the heart of both practical and theoretical research. Due to their complexity, many problems cannot be solved via exact methods in reasonable time; hence, we resort to heuristic solution methods. In recent years, machine learning (ML) has brought immense benefits in many research areas, including heuristic solution methods for CO problems. Among ML methods, reinforcement learning (RL) seems to be the most promising method to find good solutions for CO problems. In this work, we investigate an RL framework, whose agent is based on self-attention, to achieve solutions for the knapsack problem, which is a CO problem. Our algorithm finds close to optimal solutions for instances up to one hundred items, which leads to conjecture that RL and self-attention may be major building blocks for future state-of-the-art heuristics for other CO problems.

5.1. INTRODUCTION

In recent years, machine learning (ML) has shown super-human capabilities in speech recognition, language translation, image classification, etc. (Bontemps et al., 2016; O'Shea and Nash, 2015; Vaswani et al., 2017). Lately, more and more combinatorial optimization (CO) problems have been studied under the lens of machine learning (Bengio

Parts of this chapter have been published in: Optimization and Data Science: Trends and Applications, volume 6 of 5th AIROYoung Workshop and AIRO PhD School 2021, AIRO Springer Series. 1st edition, 2021 ISBN: 978-3-030-86285-5.

et al., 2021). Among these CO problems, NP-hard problems are of interest because, so far, solving them to optimality (via so-called exact methods) takes exponential time; thus, for many classes of CO problems, obtaining good solutions for large or even medium sized instances in reasonable time can only be achieved by exploiting handcrafted heuristics. Instead of creating a heuristic by hand, one can also use ML to train a neural network to predict an almost optimal solution for given or randomly generated CO instances (Bengio et al., 2021). This way heuristics can be learned without expert knowledge of the problem domain, which is also called *end-to-end training*. Reinforcement learning (RL) seems to be the most promising end-to-end method to solve combinatorial problems (Bello et al., 2016). In fact, in difference to supervised ML, RL does not need to know the solutions to given training instances to learn a good heuristic. This way one can learn a heuristic without any domain knowledge and, in principle, one could find a heuristic that works better than any a human would be able to design. RL has been used to train the neural networks used by heuristics designed to solve CO problems (Joshi et al., 2020; La Maire and Mladenov, 2012; Nazari et al., 2018), including the knapsack problem (KP, Bello et al., 2016). The aim of this work is to develop an RL end-to-end algorithm for the knapsack problem based on attention (Vaswani et al., 2017), in difference to prior work that used either recurring neural networks (RNN, Bontemps et al., 2016) or convolutional neural networks (CNN, O'Shea and Nash, 2015) which are popular NN for end-to-end methods. By developing such an algorithm for a relatively easy CO problem (the KP, Pisinger, 2005), we want to assess if RL with attention can be a fruitful method to tackle other, more complex, CO problems, which will be the focus of future research.

The remainder of the chapter is organized as follows. The formulation of the KP, our motivations on how and why we use attention (and not RNNs or CNNs) and model architecture are presented in Section 5.2. The training distributions (i.e. benchmarks of instances) used for testing and evaluating as well as the computational results are detailed in Section 5.3. Finally, in Section 5.4, we illustrate our conclusions.

5.2. PROBLEM FORMULATION AND BACKGROUND INFORMATION

The knapsack problem (KP) is one of the most studied CO problems (Pisinger, 2005). As input, we have a set of objects (denoted by set N) and a knapsack of capacity W. Each object $i \in N$ has a positive profit p_i and a positive weight w_i. The objective of the problem is to maximize the sum of the profits of the collected objects without violating the capacity constraint. Introducing binary variables x_i, which assume value one if object $i \in N$ is selected and zero otherwise, we can write the problem as follows:

$$\max \sum_{i \in N} x_i p_i \tag{5.1}$$

$$\sum_{i \in N} x_i w_i \leq W \tag{5.2}$$

$$x_i \in \{0,1\} \qquad \forall i \in N. \tag{5.3}$$

The objective function (5.1) maximises the total profit of the selected objects, Con-

straint (5.2) acts as the capacity constraint and Constraints (5.3) force the variables to be binary. This integer linear program (ILP) belongs to the class of NP-hard problems (Pisinger, 2005), which means that the computation time for obtaining optimal solutions with known exact solution methods grows exponentially with the number of objects. A simple yet very powerful heuristic is to sort the objects in non-increasing order of their ratio, i.e., $q_i = \frac{p_i}{w_i}$ for $i \in N$ and collect them in order as long as Constraint (5.2) is respected (collecting non-consecutive objects is allowed). In the following, we refer to this as the *simple heuristic*.

5.2.1. REINFORCEMENT LEARNING FRAMEWORK

In this section, we give an overview of how we implemented our algorithm. For more details on RL, we refer the reader to Sutton and Barto, 2018. Our algorithm belongs to the area of multi-task RL (Vithayathil Varghese and Mahmoud, 2020), where a task is an instance of the knapsack problem. The difference between single and multi-tasks is that: in single-task, we want to learn a policy to always solve the same (instance of a) problem; in multi-task, we want to learn a policy to solve a family of different instances of a problem (or even different problems). Moreover, while in single-task the initial state is always the same, this does not hold in multi-tasks. In order to describe a state in our case, we first define how we embed the objects into vectors. At any given time step, each object $i \in N$ is uniquely associated to a vector. Each vector is of the form $t_i = [p_i, w_i, q_i, \bar{x}_i, u]$, where $\bar{x}_i$ is a binary parameter assuming value zero when object i has already been selected or cannot be selected due to the capacity constraint (5.2) and one otherwise and u is the residual capacity of the knapsack (i.e., W minus the weights of the already selected objects).

We name the selection of an object an *action*. Actions (A) are chosen based on the Q-value of each object (see Section 5.2.1). The algorithm that determines the Q-values is called the *agent* (see Section 5.2.2). In RL, a *state* represents the available information about the process at a given moment. We represent the observation of a state by the matrix obtained stacking all the $|N|$ object vectors together. Given a non-final state, the agent has to select an action; however, not all objects can be chosen in any state. While choosing an action, the non-selectable objects are momentarily removed, which is called masking. In our case, generic action (object) i is masked when $\bar{x}_i$ equals zero. The initial state has $u = W$ and $\bar{x}_i = 1$ for all $i \in N$, while we define a state as final if $\bar{x}_i = 0$ for all $i \in N$. Our algorithm sequentially selects objects until no additional object can be selected, in which case the algorithm terminates.

Given a state, each action leads to a new state and a reward. In our case, the reward r of choosing object $i \in N$ is the profit of the chosen object (i.e. $r = p_i$ when choosing object $i \in N$). The series of states in between an initial and a final state is called an *episode*. The final objective of an RL algorithm is to maximize the (discounted) cumulative reward observed in an episode. In general, we discount the future reward to avoid problems arising with very long or non-finishing episodes. In our case, episodes are relatively short and they always terminate (worst case scenario, they terminate in $|N|$ steps); so, there is no need to discount the future rewards. We call the sequence of old state s, chosen action a, observed reward r and new state s' a *transition*. A set (of fixed

name	definition
Task	An instance of the KP
Multi-task RL	RL algorithm to solve a family of tasks (virtually any KP problem in our case)
Action	The selection of an object
State	The available information (profits, weights, which objects have been selected and which have not,..) at a given time moment
Action	The selection of an object
Initial state	State where no objects have been selected yet
Final state	State where no more objects can been selected
Episode	Series of visited states from the initial state to the final state
Masking	The removal of the unselectable actions
Reward	The profit of the selected object
Transition	A sequence of an old state, a chosen action, an observed reward and a new state
Minibatch	A set of non-consecutive[1] transitions
Training distribution	Distribution from which we draw the instances to train the algorithm

Table 5.2.1: Summary of the definitions needed for our reinforcement learning framework

size, in our case) of non-consecutive[1] transitions is called a *minibatch*. The transitions in the minibatches are used to compute the loss (which is needed in order to learn) in the learning step (Section 5.2.1). Finally, we train and evaluate the algorithm by solving randomly drawn tasks from the so called *training distribution*. Table 5.2.1 summarises the introduced definitions.

DOUBLE Q-LEARNING

Our RL algorithm falls under the general umbrella of Q-learning (Watkins and Dayan, 1992). Given a state s, and a set of possible actions A, the idea of Q-learning is to estimate the expected future cumulative rewards for each possible action (called Q-values $Q(s,a), \forall a \in A$) and select one action based on a exploration/exploitation strategy. On one hand, exploration is fundamental to search the state-action space. In fact, in (non-deep) Q-learning, if one could explore for an infinite amount of time, the optimal Q-values would be retrieved. On the other hand, the agent should concentrate more on promising actions to improve convergence to an optimal policy. As exploration/exploitation strategy, we use ϵ-greedy, which greedily chooses the best action (i.e. the action with the highest Q-value) with probability 1-ϵ or a random action with probability ϵ. Often the Q-learning algorithm can be too optimistic while estimating the Q-values. One common solution to this problem is to adopt double Q-learning (Hasselt, 2010). In deep RL, double Q-learning is enforced by having two identically structured neural networks. The current network is used to select the best action at the next state while the other one (called the *target* network) is used to compute the Q-value of the next state. In this work, we use a similar method which helps stabilizing our results. The

[1] Transitions do not have to be consecutive, but, by chance, they could be.

difference being that the Q-values are always computed via the current network and the target network is used to determine the action. Naming Q' the function to compute the Q-values associated with the target network, our revised Bellman equation becomes:

$$Q(s,a) = r(s,a) + Q(s', \arg\max_a(Q'(s',a))). \tag{5.4}$$

Equation (5.4) is needed in the learning step, where the parameters of the Q function are tuned in such a way that the distance between $Q(s,a)$ and $r(s,a) + Q'(s', \arg\max_a(Q(s',a)))$ is minimized.

LEARNING STRATEGY

Our algorithm works by generating and solving new tasks of different dimensions (i.e. $|N|$ is not a constant between two different tasks). Let us assume that we train our algorithm to solve instances of k different sizes. Every time a new instance is generated and solved, all the transitions are stored in a replay buffer (S. Zhang and Sutton, 2017). Our algorithm has k different fixed-size replay buffers (one for each possible dimension of $|N|$) where transitions are stored with a FIFO (first in first out) strategy. A FIFO policy guarantees that the algorithm always keeps in memory the newest generated information. Transitions of instances with the same dimensions are stored in the same buffer. When a minibatch is needed, we randomly choose one of the k replay buffers and extract a minibatch from there. Given the multiple replay buffers, each state in the minibatch has the same dimension and, thus, can be stack together, easing the computation. It is important to note that different tasks have different gradient magnitude: a task with 100 objects is likely to have a different gradient than a task with 2 objects. In fact, we are using a NN to approximate the Q-values and, reasonably, the approximation becomes more and more difficult (thus less and less accurate) with an increasing number of objects. A less accurate Q-value approximation would likely lead to greater gradient magnitudes; thus, different tasks present different gradient magnitude. However, since each time we choose the replay buffer uniformly at random, we are averaging the gradients; thus, we are not introducing any bias. When the algorithm has accumulated *enough* transitions in the replay buffers, it begins to learn. We do so by selecting, uniformly at random, from one random replay buffer, t transitions (or all the transitions if less than t transitions are present in that replay buffer). Transitions which have never been selected before have priority over transitions that were. We call these t transitions a *minibatch*. For generic transition i (s_i, a_i, r_i, s'_i) in the minibatch, we compute the loss as:

$$loss_i = \left(Q(s_i,a_i) - \left(r_i + Q(s'_i, \arg\max_a(Q'(s'_i,a))\right)\right)^2 \tag{5.5}$$

Then, we backpropagate the average of the t losses. Sometimes, the loss function is so steep that blindly following its gradient would lead outside of the region where the gradient is meaningful. To prevent this, we clip the gradient (J. Zhang et al., 2020) to a maximum length of 0.1. The parameters of the agent are updated via the RMSprop method[2] (Duchi et al., 2011). Finally, the *target* network is updated via a soft-update (Fox et al., 2016), i.e., naming p any generic parameter of the agent, p_t its corresponding one in the target network and τ (constant equal to 0.05 in our case) the soft-update parameter: $p_t \leftarrow (1-\tau)p_t + \tau p$.

[2] http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf

5.2.2. AGENT

The agent receives the observations of the states and outputs the Q-values. It is composed by three main blocks, all using ReLU as activation function. The first and last block are composed of two fully connected linear layers of dimension 512 each. The first block enlarges the feature space of each object vector from five to 512 and the last block reduces the features to one (the Q-value). The second block is a transformer (Section 5.2.2). In most CO problems, there is no clear ordered object structure. Even if we introduce an arbitrary order, the problem would be permutation invariant. In the KP, a permutation of the elements would neither change the optimal solution of the problem nor its structure. For this reason, we decide to base our agent on self-attention, which is permutation invariant (unlike CNNs or RNNs). While most agents for end-to-end approaches involve CNNs and/or RNNs (Nazari et al., 2018), we conjecture that, for the KP and other CO problems, the effectiveness of an algorithm does not lie within those structures. In fact, CNNs are an excellent tool to extract local features (O'Shea and Nash, 2015), but they are only useful when there is a clear ordered object structure (such as pixels in an image). RNNs sequentially embed a sequence of inputs, where each output depends also on the sequence of previous inputs. This is very useful when states are partially observable (Bontemps et al., 2016); however, the KP satisfies the Markov property, i.e., the distribution of future states depends only on the current state. This memoryless property makes the problem Markovian. Thus, given the Markovian property of our problem and the absence of an underlying ordered structure, we decide to base our implementation on a variation of the transformer (Vaswani et al., 2017) without CNNs or RNNs. The transformer accepts as input a variable-length (d_t) tuple of objects (where all objects have the same dimension d_o) and returns a tuple of same length and dimension (d_o for each single output, d_t for the whole tuple). It is composed by a series of multi-head attention mechanisms in a layer structure (see Section 5.2.2). Attention is a powerful mechanism that allows to look at the input and generate a context vector based on how much each part of the input is relevant for the output. Doing so, the algorithm learns to isolate from a set of features the one(s) relevant for that particular state.

SELF-ATTENTION, MULTI-HEAD AND MULTI-LAYER TRANSFORMER

Self-attention is a powerful ML technique that takes a set of objects and returns an equally sized set of vectors. In our case, the objects taken as input are matrices obtained via linear transformations of the object vectors. These matrices, called queries Q, keys K and values V, have size d_q, d_n and d_n, respectively. Self-attention is a function that measures the similarity of queries and keys with a dot product; then, a softmax of that similarity is used to weight the values in a linear combination. So, naming W^Q, W^K and W^V the matrices of learnable parameters for the linear transformations and $\bar{S} \in \mathbb{R}^{n \times d_n}$ the embedded state observation (i.e., the matrix obtained by stacking the object vectors of dimension 512), we obtain:

$$\text{Attention}(Q,K,V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_n}}\right)V = \text{softmax}\left(\frac{(\bar{S}W^Q)(\bar{S}W^K)^\top}{\sqrt{d_n}}\right)(\bar{S}W^V).$$

Instead of a single self-attention mechanism on vectors of dimension d_n, Vaswani et al., 2017 discovered that it was beneficial to linearly project the queries, keys and values

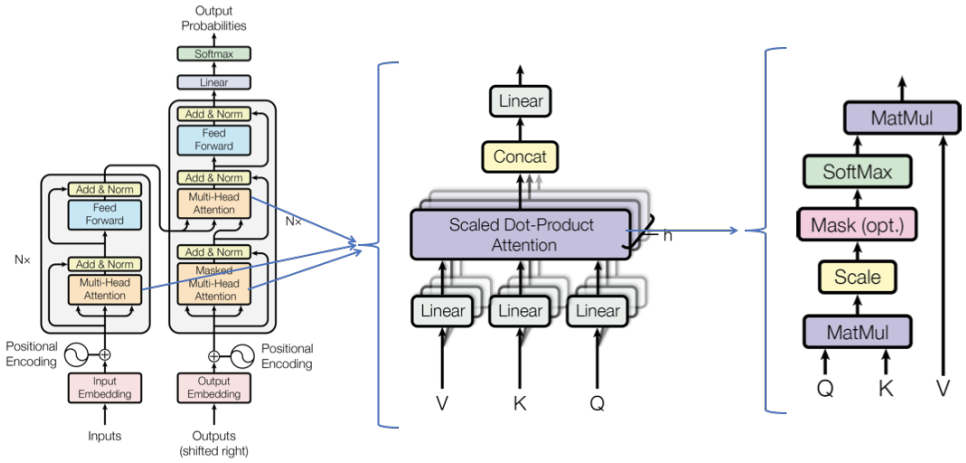


Figure 5.2.1: Full model architecture of the transformer. Image source: Vaswani et al., 2017, the positional encoding is used in language translation, not in our problem.

5

h times (called a *head*; hence, multi-head) with different, learned linear projections on a smaller dimension of size $d_v = \frac{d_n}{h}$. The outputs are computed in parallel, concatenated and reprojected once again (via a learnable matrix W^0). Formally, this becomes:

$$\text{MultiHead}(\bar{S}) = [head_1, \cdots, head_h]W^0,$$

where $head_i = \text{Attention}(\bar{S}W_i^Q, \bar{S}W_i^K, \bar{S}W_i^V)$ and W_i^Q, W_i^K, W_i^V are all learnable matrices for all $i \in [1, ..., h]$. This multi-head self-attention mechanism is repeated for L layers. Each layer is composed of two units which both produce outputs of the same dimension as their input, i.e., d_n. The first unit is indeed the multi-head self-attention mechanism, the second unit is a fully connected feed-forward network with ReLUs. Both these units adopt also a residual connection and a layer normalization (Ba et al., 2016); i.e. naming x the input to the unit itself and $f(x)$ its function (the multi-head self-attention or the fully connected feed-forward network), the output of each unit is the normalization of $x + f(x)$. In fact, if we assume function $y = f(x)$ to be learnable; then, it is reasonable to assume function $y = f(x) - x$ to be learnable as well. The residual connection was proven to facilitate learning (He et al., 2016).

This overall structure is called the transformer and Figure 5.2.1 displays its graphical representation.

5.2.3. MODEL ARCHITECTURE

In each instance, all objects are normalized such that the maximum profit and weight is one. The agent has a two layer fully connected neural network to expand the 5 features of a vector into 512 features. The resulting vector is fed to a transformer encoder[3]

[3] for details see https://pytorch.org/docs/stable/generated/torch.nn.TransformerEncoder.html. Many optional parameters were set to the default values, such as the feedforward dimension was set to 512 and the probability of dropout to 0.1

with six layers and eight heads per layer. Normalization is applied after each layer. After the transformer, another two fully connected neural network layers are used to reduce the 512 features to a single one (the Q-value associated with the action of selecting the corresponding object). The learning rate of the optimizer is set to 10^{-6} and ϵ linearly decreases with the episode number from one to 0.05. Each replay buffer can store up to a maximum of 10^5 transitions, the minibatch size is set to 512 and the soft update parameter τ is set to 0.05. The overall structure of the algorithm is given in Algorithm 3. For a total of 10^5 times, the algorithm generates and solves one instance. Its transitions are saved in the replay buffer and the algorithm takes a learning step. In order to partially fill the replay buffers, the algorithm starts to learn only after the 512th iteration. During the training, ten equally spaced greedy test evaluations over one hundred randomly generated instances are conducted in order to assess the algorithm progress.

Algorithm 3 RL algorithm overview

```
for i = 0, ··· , 10^5 do
    task ← generate new task
    transitions ← solve the task with an ε-greedy policy
    store transitions in replay buffer
    if i ≥ 512 then
        learning step
    end if
    if i mod 10^4 = 0 then
        evaluate the algorithm with a greedy policy
    end if
end for
```


5.3. COMPUTATIONAL RESULTS

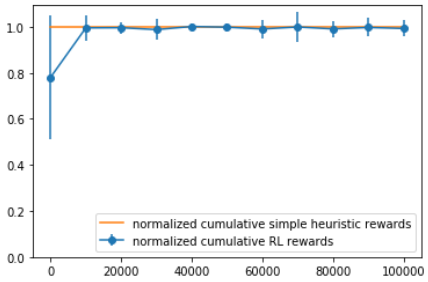
Two different training distributions are used to generate the tasks. In the first distribution, $|N|$ is chosen uniformly at random between 2 and 100 every time a new instance is generated. Moreover, the profit and weight of each object are also chosen uniformly at random in the closed interval $[10^{-6}, 1]$. An upper bound of one is enforced because it is well-known that neural networks perform better when the input data has absolute value lower than or equal to one. A lower bound of 10^{-6} is enforced to avoid numerical errors. We call this distribution *random*. The second distribution (named *Pisinger*) are some of the *small, large* and *hard* instances taken from Pisinger, 2005. These Pisinger instances were generated in order to be difficult to be solved via a MILP solver. These small, large and hard instances are further subdivided in six, six and five groups, respectively. From these groups, we select instances with 20, 50 and 100 objects. Each pair group-number of objects contains one hundred instances, for a total of 3200 instances (because not all groups have the 20 objects instances).

We train our algorithm twice from scratch, thus obtaining two different versions of the same model. We train the first version exclusively on the random instances while we train the second one exclusively on the Pisinger instances. We evaluate the trained algorithms both on random instances and on Pisinger's. When evaluating and testing, we compare our results with the simple heuristic (see Section 5.2) which achieves, on av-

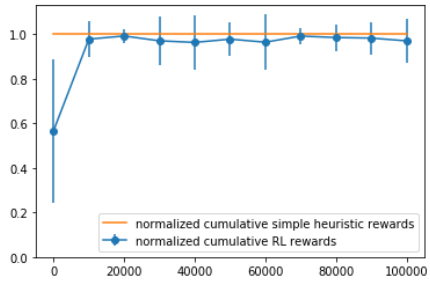
erage, 99% of the optimal solution's value (hence, it is a good measure for comparison). In Figures 5.4.1 and 5.4.2, every result is normalized with respect to the optimal solutions (in the Pisinger distribution) or with respect to the heuristic solution. Figure 5.4.1 displays the evaluations of the algorithm during training on one hundred random instances. For the sake of brevity, we report only the most meaningful results, i.e., the *hard* Pisinger instances with one hundred objects. Figure 5.4.2i shows the boxplot of the gap to the optimal solution for the *hard* Pisinger instances of the algorithm trained on the random distribution. Although the results are overall satisfactory, the algorithm trained on random instances performs badly on some types of Pisinger instances. The most likely reason is that the algorithm trained on random instances has an extremely small probability of seeing some Pisinger instances (which have been hand crafted), thus it does not generalise over those particularly complex instances. On the other hand, when the algorithm is evaluated on randomly generated instances (Figure 5.4.1i), results are very close to the heuristic solution, thus, to the optimal solution. Figure 5.4.2ii displays the same gap for the algorithm trained on the Pisinger distribution. In this case, results are very satisfactory since the algorithm consistently achieves near-optimal solutions. Also while evaluating on randomly generated instances (Fig. 5.4.1ii), results are very close the heuristic solution, thus to the optimal solution; however, results are slightly worse than the results obtained by the algorithm trained on the random distribution. As expected, we conclude that training the algorithm on randomly generated instances boosts performance in the average case, but it is less effective to complex instances, while training the algorithm on the Pisinger distribution performs (sligthly) worse on the average case, but is much more robust (both on the random and on the complex Pisinger instances).

5.4. CONCLUSION

In this work, we introduced a deep Q-learning framework with a transformer as the main deep architecture for the KP problem. The algorithm achieves results very close to optimality within a split second on instances up to one hundred objects. These results are promising; however, in the KP, also a simple conventional heuristic returns very solid results. Nonetheless, our results suggests that "attention is all you need" may also hold in end-to-end methods for CO problems. Future research will explore a transformer-based RL method on other CO problems where conventional heuristics fail to give good solutions in a short amount of time. Moreover, our algorithm computes the Q-values which are difficult quantities to estimate. Instead, one could aim to learn directly the policy with whom to take actions (i.e. the probability distribution of the actions for a given state). This policy could be learned by firstly using behavioural cloning (Torabi et al., 2018) (to imitate the heuristic) and, secondly, RL to explore more possible state-action combinations.

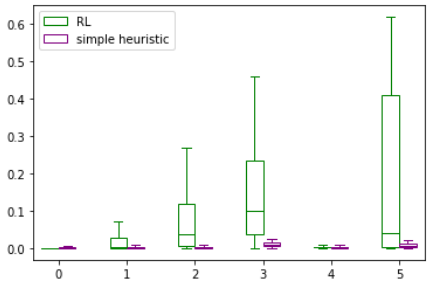


(i) Training on the random distribution

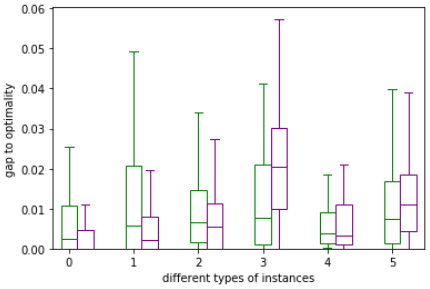


(ii) Training on the Pisinger distribution

Figure 5.4.1: During training evaluation on 100 random instances. On the x-axis, the number of iterations and on the y-axis, the averaged normalized cumulative reward are shown. The blue dots indicate the average cumulative reward, the vertical lines indicate the standard deviation.



(i) training on random distribution



(ii) training on Pisinger distribution

Figure 5.4.2: Evaluation on the hard, 100 objects Pisinger instances. Green lines for the RL and purple lines for the simple heuristic. On the x-axis, different groups of instances, on the y-axis, the average normalized cumulative rewards are displayed. Please note the different scale of the y-axis.

6

CONCLUSION

In this chapter, we summarize all results from this thesis and discuss them in light of their scientific and technical implications for society. In this dissertation, we discussed three novel routing problems (Chapter 2, Chapter 3 and Chapter 4) and a new approach on a classical combinatorial combinatorial optimization problem (Chapter 5). All of the treated problems are NP-hard. Their NP-hardness implies that exact methods and commercial software will eventually fail to return optimal solutions when the size of the instances increases; hence, efficient heuristics are needed.

In Chapter 2, we described our contribution for the MESS2018 competition, where we designed a metaheuristic to solve the balanced travelling salesman problem. Our algorithm stems from the well-known (adaptive) iterated local search algorithm and adds two features. The first one is an uneven reward-and-punishment rule. In our opinion, unevenly adapting the operators' weights lets the algorithm be more responsive to sudden changes in the structure of the current solution, which is a dynamic component, rather than to the structure of the network, which is static. Given the competition instances that use graphs with no known metric and high cost, each change performed by the algorithm, independently from the operator, would result in a considerable modification of the solution structure, to which the algorithm would have to rapidly adapt. This idea does not have to be limited to this particular problem or instances; on the contrary, we think it can be easily applied wherever the adaptive metaheuristic (not necessarily iterated local search) substantially modifies the current solution.

Many CO problems, among which the TSP, have the characteristic of having many local optima very close to each other. Intuitively, we can see that by imagining a solution where most of the nodes are in the optimal order and only a few nodes are swapped. A solution may well be a local optimum and there can be more in its proximity. This means that once a local optimum is found, an intense local search should be carried out in its neighbourhood. In order to perform such an intense local search, we introduced a second feature, namely *random restarts*. Randomly restarting the algorithm from a local optimum ensures us to abundantly and intensively search the solution space close to known good solutions. Clearly, searching just in the neighbourhood of a subset

of solutions does not allow the algorithm to explore enough throughout the whole solution space. For this reason, the random restarts become less and less frequent as fewer new good solutions are found. This trade-off allows us to profoundly search the neighbourhoods of known good solutions without getting endlessly stuck.

Chapter 3 discusses the dial-a-ride-problems with transfers. Dial-a-ride-problems are a well-studied class of problems where vehicles have to pick up and deliver requests under specific constraints, e.g. capacity and time constraints. Often, given the complexity of the problem, it is simplified in order to solve medium sized instances.

Instead of adopting simplifying assumptions, we introduce very articulated models in order to be as close as possible to reality. Moreover, we allow requests to be transported by multiple vehicles instead of adopting the conventional unique pairing between vehicles and requests. This is done by the possibility of transfers and was inspired by some commercial enterprises. In fact, in some industrial applications, goods are picked up and delivered in intermediate hubs where they wait for the next vehicle to pick them up. On the one hand, usually, there are just a few of these hubs and, to limit the complexity of the problem, just one transfer is considered. On the other hand, when transporting people, multiple transfers may come in handy and they do not have to be limited to the presence of hubs. In fact, generally speaking, all the nodes in the network can be possible transfer points.

Moreover, most authors in the literature on routing problems do not allow vehicles or requests to travel the same arc multiple times, while this can well be the case in practice (imagine taxis going back and forth from the airport). To model this leap in complexity (travelling multiple times the same arc, providing multiple transfer nodes and allowing multiple transfers) we introduce the *flow formulation*. The idea is to model both vehicles' and requests' paths as flows between nodes in time and then force the requests' flow on the arcs to be paired with any of the vehicles' flow. Doing so, we do not have to explicitly model any transfer.

The articulated model limits us to be able to solve only small instances; nevertheless, even in these small instances, we can already see the benefits of introducing the option of transfers. In addition, we proposed core and extended models (i.e., more and more complex models to capture often neglected small features), both in continuous and discrete time. Conclusions were drawn on how much the extended models were more complicated than the core ones. Similar analyses were conducted to highlight the difference between the continuous and discrete time models. Finally, we conjecture that bigger and bigger instances will amplify the benefits of transfers and future work will be directed in the direction of designing metaheuristics able to solve the dial-a-ride problem with transfers and prove (or disprove) our conjecture.

Also the project described in Chapter 4 stems from a real-world problem. In this case, we collaborated with a school for students with special needs in the county of Kent (United Kingdom). The aim of this project is to ease the commuting experience of special education needs students without sacrificing attention to travel costs. Indeed, transporting SEN students, in addition to being stressful, is a very expensive task to perform due to special busses and attendants (i.e. staff) costs. Having two conflicting objectives

(minimizing travel costs and maximizing the quality of the service) means that there does not exist one single optimal solution, but there exists a family of optimal solutions. This increases the complexity of solving such a problem.

In Chapter 4, we described how to model and solve the biobjective school bus routing problem for students with special needs, both from an exact and a metaheuristic point of view. We applied the exact method to small instances derived from the case study because they could be solved to optimality in reasonable time. Our valid inequalities are based on symmetry breaking constraints and on an overestimation of the quality of the service. Even though an integer quadratic problem needs to be solved to introduce these valid inequalities, these valid inequalities caused a decrease of about one quarter of the overall computation time.

Despite the speedups, large instances are still too complex to be solved via exact methods. Hence, we designed a metaheuristic. We developed a simple, robust and efficient general metaheuristic for multi-objective problems and we tested it on instances of increasing size. Metaheuristics for vehicle routing problems usually entails the use of many different operators such as insertion, swap, 2-opt, longest route selection, etc. We unified all of these different operators in a single general operator that chooses at random two routes to modify and one, possibly empty, chain of students per route. All the above mentioned decisions (which routes to pick, how to choose the chains, etc.) are taken following an uniformly distributed probability distribution. Although one could argue that other more effective choices should be performed, for instance, giving priority to longer routes, we motivate our choice stating that these more effective decisions might not be worth the extra computation power needed.

Medium sized instances (50 to 100 students) were reasonably solved in about 15 minutes. For the case study instead, we added an intensified search feature, meaning that the algorithm was more prone to select smaller chains, and confront the performance with and without the intensified feature. Both versions were run for one hour. Although the algorithm with the intensified search feature converged faster to a Pareto frontier, both algorithms returned stable Pareto frontiers within an hour of computation time. This leads us to conjecture that, instead of letting almost all decisions be decided via a uniformly distributed probability, other distributions, which could be learned, can be used and they could lead to an increase in performance.

The design of our metaheuristic arises from the work that Martin Josef Geiger developed for the Verolog challenge 2019[1]. In this work, an almost aparametric metaheuristic was introduced and it achieved incredible results obtaining second place in the competition. In our opinion, aparametric metaheuristics are extremely interesting because, even if they might not be as performing as metaheuristics with plenty of carefully hand-tuned parameters, they usually are much more robust. In fact, while carefully handpicking parameters may increase the performance of an algorithm on a specific problem, it is very likely to decrement its performance on all other problems where the same algorithm can be applied. Future directions of research would include modifying our almost aparametric algorithm such that the few tunable parameters would self-tune, either using adaptiveness or using some hybrid machine learning technique.

[1] https://verolog2019.ortec.com/ link accessed October 2021

Finally, in Chapter 5, we present our contribution in one of the most promising directions in the metaheuristic field, machine learning-based heuristics. Artificial intelligence and machine learning seemingly endless potential revolutionised entire areas of human knowledge and, in our opinion, they will play a crucial role in future metaheuristics. Some may say that they are already present in state-of-the-art heuristics; in fact, one can see adaptiveness as an early form of artificial intelligence.

In order to quickly generate heuristic solutions from instances, reinforcement learning seems to be the most promising algorithm. In addition, reinforcement learning comes with some effective features. Firstly, it does not need a data set of known good or optimal solutions in order to learn, which means it can directly be applied to new, yet unstudied problems. Second, since little to no human information regarding the specific problem to be solved has to be passed to the algorithm, reinforcement learning seems to be an extremely versatile tool to solve large classes of problems. For instance, almost all Atari 2600 games can be played with superhuman performances by the same algorithm[2].

Our contribution is to design, implement and test a reinforcement learning structure that could learn to solve the knapsack problem without any problem-specific knowledge, except for masking out the unselectable objects. We chose the knapsack problem because we wanted to test the effectiveness of our algorithm on a very well-known combinatorial problem. Moreover, many current machine learning architectures are either bounded by a constant-size vector, for instance, recursive neural network (RNN), or can work only with inputs of specified dimensions, for instance, convolutional neural network (CNN). To avoid those limitations, we chose to rely on the transformer (a self-attention based mechanism). This has the advantage of not being size-dependent, i.e. our algorithm can solve instances with 2 or with 100 objects without any modifications. Furthermore, none of the intermediate context vectors are bounded by a constant-size threshold; this means that the length of the context vector grows with the size of the instance being solved.

After training, our algorithm returns close to optimal solutions, on average within one percent of the optimal solution, in a split second. These results are promising, but in the knapsack problem, there already exist non-machine learning based heuristics with similar results. Nonetheless, our results are encouraging and future work will be directed in applying a similar reinforcement learning framework to various other CO problems.

In this dissertation, on the one hand, we have solved very complex models to optimality; yet, this encompasses that the size of the instances must be sacrificed, even when using speed-ups techniques such as valid inequalities. On the other hand, when using heuristics, we demonstrated that aparametric, meaning with few parameters or with self-tuning parameters, algorithms can achieve optimal or close to optimal results in both single and multi-objective problems. Finally, we gave a contribution in the blossoming field of machine learning and artificial intelligence for combinatorial optimization.

[2] The algorithm structure is the same, the training is game-specific.

BIBLIOGRAPHY

Agatz, N., Erera, A., Savelsbergh, M., & Wang, X. (2012). Optimization for dynamic ride-sharing: A review. *European Journal of Operational Research, 223*(2), 295–303.

Ahmed, S., Adnan, M., Janssens, D., & Wets, G. (2020). A route to school informational intervention for air pollution exposure reduction. *Sustainable Cities and Society, 53*.

Applegate, D. L., Bixby, R. E., Chvátal, V., & Cook, W. J. (2011). The traveling salesman problem. *Princeton University Press*.

Artigues, C., & Roubellat, F. (2000). A polynomial activity insertion algorithm in a multi-resource schedule with cumulative constraints and multiple modes. *European Journal of Operational Research, 127*(2), 297–316.

Ba, J. L., Kiros, J. R., & Hinton, G. E. (2016). Layer normalization. *arXiv preprint arXiv:1607.06450*.

Balcan, M.-F., Dick, T., Sandholm, T., & Vitercik, E. (2018). Learning to branch. *International Conference on Machine Learning*, 344–353.

Baniasadi, P., Ejov, V., Filar, J. A., Haythorpe, M., & Rossomakhine, S. (2014). Deterministic "Snakes and Ladders" Heuristic for the Hamiltonian cycle problem. *Mathematical Programming Computation, 6*(1), 55–75.

Barma, P. S., Dutta, J., & Mukherjee, A. (2019). A 2-opt guided discrete antlion optimization algorithm for multi-depot vehicle routing problem. *Decision Making: Applications in Management and Engineering, 2*(2), 112–125.

Bello, I., Pham, H., Le, Q. V., Norouzi, M., & Bengio, S. (2016). Neural combinatorial optimization with reinforcement learning. *arXiv preprint arXiv:1611.09940*.

Bengio, Y., Lodi, A., & Prouvost, A. (2021). Machine learning for combinatorial optimization: A methodological tour d'horizon. *European Journal of Operational Research, 290*(2), 405–421.

Bollobás, B. (1998). Modern graph theory (Vol. 184). *Springer Science & Business Media*.

Bontemps, L., McDermott, J., & Le-Khac, N. (2016). Collective anomaly detection based on long short-term memory recurrent neural networks. *International Conference on Future Data and Security Engineering*, 141–152.

Bradley, R., Danielson, L., & Hallahan, D. P. (2002). Identification of learning disabilities: Research to practice. *Lawrence Erlbaum Associates, Inc.*

Buliung, R., Bilas, P., Ross, T., Marmureanu, C., & El-Geneidy, A. (2021). More than just a bus trip: School busing, disability and access to education in Toronto, Canada. *Transportation Research Part A: Policy and Practice, 148*, 496–505.

Caceres, H., Batta, R., & He, Q. (2019). Special need students school bus routing: consideration for mixed load and heterogeneous fleet. *Socio-Economic Planning Sciences, 65*(100), 10–19.

Caserta, M., & Voß, S. (2014). A hybrid algorithm for the DNA sequencing problem. *Discrete Applied Mathematics, 163*, 87–99.

Censor, Y. (1977). Pareto optimality in multiobjective problems. *Applied Mathematics and Optimization*, *4*(1), 41–59.

Chalkia, E., Grau, J. M. S., Bekiaris, E., Ayfandopoulou, G., Ferarini, C., & Mitsakis, E. (2016). Safety bus routing for the transportation of pupils to school. *Traffic Safety* (283–299). John Wiley & Sons Ltd.

Christofides, N. (1976). Worst-case analysis of a new heuristic for the travelling salesman problem (Technical Report No. 388). *Graduate School of Industrial Administration, Carnegie Mellon University*.

Conceição, L., Correia, G. H., & Tavares, J. P. (2017). The deployment of automated vehicles in urban transport systems: A methodology to design dedicated zones. *Transportation Research Procedia*, *27*, 230–237.

Cordeau, J., & Laporte, G. (2003). A tabu search heuristic for the static multi-vehicle dial-a-ride problem. *Transportation Research Part B: Methodological*, *37*(6), 579–594.

Correia, G. H., & van Arem, B. (2016). Solving the user optimum privately owned automated vehicles assignment problem (UO-POAVAP): A model to explore the impacts of self-driving vehicles on urban mobility. *Transportation Research Part B: Methodological*, *87*, 64–88.

Cortés, C. E., Matamala, M., & Contardo, C. (2010). The pickup and delivery problem with transfers: formulation and a branch-and-cut solution method. *European Journal of Operational Research*, *200*(3), 711–724.

Danloup, N., Allaoui, H., & Goncalves, G. (2018). A comparison of two meta-heuristics for the pickup and delivery problem with transshipment. *Computers & Operations Research*, *100*, 155–171.

Dantzig, G. B., Orden, A., & Wolfe, P. (1955). The generalized simplex method for minimizing a linear form under linear inequality restraints. *Pacific Journal of Mathematics*, *5*(2), 183–195.

Deleplanque, S., & Quilliot, A. (2013). Transfers in the on-demand transportation: the DARPT dial-a-ride problem with transfers allowed. *Multidisciplinary International Scheduling Conference: Theory and Applications (MISTA)*, 185–205.

Desaulniers, G., Desrosiers, J., & Solomon, M. M. (2006). Column generation (Vol. 5). Springer Science & Business Media.

Dijkstra, E. W. et al. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik*, *1*(1), 269–271.

Donati, A. V., Montemanni, R., Casagrande, N., Rizzoli, A. E., & Gambardella, L. M. (2008). Time dependent vehicle routing problem with a multi ant colony system. *European Journal of Operational Research*, *185*(3), 1174–1191.

Dorigo, M., & Gambardella, L. M. (1997). Ant colonies for the travelling salesman problem. *Biosystems*, *43*(2), 73–81.

Duchi, J., Hazan, E., & Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, *12*(7).

Edmonds, J. (1965). Paths, trees, and flowers. *Canadian Journal of mathematics*, *17*, 449–467.

Ellegood, W. A., Solomon, S., North, J., & Campbell, J. F. (2020). School bus routing problem: Contemporary trends and research directions. *Omega*, *95*(100).

Escario, J. B., Jimenez, J. F., & Giron-Sierra, J. M. (2015). Ant colony extended: Experiments on the travelling salesman problem. *Expert Systems with Applications*, *42*(1), 390–410.

Fagnant, D. J., & Kockelman, K. (2015). Preparing a nation for autonomous vehicles: Opportunities, barriers and policy recommendations. *Transportation Research Part A: Policy and Practice*, *77*, 167–181.

Fagnant, D. J., & Kockelman, K. M. (2014). The travel and environmental implications of shared autonomous vehicles, using agent-based model scenarios. *Transportation Research Part C: Emerging Technologies*, *40*, 1–13.

Feo, T. A., & Resende, M. G. (1995). Greedy randomized adaptive search procedures. *Journal of Global Optimization*, *6*(2), 109–133.

Fox, R., Pakman, A., & Tishby, N. (2016). Taming the noise in reinforcement learning via soft updates. *Proceedings of the Thirty-Second Conference on Uncertainty in Artificial Intelligence*, 202–211.

Garey, M., Johnson, D., & Tarjan, R. (1976). The Planar Hamiltonian Circuit Problem is NP-Complete. *SIAM Journal on Computing*, *5*(4), 704–714.

Geiger, M. J., & Graf, B. (2019). VeRoLog Solver Challenge 4: implementierungswettbewerb der EURO arbeitsgruppe vehicle routing and logistics optimization. *Gesellschaft für Operations Research*.

Geng, X., Chen, Z., Yang, W., Shi, D., & Zhao, K. (2011). Solving the traveling salesman problem based on an adaptive simulated annealing algorithm with greedy search. *Applied Soft Computing*, *11*(4), 3680–3689.

Gillett, B. E., & Miller, L. R. (1974). A heuristic algorithm for the vehicle-dispatch problem. *Operations Research*, *22*(2), 340–349.

Goeke, D., Roberti, R., & Schneider, M. (2019). Exact and heuristic solution of the consistent vehicle-routing problem. *Transportation Science*, *53*(4), 1023–1042.

Goldreich, O. (2010). P, np, and np-completeness: The basics of computational complexity. Cambridge University Press.

Gurobi Optimization, LLC. (2021). Gurobi Optimizer Reference Manual. https://www.gurobi.com

Hahsler, M., & Hornik, K. (2007). TSP-infrastructure for the traveling salesperson problem. *Journal of Statistical Software*, *23*(2), 1–21.

Hansen, P., Mladenović, N., & Pérez, J. A. M. (2010). Variable neighbourhood search: Methods and applications. *Annals of Operations Research*, *175*(1), 367–407.

Hartmanis, J. (1982). Computers and intractability: A guide to the theory of NP-Completeness. *SIAM Review*, *24*(1), 90.

Hasselt, H. (2010). Double Q-learning. *Advances in Neural Information Processing Systems*, *23*, 2613–2621.

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. *Proceedings of the IEEE conference on computer vision and pattern recognition*, 770–778.

Ho, S. C., Szeto, W., Kuo, Y.-H., Leung, J. M., Petering, M., & Tou, T. W. (2018). A survey of dial-a-ride problems: Literature review and recent developments. *Transportation Research Part B: Methodological*, *111*, 395–421.

Hou, Y., Zhong, W., Su, L., Hulme, K., Sadek, A. W., & Qiao, C. (2016). TASeT: Improving the efficiency of electric taxis with transfer-allowed rideshare. *IEEE Transactions on Vehicular Technology*, *65*(12), 9518–9528.

Ichoua, S., Gendreau, M., & Potvin, J.-Y. (2003). Vehicle dispatching with time-dependent travel times. *European Journal of Operational Research*, *144*(2), 379–396.

International Transport Forum. (2015). Urban mobility system upgrade. (6). https://doi.org/10.1787/5jlwvzdk29g5-en

Intini, P., Colonna, P., & Ryeng, E. (2019). Route familiarity in road safety: A literature review and an identification proposal. *Transportation Research Part F: Traffic Psychology and Behaviour*, *62*, 651–671.

Jagtenberg, C. J., Bhulai, S., & van der Mei, R. D. (2015). An efficient heuristic for real-time ambulance redeployment. *Operations Research for Health Care*, *4*, 27–35.

Johnson, D. S. (1990). Local optimization and the traveling salesman problem. *International Colloquium on Automata, Languages, and Programming*, 446–461.

Joint Research Centre. (2021). Transport sector economic analysis [website accessed in December 2021]. https://ec.europa.eu/jrc/en/research-topic/transport-sector-economic-analysis

Joshi, C. K., Cappart, Q., Rousseau, L.-M., Laurent, T., & Bresson, X. (2020). Learning TSP requires rethinking generalization. *arXiv preprint arXiv:2006.07054*.

Juneja, S. S., Saraswat, P., Singh, K., Sharma, J., Majumdar, R., & Chowdhary, S. (2019). Travelling Salesman Problem Optimization Using Genetic Algorithm. *2019 Amity International Conference on Artificial Intelligence*, 264–268.

Kojima, M., Mizuno, S., & Yoshise, A. (1989). A primal-dual interior point algorithm for linear programming. *Progress in Mathematical Programming* (29–47). Springer.

Köksal Ahmed, E., Li, Z., Veeravalli, B., & Ren, S. (2020). Reinforcement learning-enabled genetic algorithm for school bus scheduling. *Journal of Intelligent Transportation Systems*, 1–19.

Korte, B. H., Vygen, J., Korte, B., & Vygen, J. (2011). Combinatorial optimization (Vol. 1). *Springer*.

La Maire, B. F. J., & Mladenov, V. M. (2012). Comparison of neural networks for solving the travelling salesman problem. *11th Symposium on Neural Network Applications in Electrical Engineering*, 21–24.

Land, A. H., & Doig, A. G. (1960). An automatic method of solving discrete programming problems. *Econometrica*, *28*(3), 497–520.

Lange, K. (1994). An adaptive barrier method for convex programming. *Methods and Applications of Analysis*, *1*(4), 392–402.

Lawler, E. L., Lenstra, J. K., Rinnooy Kan, A. H. G., & Shmoys, D. B. (1985). The traveling salesman problem: A guided tour of combinatorial optimization. *Wiley-Interscience Series in Discrete Mathematics*.

Lenstra, J. K. (1974). Clustering a data array and the traveling-salesman problem. *Operations Research*, *22*(2), 413–414.

Lenstra, J. K., & Kan, A. H. G. R. (1975). Some simple applications of the travelling salesman problem. *Journal of the Operational Research Society*, *26*(4), 717–733.

Lenstra, J. K., & Kan, A. H. G. R. (1981). Complexity of vehicle routing and scheduling problems. *Networks*, *11*(2), 221–227.

Lewis, R., & Smith-Miles, K. (2018). A heuristic algorithm for finding cost-effective solutions to real-world school bus routing problems. *Journal of Discrete Algorithms*, *52*, 2–17.

Li, L. Y. O., & Fu, Z. (2002). The school bus routing problem: A case study. *Journal of the Operational Research Society*, *53*, 552–558.

Liang, X., Correia, G. H., & van Arem, B. (2017). An optimization model for vehicle routing of automated taxi trips with dynamic travel times. *Transportation Research Procedia*, *27*, 736–743.

Liu, J., Kockelman, K. M., Boesch, P. M., & Ciari, F. (2017). Tracking a system of shared autonomous vehicles across the Austin, Texas network using agent-based simulation. *Transportation*, *44*(6), 1261–1278.

Lodi, A., & Zarpellon, G. (2017). On learning and branching: A survey. *Top*, *25*(2), 207–236.

Lourenço, H. R., Martin, O. C., & Stützle, T. (2003). Iterated local search. *Handbook of metaheuristics* (320–353). Springer.

Madsen, O. B. G. (1988). An application of travelling-salesman routines to solve pattern-allocation problems in the glass industry. *Journal of the Operational Research Society*, *39*(3), 249–256.

Malandraki, C., & Daskin, M. S. (1992). Time dependent vehicle routing problems: Formulations, properties and heuristic algorithms. *Transportation Science*, *26*(3), 185–200.

Malek, M., Guruswamy, M., Pandya, M., & Owens, H. (1989). Serial and parallel simulated annealing and tabu search algorithms for the traveling salesman problem. *Annals of Operations Research*, *21*(1), 59–84.

Marler, R. T., & Arora, J. S. (2010). The weighted sum method for multi-objective optimization: New insights. *Structural and Multidisciplinary Optimization*, *41*(6), 853–862.

Martinez, L. M., Correia, G. H., & Viegas, J. M. (2015). An agent-based simulation model to assess the impacts of introducing a shared-taxi system: An application to Lisbon (Portugal). *Journal of Advanced Transportation*, *49*(3), 475–495.

Masson, R., Lehuédé, F., & Péton, O. (2013). An adaptive large neighborhood search for the pickup and delivery problem with transfers. *Transportation Science*, *47*(3), 344–355.

Masson, R., Lehuédé, F., & Péton, O. (2014). The dial-a-ride problem with transfers. *Computers & Operations Research*, *41*, 12–23.

Mayer, T., Uhlig, T., & Rose, O. (2018). Simulation-based autonomous algorithm selection for dynamic vehicle routing problems with the help of supervised learning methods. *2018 Winter Simulation Conference (WSC)*, 3001–3012.

Miranda, D. M., de Camargo, R. S., Conceição, S. V., Porto, M. F., & Nunes, N. T. (2021). A metaheuristic for the rural school bus routing problem with bell adjustment. *Expert Systems with Applications*, *180*.

Mitchell, T. (1997). Machine learning. *McGraw Hill Burr Ridge*.

Mokhtari, N., & Ghezavati, V. (2018). Integration of efficient multi-objective ant-colony and a heuristic method to solve a novel multi-objective mixed load school bus routing model. *Applied Soft Computing*, *68*, 92–109.

Molenbruch, Y., Braekers, K., & Caris, A. (2017). Typology and literature review for dial-a-ride problems. *Annals of Operations Research*, *259*(1), 295–325.

Nazari, M., Oroojlooy, A., Takáč, M., & Snyder, L. V. (2018). Reinforcement learning for solving the vehicle routing problem. *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, 9861–9871.

Newton, R. M., & Thomas, W. H. (1969). Design of school bus routes by computer. *Socio-Economic Planning Sciences*, *3*(1), 75–85.

O'Shea, K., & Nash, R. (2015). An introduction to convolutional neural networks. *arXiv preprint arXiv:1511.08458*.

Pacheco, J., Caballero, R., Laguna, M., & Molina, J. (2013). Bi-objective bus routing: an application to school buses in rural areas. *Transportation Science*, *47*(3), 397–411.

Paquete, L., & Stützle, T. (2003). A two-phase local search for the biobjective traveling salesman problem. *International Conference on Evolutionary Multi-Criterion Optimization*, 479–493.

Park, J., Tae, H., & Kim, B. (2009). The effects of allowing mixed loads in the commuter bus routing problem. *Proceedings of the 10th Asia Pacific Industrial Engineering and Management Systems Conference*, 14–21.

Parvasi, S. P., Mahmoodjanloo, M., & Setak, M. (2017). A bi-level school bus routing problem with bus stops selection and possibility of demand outsourcing. *Applied Soft Computing*, *61*, 222–238.

Peng, Z., Al Chami, Z., Manier, H., & Manier, M. A. (2019). A hybrid particle swarm optimization for the selective pickup and delivery problem with transfers. *Engineering Applications of Artificial Intelligence*, *85*, 99–111.

Petersen, H. L., & Ropke, S. (2011). The pickup and delivery problem with cross-docking opportunity. *Computational Logistics*, 101–113.

Pisinger, D. (2005). Where are the hard knapsack problems? *Computers & Operations Research*, *32*(9), 2271–2284.

Poole, D., Mackworth, A., & Goebel, R. (1998). Computational intelligence. *Oxford University Press*.

Posada, M., Andersson, H., & Häll, C. H. (2017). The integrated dial-a-ride problem with timetabled fixed route service. *Public Transport*, *9*(1), 217–241.

Prah, K., Keshavarzsaleh, A., Kramberger, T., Jereb, B., & Dragan, D. (2018). Optimal bus stops' allocation: A school bus routing problem with respect to terrain elevation. *Logistics & Sustainable Transport*, *9*, 1–15.

Prouvost, A., Dumouchelle, J., Scavuzzo, L., Gasse, M., Chételat, D., & Lodi, A. (2020). Ecole: A gym-like library for machine learning in combinatorial optimization solvers. *arXiv preprint arXiv:2011.06069*.

Rais, A., Alvelos, F., & Carvalho, M. (2014). New mixed integer-programming model for the pickup-and-delivery problem with transshipment. *European Journal of Operational Research*, *235*(3), 530–539.

Reinhardt, L. B., Clausen, T., & Pisinger, D. (2013). Synchronized dial-a-ride transportation of disabled passengers at airports. *European Journal of Operational Research*, *225*(1), 106–117.

Ribeiro, G. M., & Laporte, G. (2012). An adaptive large neighborhood search heuristic for the cumulative capacitated vehicle routing problem. *Computers & Operations Research*, *39*(3), 728–735.

Rodrigue, J.-P., Comtois, C., & Slack, B. (2016). The geography of transport systems. *Routledge*.

Ropke, S., & Pisinger, D. (2006). An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation Science*, *40*(4), 455–472.

Salkin, H. M., & De Kluyver, C. A. (1975). The knapsack problem: A survey. *Naval Research Logistics Quarterly*, *22*(1), 127–144.

Schittekat, P., Kinable, J., Sörensen, K., Sevaux, M., Spieksma, F., & Springael, J. (2013). A metaheuristic for the school bus routing problem with bus stop selection. *European Journal of Operational Research*, *229*(2), 518–528.

Schmid, V., & Doerner, K. F. (2010). Ambulance location and relocation problems with time-dependent travel times. *European Journal of Operational Research*, *207*(3), 1293–1303.

Schrijver, A. (2003). Combinatorial optimization: Polyhedra and efficiency (Vol. 24). *Springer Science & Business Media*.

Shafahi, A., Wang, Z., & Haghani, A. (2018). Speedroute: Fast, efficient solutions for school bus routing problems. *Transportation Research Part B: Methodological*, *117*, 473–493.

Shaw, P. (1997). A new local search algorithm providing high quality solutions to vehicle routing problems. *APES Group, Dept of Computer Science, University of Strathclyde, Glasgow, Scotland, UK*.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. (2016). Mastering the game of Go with deep neural networks and tree search. *Nature*, *529*(7587), 484–489.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., et al. (2017). Mastering the game of Go without human knowledge. *Nature*, *550*(7676), 354–359.

Souza Lima, F. M., Pereira, D. S., Conceição, S. V., & Ramos Nunes, N. T. (2016). A mixed load capacitated rural school bus routing problem with heterogeneous fleet: Algorithms for the brazilian context. *Expert Systems with Applications*, *56*, 320–334.

Spieser, K., Treleaven, K., Zhang, R., Frazzoli, E., Morton, D., & Pavone, M. (2014). Toward a systematic approach to the design and evaluation of automated mobility-on-demand systems: A case study in Singapore. *Road Vehicle Automation* (229–245). Springer.

Sutton, R. S., & Barto, A. G. (2018). Reinforcement learning: An introduction. *MIT press*.

Thangiah, S. R., Fergany, A., & Awan, S. (2007). Real-time split-delivery pickup and delivery time window problems with transfers. *Central European Journal of Operations Research*, *15*(4), 329–349.

Torabi, F., Warnell, G., & Stone, P. (2018). Behavioral cloning from observation. *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, 4950–4957.

Toth, P., & Vigo, D. (2002). The vehicle routing problem. *SIAM*.

Toth, P., & Vigo, D. (2003). The granular tabu search and its application to the vehicle-routing problem. *Informs Journal on Computing*, *15*(4), 333–346.

van den Berg, P. L., & van Essen, J. T. (2019). Scheduling non-urgent patient transportation while maximizing emergency coverage. *Transportation Science*, *53*(2), 492–509.

van den Bergh, J., Quttineh, N.-H., Larsson, T., & Beliën, J. (2016). A time-indexed generalized vehicle routing model for military aircraft mission planning. *Operations Research Proceedings 2014*, 605–611.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. *Advances in Neural Information Processing Systems*, *30*.

Vithayathil Varghese, N., & Mahmoud, Q. H. (2020). A survey of multi-task deep reinforcement learning. *Electronics*, *9*(9), 1363.

Voudouris, C., & Tsang, E. (1999). Guided local search and its application to the traveling salesman problem. *European Journal of Operational Research*, *113*(2), 469–499.

Wang, Z., & Haghani, A. (2020). Column generation-based stochastic school bell time and bus scheduling optimization. *European Journal of Operational Research*, *286*(3), 1087–1102.

Watkins, C., & Dayan, P. (1992). Q-learning. *Machine Learning*, *8*(3-4), 279–292.

Whitley, L. D., Starkweather, T., & Fuquay, D. (1989). Scheduling problems and traveling salesmen: The genetic edge recombination operator. *3rd International Conference on Genetic Algorithms*, *89*, 133–40.

Zajac, S., & Huber, S. (2021). Objectives and methods in multi-objective routing problems: A survey and classification scheme. *European Journal of Operational Research*, *290*(1), 1–25.

Zhang, A., Kang, J. E., & Kwon, C. (2017). Incorporating demand dynamics in multi-period capacitated fast-charging location planning for electric vehicles [Green Urban Transportation]. *Transportation Research Part B: Methodological*, *103*, 5–29.

Zhang, J., He, T., Sra, S., & Jadbabaie, A. (2020). Why gradient clipping accelerates training: A theoretical justification for adaptivity. *International Conference on Learning Representations*.

Zhang, S., & Sutton, R. S. (2017). A deeper look at experience replay. *arXiv preprint arXiv:1712.01275*.

ACKNOWLEDGEMENTS

In the first graduate school course I attended, they defined PhD candidates as *mushrooms* because they are kept alone in dark rooms for months until supervisors check on them for harvest. My slightly tanned skin begs to differ!

I was not alone in a room, I was lucky to have an office in a building full of interesting and inspiring people. In addition to them, there were also Tom and Yuki. Yuki, I am grateful that you come to me for relationship advises, but you are not woke enough to realize that I am always single. Tom, I appreciate that you have let me in your life and it was a pleasure to be there for you when you needed me. For instance, I was there when Eline cheated on you. I was *there*. Thanks for tolerating all the chaos we were doing to all the others PhD candidates in our group: Qiaochu, Naqi, Remie, Esther, ιLara, Josse, Renfei, Merel and Willem. Thanks also to the optimization professors team, Anurag, David, Teun, Fernando, Krzysztof, the second nicest hair in the office Mark, de echte italiaans Nikolaas, Leo with his happy and contagious laugh, Sherlock Holmes Dion, and my lunchmaatje Jos (my last hope to learn Dutch is to attend your wife's classes).

I'm grateful for the chance to work with outstanding researchers such as Laura Pozzi, Maxi, Javier, our lord and saviour Wendelin and the barley doctor Lorenzo. Thanks also to all the people of LNMB for the mind-stimulating conversations. I am indebted to the support staff, thanks Dorothée, Joeffrey and Xi Wei and I am especially indebted to Kees, who always has the patience to explain me that I cannot solve all my Linux problems with *ls* and *cd*, no matter how hard I bruteforce it. A word of appreciation goes also to the ones who briefly shared their journey with us at TU Delft: thanks Etienne, Trevis, Berry, Guillermo and especially Luigi.

In addition to the people who helped me in the groundbreaking and revolutionary research that I surely conducted at TU Delft, I want to thank all the fioi with whom I shared dinners, nights and 'more, much 'more. Grazie di cuore to my boyfriend Edo, you are the most propositve person I know, Jack, the one who most desperately tries to copycat me perché le sai far tutte (sport, spettacolo, nazionali ed internazionali? Ciccio ti dico che le so far tutte!), l'asse Doelenplein-Spoorsingel, Giups -si sta una crema con te-, 'weekly pleasure' Billy, Spritz, my little sister and bronsa coerta Isa, her twin miss Laretta che mi charga up sempre, Dani with our bromance, the best roomie ever Irene, the worst roomie ever Benji, la mia futura cicala Roci (yo te voy a buitrear), la Franci, Benni (ma quanto son ganze le Grazian?), my second favourite Mexican Emily, Simo, Anna, Ale, Cate, Fabio, Carlos, Marcello, Alan, Frank, gli amici di una vita Davide, Mu, Samuel, Gloria, Cla and all the other wonderful people I can't think of right now.

Thanks to my close family, fratello Francesco, sorella Nicole e mamma Fiore per essere stata ed essere ancora la mia roccia (e adés che ta laùret piò, pota, al par de es in ferie). Thanks to my extended family as well, nonna, zie, zii, cugini vari e parenti acquisiti (troppi per nominarvi tutti, ma vi voglio bene lo stesso!). Voglio peró fare un ringraziamento speciale alla zia Monica per il supporto e per non farmi sentire solo anche quando

ballo -dignitosamente brillo- sui tavoli.

Finally, I want to state my outermost appreciation and gratitude for the two people who made my PhD happen. Thank you Karen so much for having trusted me back in 2017, when I sent you an email saying I was interested in doing a PhD with your group. I can now admit it. I had no idea what I was getting into, but I am so lucky it all worked out meravigliosamente. And it happens so perché sei una promotor fantastica e metti tutta te stessa in ció che fai. Theresia, I have told you one hundred times and now once more. You are way too good and kind. You made me feel gezellig thuis and you were there to support me even when I was slacking more than a non-binding inequality constraint. You two motivated me to put my energy into my work and you encouraged me to learn new skills and experiment with those. I will be eternally grateful to you for these four geweldige years. I can in all honesty say that you are two excellent supervisors.

To you all, grazie di cuore, é stato un viaggio incredibile! Per aspera ad astra

CURRICULUM VITÆ

Jacopo PIEROTTI

26-04-1993 Born in Bergamo, Italy.

EDUCATION

2012–2015 B. Automation Engineering
Politecnico di Milano

2015–2017 M. Control Engineering
Politecnico di Milano
Thesis: The EVRP-TW with heterogeneous recharging stations. An exact branch-and-price method.
Supervisors: Prof. Dr. F. Malucelli, Prof. Dr. G. Desaulniers, Dr. F. Errico

2018–2022 PhD Optimization
Delft University of Technology
Thesis: Models and Heuristics for Hard Routing and Knapsack Problems
Promotor: Prof. dr. ir. K.I. Aardal
Copromotor: Dr. ir. J.T. van Essen

LIST OF PUBLICATIONS

3. **Pierotti, J.**, Kronmüller, M., Alonso-Mora, J., van Essen, J. T. & Böhmer, W. (2021). "Reinforcement learning for the knapsack problem". *Optimization and Data Science: Trends and Applications, volume 6 of 5th AIROYoung Workshop and AIRO PhD School 2021, AIRO Springer Series. 1st edition*

2. **Pierotti, J.**, & van Essen, J. T. (2021). "MILP models for the Dial-a-ride problem with transfers". *EURO Journal on Transportation and Logistics, 10*.

1. **Pierotti, J.**, Ferretti, L., Pozzi, L. & van Essen, J. T. (2021). "Adaptive Iterated Local Search with Random Restarts for the Balanced Travelling Salesman Problem". *Metaheuristics for Combinatorial Optimization. Springer Nature*.