A Dispatching Algorithm with Application to Fleets of Shared Autonomous Vehicles

Master's thesis in Engineering Mathematics and Computational Sciences

Erik Hellsten

Department of Mathematical Sciences
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2017

Master's thesis 2017

A Dispatching Algorithm with Application to Fleets of Shared Autonomous Vehicles

ERIK HELLSTEN

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Mathematical Sciences
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2017

A dispatching algorithm with application to fleets of shared autonomous vehicles

Supervisors:
Robert Nilsson, Volvo Cars Corporation
Ann-Brith Strömberg, Department of Mathematical Sciences, Chalmers University of Technology

Examiner:
Ann-Brith Strömberg, Department of Mathematical Sciences, Chalmers University of Technology

Master's Thesis 2017
Department of Mathematical Sciences
Chalmers University of Technology
University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Image from Vecteezy.com

Typeset in LaTeX
Printed by the Department of Mathematical Sciences

Gothenburg, Sweden 2017

A dispatching algorithm with application to fleets of shared autonomous vehicles

Erik Hellsten
Department of Mathematical Sciences
Chalmers University of Technology

Abstract

Fleets of shared autonomous vehicles have been predicted to dominate the transport sector within a near future. For this to work efficiently—including the handling of spontaneous requests—the associated routing problems need to be modelled dynamically and solved efficiently. We formulate and model the problem of routing a fleet of shared autonomous vehicles over a period of time. For each vehicle and each moment in time, it must be decided which customers to serve and which routes to take. The resulting model is solved using a rolling horizon optimisation methodology together with an insertion heuristic for new requests. The optimisation problems resulting from the rolling horizon methodology are solved using column generation, where the subproblems, being elementary shortest path problems with side constraints, are solved using both a local-search heuristic and a dynamic programming algorithm. Our computational experiments show that real-world sized problem instances can be solved to near-optimality within a reasonable computing time.

Keywords: Optimisation, Dynamic routing, Dial-a-Ride, Column Generation, Rolling-Horizon

Acknowledgements

First I would like to thank Robert Nilsson, for having faith in me and supporting me, both in this project and in our earlier endeavours, always the source of positive energy. I would also like to thank Ann-Brith Strömberg, for advising me, for her meticulous corrections of my texts, and for taking time for me when I needed it the most. Without you two, this projects would not have been what it is.

I would also like to thank Volvo Cars Corporation for providing with me with a place to work as well many meaningful discussions about the practical viewpoints of this and similar projects.

Erik Hellsten, Gothenburg, Aug 2017

Table of Acronyms

Acronym	**Definition**
BFS	*Basic Feasible Solution*
CVRP	*Capacitated Vehicle Routing Problem*
CVRPPD	*Capacitated Vehicle Routing Problem with Pickup and Delivery*
DARP	*Dial-a-Ride Problem*
DDARP	*Dynamic Dial-a-Ride Problem*
DRMP	*Dual Restricted Master Problem*
ESPPTWRCPD	*Elementary Shortest Path Problem with Time Windows, Resource Constraints and Pickup and Delivery*
FDP	*Fleet Dispatching Problem*
ILP	*Integer Linear Programing*
IMP	*Integer Master Problem*
IRMP	*Integer Restricted Master Problem*
LNS	*Large Neighbourhood Search*
LP	*Linear Programming*
MP	*Master Problem*
PDP	*Pickup and Delivery Problem*
RMP	*Restricted Master Problem*
SAV	*Shared Autonomous Vehicle*
SDARP	*Stochastic Dial-a-Ride Problem*
SFDP	*Snapshot Fleet Dispatching Problem*
SP	*Subproblem*
TSP	*Travelling Salesman Problem*
VRP	*Vehicle Routing Problem*
VRPPD	*Vehicle Routing Problem with Pickup and Delivery*
VRPPDTW	*Vehicle Routing Problem with Pickup and Delivery and Time Windows*
VRPTW	*Vehicle Routing Problem with Time Windows*

Contents

1 Introduction **1**

2 Previous Work in Vehicle Routing **5**
2.1 Vehicle Routing Problems 5
2.2 Pickup and Delivery and Dial-a-Ride Problems 6
2.3 Dynamic Dial-a-Ride Problems 7
2.4 Fleets of Shared Autonomous Vehicles 8

3 Linear and Integer Linear Programming and Column Generation **9**
3.1 Linear Programming 9
3.1.1 The Simplex Method 10
3.2 Column Generation 12
3.2.1 Dantzig-Wolfe Decomposition 12
3.3 Integer and Mixed Integer Linear Programs 15
3.3.1 Branch-and-Bound 16
3.3.2 Tabu Search 16
3.4 Applying Column Generation to Integer Linear Problems 16
3.4.1 Branch-and-Price 17

4 Problem Formulation and Modelling **19**
4.1 The Fleet Dispatching Problem 19
4.2 Modelling the Fleet Dispatching Problem 20
4.3 The Snapshot Fleet Dispatching Problem 21

5 The Column Generation Scheme Applied to the Snapshot Fleet Dispatching Problems **27**
5.1 The Set Covering Formulation 27
5.1.1 The Integer Master problem 27
5.1.2 The Integer Restricted Master problem 29
5.1.3 The Dual Restricted Master Problem 29
5.1.4 The Subproblems 29
5.2 Solving the Snapshot Problem 31
5.2.1 Generating an Initial Set of Routes 31
5.2.2 Solving the Integer Restricted Master Problem and the Dual Restricted Master Problem 32
5.3 Solving the Sub Problems 32

5.3.1 A Local Search Heuristic 33
5.3.2 A Dynamic Programming Approach to Solve the Subproblem 34
5.3.3 A combination of the Local-Search Heuristic and the Labelling Algorithm 40
5.4 Post-Processing 40

6 Solving the Fleet Dispatching Problem 41
6.1 A Solution Algorithm for the Fleet Dispatching Problem 42
6.2 The Insertion Heuristic 44

7 Computational Experiments 45
7.1 Testing the Snapshot Fleet Dispatching Problem 45
7.1.1 Results for the Small Instances of the Snapshot Fleet Dispatching Problem 46
7.1.2 Results for the Large Instances of the Snapshot Problem . . . 47
7.2 Testing the Fleet Dispatching Problem 47
7.2.1 Results for the Fleet Dispatching Problem 49

8 Conclusion 55

9 Reflections and Outlook 57
9.1 The MATSim Heritage 57
9.2 Outlook 58

Bibliography 59

A Appendix I

1

Introduction

We live in a time when autonomous vehicles are soon to be released to the public. For example, do Volvo Cars plan on releasing their first autonomous cars in 2021. In addition to being a comfortable feature it has been predicted to have drastic impacts on future transport solutions. The major belief is that fleets of shared autonomous vehicles, SAVs, will become a dominant way of transportation in the local passenger transport sector (see [1, 2]). The concept is that instead of people having their own cars they request a car from the fleet whenever they want to go somewhere, much like a taxi system. For this to work efficiently, relevant models and customised algorithms are required, deciding for each car, at each moment, which route to take and which customers to serve.

The problem is similar to dispatching normal taxi fleets, with the exception that there are no drivers. This will affect the base cost of using a vehicle as there will be no salary to pay. It also differs in that such a fleet of SAVs may use so called ”car sharing”, in which a vehicle may serve multiple customers simultaneously. This has been shown to greatly reduce the number of vehicles required, (see [3]). Further, with better communication technology, more data could be sent to and handled by a central hub, thus increasing the potential for optimisation across the whole fleet. Additionally, the SAVs could be more adaptive, constantly changing their driving schedules when new information is revealed, something which might be tiring for regular human drivers. This results in that, while taxi fleets today tend to use manual dispatching or simple heuristic solutions, fleets of SAVs are better suited for more advanced optimisation schemes.

The field of route optimisation has been around for a long time. In 1959, Dantzig and Ramser [4] first described the *vehicle routing problem* (VRP) and since then the topic has been studied widely. The vast majority of the literature regards *static* VRPs, in which all information is known before the execution of the vehicle routes. Assuming that the problem of dispatching a fleet of SAVs could be considered *static*, it would be well described as a *dial-a-ride problem*, as presented by Cordeau [5]. As customers will request rides with short notice, the problem at hand cannot, however, be considered *static* and the literature regarding *dynamic dial-a-ride problems* is certainly more scarce.

In this thesis, the problem of routing a fleet of SAVs is presented. Due to the dynamic nature of the problem, it cannot be modelled as a classical optimisation problem. Instead, it is treated as a simulation model, for which the routes are updated with regular intervals. The new routes are created using a reoptimisation scheme, which optimises the routes for all vehicles, based on the currently available information. Additionally, an *insertion heuristic* is used to quickly integrate new requests into the routes. The reoptimisation is performed through solving an optimisation problem, which represents a routing of the vehicles in the simulation model. It is a rolling-horizon method, meaning that in each reoptimisation, the simulation is not optimised over its full time span, but only from the time of the reoptimisation and for a fixed period ahead. These problems are modelled as variations of *static dial-a-ride problems* and are solved using *column generation* (see [6]), which has been shown to yield good results for similar problems. The *column generation* method subdivides the problem into smaller single-vehicle problems, which are solved using customised solution methods. Both a *local-search heuristic* and a so called *labelling algorithm* are presented as solution methods for the single-vehicle problems, as well as a combination of the two.

To the best of our knowledge, this is the first time such a rolling horizon methodology is used for handling the dynamic aspects of this problem, in conjunction with column generation. It also differs from earlier work in that the problem is modelled using a graph representing an actual road network instead of just the start and end nodes of the customers. This has no real advantages in *static vehicle routing problems*, but in *dynamic vehicle routing problems*, vehicles may start driving towards a node and then change destination, when new information is revealed. In the classic *dial-a-ride problem*, no information is stored about the vehicles whereabouts, in-between visiting customer nodes. This makes it impossible to model such route changes. With the modelling employed in this thesis, a vehicle is only constrained to drive to the next intersection before it can change its route. This makes our formulation more flexible and adaptive for dynamic problem settings.

It has been developed, as will be presented later, a number of approaches in which new requests are anticipated, by assuming a random distribution for new requests or using data from earlier simulations. No such methods will, however, be used in this thesis. When using column generation, to guarantee the solution to be optimal, a *branch-and-price* method is usually utilised. In this work, as not all information is known and the reoptimisation problem needs to be solved many times, a lower computational complexity is more important than a guarantee of optimality; hence *branch-and-price* has not been implemented.

To test the solution methods proposed, a number of computational experiments are conducted and the results show that reasonably sized instances can be solved with good results in manageable computation times.

The remainder of this thesis is organised as follows. Chapter 2 is a literature review on VRPs. Chapter 3 briefly introduces the optimisation theory needed for the

methods used. Chapter 4 presents and formalises the problem of routing the SAVs and Chapter 5 describes the reoptimisation problems. In Chapter 6 the solution procedure for solving the problem of routing the fleet of SAVs is described. Chapter 7 presents tests and computational results for the developed solution method. In chapter 8 the conclusions are presented and then Chapter 9 contains a brief discussion of the results and possible future outlooks.

2

Previous Work in Vehicle Routing

This chapter presents a brief review of earlier work in the area of *vehicle routing problems*. It starts rather broad and then gradually focuses more on literature relevant for the problem at hand. Necessary nomenclature is presented as it appear. The aim is to both introduce a less informed reader to the general area and to help put this work in a context, as to better understand the environment in which it has been created. This chapter includes the most prominent publications in the respective areas as perceived by the author.

2.1 Vehicle Routing Problems

The idea of route optimisation is not new. The applications in which it is used range from train to freight ship routing (see [7, 8]). When it comes to transport problems, one usually talks about *vehicle routing problems* (VRP). The original VRP was first described by Dantzig and Ramser [4] in 1959. In short, it describes the problem in which a number of vehicles start at a common depot and are to deliver goods to a set of customers so that the total distance, driven by all the vehicles together, is minimised. As it is a generalisation of the *travelling salesman problem* (TSP), it is NP-hard (non-deterministic polynomial-time hard; see [9]), and so are almost all versions and variations of it.

As more and more applications of the VRP have arisen, so has different versions of it. The most common are the *capacitated vehicle routing problem* (CVRP), the *vehicle routing problem with time windows* (VRPTW) and the *vehicle routing problem with pickup and deliveries* (VRPPD). In the CVRP the vehicles has a limited capacity for carrying goods and can hence only deliver to a subset of the customers. In the VRPTW each delivery has a time window in which the delivery must be made. Lastly, in VRPPD, instead of delivering goods from the depot, the vehicles pick up goods at some nodes and then deliver the goods to other nodes. There are also combinations of these, such as the *vehicle routing problem with pickup and deliveries and time windows* (VRPPDTW), which is a VRPPD where the pick-ups and deliveries have to be made within certain time windows. A thorough descrip-

tion of the field of VRPs can be found in *Vehicle routing: problems, methods, and applications* by Toth and Vigo [10].

Further, a CVRPPD in which each vehicle has unit capacity and all goods take up unit space, such that each vehicle only can deliver one good at a time, is called a *stacker-crane problem* (see [11]).

2.2 Pickup and Delivery and Dial-a-Ride Problems

Pickup and delivery problems (PDPs) and *dial-a-ride problems* (DARPs) are similar in their structure. Both aim to route a set of vehicles to deliver goods from their origins to their destinations. A PDP is generally called a DARP when the goods to deliver are customers and the problem is concerned with customer satisfaction. This often means that DARPs have tighter time windows and/or tries to minimise the travelling and waiting times for the customers.

One of the first approaches to the DARP was made by Psaraftis [12, 13], who developed exact methods using dynamic programming (see [14]) for the single-vehicle case. An improved *labelling algorithm* was later presented by Desrosiers et al. [15]. For the multi-vehicle problems the methods are usually divided into exact solution methods and heuristic and meta-heuristic approaches. Due to the complexity of the problem the heuristic approaches are more prominent than the exact ones in the literature. For the exact approaches there are two main solution procedures, *branch-and-cut* and *branch-and-price*.

The first *branch-and-price* approach to the DARP was made by Dumas, Desrosiers and Soumis [16] who used the *labelling algorithm* by Desrosiers et al. [15]. Savelsbergh and Sol [17] presented another *branch-and-price* algorithm which differs significantly from the work in [15] in that [17] employed a heuristic solution for the subproblems. The heuristic contained an initial construction phase followed by a local-search improvement phase. Further contributions has been made by Xu et al. [18] and Sigurd, Pisinger and Sig [19]. In [18], a more practical problem, including for example government laws regulating the drivers working hours, was modelled. The subproblems were solved using a heuristic based on modifying and merging earlier generated columns. A dynamic programming was further employed to achieve a lower bound on the heuristic. In [19], a *branch-and-price* algorithm was applied to animal transport in Denmark; the model included precedence constraints, stating that animals cannot be transported in a vehicle, which has earlier transported unhealthy animals, unless that vehicle is cleaned first. The subproblems were solved both exactly, with dynamic programming, and using two different heuristics. In later years, Parragh [20] has presented a *hybrid-column generation* approach with *large-Neighbourhood search* (LNS). They alternated between using column generation and

LNS to generate new columns.

Branch-and-cut methods was successfully used by Cordeau [21] and Cordeau and Laporte [22] in 2006 and 2007 respectively. In 2009 the two approaches were used together in a *branch-and-cut-and-price* algorithm by Ropke and Cordeau [23].

One of the first heuristic solutions for the multi-vehicle DARP was presented by Jaw et al. [24] in which they used an insertion heuristic, sequentially deciding for each customer how to optimally insert that customer into the current schedule. Later, Toth and Vigo [25, 26] useed local search and tabu-thresholding to solve a real-life problem in Bologna and in Cordeau and Laporte [27] described a tabu-search algorithm. Ropke and Pisinger [28] presented an adaptive LNS. It was an remove-and-reinsert heuristic where the methods used for removing and reinserting requests were chosen adaptively based on their previous performance.

2.3 Dynamic Dial-a-Ride Problems

Dynamic dial-a-ride problems (DDARPs) are DARPs in which some or all information is revealed during the execution of the routes. This means that the solution needs to be updated during the execution of the routes or "online", as it is also called. The most common category of *dynamic dial-a-ride problems* is when not all requests are known at the execution start but are instead revealed gradually. The literature regarding *dynamic dial-a-ride problems* is certainly more scarce than for its *static* counterpart, but there have still been a number of successful approaches. A survey over *dynamic* PDPs was written by Berbeglia, Laporte and Cordeau [11] in 2010 and a more recent survey over *dynamic* VRPs was published by Psaraftis [29] in 2016.

The work by Psaraftis [12, 13], presented in the Section 2.2, also handled *dynamic* cases. It first solved the *static* problem, as described, and then used an insertion heuristic for new requests during the execution. Also Madsen et al. [30] used an insertion heuristic to handle new requests.

In some cases, some of the unknown information follows a known statistical distribution, which can then be used to predict unknown information. Such problems are called *stochastic dial-a-ride problems* (SDARPs). An example is the work by Swihart and Papastavrou [31], in which they assumed that incoming requests followed a Poisson distribution. When no such distribution exists, one may instead use information from previous simulations for the same purpose. This has been utilised by Van Hentenryck and Bent [32] in 2004.

In 2004 Mitrovic-Minic and Laporte [33] presented a dynamic VRPPD which they solved using a rolling horizon-based algorithm, in which new customers were inserted using an insertion heuristic and the problem was periodically reoptimised using

tabu-search. This resembles the methodology developed in this thesis. They also presented a number of waiting strategies. Mitrovic-Minic [34], later presented a model with a double horizon objective, where the *short horizon objective* aimed at minimising the route length and the *long horizon objective* aimed at keeping a slack schedule to avail new requests to be added.

Other types of methods, such as for example genetic algorithms, has also been used by for example Saez et al. [35], which used a genetic algorithm in combination with fuzzy clustering to solve a dynamic PDP. Han et al. [36] has further used deep neural networks to route a virtual taxi fleet in Singapore.

2.4 Fleets of Shared Autonomous Vehicles

In addition to the literature regarding DARPs and PDPs, some interesting literature connects more directly to the concept of SAVs. Fagnant and Kockelman [37, 38] has published two studies in which they use transport modelling to estimate the efficiency of SAVs. Their conclusion was that each SAV can replace around ten private cars if all private car-commuters would instead use the fleet.

Some publications use the transport simulation software MATSim [39] to implement different taxi dispatching algorithms. One example is the work by Maciejewski et al. [40].

3

Linear and Integer Linear Programming and Column Generation

This chapter aims to introduce the theory necessary to understand the models and methods used later in this thesis as well as providing a brief introduction to *mixed integer linear optimisation*. Section 3.1 presents *linear programming* as well as the simplex method. Section 3.2 introduces the *column generation* method for linear programs. Section 3.3 then introduces the integer and mixed integer linear problems along with different *branch-and-bound* procedures and *meta-heuristics* for their solutions. Lastly, Section 3.4 describes how the *column generation* methodology can be used to solve mixed integer problems with, for example, *branch-and-price* methods.

3.1 Linear Programming

Linear programming (LP) is a category of linear mathematical optimisation models, in which the objective and the constraints are all defined by linear functions, and corresponding solution techniques. Linear optimisation models have proved useful in modelling a variety of problems, for example in transport and in economics. It is widely used due to the numerous efficient solution methods developed, one of which the most common is the simplex method. Linear programs are optimisation problems which can be formulated as to

$$\underset{\mathbf{x}}{\text{minimise}} \quad \mathbf{c}^{\mathrm{T}}\mathbf{x}, \tag{3.1a}$$

$$\text{subject to} \quad \mathbf{A}\mathbf{x} = \mathbf{b}, \tag{3.1b}$$

$$\mathbf{x} \geq \mathbf{0}, \tag{3.1c}$$

where $\mathbf{x} \in \mathbb{R}^n$ are the decision variables and $\mathbf{c} \in \mathbb{R}^n$, $\mathbf{A} \in \mathbb{R}^{m \times n}$ and $\mathbf{b} \in \mathbb{R}^m$ are input parameters. For a further introduction to linear optimisation, see [41].

Linear optimisation has some interesting properties such that the solution space

is a convex polyhedron, which is the intersection of finitely many halfspaces by the constraints (3.1b)–(3.1c). It can further be shown that if the problem has an optimal solution, the optimal value will be found in at least one *extreme point* of the solution space.

3.1.1 The Simplex Method

The simplex method for LP utilises the fact that the solution can be found in at least one of the *extreme points* of the solution space. Theoretically one could just check the value of the objective in each *extreme point* and choose the lowest, but this is usually inefficient due to the vast amount of *extreme points*. Instead, the simplex method visits only a limited number of *extreme points*. For simplicity, for the rest of this section the feasible set will be assumed to be bounded.

Assume that the matrix $\mathbf{A}$ is of rank m. If this is not the case, the constraints are linearly dependent, which means that some constraints are redundant and can be removed. An *extreme point*, $\hat{\mathbf{x}} = (\hat{x}_1, \ldots, \hat{x}_n)$, of the polytope, defined by the constraints (3.1b)–(3.1c), is an element of the feasible region, such that the subset of columns of $\mathbf{A}$ corresponding to the non-zero elements in $\hat{\mathbf{x}}$ are linearly independent. Since $\mathbf{A}$ is of rank m, $\hat{\mathbf{x}}$ then includes a maximum of m non-zero elements. Let us call such an *extreme point* a *basic feasible solution* (BFS) and the non-zero variables for the *basic variables*. The remaining (i.e, zero-valued) variables in $\hat{\mathbf{x}}$ are further called the *non-basic variables*.

A BFS can be shown to be non-optimal if that BFS is part of an edge of the convex polytope and the objective value is strictly decreasing when moving along that edge away from the BFS. A geometric viewpoint of the method is that it starts at a BFS and then jumps to adjacent BFSs in the polytope until it can no longer find an edge along which the objective value decreases.

Now, how is this done in practice? Assume that a *BFS*, $\hat{\mathbf{x}}$, is known. One can then partition $\hat{\mathbf{x}}$ into the *basic variables* $\hat{\mathbf{x}}_B$ and the *non-basic variables* $\hat{\mathbf{x}}_N$, $\mathbf{c}$ into the corresponding $\mathbf{c}_B$ and $\mathbf{c}_N$ and $\mathbf{A}$ into the corresponding matrices $\mathbf{A}_B$ and $\mathbf{A}_N$ and rewrite equations (3.1) as:

$$\begin{aligned} \underset{\mathbf{x}_B, \mathbf{x}_N}{\text{minimise}} \quad & \mathbf{c}_B\mathbf{x}_B + \mathbf{c}_N\mathbf{x}_N, && \text{(3.2a)} \\ \text{subject to} \quad & \mathbf{A}_B\mathbf{x}_B + \mathbf{A}_N\mathbf{x}_N = \mathbf{b}, && \text{(3.2b)} \\ & \mathbf{x}_B, \mathbf{x}_N \geq \mathbf{0}, && \text{(3.2c)} \end{aligned}$$

In order to move from one BFS to an adjacent BFS one could introduce a *non-basic variable* into the set of *basic variables* and move one current *basic variable* to the set of *non-basic variables*. But to find out whether this operation is going to improve the

solution the model first needs to be reformulated in terms of the *non-basic variables*. From (3.2b) follows that

$$\mathbf{x}_B = \mathbf{A}_B^{-1}(\mathbf{b} - \mathbf{A}_N\mathbf{x}_N) \tag{3.3}$$

and inserting (3.3) in the objective function (3.2a) yields

$$\mathbf{c}^{\mathrm{T}}\mathbf{x} = \mathbf{c}_B^{\mathrm{T}}\mathbf{A}_B^{-1}\mathbf{b} + (\mathbf{c}_N^{\mathrm{T}} - \mathbf{c}_B^{\mathrm{T}}\mathbf{A}_B^{-1}\mathbf{A}_N)\mathbf{x}_N. \tag{3.4}$$

Further, clearly

$$(\mathbf{c}_B^{\mathrm{T}} - \mathbf{c}_B^{\mathrm{T}}\mathbf{A}_B^{-1}\mathbf{A}_B)\mathbf{x}_B = 0 \tag{3.5}$$

holds, so one can combine (3.5) and (3.4), getting

$$\mathbf{c}^{\mathrm{T}}\mathbf{x} = \mathbf{c}_B^{\mathrm{T}}\mathbf{A}_B^{-1}\mathbf{b} + (\mathbf{c}^{\mathrm{T}} - \mathbf{c}_B^{\mathrm{T}}\mathbf{A}_B^{-1}\mathbf{A})\mathbf{x}. \tag{3.6}$$

This gives that for a variable x_j in the set of *non-basic variables*, the objective value will decrease, and hence the solution will improve, when x_j is introduced into the set of *basic variables* if and only if it holds that

$$c_j^{\mathrm{T}} - \mathbf{c}_B^{\mathrm{T}}\mathbf{A}_B^{-1}\mathbf{A}_j < 0. \tag{3.7}$$

The value $c_j^{\mathrm{T}} - \mathbf{c}_B^{\mathrm{T}}\mathbf{A}_B^{-1}\mathbf{A}_j$ is called the *reduced cost* of the variable x_j. The variable moved from the set of *basic variables* to the set of *non-basic varibles* is called the *leaving variable*. To keep the solution feasible the *leaving variable* needs to be chosen according to:

$$x_k \in \arg\min_{x_i \in \mathbf{x}_B} \left(\frac{(\mathbf{A}_B^{-1}\mathbf{b})_i}{(\mathbf{A}_B^{-1}(\mathbf{A}_N)_j)_i} \middle| (\mathbf{A}_B^{-1}(\mathbf{A}_N)_j)_i > 0 \right). \tag{3.8}$$

Else, if another *basic variable* x_k is chosen as the leaving variable, some *basic variables* will possess a negative value after the inclusion of the new *non-basic variable*, and the solution renders infeasible. Further, an *extreme point* in which no variable has a negative *reduced cost* is an optimal solution to the problem.

Statement. A BFS in a linear optimisation problem, such that the *reduced cost* of each variable is non-negative, is an optimal solution for that problem, i.e. it exists no other feasible solution with lower objective value.

Proof. Let $\mathbf{x}^*$ be a BFS such that the *reduced cost* of each variable is non-negative, i.e.

$$c^{\mathrm{T}} \geq \mathbf{c}_B^{\mathrm{T}}\mathbf{A}_B^{-1}\mathbf{A} \tag{3.9}$$

holds. The equivalences (3.4) state that the cost of $\mathbf{x}^*$ is $\mathbf{c}_B\mathbf{A}_B^{-1}\mathbf{b}$ as $\mathbf{x}_N = \mathbf{0}$. Now let $\mathbf{y} \in \mathbb{R}^n$ be another BFS. As $\mathbf{y}$ is feasible, $\mathbf{A}\mathbf{y} = \mathbf{b}$ and $\mathbf{y} \geq \mathbf{0}$ hold from (3.1b) and (3.1c), respectively. This, together with (3.1a) and (3.9), yields that the cost of $\mathbf{y}$ is

$$\mathbf{c}^{\mathrm{T}}\mathbf{y} \geq \mathbf{c}_B^{\mathrm{T}}\mathbf{A}_B^{-1}\mathbf{A}\mathbf{y} = \mathbf{c}_B^{\mathrm{T}}\mathbf{A}_B^{-1}\mathbf{b} = \mathbf{c}^{\mathrm{T}}\mathbf{x}. \tag{3.10}$$

Hence the cost of any other basic solution is at least as high as the cost for $\mathbf{x}^*$ and $\mathbf{x}^*$ is an optimal solution. □

For a more detailed introduction to the simplex method, see [42].

3.2 Column Generation

Some integer linear optimisation problems tend to have a huge number of variables. When, in the simplex method, a new *non-basic variable* is to be introduced into the set of *basic variables* it could be computationally intractable to search through *all non-basic variables*. Gilmore and Gomory [43] came up with a procedure in which new variables (or columns) are generated when they are to be introduced into the set of *basic variables*. This is done by solving a so called *subproblem*. In this section the Dantzig-Wolfe decomposition method will be introduced which decomposes general linear optimisation problems, to a form which is often more suitable for *column generation*, before it applies the *column generation* method. Later, in Section 4 the *column generation* method will be shown to be useful for solving *mixed integer linear problems* as well.

3.2.1 Dantzig-Wolfe Decomposition

Decomposing problems into a sequence of simpler/smaller problems is one of the classic methods to solve optimisation problems. Dantzig-Wolfe decomposition is one common technique where the problem is decomposed into a *master problem* and one or several *sub problems*. The idea is to initially only consider a few variables for the *master problem* and then through *column generation* generate more variables by solving the *subproblem*. But first the problem is partially reformulated into an *inner representation* as is described below. Consider the LP to

$$\begin{aligned} \underset{\mathbf{x}}{\text{minimise}} \quad & \mathbf{c}^{\mathrm{T}}\mathbf{x}, && \text{(3.11a)} \\ \text{subject to} \quad & \mathbf{A}_1\mathbf{x} = \mathbf{b}_2 && \text{(3.11b)} \\ & \mathbf{A}_2\mathbf{x} = \mathbf{b}_2 && \text{(3.11c)} \\ & \mathbf{x} \geq \mathbf{0}. && \text{(3.11d)} \end{aligned}$$

Which is a reformulation of the LP (3.1) with $\mathbf{A} = \begin{bmatrix} \mathbf{A}_1 \\ \mathbf{A}_2 \end{bmatrix}$ and $\mathbf{b} = \begin{bmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \end{bmatrix}$. Now this can be written as

$$\begin{aligned} \underset{\mathbf{x}}{\text{minimise}} \quad & \mathbf{c}^{\mathrm{T}}\mathbf{x}, && \text{(3.12a)} \\ \text{subject to} \quad & \mathbf{A}_1\mathbf{x} = \mathbf{b}_1 && \text{(3.12b)} \end{aligned}$$

with the additional constraints

$$\mathbf{A}_2\mathbf{x} = \mathbf{b}_2 \tag{3.12c}$$
$$\mathbf{x} \geq \mathbf{0}. \tag{3.12d}$$

(3.12b) are the so called "complicating" constraints and (3.12c)–(3.12d) are the so called "simple" constraints. The set of variables fulfilling the "simple" constraints make up a convex polyhedron $X = \{\mathbf{x} \in \mathbb{R}^n : \mathbf{A}_2\mathbf{x} = \mathbf{b}_2,\ \mathbf{x} \geq \mathbf{0}\}$. By the representation theorem for convex polyhedra by Minkowski and Weyl [44], any point $\hat{\mathbf{x}}$ in the polyhedron X can be expressed as convex combination of its *extreme points* ($\mathbf{u}^i \in \Omega$) and a positive linear combination of its *extreme rays* ($\mathbf{v}^j \in \Lambda$).

$$\hat{\mathbf{x}} = \sum_{i=1}^{|\Omega|} \lambda_i \mathbf{u}^i + \sum_{j=1}^{|\Lambda|} \mu_j \mathbf{v}^j, \tag{3.13a}$$
$$\sum_{i=1}^{|\Omega|} \lambda_i = 1, \tag{3.13b}$$
$$\lambda_i \geq 0, \qquad i = 1, \ldots, |\Omega|, \tag{3.13c}$$
$$\mu_j \geq 0, \qquad j = 1, \ldots, |\Lambda|. \tag{3.13d}$$

Then the LP (3.1) can be rewritten as

$$\underset{\lambda_i, \mu_j}{\text{minimise}} \quad \mathbf{c}^{\mathrm{T}} \left(\sum_{i=1}^{|\Omega|} \lambda_i \mathbf{u}^i + \sum_{j=1}^{|\Lambda|} \mu_j \mathbf{v}^j \right), \tag{3.14a}$$
$$\text{subject to} \quad \mathbf{A}_1 \left(\sum_{i=1}^{|\Omega|} \lambda_i \mathbf{u}^i + \sum_{j=1}^{|\Lambda|} \mu_j \mathbf{v}^j \right) = \mathbf{b}_1 \tag{3.14b}$$
$$\sum_{i=1}^{M} \lambda_i = 1 \tag{3.14c}$$
$$\lambda_i \geq 0, \quad i = 1, \ldots, |\Omega| \tag{3.14d}$$
$$\mu_j \geq 0, \quad j = 1, \ldots, |\Lambda|. \tag{3.14e}$$

If the variables further are separable by the constraints (3.12c)–(3.12d), i.e. the variables $\mathbf{x}$ can be partitioned into p sets such that each of these constraints only affects variables in one of the sets, the problem can be further decomposed. Then one can partition $\mathbf{x}$ into $(\mathbf{x}_1, \ldots, \mathbf{x}_p)$, $\mathbf{x}_k \in \mathbb{R}^{n_k}$, where n_k is the dimension of $\mathbf{x}_k$, and write the LP (3.1) as to

$$\underset{\mathbf{x}}{\text{minimise}} \quad \sum_{k=1}^{p} \mathbf{c}_k^{\mathrm{T}} \mathbf{x}_k, \tag{3.15a}$$
$$\text{subject to} \quad \sum_{k=1}^{p} \mathbf{D}_k \mathbf{x}_k = \mathbf{d} \tag{3.15b}$$
$$\mathbf{F}_k \mathbf{x}_k = \mathbf{b}_k, \quad k = 1, \ldots, p \tag{3.15c}$$
$$\mathbf{x}_1, .., \mathbf{x}_p \geq 0. \tag{3.15d}$$

Here (3.15b), (3.15c) and (3.15d) are reformulations of (3.11b),(3.11c) and (3.11d), respectively. The relation between the matrices $\mathbf{D}_k$, $\mathbf{F}_k$ and the original matrix $\mathbf{A}$ is illustrated in figure 3.1. Let Ω_k and Λ_k be the *extreme points* and *extreme rays*, respectively, for the polyhedron defined by the constraints (3.15c)–(3.15d), which affects $\mathbf{x}_k$, i.e. $X_k = \{\mathbf{x}_k \in \mathbb{R}^{n_k} : \mathbf{F}_k\mathbf{x}_k = \mathbf{b}_k, \mathbf{x}_k \geq \mathbf{0}\}$. Then $\mathbf{x}_k$ can be expressed as a convex combination of Ω_k and a positive linear combination Λ_k. This finally yields the LP to

$$\underset{\lambda_{ki},\mu_{kj}}{\text{minimise}} \quad \sum_{k=1}^{p} \mathbf{c}_k \left(\sum_{i=1}^{|\Omega_k|} \lambda_{ki}\mathbf{u}_k^i + \sum_{j=1}^{|\Lambda_k|} \mu_{kj}\mathbf{v}_k^j \right) \tag{3.16a}$$

$$\text{subject to} \quad \sum_{k=1}^{p} \mathbf{D}_k \left(\sum_{i=1}^{|\Omega_k|} \lambda_{ki}\mathbf{u}_k^i + \sum_{j=1}^{|\Lambda_k|} \mu_{kj}\mathbf{v}_k^j \right) = \mathbf{d} \tag{3.16b}$$

$$\sum_{i=1}^{|\Omega_k|} \lambda_{ki} = 1, \quad k = 1, \ldots, p \tag{3.16c}$$

$$\lambda_{ki} \geq 0, \quad i = 1, \ldots, |\Omega_k|, k = 1, \ldots, p \tag{3.16d}$$

$$\mu_{kj} \geq 0, \quad j = 1, \ldots, |\Lambda_k|, k = 1, \ldots, p, \tag{3.16e}$$

where λ_{ki} are the weights for $\mathbf{u}_k^i \in \Omega_k$ and μ_{kj} are the weights for $\mathbf{v}_k^j \in \Lambda_k$. Now, with λ_{ki} and μ_{kj} as the new decision variables, this is the so called *inner representation*. There are fewer constraints, in (3.16), than in the original formulation (3.11), but usually a much larger number of variables. Let us for the remainder of this section assume that we are working with a bounded LP, meaning that there are no *extreme rays*, i.e $\Lambda_k = \varnothing, k = 1, \ldots, p$.

Even though the model (3.16) might have a very large number of variables, usually only a few are needed for the representation of the optimal solution. This is what is utilised in the *column generation* method. Instead of optimising over all variables, one starts with a subset of the variables and then iteratively adds variables, which can be proven to improve the current solution, to the problem. The problem (3.16) is generally called the *master problem*, and a *master problem* including only a subset of the variables is called the *restricted master problem*. Denote by $\tilde{\Omega}_k \subset \Omega_k$ for $k = 1, \ldots, p$, subsets of the variable spaces. Then the *restricted master problem* can be expressed as to

$$\underset{\lambda_{ki}}{\text{minimise}} \quad \sum_{k=1}^{p} \sum_{i=1}^{|\tilde{\Omega}_k|} \mathbf{c}_k^{\mathrm{T}}\mathbf{u}_k^i\lambda_{ki}, \tag{3.17a}$$

$$\text{subject to} \quad \sum_{k=1}^{p} \sum_{i=1}^{|\tilde{\Omega}_k|} \mathbf{D}_k\mathbf{u}_k^i\lambda_{ki} = \mathbf{d} \tag{3.17b}$$

$$\sum_{i=1}^{|\tilde{\Omega}_k|} \lambda_{ki} = 1, \qquad k = 1, \ldots, p \tag{3.17c}$$

$$\lambda_{ki} \geq 0, \;\; i = 1, \ldots, |\tilde{\Omega}_k|, \;\; k = 1, \ldots, p. \tag{3.17d}$$

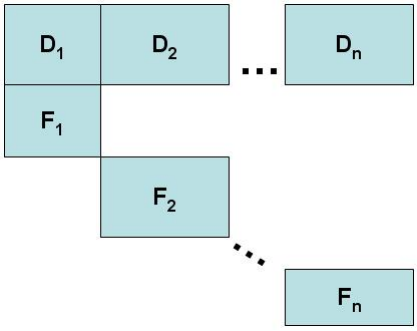


Figure 3.1: Structure of the constraint matrix for an LP where the variables are separable by the "simple" constraints. The D-matrices represent the "complicating" constraints and the F-matrices represent "simple" constraints.

Clearly, an optimal solution to the *reduced master problem* does not need to be optimal for the full *master problem*. But from the theory of the simplex method it is known that a new variable would only enter and improve the solution if it has a negative *reduced cost* in connection with the current solution of the *reduced master problem*. The procedure is then to generate new variables $\mathbf{u}_k \in \Omega_k \setminus \tilde{\Omega}_k, k = 1, \ldots, p$ with negative *reduced cost* and introduce them into the sets $\tilde{\Omega}_k$.

Let $\bar{\pi}$ and $\bar{\gamma}_k$ be the dual variables corresponding to the constraints (3.17b) and (3.17c) respectively for the optimal dual solution to (3.17). The *reduced cost* for a new variable $\mathbf{u}_k$ will then be $(\mathbf{c}_k^{\mathrm{T}} - \pi^{\mathrm{T}}\mathbf{D})\mathbf{u}_k - \gamma_k$. The new variables are generally generated by solving the so called *subproblems* to

$$\underset{\mathbf{u}^k \in \Omega_k \setminus \tilde{\Omega}_k}{\text{minimise}} \left(\mathbf{c}_k^{\mathrm{T}} - \bar{\pi}^{\mathrm{T}}\mathbf{D}\right)\mathbf{u}_k - \bar{\gamma}_k. \tag{3.18}$$

The constraints in the optimisation problem (3.18) comes from that u_k must be included in the set Ω_k.

3.3 Integer and Mixed Integer Linear Programs

Integer Linear Programming (ILP) denotes LP where some or all variables also are restricted to take integer values. Most ILP's that appear in practice are computationally infeasible to solve by brute force. Instead more sophisticated approaches has evolved where the most common exact solution algorithm is the so called *branch-and-bound method*. For some large instances, finding and verifying an optimal solution might be unreasonably time-consuming and one could instead use meta-heuristics like *tabu search* or *simulated annealing*. Meta-heuristics do not guarantee to find the optimal solution but are in general less time-consuming than exact methods. In this section the *branch-and-bound* and *tabu search* methods are briefly introduced.

3.3.1 Branch-and-Bound

As searching the full solution space is computationally intractable, *branch-and-bound* methods try to prune the search space of solutions which can be proven not to be optimal. The idea is to split the problem into smaller and smaller sub problems in a tree structure, called branching, and to remove branches whenever possible, which constitutes the bounding. The branching works such that a problem is split into several smaller problems by fixing an integer variable to take only a subset of its possible values in each sub problem. If no bounding is done the procedure will result in that all candidates are checked in a bruteforce solution. The bounding works such that the best value found so far is stored and whenever a branch could be proven unable to produce a better result than the best solution already found, it does not need further exploring and can be pruned. This can be done by finding a lower bound on the optimal value for that branch by, for example, using a continuous relaxation. For a more thorough introduction to *branch-and-bound* methods, see [45].

3.3.2 Tabu Search

Tabu search is a meta-heuristic solution approach to solve integer optimisation problems. It uses local search methods or neighbourhood search to find new solutions but keeps a list of solutions, or traits for solutions, which are tabu and hence disallowed for the search. The tabu list aims to prevent the search to return to solutions that have been previously visited in order to receive a wider spread of solutions searched. More information about *tabu search* and other meta-heuristics can be found in [46].

3.4 Applying Column Generation to Integer Linear Problems

To use the method *column generation*, technically, the optimisation problem needs to be linear, as it relies on LP the simplex method. There are, however, ways to use *column generation* to get both approximate and optimal solutions to ILPs as well. One can for example solve a continuous relaxation of the problem using *column generation* and then solve the original problem using only the columns generated from solving the relaxation. This restricted problem includes significantly fewer variables than the original one, but there are no guarantees that the restricted problem will have the same optimal value as the original one. It has been shown to provide good results and for some problems it can be used to find good solutions quickly. The bounds of the objective function from Dantzig-Wolfe decomposition are the same as the dual bounds (see [47]) and better or equal than or equal to the bounds received from solving a continuous relaxation of the problem. If one wants to find an optimal solution one could use the solution to the relaxed problem as a

lower bound in a method similar to a *branch-and-bound* procedure.

3.4.1 Branch-and-Price

The *branch-and-price* method is a combination of *column generation* and *branch-and-bound*. It uses *column generation*, as described in Section 4.2, on the LP-relaxed *master problem* until an optimal solution is found. Then, if there are any variables which are non-integer it branches as in a *branch-and-bound* method. Several different branching rules can be used but it is most common, if a Dantzig-Wolfe decomposition is used, to branch on the variables of the original formulation. The columns created so far can generally be divided into two sets, one for each branch, so that the *column generation* does not need to start from scratch in each new branch. In each node of the branching tree, new columns are generated. The *column generation* then provides a lower bound, which can be used in the bounding part of the *branch-and-bound* method. A thorough description of the *branch-and-price* methodology can be found in [6].

4

Problem Formulation and Modelling

In this chapter the problem of routing a fleet SAVs is presented, and a modelling of it is proposed. Let us start by naming the problem of routing the fleet of SAVs the *fleet dispatching problem* (FDP).

4.1 The Fleet Dispatching Problem

The FDP, which is studied in this thesis, is the problem of a fleet of shared autonomous vehicles trying to serve a set of customers going about on their daily business. The general idea is that each customer requests a vehicle to pick him/her up around a given time, to bring him/her from his/her specific origin to his/her specific destination. As most people do not know long in advance when they wish to travel, most requests will be presented to the system just a short while before the journey's expected start. Further, the vehicles will be assumed to be identical and each vehicle will have a maximum capacity of simultaneous customers.

The problem is defined over a fixed time period, and practicalities, such as the need for refuelling and repair, are not taken into account. The vehicles are basically considered up and running at all times. This is must clearly be addressed before this model can be used in practice. It does, however, suffice to start understanding the basic properties of the problem.

The map is modelled as a graph and requests are required to start and end at nodes in the graph. The customers would in reality probably want to be picked up and delivered to positions away from those nodes, but this extra distance might be considered negligible for the larger problem, depending on how close the nodes are to each other in a particular instance.

The initial positions of the vehicles are usually at one or more common depots. But under the assumption that the vehicles just keep on running, the vehicles are, in

our definition of the problem, instead randomly distributed over the nodes, at the beginning of the time period over which the problem is defined. This distribution would further probably depend on a variety of factors, but for the scope of this problem, a uniform distribution will be considered.

Another major decision, when formulating this kind of a problem, is the optimisation criterion, as well as how to handle customer satisfaction. The most common approach is to minimise the total distance travelled while picking up and delivering customers within their respective given time windows, as is done by for example Cordeau [5]. The problem presented in this thesis differs by not explicitly minimising vehicle distance, but instead focusing on minimising waiting times and travelling times for the customers. Further, it will only have one-sided time windows on pick-ups, since customers cannot be picked up before their journeys' start time. This will lead to that some customers might need to wait long times, but will also most likely lead to a smaller average waiting time.

4.2 Modelling the Fleet Dispatching Problem

The goal of the FDP is to route a fleet K of m autonomous vehicles to serve a set of customers $\mathcal{U}$, during a time period, T. Let $\mathcal{G} = (\mathcal{V}, \mathcal{A})$ be a graph representing a road network, where each node $v \in \mathcal{V}$ represents an intersection in the road network. The intersections will also be the places where customers can start and end their journeys. Each customer $u \in \mathcal{U}$ has an origin node $o_u \in \mathcal{V}$ and a destination node $d_u \in \mathcal{V}$, a time s_u when it wants to be picked up and a time $s'_u \leq s_u$ when it alerts the fleet of its request. Let $\tau_{i,j}$ be the time it takes for a vehicle to go from node $i \in \mathcal{V}$ to node $j \in \mathcal{V}$. Let W_u be the waiting time for customer u and let H_u be the travelling time for customer u. The goal is to route the fleet as to minimise a linear combination of the waiting times and the travelling times for the customers, i.e to

$$\underset{H_u, W_u}{\text{minimise}} \sum_{u \in \mathcal{U}} (\alpha_1 H_u + \alpha_2 W_u), \tag{4.1}$$

where $\alpha_1, \alpha_2 > 0$. For customers not yet picked up at the end of the time period (at time T), W_u is set to $T - s_u$, and for customers picked up but not delivered, H_u is set to $T - w_u - s_u$. Further, no vehicle can carry more customers than its capacity C.

As not all information is known beforehand, it is impossible to model and solve the FDP as a static optimisation problem. Instead, the FDP will be treated more as a simulation model. It keeps track of the time in the problem and all the customers and vehicles during the execution of the problem. The vehicle routes are generated and updated during the execution, in two ways. The first is through a reoptimisation procedure, in which repeatedly, every T_{reopt} time units, the schedule is reoptimised for T_{horizon} time units forward in time, and with all requests received up until then. The problem, to reoptimise the schedule in this way, is denoted the *snapshot fleet*

dispatching problem (SFDP). The second way, to generate vehicle routes, is that in between the solution of the SFDPs, routes for newly received requests will be generated using an *insertion heuristic*. The FDP will, hence, receive routes this way, during the execution, and execute them.

4.3 The Snapshot Fleet Dispatching Problem

The *snapshot fleet dispatching problem* (SFDP) is defined as a static problem, i.e., it takes into account only requests that are known so-far. It further only optimises for T_{horizon} minutes forward in time. As the SFDP is a problem aiming to optimise the routes for a stage in the FDP, many of the sets and variables are the same. All vehicles $\mathcal{K}$ in the FDP are also in the SFDP and a subset $\tilde{\mathcal{U}}$, of all customers $\mathcal{U}$ in the FDP, are in the SFDP. The graph used in the SFDP, however, differs significantly from the one used in the FDP. While the nodes of the graph in the FDP represent intersections in the road network, the nodes in the graph of the SFDP represent, among other, customers' origins and destinations, similar to the structure of the one used in [5]. The graph in the SFDP, is easier to work with, and the nodes included in that representation is sufficient to model the SFDP, as it is static. The additional information in the graph for the FDP works better, however, for the dynamic FDP, as is discussed in Chapter 8.

Let $\tilde{\mathcal{G}} = \{\tilde{\mathcal{V}}, \tilde{\mathcal{A}}\}$ denote the directed graph used in the SFDP. Let $\tilde{\mathcal{V}}_{\text{orig}} = \{1, .., n\}$ and $\tilde{\mathcal{V}}_{\text{dest}} = \{n+1, .., 2n\}$ be ordered sets with the customers' origin and destination nodes respectively. Due to that the SFDP is defined as a problem aiming to optimse a period of time in the FDP, at the start of the SFDP each vehicle will be either on its way to, or at, a node in $\mathcal{G}$. Let $\tilde{\mathcal{V}}_{\text{vehicle}} = \{2n+1, .., 2n+m\}$ be an ordered set of nodes in $\tilde{\mathcal{G}}$ to be visited next by the repective vehicles in K. Some of the vehicles will already have picked up customers at the start of the SFDP. Let there be l already picked up customers and let $\tilde{\mathcal{V}}_{\text{init}} = \{2n+m+1, .., 2n+m+l\}$ be an ordered set with nodes representing the destinations of those customers. Further, since, in the SFDP, any vehicle may end its journay at any node, let $\tilde{\mathcal{V}}_{\text{end}} = 2n+m+l+1$ be a virtual node, reachable in zero time from all other nodes, representing the end of any vehicle route. Now, $\tilde{\mathcal{V}}$ denotes the union of named nodes, i.e. $\tilde{\mathcal{V}} = \tilde{\mathcal{V}}_{\text{orig}} \cup \tilde{\mathcal{V}}_{\text{dest}} \cup \tilde{\mathcal{V}}_{\text{init}} \cup \tilde{\mathcal{V}}_{\text{vehicle}} \cup \tilde{\mathcal{V}}_{\text{end}}$. Further, let $\tilde{\mathcal{V}}' = \tilde{\mathcal{V}}_{\text{orig}} \cup \tilde{\mathcal{V}}_{\text{dest}} \cup \tilde{\mathcal{V}}_{\text{init}}$. Let the set of directed edges be $\tilde{\mathcal{A}} = \{(i,j) | i \in \tilde{\mathcal{V}} \setminus \tilde{\mathcal{V}}_{\text{end}}, j \in \tilde{\mathcal{V}} \setminus \tilde{\mathcal{V}}_{\text{vehicle}}\}$ and let t_{ij} be the time it takes to traverse the edge (i,j), $(i,j) \in \tilde{\mathcal{A}}$. The values of t_{ij} can be calculated from τ_{ij} using for example Floyd-Warshall's algorithm (see [48]).

Let e_i be the time when the customer $u \in \tilde{\mathcal{U}}$, which origin node is $i \in \tilde{\mathcal{V}}_{\text{orig}}$, is available for pick-up. Let β_i^k be a binary variable taking the value 1 if the customer on its way to node i (at the start time of the SFDP) starts in vehicle k, and 0 otherwise. Let ξ_k be the time at which vehicle k reaches its first node in $\tilde{\mathcal{V}}_{\text{vehicle}}$, and let η_k be the initial load of vehicle k.

Now the decision variables are $x_{i,j}^k$, y_i, B_i^k, H_i^k and Q_i^k, where $x_{i,j}^k$ equals 1 if vehicle k uses the arc (i, j) in its route and 0 otherwise, y_i equals 1 if the customer with the destination node $i \in \tilde{\mathcal{V}}_{\text{dest}} \cup \tilde{\mathcal{V}}_{\text{init}}$ is not delivered by any vehicle and 0 otherwise, B_i^k is the arrival time of vehicle k at node i, H_i^k is the ride time for the customer arriving at its destination at node i in vehicle k and Q_i^k is the load of vehicle k immediately after visiting node i. There is a maximum capacity, C, in each vehicle, and a maximum route time, T_{horizon}. The parameter q_i denotes the change of load for each vehicle visiting node i and it will be -1 at the drop-off nodes, $\tilde{\mathcal{V}}_{\text{dest}}$ and $\tilde{\mathcal{V}}_{\text{init}}$, and +1 at the pick-up nodes, $\tilde{\mathcal{V}}_{\text{orig}}$. Lastly, let d_i be the cost of not serving the customer heading to node i. The goal is then to

Table 4.1: The quantities used in modelling the SFDPs. The waiting (picked-up) customers have not been picked up (have been picked up but not yet been dropped off) at the start time of the SFDP.

Ordered sets:

Quantity	Description	Size
$\tilde{\mathcal{V}}_{\text{orig}}$	Origin nodes for the *waiting customers*	n
$\tilde{\mathcal{V}}_{\text{dest}}$	Destination nodes for the *waiting customers*	n
$\tilde{\mathcal{V}}_{\text{init}}$	Destination nodes for the *picked up customers*	l
$\tilde{\mathcal{V}}_{\text{vehicle}}$	Start nodes for the vehicles	m
$\tilde{\mathcal{V}}_{\text{end}}$	End node the the vehicles	1

Parameters:

Quantity	Description	Size
t_{ij}	Travel time from node i to j	$\lvert\tilde{\mathcal{V}}\rvert \times \lvert\tilde{\mathcal{V}}\rvert$
η_k	Initial load of vehicle k	m
ξ_k	Arrival time at the start node for vehicle k	m
β_i^k	1 if the customer with destination i starts in vehicle k; 0 otherwise	$l \times m$
q_i	Load change at node i	$\lvert\tilde{\mathcal{V}}\rvert$
e_i	Start time for the customer with origin node i	n
T_{horizon}	Maximum route length for each vehicle	1
C	Vehicle capacity	1

Variables:

Quantity	Description	Size
B_i^k	Service time at node v_i by vehicle k	$\lvert\tilde{\mathcal{V}}\rvert \times m$
Q_i^k	Load of vehicle k just after visiting node i	$\lvert\tilde{\mathcal{V}}\rvert \times m$
H_i^k	Travel time for the customer with the destination i in vehicle k	$(n+l) \times m$
$x_{i,j}^k$	1 if vehicle k uses arc (i,j); 0 otherwise	$\lvert\tilde{\mathcal{V}}\rvert \times \lvert\tilde{\mathcal{V}}\rvert \times m$
y_i	0 if the customer with destination i is delivered; 1 otherwise	$n+l$

$$\underset{H_i^k, Q_i^k, B_i^k, y_i, x_i^k}{\text{minimise}}$$

$$\alpha_1 \sum_{i \in \tilde{\mathcal{V}}_{\text{dest}} \cup \tilde{\mathcal{V}}_{\text{init}}} \sum_{k=1}^{m} H_i^k + \alpha_2 \sum_{i \in \tilde{\mathcal{V}}_{\text{orig}}} \sum_{k=1}^{m} (B_i^k - e_i) + \sum_{i \in \tilde{\mathcal{V}}_{\text{dest}} \cup \tilde{\mathcal{V}}_{\text{init}}} d_i y_i, \tag{4.2a}$$

subject to

$$\sum_{j \in \tilde{\mathcal{V}}} \sum_{k=1}^{m} x_{i,j}^k + y_i = 1 \qquad i \in \tilde{\mathcal{V}}_{\text{dest}} \cup \tilde{\mathcal{V}}_{\text{init}} \tag{4.2b}$$

$$\sum_{j \in \tilde{\mathcal{V}} \setminus \tilde{\mathcal{V}}_{\text{vehicle}}} x_{2n+l+k,j}^k = 1 \qquad k = 1, \ldots, m \tag{4.2c}$$

$$\sum_{j \in \tilde{\mathcal{V}} \setminus \tilde{\mathcal{V}}_{\text{end}}} x_{j,2n+l+m+1}^k = 1 \qquad k = 1, \ldots, m \tag{4.2d}$$

$$\sum_{j \in \tilde{\mathcal{V}}} x_{j,i}^k - \sum_{j \in \tilde{\mathcal{V}}} x_{i,j}^k = 0 \qquad v_i \in \tilde{\mathcal{V}}', k = 1, \ldots, m \tag{4.2e}$$

$$\sum_{j \in \tilde{\mathcal{V}}} x_{i,j}^k - \sum_{j \in \tilde{\mathcal{V}}} x_{n+i,j}^k = 0 \qquad i \in \tilde{\mathcal{V}}_{\text{orig}}, k = 1, \ldots, m \tag{4.2f}$$

$$\beta_i^k - \sum_{j \in \tilde{\mathcal{V}}} x_{i,j}^k \geq 0 \qquad i \in \tilde{\mathcal{V}}_{\text{init}}, k = 1, \ldots, m \tag{4.2g}$$

$$B_{2n+k}^k \geq \xi_k \qquad k = 1, \ldots, m \tag{4.2h}$$

$$B_j^k - t_{i,j} + M(1 - x_{i,j}^k) \geq B_i^k \qquad k = 1, \ldots, m, \quad i \in \tilde{\mathcal{V}} \setminus \tilde{\mathcal{V}}_{\text{end}}, v_j \in \tilde{\mathcal{V}} \setminus \tilde{\mathcal{V}}_{\text{vehicle}} \tag{4.2i}$$

$$B_{2n+l+m+1}^k \leq T_{\text{horizon}} \qquad k = 1, \ldots, m \tag{4.2j}$$

$$B_i^k \geq e_i \qquad i \in \tilde{\mathcal{V}}_{\text{orig}}, k = 1, \ldots, m \tag{4.2k}$$

$$H_i^k = B_i^k - B_{i-n}^k \qquad i \in \tilde{\mathcal{V}}_{\text{dest}}, k = 1, \ldots, m \tag{4.2l}$$

$$H_i^k = B_i^k \qquad i \in \tilde{\mathcal{V}}_{\text{init}}, k = 1, \ldots, m \tag{4.2m}$$

$$H_i^k \geq 0 \qquad i \in \tilde{\mathcal{V}}_{\text{init}}, k = 1, \ldots, m \tag{4.2n}$$

$$Q_{2n+l+k}^k = \eta_k \qquad k = 1, \ldots, m \tag{4.2o}$$

$$Q_j^k - q_j + M(1 - x_{i,j}^k) = Q_i^k \qquad i, j \in \tilde{\mathcal{V}}', k = 1, \ldots, m \tag{4.2p}$$

$$Q_i^k \leq C \qquad i \in \tilde{\mathcal{V}}, k = 1, \ldots, m \tag{4.2q}$$

$$x_{i,j}^k \in \{0, 1\} \qquad i, j \in \tilde{\mathcal{V}}, k = 1, \ldots, m \tag{4.2r}$$

$$y_i \in \{0, 1\} \qquad i \in \tilde{\mathcal{V}}. \tag{4.2s}$$

Here, (4.2a) is the objective function aiming to minimise a weighted sum of the total travel time, the total waiting time and the number of unserved customers, weighted by the constants $\alpha_1 > 0$, $\alpha_2 > 0$ and $d_i > 0$. The constraints (4.2b) state that each customer is either served or marked as unserved ($y_i = 1$).The constraints (4.2c)–(4.2e) ensure the flow balance, where (4.2c) is noteworthy as, in contrast to most other DARPs, each car has its own starting node. The constraints (4.2f) ensure that each customer who is picked up is also dropped off by the same vehicle. The constraints (4.2g) say that a vehicle cannot drop off a customer starting in another

vehicle, which is similar to (4.2f) but for the customers starting in the vehicles, instead of the ones waiting to be picked up. The constraints (4.2h)–(4.2k) are the main time constraints stating when each car can start its route, when it has to end, that no user can be picked up before he/she is ready, as well as ensuring consistency of the time variables. (4.2l) and (4.2m) define the ride times for the customers and (4.2n) acts as the precedence constraint, meaning that a vehicle cannot drop off a customer before that customer has been picked up. The constraints (4.2o)–(4.2q) ensure consistency of the load variables, state the initial load and impose the maximum load. Lastly, (4.2r) and (4.2s) says that $x_{i,j}^k$ and y_i must be binary.

5

The Column Generation Scheme Applied to the Snapshot Fleet Dispatching Problems

The SFDPs, as formulated in Chapter 4, can be solved with different ILP methods, such as for example *branch-and-bound*. Similar problems have, however, been shown to be of high computational complexity. We present a reformulation using Dantzig-Wolfe decomposition together with a *column generation* method, which has been shown to be computationally efficient, for similar problems, in several earlier papers (see Section 2.2).

5.1 The Set Covering Formulation

The Dantzig-Wolfe decomposition, described in Section 3.2, results in a *master problem* (MP) and a set of *subproblems* (SPs). The MP, for the Dantzig-Wolfe decomposition of the SFDP, is a *set-covering problem* in which the variables are *routes* for the vehicles. The SPs aim to create *routes* which are feasible with regards to, for example, time and load. The Dantzig-Wolfe decomposition only works for linear problems, and the MP is, hence, the Dantzig-Wolfe decomposition of the continuous relaxation of the SFDP. Let us further introduce the *integer master problem* (IMP), which is the MP with reinstated integrality constraints. Clearly then, the MP will be the continuous relaxation of the IMP.

5.1.1 The Integer Master problem

In this reformulation, some new quantities are introduced. They are presented in 5.1. Let us first introduce the concept of a *route*. A *route*, r_k, for a vehicle k, is defined as an ordered set of nodes, which the vehicle k would then visit in order. Let $\mathcal{R}_k$ be the set of feasible *routes* for vehicle k and let λ_r^k be a binary decision

Table 5.1: New quantities for the column generation formulation.

Quantity	Description	Size	Type of quantity
$\mathcal{R}^k$	All feasible *routes* for vehicle k		Ordered set
$\tilde{\mathcal{R}}^k$	Generated feasible *routes* for vehicle k		Ordered set
$a^i_{k,r}$	1 if node v_i is in *route* $r \in \mathcal{R}_k$; 0 otherwise		Parameter
c^k_r	Cost of using *route* $r \in \mathcal{R}_k$		Parameter
d_i	Cost of not delivering the customer to v_i	$n+l$	Parameter
λ^k_r	1 if *route* $r \in \mathcal{R}_k$ is used; 0 otherwise		Variable
π_i	Dual variable for the master problem	$n+l$	Dual variable
γ_k	Dual variable for the master problem	m	Dual variable

variable stating if *route* $r \in \mathcal{R}_k$ is in the solution or not. Let further c^k_r be the corresponding cost of the *route* r_k and let $a^i_{k,r}$ be equal to 1 if node i is visited in *route* $r \in \mathcal{R}_k$ and 0 otherwise. In our Dantzig-Wolfe decomposition of the problem (4.2), the constraints (4.2c)–(4.2r) are considered the "simple" constraints and they separate over the vehicles, while (4.2b) and (4.2s) are considered the "complicating" constraints. A *route* r_k for a vehicle k is said to be feasible if it satisfies the "simple" constraints connected to that vehicle. In the IMP, λ^k_r are the decision variables and it is defined as to

$$\underset{\lambda^k_r, y_i}{\text{minimise}} \quad \sum_{k=1}^{m} \sum_{r \in \mathcal{R}_k} c^k_r \lambda^k_r + \sum_{i \in \tilde{\mathcal{V}}_{\text{dest}} \cup \tilde{\mathcal{V}}_{\text{init}}} d_i y_i \tag{5.1a}$$

$$\text{subject to} \quad \sum_{k=1}^{m} \sum_{r \in \mathcal{R}_k} a^i_{k,r} \lambda^k_r + y_i \geq 1 \qquad i \in \tilde{\mathcal{V}}_{\text{dest}} \cup \tilde{\mathcal{V}}_{\text{init}} \tag{5.1b}$$

$$\sum_{r \in \mathcal{R}_k} \lambda^k_r \leq 1 \qquad k = 1, \ldots, m \tag{5.1c}$$

$$\lambda^k_r \in \{0,1\} \quad r \in \mathcal{R}_k, \; k \in 1, \ldots, m \tag{5.1d}$$

$$y_i \in \{0,1\} \quad i \in \tilde{\mathcal{V}}_{\text{dest}} \cup \tilde{\mathcal{V}}_{\text{init}}, \tag{5.1e}$$

where (5.1a) is objective function, (5.1b) is equivalent to (4.2b) in the original formulation and (5.1c) states that each vehicle can use at most one *route*. Lastly, (5.1d) and (5.1e) states that the decision variables need to be integer. The IMP is a *set-covering problem*, and not a *set-partitioning problem* as the constraint (5.1b) is formulated with an inequality rather than with an equality, with *set-packing constraints* in (5.1c). Both formulations would be viable; the *set-partitioning formulation* actually better represents the real life problem, as no customer should be picked up by multiple vehicles. It is, however, harder to solve, as it is a more restricted problem, where the chosen routes need to fit together exactly, and hence the *set-covering formulation* is chosen and the problem of assigning multiple vehicles to one customer will be solved by a heuristic, as a post-processing step (see Section 5.4).

5.1.2 The Integer Restricted Master problem

To use the *column generation* method we first define the *integer restricted master problem* (IRMP). Let $\tilde{\mathcal{R}}_k \subset \mathcal{R}_k$, $k = 1, \ldots, m$, be subsets of all feasible *routes* for the vehicles. The RMP is a restriction of the MP, defined only over the *routes* in the subsets $\tilde{\mathcal{R}}_k$. The RMP is defined as to

$$\underset{\lambda_r^k, y_i}{\text{minimise}} \quad \sum_{k=1}^{m} \sum_{r \in \tilde{\mathcal{R}}_k} c_r^k \lambda_r^k + \sum_{i \in \tilde{\mathcal{V}}_{\text{dest}} \cup \tilde{\mathcal{V}}_{\text{init}}} d_i y_i, \tag{5.2a}$$

$$\text{subject to} \quad \sum_{k=1}^{m} \sum_{r \in \tilde{\mathcal{R}}_k} a_{k,r}^i \lambda_r^k + y_i \geq 1 \qquad i \in \tilde{\mathcal{V}}_{\text{dest}} \cup \tilde{\mathcal{V}}_{\text{init}} \tag{5.2b}$$

$$\sum_{r \in \tilde{\mathcal{R}}_k} \lambda_r^k \leq 1 \qquad k = 1, \ldots, m \tag{5.2c}$$

$$\lambda_r^k \in \{0, 1\} \quad r \in \tilde{\mathcal{R}}_k, \; k = 1, \ldots, m \tag{5.2d}$$

$$y_i \in \{0, 1\} \quad i \in \tilde{\mathcal{V}}_{\text{dest}} \cup \tilde{\mathcal{V}}_{\text{init}} \tag{5.2e}$$

Relaxing the integrality constraints on λ_r^k and y_i, yields the *restricted master problem* (RMP).

5.1.3 The Dual Restricted Master Problem

To find the *reduced cost* for new *route*s generated in the SPs, it is handy to use the optimal dual variables. Those are achieved by solving the *dual restricted master problem* (DRMP). The dual variables π_i and γ_k are associated with the constraints (5.2b) and (5.2c) respectively. The dual problem is to

$$\underset{\pi_i, \gamma_k}{\text{maximise}} \quad \sum_{i \in \tilde{\mathcal{V}}_{\text{dest}} \cup \tilde{\mathcal{V}}_{\text{init}}} \pi_i + \sum_k \gamma_k, \tag{5.3a}$$

$$\text{subject to} \quad \sum_{i \in \tilde{\mathcal{V}}_{\text{dest}} \cup \tilde{\mathcal{V}}_{\text{init}}} a_{k,r}^i \pi_i + \gamma_k \leq c_r^k \quad r \in \tilde{\mathcal{R}}_k, \; k = 1, \ldots, m \tag{5.3b}$$

$$\pi_i \leq d_i \quad i \in \tilde{\mathcal{V}}_{\text{dest}} \cup \tilde{\mathcal{V}}_{\text{init}} \tag{5.3c}$$

$$\pi_i \geq 0 \quad i \in \tilde{\mathcal{V}}_{\text{dest}} \cup \tilde{\mathcal{V}}_{\text{init}} \tag{5.3d}$$

$$\gamma_k \leq 0 \quad k = 1, \ldots, m \tag{5.3e}$$

5.1.4 The Subproblems

The aim of the SPs is to generate new feasible *routes* for the vehicles which improves the current solution of the RMP, i.e. with negative *reduced cost*. For a *route* to be feasible it needs to satisfy all the ”simple” constraints not taken into account by

the MP, i.e. it needs to satisfy the constraints (5.4b)–(5.4q). The *reduced cost* for a *route* is the original cost for that *route* minus a bonus for each customer served plus a penalty for the vehicle. Both the bonus and the penalty can be expressed in terms of the optimal dual variables from the DRMP. The SP for vehicle k is to[1]

$$\underset{H_i,Q_i,B_i,x_{i,j}}{\text{minimise}}$$

$$\alpha_1 \sum_{i\in\tilde{\mathcal{V}}_{\text{dest}}\cup\tilde{\mathcal{V}}_{\text{init}}} H_i + \alpha_2 \sum_{i\in\tilde{\mathcal{V}}_{\text{orig}}} (B_i - e_i) - \sum_{i\in\tilde{\mathcal{V}}_{\text{dest}}\cup\tilde{\mathcal{V}}_{\text{init}}} \sum_{j\in\tilde{\mathcal{V}}} \pi_i x_{i,j} - \gamma_k, \tag{5.4a}$$

subject to

$$\sum_{j\in\tilde{\mathcal{V}}\setminus\tilde{\mathcal{V}}_{\text{vehicle}}} x_{2n+l+k,j} = 1 \tag{5.4b}$$

$$\sum_{j\in\tilde{\mathcal{V}}\setminus\tilde{\mathcal{V}}_{\text{end}}} x_{j,2n+l+m+1} = 1 \tag{5.4c}$$

$$\sum_{j\in\tilde{\mathcal{V}}} x_{j,i} - \sum_{j\in\tilde{\mathcal{V}}} x_{i,j} = 0 \qquad i \in \tilde{\mathcal{V}}' \tag{5.4d}$$

$$\sum_{j\in\tilde{\mathcal{V}}} x_{i,j} - \sum_{j\in\tilde{\mathcal{V}}} x_{n+i,j} = 0 \qquad i \in \tilde{\mathcal{V}}_{\text{orig}} \tag{5.4e}$$

$$\beta_i - \sum_{j\in\tilde{\mathcal{V}}} x_{i,j} \geq 0 \qquad i \in \tilde{\mathcal{V}}_{\text{init}} \tag{5.4f}$$

$$B_{2n+k} \geq \xi_k \tag{5.4g}$$

$$B_j - t_{i,j} + M(1 - x_{i,j}) \geq B_i \qquad i \in \tilde{\mathcal{V}} \setminus \tilde{\mathcal{V}}_{\text{end}}, j \in \tilde{\mathcal{V}} \setminus \tilde{\mathcal{V}}_{\text{vehicle}} \tag{5.4h}$$

$$B_{2n+l+m+1} \leq T_{\text{horizon}} \tag{5.4i}$$

$$H_i = B_i - B_{i-n} \qquad i \in \tilde{\mathcal{V}}_{\text{dest}} \tag{5.4j}$$

$$H_i = B_i \qquad i \in \tilde{\mathcal{V}}_{\text{init}} \tag{5.4k}$$

$$H_i \geq 0 \qquad i \in \tilde{\mathcal{V}}_{\text{dest}} \cup \tilde{\mathcal{V}}_{\text{init}} \tag{5.4l}$$

$$Q_{2n+l+k} = \gamma_k \tag{5.4m}$$

$$Q_j - q_j + M(1 - x_{i,j}) \geq Q_i \qquad i,j \in \tilde{\mathcal{V}}' \tag{5.4n}$$

$$Q_i \leq C \qquad i \in \tilde{\mathcal{V}} \tag{5.4o}$$

$$B_i \geq e_i \qquad i \in \tilde{\mathcal{V}}_{\text{orig}} \tag{5.4p}$$

$$x_{i,j} \in \{0,1\} \qquad i,j \in \tilde{\mathcal{V}} \tag{5.4q}$$

The SPs are so called *elementary shortest path problems with time windows, resource constraints and pickup and delivery* (ESPPTWRCPD). The most successful approach, yet, to solve such problems to optimality is using so called *labelling algorithms*, which has been done by, for example, Ropke and Cordeau (see [23]).

[1] the index k has been removed wherever possible for clarity

5.2 Solving the Snapshot Problem

In this thesis, the *set-covering formulation* of the SFDP is solved by a *column generation* method which is presented here. The first step is to create an initial set $\tilde{\mathcal{R}}_k$, of *routes*, for each vehicle $k \in \{1, \ldots, m\}$. Then, let c^k_{red} denote the *reduced cost* for vehicle k's latest generated *route* and set $c^k_{\text{red}} = 0$, $k = 1, \ldots, m$. With $\tilde{\mathcal{R}}_k$, the DRMP is solved, yielding the optimal dual variables $\bar{\pi}_i$ and $\bar{\gamma}_k$. The dual variables are then used to solve m SPs, one for each vehicle, generating m *routes* r_k, which is then added to the sets of *routes*, $\tilde{\mathcal{R}}_k$. Let the *reduced costs*, for the new routes r_k, be denoted $c^{r_k}_{\text{red}}$, and they are used to update c^k_{red}. The DRMP is then solved again using the updated $\tilde{\mathcal{R}}_k$, and so on. This is done until all vehicles latest routes' *reduced costs*, c^k_{red}, are non-negative. The generated *routes* are used to formulate the IRMP, which is then solved. The complete algorithm is presented in Algorithm 1.

Algorithm 1 Solving the snapshot problem

1: generate the initial sets of *routes*, $\tilde{\mathcal{R}}_k$, $k = 1, \ldots, m$
2: **for** $k = 1, \ldots, m$ **do**
3: $\quad c^k_{\text{red}} = 0$
4: solve the DRMP over $\bigcup_{k=1}^{m} \tilde{\mathcal{R}}_k$, yielding $\bar{\pi}_i$ and $\bar{\gamma}_k$
5: **for** $k \in K$ **do**
6: $\quad$ solve the SP (5.4) for k with $\bar{\pi}_i$ and $\bar{\gamma}_k$, yielding r_k
7: $\quad$ add r_k to the set $\tilde{\mathcal{R}}_k$
8: $\quad c^k_{\text{red}} \leftarrow c^{r_k}_{\text{red}}$
9: **if** $\min\{r | r \in \text{reduced\_costs}\} < 0$ **then**
10: $\quad$ **goto** 2
11: solve the IRMP over $\bigcup_{k=1}^{m} \tilde{\mathcal{R}}_k$

If the SPs are solved to optimality, then, when no SP yields a negative *reduced cost*, the solution to the resulting RMP is also optimal to the MP. Hence, the objective value for the solution to the RMP, over all the generated columns, is a lower bound for the objective value of the IMP. Further, the solution to the IRMP must be equal or worse than the optimal solution to the IMP. This means that the optimal objective value for the IMP is bounded between the achieved objective values for the RMP and the IRMP respectively.

5.2.1 Generating an Initial Set of Routes

There are several ways to generate an initial set of *routes*. In this thesis, two ways are used. The first is to initialise with a minimal *route* for each vehicle, i.e. in which no customers are served. This means that each vehicle visits its starting node, as it has to, and then goes directly to the end node.

Going into the SFDP, each vehicle has already an assigned *route* from the previous SFDP, possibly modified by the *insertion heuristic*. As these *routes* are likely to be rather good they are used as initial *routes* in the new SFDP. This is called a "warm start" and is used to speed up the solution procedure as generally fewer columns need to be generated with a better initial column pool.

5.2.2 Solving the Integer Restricted Master Problem and the Dual Restricted Master Problem

Given a set of *routes*, solving the IRMP could be done with, for example, a *branch-and-bound* method and solving the DRMP could, for example, be done with the simplex method. It is, however, also possible to solve them using one of the modern solvers, such as CPLEX [49] or CBC [50], which is done in this thesis. As the major part of the computation time is used to solve the SPs, the solution method used to solve the IRMP and DRMP has a little impact on the computation time for the SFDP.

5.3 Solving the Sub Problems

Solving the SPs aims to generate new feasible *routes* with negative *reduced cost*, which can then be added to the set of *routes* in the RMP. A *route* being feasible means that is satisfies the constraints (5.4b)–(5.4q). The solution of the SPs are usually the most time consuming part of a *column generation* method, since they retain the integrality constraints. Hence it is often a good idea to use customised solution methods, exploiting the structure of the given SP, in an efficient way.

The SPs in the *set-covering formulation* of the SFDP, as described in Section 5.1.4, are ESPPTWCPDs, which are most often solved using so called *labelling algorithms*. They use dynamic programming to gradually build up feasible *routes* and utilises dominance rules to remove partial solutions, which can be proven not to be part of the optimal solution.

Another possibility is to use a heuristic to solve the SPs, as it suffices to find solutions to the SPs with negative *reduced costs* to improve the solution to the RMP. If a heuristic is used to solve the SPs, however, it is no longer true that when no further solution can be found with a negative *reduced cost*, then the solution to the RMP is optimal also for the MP. This follows from the fact that there might be solutions to the SPs with negative *reduced cost* which are just not found by the heuristic.

In this thesis both a *labelling algorithm* and a *local-search heuristic* is implemented. A *combination* of the two is further proposed.

5.3.1 A Local Search Heuristic

As seen in Section 2.2, a variety of heuristics has been used to solve the SPs. In the decision of what heuristic to use, both the type of problem and the expected instances of the problem are relevant. For a larger instance, generally, a more advanced heuristic is necessary to yield good results. One of the prominent features of the expected instances of the SFDPs is that the planning horizon is rather short, especially in comparison to static DARPs, for which the planning horizon covers the full problem length. On the other hand, the SFDP is solved multiple times during the execution of the *routes* and in practice they need to be solved and applied in real time. Hence, keeping the computational complexity to a minimum is of the essence. Both the short planning horizons and the need for low computational complexity motivates a quick, rather than a complex, heuristic.

The heuristic used in this thesis consists of two phases. The first phase is a *greedy insertion heuristic*, starting with a minimal *route* and then iteratively adding the requests that result in the lowest reduced cost, while keeping the *route* feasible. There are two types of possible requests to add. The first concerns customers which are already in the vehicle at the start of the SFDP. For such a request only the destination node for the customer is inserted into the *route*. The second type concerns customers which has not yet been picked up. For those requests both the origin and the destination nodes are inserted. Each time a node is inserted, it is done such that the resulting *reduced cost* is minimised while keeping the *route* feasible. Requests are added until no customer can be inserted such that the *reduced cost* is improved, i.e. decreased.

As the requests are added in one by one, only taking into account which requests is best to insert at the moment, there is no guarantee that the resulting order is optimal. Hence, to increase the quality of the solution, in a second phase, each request, one by one, is removed from the *route* and then reinserted optimally. This does not guarantee optimal ordering but it is a quick way to improve the solution. This second phase makes it a *local-search heuristic*.

Before describing the algorithm in more detail, some notation needs to be introduced. Solving the SP for vehicle k, given $\bar{\pi}_i$ and $\bar{\gamma}_k$ from the last solution of the DRMP, let r_k denote the *route* to create. From (5.4b) and (5.4c) we know that the first (last) element of r_k need to be the start node (end node) for vehicle k. Let us denote them $v_k \in \tilde{\mathcal{V}}_{\text{vehicle}}$ and $v_{\text{end}} \in \tilde{\mathcal{V}}_{\text{end}}$, respectively. As they have to be in the *route* for it to be feasible we initialise r_k as $r_k := \{v_k, v_{\text{end}}\}$. Now let us denote the customers starting in vehicle k by $\tilde{\mathcal{U}}_k$ and the customers waiting to be picked up by $\tilde{\mathcal{U}}_{\text{waiting}}$. Let further both $\tilde{\mathcal{U}}_k$ and $\tilde{\mathcal{U}}_{\text{waiting}}$ be ordered sets. Given a *route* r_k and a customer $u_i \in \tilde{\mathcal{U}}_k$, the end node of u_i can be inserted into r_k at $|r_k| - 1$ different positions. Let $r_k \wedge u_i$ denote an ordered set of *routes* resulting from all feasible insertions of the end node of u_i into r_k. Further, given a *route* r_k and a node $u_j \in \tilde{\mathcal{U}}_{\text{waiting}}$, there are $|r_k|(|r_k| - 1)/2$ ways to insert the start and end nodes of u_j into r_k (such that the start node is inserted prior, in the set, to the end node). Let $r_k \wedge u_j$ denote an

ordered set of *routes* resulting from inserting the start and end node, of customer u_j, into the *route* r_k such that the resulting *route* is feasible. Lastly, let $c_{red}^{r_k}$ be the *reduced cost* of *route* r_k as defined in (5.4a). Now, the algorithm is summarised in Algorithm 2.

Algorithm 2 A local search heuristic to solve the SP for vehicle k

input :π_i, γ_k
initialise the *route* r as the ordered set $\{v_k, v_{\text{end}}\}$
initialise $\tilde{\mathcal{U}}_k$ and $\tilde{\mathcal{U}}_{\text{waiting}}$

/* phase 1: insertion */
$r_{\text{best}} \leftarrow r_k$
$c_{\text{best}} \leftarrow c_{r_k}$
for $u_i \in \tilde{\mathcal{U}}_k \cup \tilde{\mathcal{U}}_{\text{waiting}}$ **do**
    for $r \in r_k \wedge u_i$ **do**
        if $c_r < c_{\text{best}}$ **then**
            $c_{\text{best}} = c_r$
            $r_{\text{best}} = r$
if $c_{\text{best}} < c_{r_k}$ **then**
    $r_k = r$
    goto : 3

/* phase 2: reinsertion */
for $u_i \in \{u \in \tilde{\mathcal{U}} : \text{end node of } u \in r_k\}$ **do**
    if $u_i \in \tilde{\mathcal{U}}_k$ **then**
        Remove end node of u_i from r_k
    else if $u_i \in \tilde{\mathcal{U}}_{\text{waiting}}$ **then**
        Remove start node and end node of u_i from r_k
    $r_{\text{best}} \leftarrow r_k$
    $c_{\text{best}} \leftarrow c_{r_k}$
    for $r \in r_k \wedge u_i$ **do**
        if $c_r < c_{\text{best}}$ **then**
            $c_{\text{best}} = c_r$
            $r_{\text{best}} = r$
    if $c_{\text{best}} < c_{r_k}$ **then**
        $r_k \leftarrow r_{\text{best}}$
output :r_k

5.3.2 A Dynamic Programming Approach to Solve the Subproblem

The second solution method used in this thesis, to solve the SPs, is a dynamic programming approach. It is a so called *labelling algorithm* as described by Desrosiers

et al. [16]. The idea is to build up new *routes*, node by node, and to use arc elimination as well as dominance criterion to remove *routes*, which are infeasible or can be proven not to lead to the optimal solution. The *labelling algorithms* are exact methods which are guaranteed to find an optimal solution eventually.

As presented above, a *route* is an ordered set of nodes, and a *route* r_k for a vehicle k is considered feasible if it satisfies the constraints (5.4b)–(5.4q). Let a *partial route* p_k, for a vehicle k, be an ordered set of nodes for the vehicle k to visit in the given order. Let further a *partial route* be considered feasible if it satisfies the route constraints (5.4b)–(5.4q) except for (5.4c) and (5.4e), i.e. it does not have to be complete. This means that a *feasible partial route* is a *partial route* which becomes a feasible *route* if one adds the destination nodes of the picked up customers as well as the end node. There is a possibility, however, that for a *partial route*, it is impossible to add the destination nodes of the picked up customers while satisfying the time constraints (5.4h)–(5.4i). Such a *partial route* is an exception from the definition of *feasible partial routes* above and is considered infeasible.

Let us further define the "labels" associated with the feasible *partial routes*. A label $\mathcal{L}_{p_k}$ is an object containing certain information about the *partial route* p_k. A label is defined as:

$$\mathcal{L}_{p_k} = \{i_{p_k}, c_{p_k}, q_{p_k}, s_{p_k}, \Omega^{p_k}_{\text{waiting}}, \Omega^{p_k}_{\text{open}}, \Omega^{p_k}_{\text{passenger}}, \Omega^{p_k}_{\text{served}}, \mathcal{O}_{p_k}, \mathcal{L}^{p_k}_{\text{parent}}\}, \tag{5.5}$$

where i_{p_k} is the last node of p_k, c_{p_k} is the *reduced cost* of p_k as defined by (5.4a), and s_{p_k} and q_{p_k} are the time and the load of vehicle k after it visits i_{p_k}. $\Omega^{p_k}_{\text{waiting}}$ ($\Omega^{p_k}_{\text{open}}$) denotes the set of origin nodes for the customers waiting to be picked up (having been picked up) after visiting the node i_{p_k}. $\Omega^{p_k}_{\text{passenger}}$ is the set of destination nodes for the customers, which started as well as remain in the vehicle, and $\Omega^{p_k}_{\text{served}}$ is the set of origin nodes for the customers, which have been delivered to their destinations. $\mathcal{O}_{p_k}$ is an ordered set containing the pick-up times for the picked up customers, $\mathcal{O}_{p_k} = \{o_i : i \in \Omega^{p_k}_{\text{open}}\}$. The label $\mathcal{L}^{p_k}_{\text{parent}}$ is described below.

We also define the concept of extending a label $\mathcal{L}_{p_k}$ to a new node j, which means to create a new label $\mathcal{L}_{\tilde{p}_k}$ associated with the *partial route* $\tilde{p}_k = p_k \cup j$. The $\mathcal{L}^{p_k}_{\text{parent}}$ element of a label $\mathcal{L}_{p_k}$ is a pointer to the label from which $\mathcal{L}_{p_k}$ was created. This enables recreation of the *partial route* p_k from the label $\mathcal{L}_{p_k}$ without having to store the whole *route*. An extension of a label $\mathcal{L}_{p_k}$ to a node j is said to be feasible if the resulting *partial route* associated with the new label is feasible.

For a label $\mathcal{L}_{p_k}$, there are four groups of possible nodes to extend to. The first is the origin nodes for the waiting customers, $\Omega^{p_k}_{\text{waiting}}$, the second is the destination nodes of the picked up customers in the vehicle k, $\{i + n \mid i \in \Omega^{p_k}_{\text{open}}\}$, the third is the destination nodes of the customers, which started and still remain in the vehicle k, $\Omega^{p_k}_{\text{passenger}}$ and the last group consists of just the end node, $\{2n + m + l + 1\}$. The procedure of extending $\mathcal{L}_{p_k}$ to node j varies depending on which group j belongs to. Further, the conditions for the extension to be feasible also depend on the group of j. The new label $\mathcal{L}_{\tilde{p}_k}$ is defined as

case 1: $j \in \Omega^{p_k}_{\text{waiting}}$

$$i_{\tilde{p}_k} = j \tag{5.6a}$$
$$s_{\tilde{p}_k} = s_{p_k} + t_{i_{p_k},j} \tag{5.6b}$$
$$q_{\tilde{p}_k} = q_{p_k} + 1 \tag{5.6c}$$
$$c_{\tilde{p}_k} = c_{p_k} + \alpha_2(s_{\tilde{p}_k} - e_j) \tag{5.6d}$$
$$\Omega^{\tilde{p_k}}_{\text{waiting}} = \Omega^{p_k}_{\text{waiting}} \setminus \{j\} \tag{5.6e}$$
$$\Omega^{\tilde{p_k}}_{\text{open}} = \Omega^{p_k}_{\text{open}} \cup \{j\} \tag{5.6f}$$
$$\Omega^{\tilde{p_k}}_{\text{passenger}} = \Omega^{p_k}_{\text{passenger}} \tag{5.6g}$$
$$\Omega^{\tilde{p_k}}_{\text{served}} = \Omega^{p_k}_{\text{served}} \tag{5.6h}$$
$$\mathcal{L}^{\tilde{p}_k}_{\text{parent}} = \mathcal{L}_{p_k} \tag{5.6i}$$

case 2: $j \in \{i + n \mid i \in \Omega^{p_k}_{\text{open}}\}$

$$i_{\tilde{p}_k} = j \tag{5.7a}$$
$$s_{\tilde{p}_k} = s_{p_k} + t_{i_{p_k},j} \tag{5.7b}$$
$$q_{\tilde{p}_k} = q_{p_k} - 1 \tag{5.7c}$$
$$c_{\tilde{p}_k} = c_{p_k} + \alpha_1(s_{\tilde{p}_k} - o_{j-n}) - \pi_j \tag{5.7d}$$
$$\Omega^{\tilde{p_k}}_{\text{waiting}} = \Omega^{p_k}_{\text{waiting}} \tag{5.7e}$$
$$\Omega^{\tilde{p_k}}_{\text{open}} = \Omega^{p_k}_{\text{open}} \tag{5.7f}$$
$$\Omega^{\tilde{p_k}}_{\text{passenger}} = \Omega^{p_k}_{\text{passenger}} \tag{5.7g}$$
$$\Omega^{\tilde{p_k}}_{\text{served}} = \Omega^{p_k}_{\text{served}} \cup \{j\} \tag{5.7h}$$
$$\mathcal{L}^{\tilde{p}_k}_{\text{parent}} = \mathcal{L}_{p_k} \tag{5.7i}$$

case 3: $j \in \Omega^{p_k}_{\text{passenger}}$

$$i_{\tilde{p}_k} = j \tag{5.8a}$$
$$s_{\tilde{p}_k} = s_{p_k} + t_{i_{p_k},j} \tag{5.8b}$$
$$q_{\tilde{p}_k} = q_{p_k} - 1 \tag{5.8c}$$
$$c_{\tilde{p}_k} = c_{p_k} + \alpha_1 s_{\tilde{p}_k} - \pi_j \tag{5.8d}$$
$$\Omega^{\tilde{p_k}}_{\text{waiting}} = \Omega^{p_k}_{\text{waiting}} \tag{5.8e}$$
$$\Omega^{\tilde{p_k}}_{\text{open}} = \Omega^{p_k}_{\text{open}} \tag{5.8f}$$
$$\Omega^{\tilde{p_k}}_{\text{passenger}} = \Omega^{p_k}_{\text{passenger}} \setminus \{j\} \tag{5.8g}$$
$$\Omega^{\tilde{p_k}}_{\text{served}} = \Omega^{p_k}_{\text{served}} \cup \{j\} \tag{5.8h}$$
$$\mathcal{L}^{\tilde{p}_k}_{\text{parent}} = \mathcal{L}_{p_k} \tag{5.8i}$$

case 4: $j = 2n + m + l + 1$

$$i_{\tilde{p}_k} = j \tag{5.9a}$$
$$s_{\tilde{p}_k} = s_{p_k} \tag{5.9b}$$
$$q_{\tilde{p}_k} = q_{p_k} \tag{5.9c}$$
$$c_{\tilde{p}_k} = c_{p_k} - \gamma_k \tag{5.9d}$$
$$\Omega^{\tilde{p_k}}_{\text{open}} = \Omega^{p_k}_{\text{open}} \tag{5.9e}$$
$$\Omega^{\tilde{p_k}}_{\text{served}} = \Omega^{p_k}_{\text{served}} \tag{5.9f}$$
$$\Omega^{\tilde{p_k}}_{\text{passenger}} = \Omega^{p_k}_{\text{passenger}} \tag{5.9g}$$
$$\Omega^{\tilde{p_k}}_{\text{served}} = \Omega^{p_k}_{\text{served}} \tag{5.9h}$$
$$\mathcal{L}^{\tilde{p}_k}_{\text{parent}} = \mathcal{L}_{p_k} \tag{5.9i}$$

For an extension $\mathcal{L}_{\tilde{p}_k}$ to be feasible, the following constraints needs to be satisfied:

$$s_{\tilde{p}_k} \leq T_{\text{horizon}} \tag{5.10}$$
$$q_{\tilde{p}_k} \leq C \tag{5.11}$$

and additionally, if $i_{\tilde{p}_k} = 2n + m + l + 1$, then

$$\Omega_{open} = \varnothing, \tag{5.12}$$

which makes the associated *partial route* satisfy the constraints (5.4c) and (5.4e). To further reduce the number branches, one can check if it is at all possible to deliver all picked up customers. This means adding the constraint, if $i_{\tilde{p}_k} \in \Omega^{p_k}_{\text{waiting}}$, then:

$$s_{\tilde{p}_k} + \eta(i_{\tilde{p}_k}, \Omega^{\tilde{p_k}}_{\text{open}}) \leq T_{\text{horizon}}, \tag{5.13}$$

where $\eta(i_{\tilde{p}_k}, \Omega^{\tilde{p_k}}_{\text{open}})$ is the smallest possible time it takes to visit all nodes in $\Omega^{\tilde{p_k}}_{\text{open}}$, starting at $i_{\tilde{p}_k}$.

Additionally, before describing the algorithm in detail, let us define the concept of dominating labels. A label $\mathcal{L}_{p'_k}$ is said to dominate another label $\mathcal{L}_{p_k}$ if it holds that

$$i_{p'_k} = i_{p_k} \tag{5.14a}$$
$$s_{p'_k} \leq s_{p_k} \tag{5.14b}$$
$$q_{p'_k} = q_{p_k} \tag{5.14c}$$
$$c_{p'_k} \leq c_{p_k} \tag{5.14d}$$
$$\Omega^{p'_k}_{\text{waiting}} = \Omega^{p_k}_{\text{waiting}} \tag{5.14e}$$
$$\Omega^{p'_k}_{\text{served}} = \Omega^{p'_k}_{\text{served}} \tag{5.14f}$$
$$\Omega^{p'_k}_{\text{open}} = \Omega^{p'_k}_{\text{open}} \tag{5.14g}$$

Statement. If a label $\mathcal{L}_{p'_k}$ is dominating another label $\mathcal{L}_{p_k}$, and the travelling times $t_{i,j}$ satisfies the triangle inequality, i.e. $t_{i,j} + t_{j,w} \geq t_{i,w}$, $i, j, w \in \mathcal{V}$, then the solution

achieved by extending $\mathcal{L}_{p'_k}$ optimally is at least as good as by extending $\mathcal{L}_{p_k}$ optimally. Hence the label $\mathcal{L}_{p_k}$ is unnecessary and can be removed from the solution procedure.

Proof. It is enough to prove that if $\mathcal{L}_{p'_k}$ dominates $\mathcal{L}_{p_k}$, then both $\mathcal{L}_{p'_k}$ and $\mathcal{L}_{p_k}$ can be extended to the same set of nodes $\Omega_{\text{extension}}$, and for each node j in $\Omega_{\text{extension}}$, the extension from $\mathcal{L}_{p'_k}$ to j will dominate the extension from $\mathcal{L}_{p_k}$ to j. Then for every chain of extensions from $\mathcal{L}_{p_k}$ leading to the end node, the same chain of extensions can be made for $\mathcal{L}_{p'_k}$. The resulting label from starting at the label $\mathcal{L}_{p'_k}$ will dominate the resulting label from starting at the label $\mathcal{L}_{p_k}$ and will have a lower or equal cost and, hence, represent an equal or better solution.

Let $\mathcal{L}_{p'_k}$ dominate $\mathcal{L}_{p_k}$ and let $\mathcal{L}_{\tilde{p}'_k}$ and $\mathcal{L}_{\tilde{p}_k}$ be their respective extensions to node j. We need to prove that the criterion (5.14) hold for $\mathcal{L}_{\tilde{p}'_k}$ and $\mathcal{L}_{\tilde{p}_k}$. We know that (5.14) hold for $\mathcal{L}_{p'_k}$ and $\mathcal{L}_{p_k}$ and that $i_{p_k} = i_{p'_k}$. Then, according to the extension rules (5.6)–(5.9), for each element in the labels, the change of that element from $\mathcal{L}_{p'_k}$ to $\mathcal{L}_{\tilde{p}'_k}$ is equivalent to the change of that element from $\mathcal{L}_{p_k}$ to $\mathcal{L}_{\tilde{p}_k}$. And as the constraints holds for $\mathcal{L}_{p'_k}$ and $\mathcal{L}_{p_k}$ and the changes in the labels are identical for the two labels, the constraints must also hold true for the new labels, $\mathcal{L}_{\tilde{p}'_k}$ and $\mathcal{L}_{\tilde{p}_k}$. □

We are now ready to describe the *labelling algorithm*. Let $\Gamma^i_{\text{untreated}}$ ($\Gamma^i_{\text{treated}}$) denote the sets of *untreated* (*treated*) labels at node i. At the start of the algorithm, all sets $\Gamma^i_{\text{untreated}}$ and $\Gamma^i_{\text{treated}}$ are empty, except for $\Gamma^{2n+k}_{\text{untreated}}$ which contains the element

$$\{2n+k, 0, \gamma_k, \xi_k, \{1, \ldots, n\}, \varnothing, \{i \in \{2n+m+1, \ldots, 2n+m+l\} \mid \beta^k_i = 1\}, \varnothing, \varnothing, \varnothing\}. \tag{5.15}$$

The label given by (5.15) represents a *partial route* for vehicle k, only including the vehicle's starting node.

Let us further expand the definition of extending a label, not only to create a new label, but also to insert it into the set $\Gamma^i_{\text{untreated}}$, where i is the node to which the label is extended. Then, while there are still 'untreated' labels: For each set $\Gamma^i_{\text{untreated}}$, remove any label, in that set, which is dominated by another label, in that set. Then extend every remaining label in that set to every node possible such that the extension is feasible, and move the extended labels from the set $\Gamma^i_{\text{untreated}}$ to the set $\Gamma^i_{\text{treated}}$. This procedure ends when there are no longer any untreated labels to extend, at which point the optimal feasible *route* corresponds to the label with the lowest cost in the $\Gamma^{2n+m+l+1}_{\text{treated}}$. The actual *route* can be extracted using the parent-label elements. The full algorithm is presented in Algorithm 3.

Algorithm 3 A labelling algorithm to solve the sub problem for vehicle k

input :π_i, γ_k
for $i \in 1, \ldots, 2n + m + l + 1$ **do**
    $\Gamma^i_{\text{untreated}} \leftarrow \varnothing$
    $\Gamma^i_{\text{treated}} \leftarrow \varnothing$
$\Gamma^{2n+k}_{\text{treated}} \leftarrow \{2n + k, 0, \gamma_k, \xi_k, \{1, \ldots, n\}, \varnothing,$
        $\{i \in \{2n + m + 1, \ldots, 2n + m + l\} \mid \beta^k_i = 1\}, \varnothing, \varnothing, \varnothing\}$

while $\bigcup_{i=1}^{2n+m+l+1} \Gamma^i_{\text{untreated}} \neq \varnothing$ **do**
    for $i \in 1, \ldots, 2n + m + l + 1$ **do**
        for $\mathcal{L} \in \Gamma^i_{\text{untreated}}$ **do**
            for $\mathcal{L}' \in \Gamma^i_{\text{untreated}}$ **do**
                if $\mathcal{L}$ dominates $\mathcal{L}'$ **then**
                    remove $\mathcal{L}'$ from $\Gamma^i_{\text{untreated}}$
                else if $\mathcal{L}'$ dominates $\mathcal{L}$ **then**
                    remove $\mathcal{L}$ from $\Gamma^i_{\text{untreated}}$
        for $\mathcal{L} \in \Gamma^i_{\text{untreated}}$ **do**
            for $j \in \Omega_{\text{waiting}}$ **do**
                extend $\mathcal{L}$ to j if the extension is feasible
            for $j \in \{x + n \mid x \in \Omega_{\text{open}}\}$ **do**
                extend $\mathcal{L}$ to j if the extension is feasible
            for $j \in \Omega_{\text{passenger}}$ **do**
                extend $\mathcal{L}$ to j if the extension is feasible
            if $\Omega_{\text{open}} = \varnothing$ **then**
                extend $\mathcal{L}$ to $2n + m + l + 1$
            move $\mathcal{L}$ from $\Gamma^i_{\text{untreated}}$ to $\Gamma^i_{\text{treated}}$
$\mathcal{L}^* \leftarrow$ the label in $\Gamma^{2n+m+l+1}_{\text{treated}}$ with the smallest cost
generate *route* r_k from $\mathcal{L}^*$
output :r_k

5.3.3 A combination of the Local-Search Heuristic and the Labelling Algorithm

The *local-search heuristic* solves the SPs fast, while the *labelling algorithm* solves them optimally. A way to combine these two traits, is to first generate columns using the *local-search heuristic* for the SPs and then, when no more columns with negative *reduced cost* can be generated, the SPs are instead solved using the *labelling algorithm* until no more columns with negative *reduced cost* can be generated. This way, most of the columns will be generated by the *local-search heuristic* while the *labelling algorithm* still guarantees that the generated columns suffice to make the solution to the RMP optimal for the MP. As the *local-search heuristic* is faster than the *labelling algorithm*, this will most likely speed up the solution of the SFDP. This is what is referenced as the *combination* of the two methods or the *combination algorithm*.

5.4 Post-Processing

As the SFDP is modelled as a *set-covering problem* and not a *set-partitioning problem* (see Section 5.1.1), a post-processing step is needed. In the *set-covering problem* formulation, it is allowed for a customer to be served by multiple vehicles, so when the SFDP is solved, each such customer, has to be removed from all but one route. In this thesis a simple, post-processing procedure has been implemented, looping through the cars, and for each car removing every customer which has been served by a car earlier in the loop.

6

Solving the Fleet Dispatching Problem

The FDP aims at optimising a fleet of vehicles to pick up and deliver customers during a period of time. As described in Chapter 4, we do not treat the FDP as a proper optimisation problem but rather as a framework or as a transport simulation model. The actual optimisation is done by the reoptimisation scheme as well as by the *insertion heuristic*. The solution procedure works such that the FDP is run as a simulation, starting at time 1 and ending at time T. Every T_{reopt} in the simulation, an SFDP is solved, based on the FDP's current state. The old routes in the simulation are then removed and replaced by the routes created by the SFDP. The SFDP optimises for a time period T_{horizon} forward. Both T_{reopt} and T_{horizon} are parameters affecting the quality of the solutions to the FDP but also the computational complexity. The lower T_{reopt}, is and the higher T_{horizon} is, the better the solutions are, but also the higher the computational complexity. The reason is that, if the FDP is reoptimsed more frequently, it is more responsive to new requests and a longer horizon let it plan further ahead. More frequent reoptimisations means that the SFDP needs to be solved more often and longer planning horizons makes for larger instances of the SFDP, both which increases the computation time. These effects will be seen in Chapter 7.

If the difference between the time, s_u, when a customer wants to be picked up, and the time, s'_u, that he/she alerts the fleet of his/her request, is small in comparison to T_{reopt}, only using the reoptimisation scheme might yield poor results. Let us illustrate this by an example: Assume that the FDP is reoptimised at time 1 but by then there are no customers so no routes are generated. Then, at time 2, a customer alerts the fleet of her journey and that she wants to be picked up immediately. At this stage all cars are free but no car will be assigned to pick her up until the next reoptimisation step. This means that she will need to wait for $T_{\text{reopt}} - 1$ seconds before even being noticed by the system. An approach to reduce this waiting time, resulting from new requests not being "noticed", an *insertion heuristic* is used, immediately inserting new requests into one of the current routes. This is done such that the increase in waiting and travelling times is minimised, out of all possible ways to insert the new request in the current schedule.

6.1 A Solution Algorithm for the Fleet Dispatching Problem

The FDP is initialised with a graph $\mathcal{G} = (\mathcal{V}, \mathcal{A})$ representing the road network, an ordered set K of m vehicles starting at a set of random nodes in the graph, and a set $\mathcal{U}$ of N customers. Each customer $u \in \mathcal{U}$ has a start node i^u_{orig}, an end node i^u_{dest}, a time, s_u, when his/her journey starts, and a time, s'_u, when the model is alerted of his/her request, as well as a time s^u_{pick} (s^u_{drop}), when he/she is picked up (dropped off). An ordered set $\mathcal{U}_{\text{unserved}}$ stores all customers known to the model that has not yet been picked up. The time in the model is treated as discrete so that the model iterate over it. The FDP loops through the time indices $t = 1, \ldots, T$, moving the vehicles along their routes and making the vehicles pick up and drop off the right customer at the right node. Each vehicle has, at each time, a route r_k, a last-visited node i_k, and a number of customers it is currently serving, in the two sets $\mathcal{U}^k_{\text{pick}}$ (customers to pick up) and $\mathcal{U}^k_{\text{drop}}$ (customers to drop off). At each time step t, each customer u, with $s'_u = t$, is added to the set $\mathcal{U}_{\text{unserved}}$, and the *insertion heuristic* is utilised to insert his/her start and end nodes into one of the routes as well as him/her into the $\mathcal{U}^k_{\text{pick}}$ for the vehicle driving that route.

If it holds that

$$t \bmod T_{\text{reopt}} = 1,$$

then all vehicles, all customers currently in the vehicles and all customers currently in $\mathcal{U}_{\text{unserved}}$ are included in formulating an SFDP. The SFDP uses a graph where the customers origins and destinations make up the nodes and the graph of the FDP uses nodes representing intersections in the road network. Hence, before the SFDP can be solved the graph for the SFDP needs to be created based on the graph for the FDP. Once the SFDP is solved, the solution is translated back to the framework of the FDP and the routes and customers are assigned to the vehicles.

To move the vehicles, a queue-based system is used, as in [37]. So when a vehicle leaves a node, the time of its arrival to its next node is saved. To this purpose, a number of ordered sets $\mathcal{K}^t_{\text{node}}$, $t = 1, \ldots, T$, are utilised, keeping track of which vehicles are visiting a node (any node) at time t. When a vehicle k is leaving a node i, driving towards the node j, it is added to the set $\mathcal{K}^{t+d(i,j)}_{\text{node}}$, where $d(i, j)$ denotes the number of time units it takes to drive from node i to node j. Which node a vehicle arrives to is instead derived from its path. It is called queue-based as the vehicles have no position in between nodes. The algorithm is presented in Algorithm 4.

Algorithm 4 Solving the FDP

input :$\mathcal{V}, \mathcal{U}, d, m$

$\mathcal{U}_{\text{unserved}} \leftarrow \varnothing$
for $k \in \{1, \ldots, m\}$ **do**
    $\mathcal{U}^k_{\text{pick}} \leftarrow \varnothing$
    $\mathcal{U}^k_{\text{drop}} \leftarrow \varnothing$
    $r_k \leftarrow \varnothing$
    $i_k \leftarrow$ random node $i \in \mathcal{V}$
$\mathcal{K}^1_{\text{node}} \leftarrow \mathcal{K}$
for $t \in \{2, \ldots, T\}$ **do**
    $\mathcal{K}^t_{\text{node}} \leftarrow \varnothing$
for $t \in \{1, \ldots, T\}$ **do**
    $\mathcal{U}_{\text{unserved}} \leftarrow \{u \in \mathcal{U} : s'_u = t\}$
    for $k \in \mathcal{K}^t_{\text{node}}$ **do**
        for $u \in \mathcal{U}^k_{\text{pick}}$ **do**
            if $i^u_{\text{orig}} = i_k$ **then**
                $\mathcal{U}^k_{\text{pick}} \leftarrow \mathcal{U}^k_{\text{pick}} \setminus u$
                $\mathcal{U}^k_{\text{drop}} \leftarrow \mathcal{U}^k_{\text{drop}} \bigcup u$
                $s^u_{\text{pick}} \leftarrow t$
        for $u \in \mathcal{U}^k_{\text{drop}}$ **do**
            if $i^u_{\text{dest}} = i_k$ **then**
                $\mathcal{U}^k_{\text{drop}} \leftarrow \mathcal{U}^k_{\text{drop}} \setminus u$
                $s^u_{\text{drop}} \leftarrow t$
        if $r_k = \{i, j, \ldots\}$ **then**
            $\mathcal{K}^{t+d(i,j)}_{\text{node}} \leftarrow \mathcal{K}^{t+d(i,j)}_{\text{node}} \bigcup k$
            $r_k \leftarrow r_k \setminus i$
            $i_k \leftarrow j$
        else if $r_k = \{i\}$ **then**
            $\mathcal{K}^{t+1}_{\text{node}} \leftarrow \mathcal{K}^{t+1}_{\text{node}} \bigcup k$
            $r_k \leftarrow r_k \setminus i$
        else if $r_k = \varnothing$ **then**
            $\mathcal{K}^{t+1}_{\text{node}} \leftarrow \mathcal{K}^{t+1}_{\text{node}} \bigcup k$
    if $t \bmod T_{\text{reopt}} = 1$ **then**
        solve the SFDP and update r_k and $\mathcal{U}^k_{\text{pick}}$ for $k \in \mathcal{K}$
    else
        for $u \in \mathcal{U} : s'_u = t$ **do**
            Apply the insertion heuristic (Algorithm 5) for u
calculate the objective value (4.2a) from s^u_{pick} and s^u_{drop}

6.2 The Insertion Heuristic

The *insertion heuristic* (Algorithm 5) is a tool to quickly integrate new customers into the current schedule. It checks all possible ways to insert a new customer into the current schedule and chooses the cheapest one. Let u be the new customer and let i^u_{orig} and i^u_{dest} be the start and end nodes for the customer u. Let the cost, as defined in (5.4a), for a route r_k and a vehicle k, be denoted c_{r_k}. The algortihm is then presented in Algorithm 5.

Algorithm 5 The insertion heuristic

input :$\mathcal{K}, u, r_k$ $(k \in \mathcal{K})$
$c_{\text{best}} \leftarrow \infty$
for $k \in \mathcal{K}$ **do**
  for $i = 1, \ldots, |r_k| - 1$ **do**
    for $j = i + 1, \ldots, |r_k| - 1$ **do**
      $\tilde{r}_k \leftarrow r_k$
      insert u_{orig} between i and $i + 1$ in $\tilde{r}_k$
      insert u_{dest} between j and $j + 1$ in $\tilde{r}_k$
      if $c_{\tilde{r}_k} - c_{r_k} < c_{\text{best}}$ **then**
        $c_{\text{best}} \leftarrow c_{\tilde{r}_k} - c_{r_k}$
        $k_{\text{best}} \leftarrow k$
        $r_{\text{best}} \leftarrow \tilde{r}_k$
$r_{k_{\text{best}}} \leftarrow r_{\text{best}}$
$\mathcal{U}^{k_{\text{best}}}_{\text{pick}} \leftarrow \mathcal{U}^{k_{\text{best}}}_{\text{pick}} \bigcup u$
output :$r_{k_{\text{best}}}, \mathcal{U}^{k_{\text{best}}}$

7

Computational Experiments

To estimate the efficiency of the solution algorithms presented in Chapter 5 and Chapter 6, a number of tests have been run. The tests, as well as the results, are presented here. The tests are divided into two categories, concerning the FDP and the SFDP, respectively. All tests are implemented in Python 3.4, using CBC [50] as LP-solver and pulp [51] as solver interface. The tests were run on an Intel i5-6600 3.3GHz, 16Gb RAM computer using Windows 10.

7.1 Testing the Snapshot Fleet Dispatching Problem

We have generated a set of test instances for the SFDP. The different solution methods (see Chapter 5) are then tested and compared on each of those instances. A test instance is made up by a set $\mathcal{U}$ of waiting customers, a set of vehicles $\mathcal{K}$, the maximum route length T, the vehicle capacity C and, for each vehicle $k \in \mathcal{K}$, set $\mathcal{U}_k$, of the customers starting in vehicle k. The start and end node of each customer $u \in \mathcal{U}$, the start node of each vehicle $k \in \mathcal{K}$ and the end node of each passenger $u \in \bigcup_{k \in \mathcal{K}} \mathcal{U}_k$ are randomly generated points within an area of $[0, 100] \times [0, 100]$ area units, each following a uniform distribution. The arc length for an arc between two nodes is the euclidean distance between those nodes. The units are such that it takes one time unit to travel one distance unit. Lastly, the customers, starting in the vehicles, will one by one be randomly distributed among the cars, following a uniform distribution with the exception that if assigning a customer to a vehicle would make that vehicle exceed its capacity, another vehicle is chosen in the same fashion.

The different methods used to solve the SPs are the *local-search heuristic* (Algorithm 2) and the *labelling algorithm* (Algorithm 3), as well as a *combination* of the two.

As no *branch-and-price* algorithm has been implemented there are no guarantees that the presented algorithms will find an optimal solution to the IMP. When using the *labelling algorithm* or the *combination* of the *labelling algorithm* and the *local-*

search heuristic (see Section 5.3.3) for the SPs, however, the solution to the RMP, when no more columns with negative *reduced cost* can be generated, is optimal to the MP and is a lower bound to the optimal objective value of the IMP. If, further, that lower bound equals the objective value of the IRMP, then that solution to the IRMP is optimal also to the IMP.

Solving the SPs using the *labelling algorithm*, is computationally more demanding than using the *local-search heurisitc*, and the number of customers and the maximum route length are the two most critical parameters. The tests for the SFDP are divided into two parts. The first part consists of small instances, solvable using the *labelling algorithm* for the SPs. Those solutions to the SFDP can be directly compared with their respective lower bounds. The second part consists of larger instances, which take too long to solve using the *labelling algorithm*. These are used to investigate which instance sizes that can be solved in reasonable computing times. For those instances, no proper lower bound is achieved but the objective value for the RMP still provides some information about the objective value of the IMP. At least it can be said that if the objective value for the IRMP and the objective value for the RMP differs a lot, it would probably be a good idea to use a *branch-and-price* algorithm.

Each test instance is created randomly based on some input parameters. These are the number of vehicles, m, the number of customers, n, the number of passengers, l, the capacity, C and the maximum route length, T. The cost of not delivering a customer, d_i, to its destination, will be equal to T, and both α_1 and α_2 will be set equal to 1. Further, the starting times for the customers are set to 0 but the starting times for the vehicles are uniformly distributed in the interval $[0, T/4]$. Those values are chosen as to mimic circumstances from when it is used in the tests for the FDP.

7.1.1 Results for the Small Instances of the Snapshot Fleet Dispatching Problem

For the small instances of the SFDP, we solve the problem using the *labelling algorithm*, the *local-search heuristic* and the *combination* for the SPs. Let $\underline{z}$ denote the lower bound, to the MPs, from solving the RMPs, using the *labelling algorithm* for the SPs, and let z denote the best objective values found for the IMPs, for the different instances. The results for the small instances are illustrated in figure 7.1, where the quotient between z and $\underline{z}$ is plotted against the computation time for the different problems instances and solvers. The numbers are further presented in the table A.1 in appendix A. When, for an instance, $z = \underline{z}$, that solution to the IMP is optimal. We see that when using the *labelling algorithm*, or the *combination algorithm*, for the SPs, the solutions acquired are optimal for most of the test instances. The *combination* seems to give similar objective values but to be faster. Using the *local-search heuristic* for the SPs is faster than the two others but gives slightly worse results.

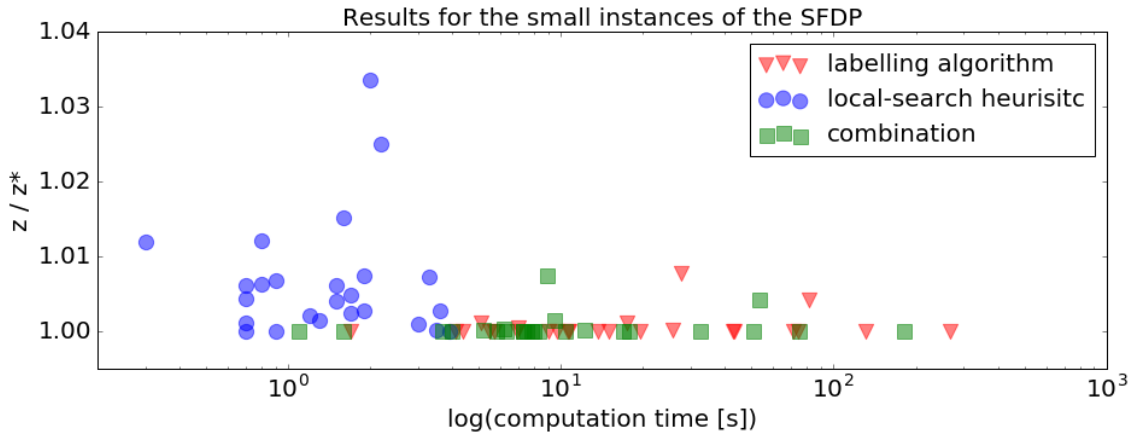


Figure 7.1: Objective values for the solutions to the IMPs, divided by their lower bounds, from solving the respective MPs, plotted against the logarithm of the computation times, for the different SFDP instances. Each instance is solved using three different methods for the SPs; the blue circles represent solutions using the *insertion heuristic*, the red triangles represent solutions using the *labelling algorithm* and the green squares represent the *combination* of the two. The numbers used and the parameters for the different instances are presented in Table A.1 in Appendix A.

7.1.2 Results for the Large Instances of the Snapshot Problem

The large instances of the *snapshot problem* would take too long to solve using the *labelling algorithm* and have, as such, only been solved using the *local-search heuristic* for the SPs. The results for the solutions to the large instances are shown Figure 7.2. The numbers are also presented in Table A.2 in the appendix. As the different parameter settings are differentiated by colour and shape, we see that the solution method performs similarly on instances with the same parameters. It also performs worse, in terms of objective value, for the instances with fewer cars and customers but with longer time horizon T (the magenta stars in the figure).

7.2 Testing the Fleet Dispatching Problem

To test the algorithms developed in this thesis, to solve the FDP, a number of test instances of the FDP are created and solved. Each test instance contains a graph, a set of n customers $\mathcal{U}$, a set of m vehicles $\mathcal{K}$ and a maximum route length T. Additionally it contains the parameters T_{reopt} and T_{horizon}, which determine how often and how far into the future to reoptimise. The maps are incomplete connected graphs, consisting of p nodes, each with a random location in the area $[0, 100] \times [0, 100]$, and a number of edges connecting them. The arcs in the graphs are created using a Delaunay triangulation (see [52]) over the nodes. An example

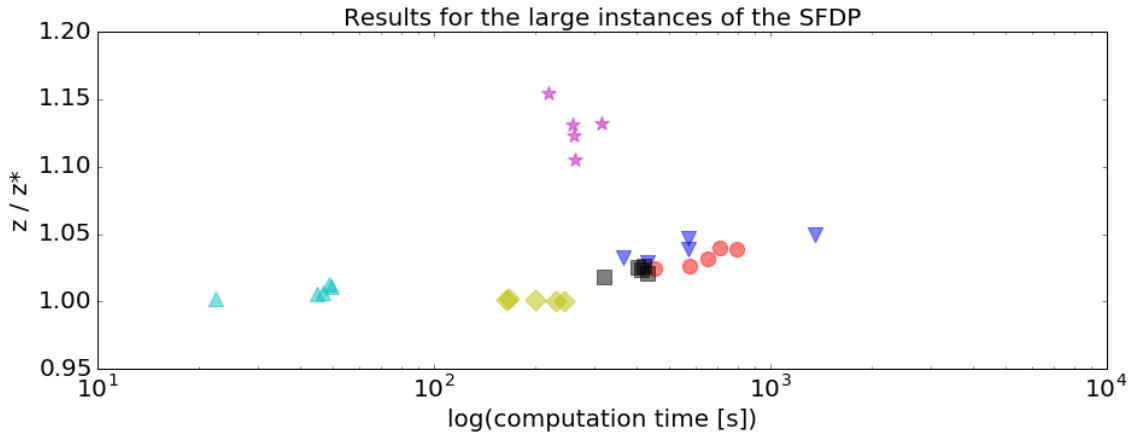


Figure 7.2: Objective values for the solutions to the IMPs, divided by the objective values for the solutions to the respective MPs, plotted against the logarithm of the computation times, for the different SFDP instances. The IMPs and the MPs has been solved using the *insertion heuristic* for the SPs. The instances of the SFDP solved can be divided into six groups of five instances each, where each instance in a group are created using the same parameters. The different colours and marker shapes represent the different groups. The numbers and the input parameters are presented in Table A.2 in Appendix A.

of such a graph is shown in Figure 7.3. The time it takes to drive along a whole edge to the other is the euclidean length of that edge. Each customer $u \in \mathcal{U}$ has an origin and a destination node on the map. He/she further has a starting time s_u when he/she wants to be picked up. In all tests, the customers want to be picked up immediately after alerting the problem, i.e., $s'_u = s_u$. The starting time of each customer is randomised uniformly in the interval $[0, T/2]$. Moreover, as in the tests for the SFDPs, the coefficients $\alpha_1 = \alpha_2 = 1$, and the price, d_i, for not delivering a customer u to his/her destination, is $T - e_u$, where e_u is the starting time for u. Each vehicle $k \in \mathcal{K}$ has a random starting node in the graph and a maximum capacity, C.

Each test instance is solved using a variety of solution methods developed in this thesis, such that the solution methods can be compared. The different methods for the FDP that will be studied are

- The reoptimisation scheme
- The reoptimisation scheme combined with the *insertion heuristic*
- The *insertion heuristic* for new customers.

These methods are used together with the following methods for the SPs

- The *labelling algorithm*

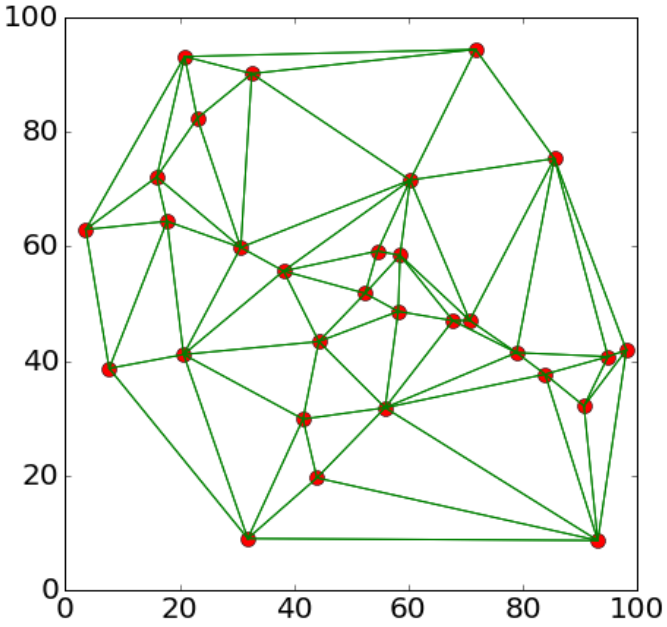


Figure 7.3: Example of a graph used for testing the FDP. The nodes are randomly generated and the edges are created through Delaunay Triangulation.

- The *local-search heuristic*
- The combination of the *labelling algorithm* and the *local-search heuristic*.

First the different methods for the FDP are tested using the *labelling algorithm* for the SPs. Then the different methods for SPs are tested using the reoptimisation scheme combined with the *insertion heuristic*.

For very small instances of the FDP, a static version of the problem, where all incoming requests are known in advance, can be solved using the *labelling algorithm* for the SPs. The solutions to the static problem are not necessarily optimal as no *branch-and-price* is used. From the tests of the SFDP, for this size of instances, we have shown that the solutions seem to be very close to optimal (see Section 7.1.1). For these instances, the solution to the FDP can be compared to the optimal solution for the corresponding static version of the problem. Note, however, that the objective values of the static versions are *n*ot lower bounds. As more information is known in the static versions of the problem it is virtually impossible to receive the same objective values for the dynamic versions.

7.2.1 Results for the Fleet Dispatching Problem

The test instances for the FDP are divided into two sets, small and large. The static version of the problem is solved for the small, but not for the large, instances. We solve the small instances both as static and dynamic problems using the *labelling*

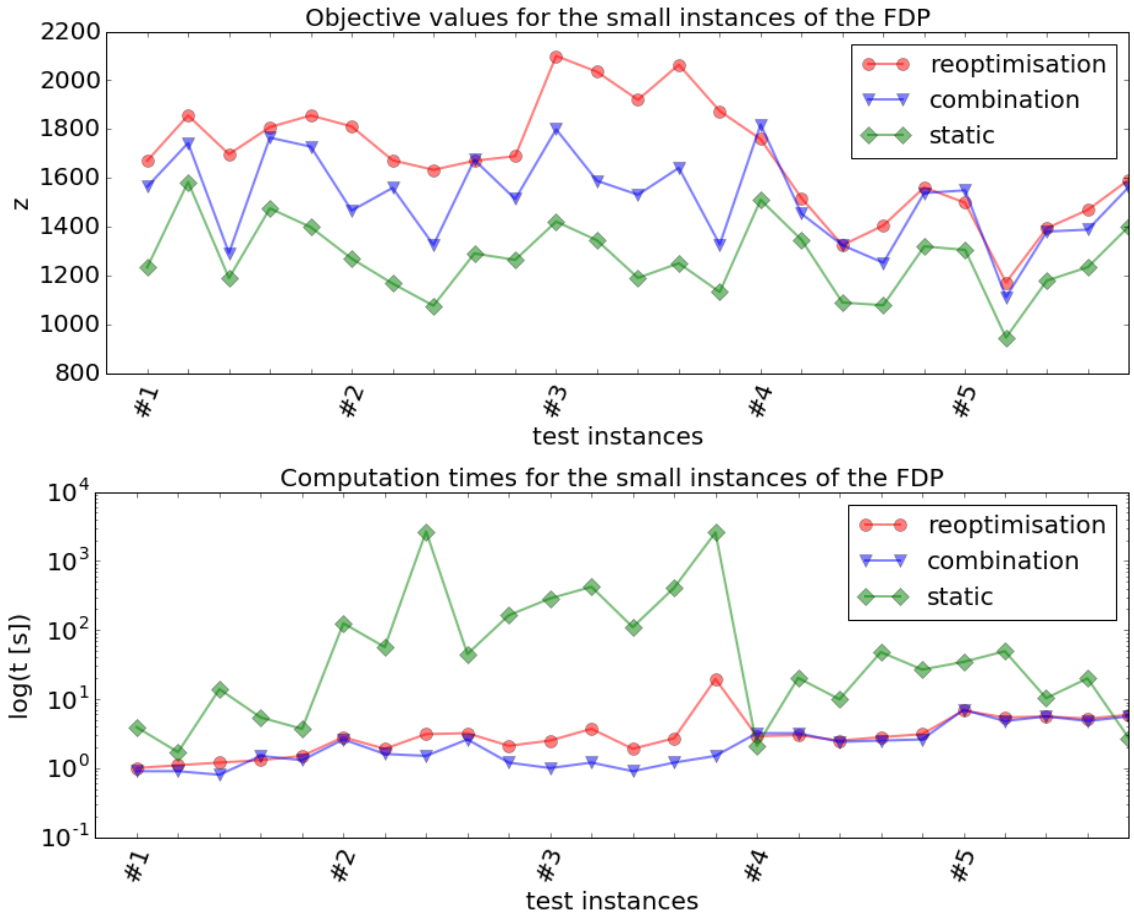


Figure 7.4: Results for the set of small instances from two different solution methods, the reoptimisation scheme (red circles) and the reoptimisation scheme combined with the *insertion heuristic* for new customers (blue triangles), for the dynamic version of the instances. Results for the corresponding static version of the instances are presented as well (green diamonds). The upper figure shows the best objective values found, and the lower figure shows the computation times, for the different instances. The numbers on the x-axis are the different parameter settings, in the order they are presented in Table A.3 in Appendix A.

algorithm for the SPs. The dynamic version of the problems are further solved with and without using the *insertion heuristic*. The objective values and computation times, for the solutions to these instances, with the different solution methods, are presented in Figure 7.4. The objective values, computation times and parameters for these instances are further presented in A.3 in Appendix A. The objective values for the static versions of the problems are lower for all instances, but the computation times are in general much longer. For the dynamic version, using the *insertion heuristic* yields better results, both in terms of objective value and computation time for most of these instances.

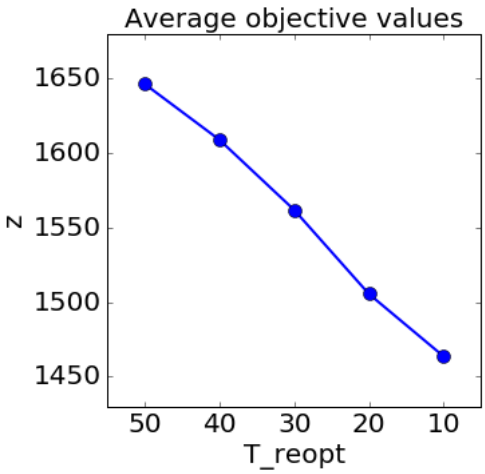
To investigate the effect of the parameter T_{reopt} on the solutions, 150 random instances with the same parameters, except for different values of T_{reopt}, were solved.

The averages of the objective values and the corresponding computation times are shown in Figure 7.5. As expected, a lower value of T_{reopt} resulted in a lower objective value but also a longer computation time. When the *insertion heuristic* is not used the results are rather straight forward, but with the *insertion heuristic* we get effects from how the heuristic interacts with the reoptimisation scheme, which might explain that the average objective value is higher for $T_{\text{reopt}} = 40$ than for $T_{\text{reopt}} = 30$. In any case, it is better both in terms of objective value and computation time to use the *insertion heuristic* for new requests instead of just lowering the T_{reopt}. Additionally, using only the *insertion heuristic* for the same set of 150 instances, without the reoptimisation, yields an average objective value of 1492. That is better (lower) than the average objective value achieved by using only the reoptimisation scheme with values of $T_{\text{reopt}} \geq 20$, but worse (higher) than the achieved average optimisation when the *combination*, for all tested values of T_{reopt}.

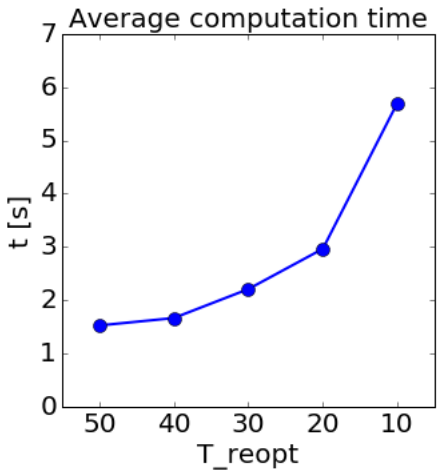
In the Figures 7.6 and 7.7 we see the objective values and computation times for the solutions to the set of large instances, for different solution methods. For the large instances a solution time cap of 300 seconds has been used to solve an instance. For each of the large instances, the graphs has been created with 30 nodes and the vehicles have a capacity of $C = 2$.

For the results in Figure 7.6 the instances has been solved using the reoptimisation scheme as well as using the reoptimisation scheme in combination with the *insertion heuristic*. In both cases the SPs has been solved using the *local-search heuristic*. For all those instances the combination produces better solutions in terms of objective value and in all but one case it is also faster. The objective values and computation times, as well as the parameters for the instances are shown in Table A.4 in Appendix A.

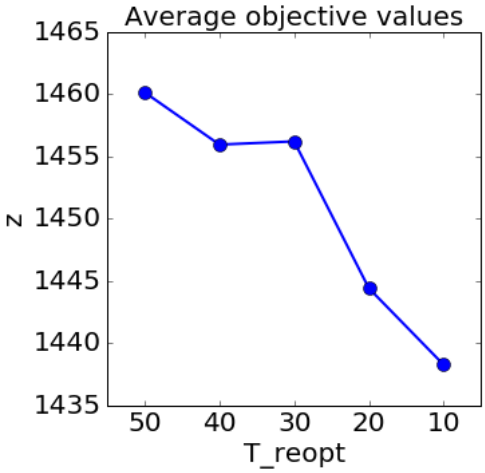
In Figure 7.7 the same set of instances has been solved, now using the combination of the reoptimisation scheme and the *insertion heuristics* but with the three different methods to solve the SPs. As we have seen in the tests for the SFDP (see Section 7.1), the *labelling algorithm* and the *combination* of the two algorithms seem to provide better results for the SFDP than the *local-search heuristic*. Mostly, that seem to be the case for those instances of the FDP as well. For these instances, using the *local-search heuristic* for the SPs results in a faster method than using the other two and using the *combination* seem to be faster than using the *labelling algorithm*. The objective values and computation times, as well as the parameters for the instances, are presented in Table A.5 in Appendix A.



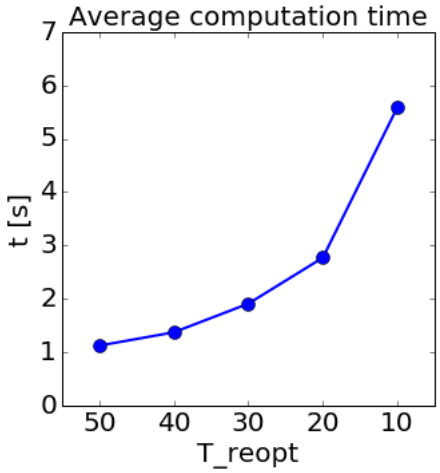
(a) Average objective values for different T_{reopt} when not using the insertion heuristic.



(b) Average computation times for different T_{reopt} when not using the insertion heuristic.



(c) Average objective values for different T_{reopt} when using the insertion heuristic.



(d) Average computation times for different T_{reopt} when using the insertion heuristic.

Figure 7.5: Average objective values and computation times for different values for T_{reopt} when using the reoptimisation scheme with and without the *insertion heuristic* and using the *labelling algorithm* for the SPs. For each value of T_{reopt} tested, 150 different random instances has been solved with the two methods and the average objective values and computation times are shown in the graphs. The instances is made with $m = 5$ vehicles, $n = 15$ customers, 20 nodes in the graph, a vehicle capacity of $C = 2$, a simulation time $T = 200$ and a planning horizon of $T_{\text{horizon}} = 200$.

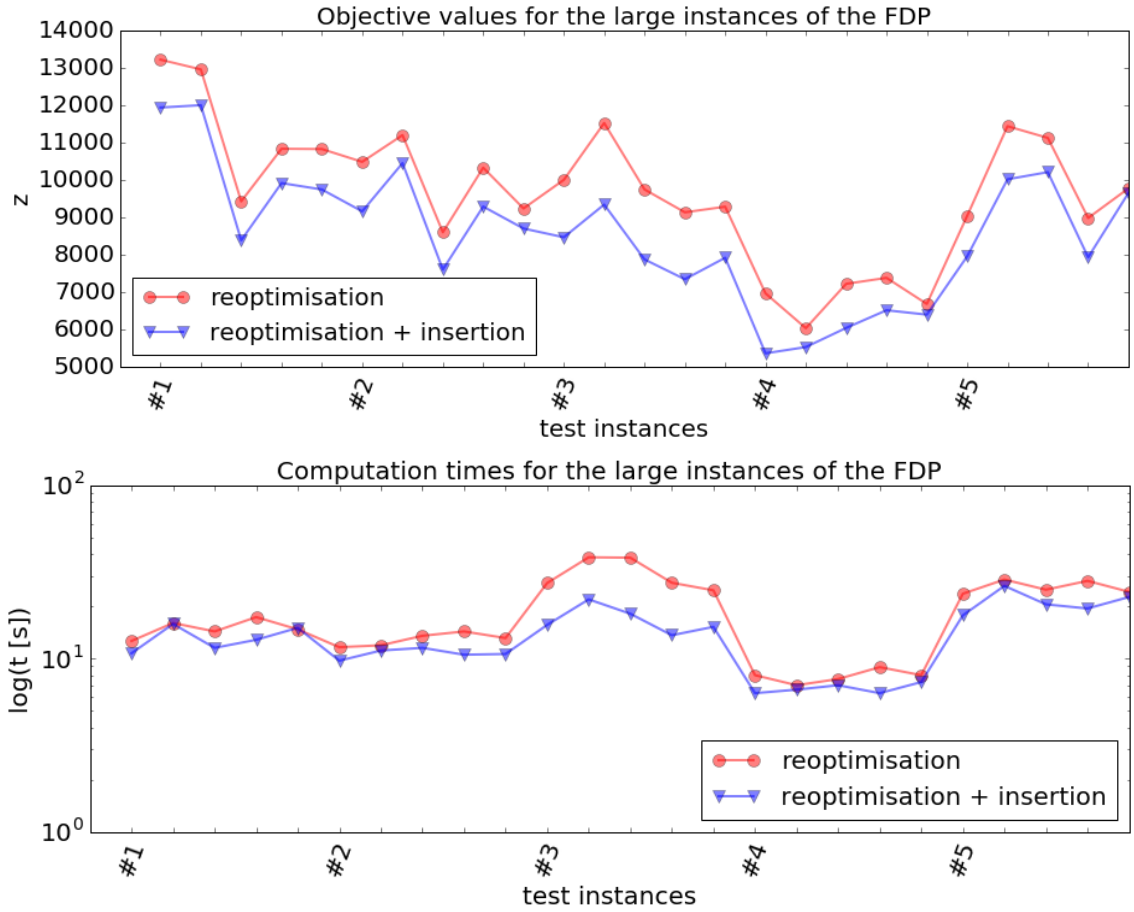


Figure 7.6: Results for the set of large instances. Each instance has been solved using the reoptimisation scheme (red circles) as well as the combination of the reoptimisation and the *insertion heuristic* (blue triangles). For both of them, the *local-search heuristic* was used for the SPs. The upper figure shows objective values for the different instances, and the lower shows the computation times. The numbers along the x-axis denotes the different parameter settings in the order they appear in Table A.4 in Appendix A.

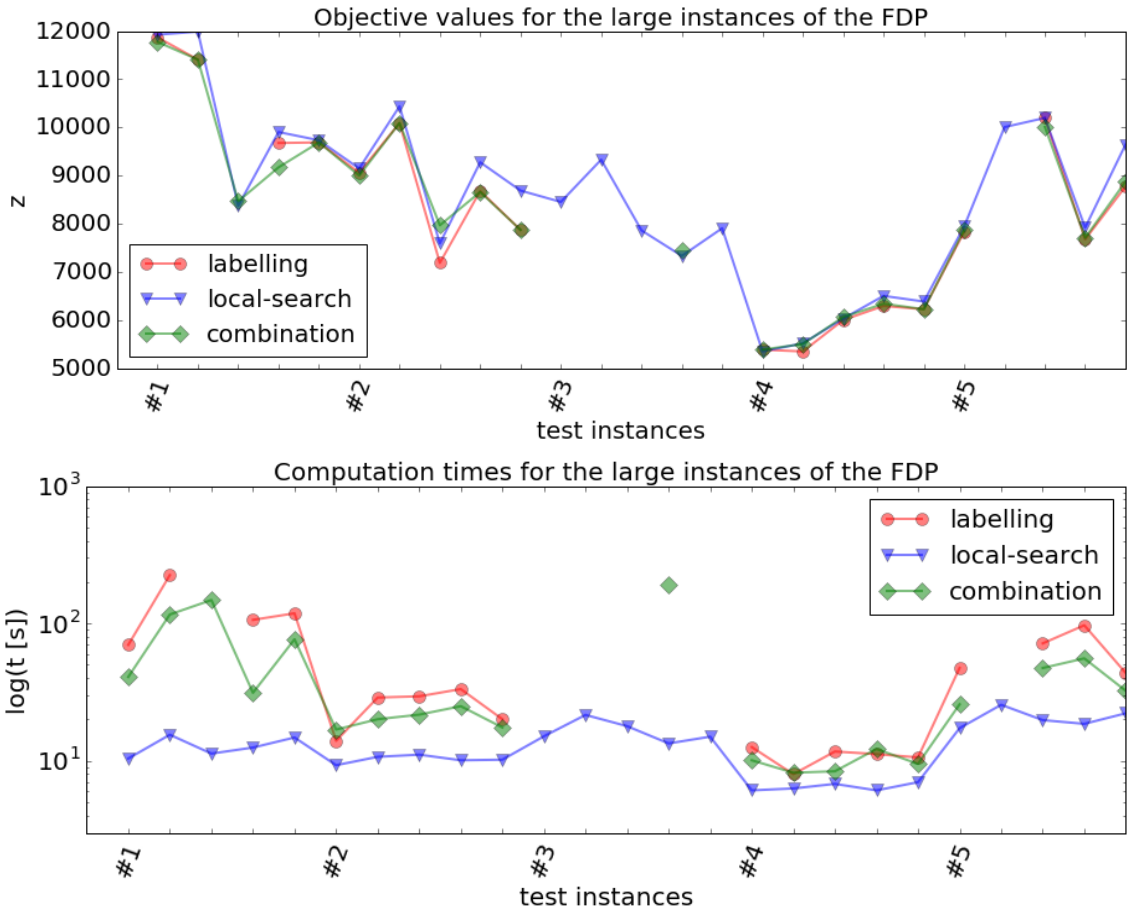


Figure 7.7: Results for the set of large instances of the FDP, each solved using the reoptimisation scheme and the *insertion heuristic* for new requests and with the *labelling algorithm* (red circles), the *local-search heuristic* (blue triangles) and the *combination* (green diamonds) for the SPs. The upper figure shows the objective values for the different instances and the lower shows the computation times. A time cap of 300 seconds was used for the solutions and the missing data is where the methods failed to reach the stopping criteria within that time limit. The numbers along the x-axis denotes the different parameter settings in the order they appear in Table A.5 in Appendix A.

8

Conclusion

In this thesis, a problem of routing shared autonomous vehicles dynamically has been presented and translated into a mathematical model. A number of solution methods are proposed, and computational experiments has been run to test their efficiency. The solution methods use a rolling horizon reoptimisation scheme, as well as an *insertion heuristic* to handle the dynamic nature of the problem. Further, a *column generation* method has been created to solve the static *snapshot fleet dispatching problems* within the reoptimisation scheme.

Utilising a rolling horizon to handle the dynamics of pickup-and-delivery problems has been used before by Mitrovic-Minic and Laporte [33], but in most other aspects their modelling and solution methods differs from what is presented here. First of all, in this thesis, contrary to the classic dial-a-ride problem, the fleet dispatching is modelled over a graph which represents an actual road network. This means that a vehicle, which has started a journey, can, if new information is revealed, change its path at the next intersection instead of at its next customer node (which would be its next node in the dial-a-ride problem formulation). This makes our problem formulation more realistic and more flexible. Further, we present a *column generation* scheme in conjunction with the rolling horizon algorithm, which, to the best of the authors knowledge, is the first of its kind. Also, it presents a *labelling algorithm* for the snapshot fleet dispatching problems, which differs from earlier problems solved by labelling algorithms both in terms of constraints and objective function.

The computational experiments shows that for small instances of the *snapshot fleet dispatching problem*, the presented *column generation* scheme generates close to optimal solutions, both when using the *labelling algorithm* and the *local-search heuristic* to solve the subproblems. For those instances the difference between the objective value for the integer restricted master problem and its lower bound is very small, especially for the *labelling algorithm*, which indicates that for such small problems, a *branch-and-price* method would be unnecessary. For the large instances of the *snapshot fleet dispatching problem*, the difference between the objective value of the integer restricted master problem and the restricted master problem increases, especially for instances with large values of T. This is less of a problem though, as the *snapshot fleet dispatching problem* is supposed to be used in conjunction with

the fleet dispatching problem where the reoptimisation horizon does not need to be that long. It has also been shown that the *local-search heuristic* and the *labelling algorithm* can be used together successfully, yielding better results in terms of both objective value and computation time, in comparison to only using the *labelling algorithm*.

The computational experiments for the *fleet dispatching problem* shows that the *insertion heuristic* for new requests improves the objective values significantly, especially for high values of T_{reopt} but also seem to improve the computation times. Due to the random nature of dynamic problems, it is hard to judge the impact of the solution to the *snapshot fleet dispatching problems* on the solution to the *fleet dispatching problems*.

9

Reflections and Outlook

The problem formulated and the solution methods presented in this thesis are not quite yet at a stage where they could be directly implemented in practice. It is an interesting approach, however, that with some tweaks and a somewhat more advanced model, better describing reality, might be useful. In reality, for example, the travelling times along roads are never fixed and cannot be known in advance. The problem formulation presented in this thesis is, however, well suited to handle such difficulties as the estimate of travelling times can be updated before each reoptimisation step. Nor will other changes, such as customers changing their destinations during their rides, be difficult to handle, as the formulation with the reoptimisation method is very flexible.

9.1 The MATSim Heritage

The masters project reported in this thesis was initially planned to create an optimisation tool for a fleet of SAVs in the transport model software MATSim (see [39]). Since the MATSim interface was not in a state where such a project could be carried out with a satisfactory result, however, the project took a turn towards a more mathematical problem formulation. This is noteworthy though, as some of the model decisions remain optimised for the MATSim framework. Examples of such decisions are the complete focus on travelling and waiting times, the lack of tight time windows and the very short time gaps between when the customers notify the system and when they want to start their journeys.

If the solution procedures presented in this thesis are to be used in practice, those things need to be changed. The change in objective function and notification times are no major changes and they could rather easily be implemented in the presented solution methods. The change to tight time windows would, however, be more difficult, especially if the time windows are hard, as there is no way to guarantee the solution to be feasible due to the problem's dynamic nature.

9.2 Outlook

In addition to changes in the modelling, as discussed above, there are three areas of the solution procedure on which it would be possible to improve upon. The first area is the solution of the SPs. Both stronger dominance and arc elimination criterion could speed up the solution procedure for the *labelling algorithm*. For the heuristics, it would be interesting to test a *tabu search algorithm* to try to find better solutions than those found by the *local-search heuristic*, and faster than the *labelling algorithm*. The second area would be to implement a *branch-and-price* algorithm. Even though much points towards it being superfluous, it would be interesting to see how it would affect the computation times and the objective values, especially for instances with longer maximum route length. The last area is the *insertion heuristic*. If many customers notify the system within a short time frame, those customers could, for example, be inserted together, yielding better results than if they were inserted one at a time.

In addition to these changes, which could be considered as tweaks to the presented solution procedure, there are a lot of other possible improvements. Especially, in practice it is likely that one has some kind of information that makes it possible to predict future requests. Such information could be utilised to relocate empty vehicles, use waiting strategies, and so on, to try to improve the solutions.

It is highly likely that when the fleets of SAVs arrive they will mostly consist of electric vehicles. It would hence be of great interest to including recharging stations in the road network and to apply charge state constraints on the vehicles in future models.

Bibliography

[1] Marcus P Enoch. How a rapid modal convergence into a universal automated taxi service could be the future for local passenger transport. *Technology Analysis & Strategic Management*, 27(8):910–924, 2015.

[2] Todd Litman. *Evaluating public transit benefits and costs*. Victoria Transport Policy Institute, Victoria, Canada, 2015.

[3] Javier Alonso-Mora, Samitha Samaranayake, Alex Wallar, Emilio Frazzoli, and Daniela Rus. On-demand high-capacity ride-sharing via dynamic trip-vehicle assignment. *Proceedings of the National Academy of Sciences of the United States of America*, pages 462–467, 2017.

[4] George B Dantzig and John H Ramser. The truck dispatching problem. *Management Science*, 6(1):80–91, 1959.

[5] Jean-François Cordeau. A branch-and-cut algorithm for the dial-a-ride problem. *Operations Research*, 54(3):573–586, 2006.

[6] Cynthia Barnhart, Ellis L Johnson, George L Nemhauser, Martin WP Savelsbergh, and Pamela H Vance. Branch-and-price: Column generation for solving huge integer programs. *Operations Research*, 46(3):316–329, 1998.

[7] Jean-Francois Cordeau, Paolo Toth, and Daniele Vigo. A survey of optimization models for train routing and scheduling. *Transportation Science*, 32(4):380–404, 1998.

[8] Anastassions N Perakis and D I Jaramillo. Fleet deployment optimization for liner shipping Part 1. Background, problem formulation and solution approaches. *Maritime Policy and Management*, 18(3):183–200, 1991.

[9] Jan Van Leeuwen. *Handbook of Theoretical Computer Science (vol. A): Algorithms and Complexity*. MIT Press, Cambridge, Massachusetts, 1991.

[10] Paolo Toth and Daniele Vigo. *Vehicle routing: Problems, Methods, and Applications*. SIAM, Philadelphia, 2014.

[11] Gerardo Berbeglia, Jean-François Cordeau, and Gilbert Laporte. Dynamic pickup and delivery problems. *European Journal of Operational Research*, 202(1):8–15, 2010.

[12] Harilaos N Psaraftis. A dynamic programming solution to the single vehicle many-to-many immediate request dial-a-ride problem. *Transportation Science*, 14(2):130–154, 1980.

[13] Harilaos N Psaraftis. An exact algorithm for the single vehicle many-to-many dial-a-ride problem with time windows. *Transportation Science*, 17(3):351–357, 1983.

[14] Richard Bellman. *Dynamic programming*. Courier Corporation, 2013.

[15] Jacques Desrosiers, Yvan Dumas, and François Soumis. A dynamic programming solution of the large-scale single-vehicle dial-a-ride problem with time windows. *American Journal of Mathematical and Management Sciences*, 6(3-4):301–325, 1986.

[16] Yvan Dumas, Jacques Desrosiers, and Francois Soumis. The pickup and delivery problem with time windows. *European Journal of Operational Research*, 54(1):7–22, 1991.

[17] Martin Savelsbergh and Marc Sol. DRIVE: Dynamic routing of independent vehicles. *Operations Research*, 46(4):474–490, 1998.

[18] Hang Xu, Zhi-Long Chen, Srinivas Rajagopal, and Sundar Arunapuram. Solving a practical pickup and delivery problem. *Transportation Science*, 37(3):347–364, 2003.

[19] Mikkel Sigurd, David Pisinger, and Michael Sig. Scheduling transportation of live animals to avoid the spread of diseases. *Transportation Science*, 38(2):197–209, 2004.

[20] Sophie N Parragh and Verena Schmid. Hybrid column generation and large neighborhood search for the dial-a-ride problem. *Computers & Operations Research*, 40(1):490–497, 2013.

[21] Jean-François Cordeau. A branch-and-cut algorithm for the dial-a-ride problem. *Operations Research*, 54(3):573–586, 2006.

[22] Stefan Ropke, Jean-François Cordeau, and Gilbert Laporte. Models and branch-and-cut algorithms for pickup and delivery problems with time windows. *Networks*, 49(4):258–272, 2007.

[23] Stefan Ropke and Jean-François Cordeau. Branch and cut and price for the

pickup and delivery problem with time windows. *Transportation Science*, 43(3):267–286, 2009.

[24] Jang-Jei Jaw, Amedeo R Odoni, Harilaos N Psaraftis, and Nigel H M Wilson. A heuristic algorithm for the multi-vehicle advance request dial-a-ride problem with time windows. *Transportation Research Part B: Methodological*, 20(3):243–257, 1986.

[25] Paolo Toth and Daniele Vigo. Fast local search algorithms for the handicapped persons transportation problem. In *Meta-Heuristics*, pages 677–690. Springer, 1996.

[26] Paolo Toth and Daniele Vigo. Heuristic algorithms for the handicapped persons transportation problem. *Transportation Science*, 31(1):60–71, 1997.

[27] Jean-François Cordeau and Gilbert Laporte. A tabu search heuristic for the static multi-vehicle dial-a-ride problem. *Transportation Research Part B: Methodological*, 37(6):579–594, 2003.

[28] Stefan Ropke and David Pisinger. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation Science*, 40(4):455–472, 2006.

[29] Harilaos N Psaraftis, Min Wen, and Christos A Kontovas. Dynamic vehicle routing problems: Three decades and counting. *Networks*, 67(1):3–31, 2016.

[30] Oli Madsen, Hans F Ravn, and Jens Moberg Rygaard. A heuristic algorithm for a dial-a-ride problem with time windows, multiple capacities, and multiple objectives. *Annals of Operations Research*, 60(1):193–208, 1995.

[31] Michael R Swihart and Jason D Papastavrou. A stochastic and dynamic model for the single-vehicle pick-up and delivery problem. *European Journal of Operational Research*, 114(3):447–464, 1999.

[32] Pascal Van Hentenryck and Russell Bent. *Online Stochastic Combinatorial Optimization*. The MIT Press, Cambridge, Massachusetts, 2009.

[33] Snežana Mitrović-Minić and Gilbert Laporte. Waiting strategies for the dynamic pickup and delivery problem with time windows. *Transportation Research Part B: Methodological*, 38(7):635–655, 2004.

[34] Snežana Mitrović-Minić, Ramesh Krishnamurti, and Gilbert Laporte. Double-horizon based heuristics for the dynamic pickup and delivery problem with time windows. *Transportation Research Part B: Methodological*, 38(8):669–685, 2004.

[35] Doris Sáez, Cristián E Cortés, and Alfredo Núñez. Hybrid adaptive predictive control for the multi-vehicle dynamic pick-up and delivery problem based on genetic algorithms and fuzzy clustering. *Computers & Operations Research*, 35(11):3412–3438, 2008.

[36] Miyoung Han, Pierre Senellart, Stéphane Bressan, and Huayu Wu. Routing an autonomous taxi with reinforcement learning. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, pages 2421–2424. ACM, 2016.

[37] Daniel J Fagnant and Kara M Kockelman. The travel and environmental implications of shared autonomous vehicles, using agent-based model scenarios. *Transportation Research Part C: Emerging Technologies*, 40:1–13, 2014.

[38] Daniel J Fagnant, Kara M Kockelman, and Prateek Bansal. Operations of shared autonomous vehicle fleet for Austin, Texas, market. *Transportation Research Record: Journal of the Transportation Research Board*, (2536):98–106, 2015.

[39] Andreas Horni, Kai Nagel, and Kay W Axhausen. The Multi-Agent Transport Simulation MATSim. *Ubiquity Press, London, UK*, vol. 9, 2016.

[40] Michal Maciejewski, Joschka Bischoff, and Kai Nagel. An assignment-based approach to efficient real-time city-scale taxi dispatching. *IEEE Intelligent Systems*, 31(1):68–77, 2016.

[41] Gerard Sierksma. *Linear and Integer Programming: Theory and Practice*. CRC Press, New York, 2001.

[42] Michael Patriksson, Niclas Andréasson, Anton Evgrafov, Emil Gustavsson, and Magnus Önnheim. *Introduction to Continuous Optimization*. Studentlitteratur AB, Lund, 2013.

[43] Paul C Gilmore and Ralph E Gomory. A linear programming approach to the cutting-stock problem. *Operations Research*, 9(6):849–859, 1961.

[44] A. Schrijver. *Theory of Linear and Integer Programming*. John Wiley & Sons, Hoboken, New Jersey, 1986.

[45] Der-San Chen, Robert G Batson, and Yu Dang. *Applied Integer Programming: Modeling and Solution*. John Wiley & Sons, Hodoken, New Jersey, 2011.

[46] Michel Gendreau and Jean-Yves Potvin. *Handbook of Metaheuristics*, volume 2. Springer, 2010.

[47] Marco E Lübbecke. Column generation. *Wiley Encyclopedia of Operations*

Research and Management Science, 2011.

[48] Eric W Weisstein. Floyd-Warshall algorithm. *MathWorld*, 2008. http://mathworld.wolfram.com/Floyd-WarshallAlgorithm.html.

[49] CPLEX. https://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/. Accessed: September 28, 2017.

[50] COIN-OR branch and cut. https://projects.coin-or.org/Cbc, note=Accessed: September 28, 2017.

[51] PuLP. https://pythonhosted.org/PuLP/. Accessed: September 28, 2017.

[52] Der-Tsai Lee and Bruce J Schachter. Two algorithms for constructing a Delaunay triangulation. *International Journal of Computer & Information Sciences*, 9(3):219–242, 1980.

A

Appendix

Table A.1: Results for the small instances of the SFDP when the SPs are solved using the *labelling algorithm*, the *local-search heuristic* and the *combination* of the two for the SPs. $\underline{z}$ and z_{LP} denote, respectively, the objective values of the solutions of the RMP; for the *labelling algorithm* and the *combination algorithm*, these values are lower bounds for objective values of the corresponding MPs, while, when using the *local-search heuristic*, they are not. z denote the objective values for the solutions to the IMPs. t denotes the computation time.

			Labelling			Insertion			Combination	
		#	$\underline{z}$	$z/\underline{z}$	t [s]	$z_{LP}/\underline{z}$	$z/\underline{z}$	t [s]	$z/\underline{z}$	t [s]
m:	2	1	3032.6	1	1.7	1.0083	1.012	0.3	1	1.1
n:	10	2	2678.7	1	5.7	1	1	0.7	1	1.6
l:	3	3	2721.9	1	9	1.0059	1.0061	0.7	1	8.3
C:	2	4	2582.2	1	13.7	1.0122	1.0122	0.8	1	7.3
T:	300	5	2420.1	1	5.5	1.0011	1.0011	0.7	1	3.7
m:	5	1	3316.9	1.008	27.5	1.0075	1.0075	1.9	1.0075	8.9
n:	20	2	3647.0	1	19.6	1.004	1.004	1.5	1	10.4
l:	5	3	3774.2	1.001	5.1	1.0063	1.0063	0.8	1.0001	5.2
C:	2	4	3862.2	1	4.4	1.0044	1.0044	0.7	1	4
T:	200	5	3516.1	1	25.8	1.0067	1.0067	0.9	1.0002	12.3
m:	5	1	2853.7	1	268	1.018	1.0336	2	1	182
n:	15	2	3361.4	1	10.6	1	1	0.9	1	7.5
l:	10	3	3415.8	1	71.5	1.0061	1.0061	1.5	1	51.1
C:	4	4	3135.1	1	74.5	1.0047	1.0048	1.7	1	32.6
T:	200	5	2983.8	1	131.1	1.0237	1.025	2.2	1	74.9
m:	20	1	5462.4	1	7	1.0027	1.0027	3.6	1.0004	6.2
n:	5	2	5660.3	1	6.9	1	1	3.9	1	6.3
l:	50	3	5135.4	1	10.7	1.0072	1.0072	3.3	1	8
C:	4	4	5488.9	1	4.1	1.0002	1.0002	3.5	1	5.8
T:	200	5	5702.3	1	15	1.001	1.001	3	1	7.3
m:	5	1	3557.9	1	43	1.0027	1.0027	1.9	1	17.8
n:	20	2	3939.5	1	43.3	1.0024	1.0024	1.7	1	17
l:	5	3	4046.6	1.001	17.5	1.0015	1.0015	1.3	1.0015	9.5
C:	4	4	4150.2	1	9.7	1.0021	1.0021	1.2	1	7.8
T:	225	5	3721.0	1.004	81.2	1.0152	1.0152	1.6	1.0042	53.3

Table A.2: Results for the large instances of the SFDPs solved using the *local-search heuristic* for the SPs. z denote the objective values of the IRMPs and z_{LP} denote the objective values of the RMPs. t denotes the computation times.

			Local-Search Heuristic		
instance		#	z_{LP}	z/z_{LP}	t [s]
m:	20	1	13 376.3	1.025	453.2
n:	80	2	11 759.8	1.026	578.3
l:	0	3	11 369.4	1.04	708.3
C:	2	4	11 981.3	1.032	650.7
T:	400	5	12 606.1	1.038	796.4
m:	15	1	16 685.3	1.05	1361
n:	100	2	17 670.1	1.039	570.6
l:	0	3	18 552.5	1.029	431.7
C:	2	4	18 010.5	1.047	569.9
T:	300	5	18 007.5	1.032	366.8
m:	20	1	19 352.7	1.025	402.9
n:	60	2	18 409.8	1.019	320
l:	60	3	18 908.9	1.023	415.3
C:	4	4	18 219.0	1.021	431.8
T:	350	5	17 152.4	1.027	422.3
m:	5	1	20 148.2	1.131	259.2
n:	60	2	20 782.3	1.105	262.8
l:	0	3	20 164.8	1.132	315.4
C:	2	4	19 281.5	1.154	219
T:	600	5	19 230.9	1.123	261.9
m:	20	1	13 152.5	1.013	48.7
n:	80	2	13 174.0	1.005	45
l:	20	3	14 521.3	1.002	22.5
C:	2	4	13 033.4	1.011	49.6
T:	200	5	13 818.1	1.007	47.1
m:	60	1	8745.9	1.001	164.1
n:	60	2	8018.3	1.001	230.6
l:	30	3	8325.7	1.001	199.8
C:	2	4	8694.8	1.002	166.8
T:	300	5	7998.2	1.001	243.7

Table A.3: Results for the set of small instances solved as dynamic and static problems. The dynamic versions are solved using two methods, the reoptimisation scheme as well as the reoptimisation scheme combined with the *insertion heuristic* for new customers In all solutions the *labelling algorithm* was used for the SPs. z denotes the best objective values found and t the computation times. For each of the instances, a vehicle capacity of 2, graphs with 30 nodes and a T_{horizon} of 200 has been used.

instance		#	Reoptimisation		Reoptimisation with the insertion heuristic		Static	
			z	t $[s]$	z	t $[s]$	z	t $[s]$
m :	5	1	1670.0	1.0	1564.0	0.9	1233.0	3.9
n :	15	2	1856.0	1.1	1743.0	0.9	1583.0	1.7
T :	200	3	1697.0	1.2	1292.0	0.8	1189.0	13.9
T_{reopt} :	50	4	1808.0	1.3	1765.0	1.5	1475.0	5.4
		5	1856.0	1.5	1728.0	1.3	1398.0	3.7
m :	5	1	1811.0	2.8	1466.0	2.6	1269.0	124.9
n :	15	2	1672.0	1.9	1559.0	1.6	1167.0	56.5
T :	250	3	1633.0	3.1	1325.0	1.5	1076.0	2671.6
T_{reopt} :	50	4	1670.0	3.2	1674.0	2.6	1290.0	44.9
		5	1689.0	2.1	1513.0	1.2	1264.0	163.2
m :	10	1	2100.0	2.5	1799.0	1.0	1421.0	286.3
n :	20	2	2035.0	3.7	1588.0	1.2	1344.0	421.5
T :	200	3	1920.0	1.9	1531.0	0.9	1190.0	109.9
T_{reopt} :	50	4	2064.0	2.7	1641.0	1.2	1250.0	408.5
		5	1874.0	19.5	1327.0	1.5	1132.0	2582.8
m :	5	1	1759.0	2.9	1819.0	3.2	1510.0	2.1
n :	15	2	1517.0	3.0	1453.0	3.2	1346.0	20.2
T :	200	3	1324.0	2.5	1324.0	2.4	1089.0	9.9
T_{reopt} :	20	4	1405.0	2.8	1252.0	2.5	1078.0	47.8
		5	1561.0	3.1	1538.0	2.6	1319.0	26.7
m :	5	1	1498.0	6.8	1549.0	7.0	1305.0	34.5
n :	15	2	1171.0	5.4	1111.0	4.8	945.0	49.2
T :	200	3	1394.0	5.6	1380.0	5.6	1179.0	10.3
T_{reopt} :	10	4	1469.0	5.2	1388.0	4.8	1234.0	19.9
		5	1592.0	5.9	1562.0	5.6	1400.0	2.7

Table A.4: Results for the set of large instances. Each instance has been solved using the reoptimisation scheme as well as the combination of the reoptimisation and the *insertion heuristic*. For both of them, the *local-search heuristic* was used for the SPs. These results clearly indicates that the combination is a better algorithm, in terms of both objective value and computation time. For each of the instances, a vehicle capacity of 2 and graphs with 30 nodes has been used.

			Reoptimisation		reopt + insert	
instance		#	z	t $[s]$	z	t $[s]$
m :	5	1	13 207.0	12.6	11 926.0	10.7
n :	60	2	12 946.0	16.0	11 989.0	15.8
T :	800	3	9415.0	14.3	8372.0	11.5
T_{reopt} :	50	4	10 822.0	17.3	9904.0	12.8
T_{horizon} :	200	5	10 815.0	14.7	9733.0	15.0
m :	5	1	10 472.0	11.6	9147.0	9.7
n :	60	2	11 188.0	11.9	10 438.0	11.1
T :	1000	3	8588.0	13.5	7590.0	11.5
T_{reopt} :	50	4	10 317.0	14.3	9276.0	10.5
T_{horizon} :	200	5	9225.0	13.1	8688.0	10.6
m :	10	1	9991.0	27.2	8454.0	15.6
n :	80	2	11 512.0	38.3	9330.0	21.9
T :	800	3	9734.0	38.1	7861.0	18.1
T_{reopt} :	50	4	9122.0	27.3	7331.0	13.6
T_{horizon} :	300	5	9272.0	24.7	7906.0	15.2
m :	5	1	6953.0	8.0	5348.0	6.3
n :	40	2	6021.0	7.0	5515.0	6.6
T :	600	3	7208.0	7.6	6028.0	7.0
T_{reopt} :	50	4	7368.0	8.9	6501.0	6.3
T_{horizon} :	200	5	6669.0	8.0	6383.0	7.3
m :	10	1	9027.0	23.6	7943.0	17.7
n :	80	2	11 426.0	28.5	10 006.0	26.2
T :	800	3	11 113.0	24.9	10 197.0	20.5
T_{reopt} :	30	4	8970.0	28.0	7918.0	19.4
T_{horizon} :	200	5	9764.0	24.2	9642.0	22.6

Table A.5: Results for a set of large instances of the FDP, each solved using the reoptimisation scheme and the *insertion heuristic* for new requests and with the *labelling algorithm*, the *local-search heuristic* and the *combination* for the SPs. z denotes the best objective values found and t the computation times. A solution time cap of 300 seconds has been used and fields for instances which did not reach its stopping criteria in that time are marked as empty. For each of the instances, a vehicle capacity of 2 and graphs with 30 nodes has been used.

instance		#	Labelling		Heuristic		Combination	
			z	t $[s]$	z	t $[s]$	z	t $[s]$
m :	5	1	11 871.0	69.8	11 926.0	10.4	11 775.0	40.8
n :	60	2	11 408.0	224.9	11 989.0	15.5	11 408.0	116.2
T :	800	3			8372.0	11.3	8465.0	148.1
T_{reopt} :	50	4	9676.0	106.2	9904.0	12.5	9170.0	31.3
T_{horizon} :	200	5	9688.0	118.5	9733.0	14.8	9688.0	76.7
m :	5	1	9058.0	14.0	9147.0	9.3	8999.0	16.9
n :	60	2	10 091.0	28.9	10 438.0	10.7	10 091.0	20.1
T :	1000	3	7186.0	29.5	7590.0	11.1	7966.0	21.7
T_{reopt} :	50	4	8679.0	33.3	9276.0	10.1	8659.0	25.0
T_{horizon} :	200	5	7871.0	20.1	8688.0	10.2	7871.0	17.5
m :	10	1			8452.0	15.1		
n :	80	2			9330.0	21.6		
T :	800	3			7861.0	17.9		
T_{reopt} :	50	4			7331.0	13.4	7458.0	192.0
T_{horizon} :	300	5			7906.0	15.0		
m :	5	1	5384.0	12.6	5348.0	6.1	5384.0	10.1
n :	40	2	5346.0	8.0	5515.0	6.3	5501.0	8.2
T :	600	3	5999.0	11.7	6028.0	6.8	6061.0	8.4
T_{reopt} :	50	4	6291.0	11.2	6501.0	6.1	6338.0	12.2
T_{horizon} :	200	5	6224.0	10.6	6383.0	7.0	6224.0	9.5
m :	10	1	7834.0	47.5	7943.0	17.3	7870.0	25.8
n :	80	2			10 006.0	25.6		
T :	800	3	10 199.0	71.6	10 197.0	19.8	10 013.0	47.4
T_{reopt} :	30	4	7676.0	97.0	7918.0	18.6	7707.0	55.7
T_{horizon} :	200	5	8772.0	43.5	9642.0	22.2	8872.0	32.3