A 4th Party Logistics Problem Optimizer

A thesis presented for

the Master of Science in Informatics

by

Preben Bucher Johannessen

Department of Informatics

University of Bergen

Norway

January 16, 2020

A 4th Party Logistics Problem Optimizer

Preben Bucher Johannessen

January 16, 2020

Abstract

This thesis considers a pickup and delivery problem with multiple time windows, a complex cost structure and factory constraints. We formulated the problem as a mathematical model and created an instance generator based on real data. We also implemented a heuristic solution method for the problem and ran extensive statistical tests. The mathematical model shows the complexity of the problem and is implemented in AMPL to give a benchmark for the proposed solution method. The instance generator was created based on real anonymized data from a 4th party logistics (4PL) company. The proposed solution method, called the 4th Party Logistics Optimizer, is a meta-heuristic approach with industry specific implementations. The solution method is refined through extensive statistical experiments. The experiments determine which parts of the solution method have a significant positive impact on the objective value. This leads to a final composition of our solution method. The final solution method is robustly giving near optimal solutions to realistic sized instances in seconds, and is a powerful tool for companies facing the proposed adaptation of the pickup and delivery problem.

Acknowledgements

First and foremost I would like to thank my professor Dr. Ahmad Hemmati, for without his counselling, guidance and expertise this work would not be possible. I would also like to thank 4flow and in particular Lars Stolletz for their cooperation on this thesis and for providing the data and the details on the problem. Thank you also to the Institute of Informatics and all my co-students who has made my Masters degree an incredibly enjoyable time of my life.

A big thanks also to Philipp Zschoche, at the Technische Universität Berlin, for reading through every chapter and helping me with corrections, exchanging ideas and sharing experience.

“Behind every great man there’s a great woman” was written in The Port Arthur News (1946), and I do not know about the great man, but I could not have written this thesis without the support and sacrifice of my wife Rosa Sophie and our newborn son Bonifazius Louie.

Contents

List of Tables

List of Figures

List of Algorithms

Notations

The following notations will be used throughout the thesis. For further mathematical expressions we refer to Chapter 2.

Mathematical expressions

Notation	*Description*
s	solution.
S_v	The vehicle schedule of vehicle v.
S_{vjk}	A sub part of a vehicle schedule of vehicle v starting at index j ending at index k.
f	objective function returning the total cost in problem.
$f(s)$	objective value, or total cost, of solution s.
$f(S_v)$	objective value of vehicle schedule S_v.
N^P	orders or pickup locations in problem.
C_i^S	Difference in objective value $f(S_v)$ for a vehicle schedule S_v when excluding order $i \in N^P$ from the schedule.
c_i	minimum increase in objective function f for adding order $i \in N^P$.
c_i^*	regret value of order $i \in N^P$.
r_{ij}	relatedness factor between two orders $i, j \in N^P$.
p	random degree factor.
y^p	used to select which index to pick in an ordered set.
h	A (neighbourhood) heuristic.
$\mathbb{H}$	A set of heuristics.
$\mathbb{Z}$	A set of integers.
$[1, 2, .., n]$	Indicate an integer set starting with 1 and ending at n.
$[0, 1]$	Indicate a set of all real numbers from 0 to 1.

Abbreviations

Notation	*Description*
4PL	4th Party Logistics
4PLP	4th Party Logistics Problem
4PLO	4th Party Logistics Optimizer
TSP	Travelling Salesman Problem
VRP	Vehicle Routing Problem
DARP	Dial-a-ride Problem
PDP	Pickup and Delivery Problem
PDPTW	Pickup and Delivery Problem with Time Windows
PDPMTW	Pickup and Delivery Problem with Multiple Time Windows

Chapter 1

Introduction

The auto-mobile and machinery manufacturing industry is under constant change in today's conditions of increased cost pressure and international competition, 4flow AG (2019). This has made supply chain and logistics of paramount importance, and the need for external expertise for these manufacturers have given rise to 4th party logistics (4PL) companies. The demand for high flexibility in the planning and modelling of transportation has made it more important than ever for these companies to have efficient and robust solutions to the *routing problem* they want to solve. The routing problem a 4PL company is facing is a type of *Pickup and Delivery Problem with Time Windows* (PDPTW) with additional requirements. In this thesis we will present a mathematical model, an instance generator and a solution method to a *Pickup and Delivery Problem with Multiple Time Windows* (PDPMTW), *factory constraints* and a *complex cost structure* typically faced by a 4PL company. We name the problem presented in this thesis the *4th Party Logisitics Problem* (4PLP).

Thesis Outline

Section 1.1 in this chapter will introduce 4flow AG, a 4PL planning company, which has contributed to the research done in this thesis. We will explain the problems a 4PL planning company are facing when planning transport for their customers, based on their typical manufacture customers in Germany and Europe and give a brief overview over why the problem they are facing is different than other pickup and delivery problems. Although we focus on the 4PL industry the problem will be on a general form and can be applied to many other industries as well. Then we will move into a literature review in Section 1.2 of research made surrounding other famous routing problems.

The rest of the thesis will be divided in the following chapters. Chapter 2 will present description of the 4PLP problem and the mathematical formulation of the 4PLP. Chapter 3 will present known solution methods, both exact and meta-heuristics, to problems similar to our 4PLP. Chapter 4 is a detailed outline of our 4PLO model. Chapter 5 then presents the experimental results that lead us to the final composition of our 4PLO model. Here we also analyse the performance of the final composition of the 4PLO in comparison to the benchmark results of implementing our mathematical model in AMPL. Chapter 6 finally concludes the work and will present the academic contributions of this thesis and possible future areas of research.

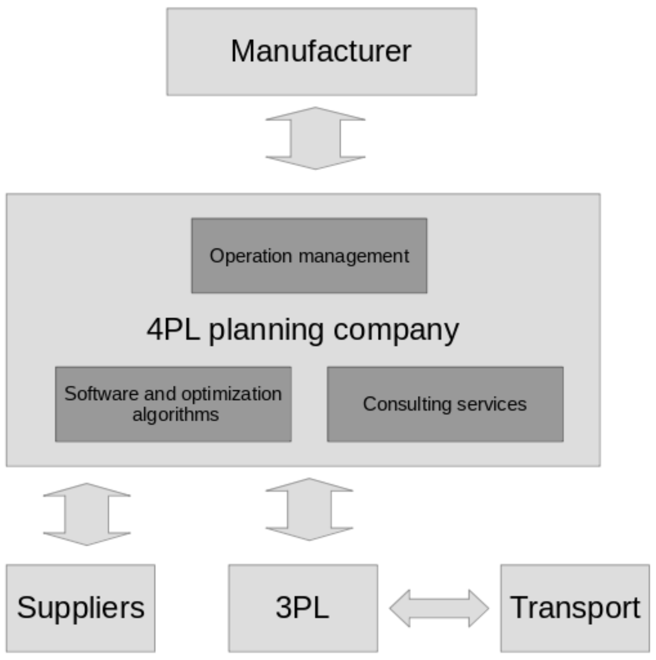


Figure 1.1: The figure tries to illustrate the role of a 4PL planning company related to a manufacturer (the customer), suppliers, the third-party logistics provider (3PL) and the transport companies. The arrows represent the flow of data and information.

1.1 4flow, A 4PL Planning Company

This thesis will present a model to solve a special routing problem we call the 4th Party Logistics Problem (4PLP), however the problem will be formulated in a general manner and can be applied to many other industries apart from the 4th Party Logistics planning industry.

The work in this thesis is done in cooperation with a German company called 4flow AG. 4flow is a 4PL consulting, management and software company located on every continent with main office in Berlin, Germany. A Fourth Party Logistic Model (4PL) is defined in logistics as follows. The manufacturer does not only outsource the organisation of its logistic tasks to third parties, but also the management thereof. Fourth party logistic service providers often check the entire supply chain. The organisational and executive activities are again often outsourced to other parties, HM Group (2019).

Figure 1.1 tries to illustrate a 4PL company and its role compared to a manufacturer (its customer), 3PL company, supplier and the transportation company. It shows how 4PL becomes a manager of data from all sources related to transportation and plans on behalf of the manufacturer how to manage all these resources.

For a company like 4flow, having software with reliable algorithms that can find near optimal solutions for a transportation or *routing problem* is the key to success. A 4PL planning company is not the manufacturer itself. Neither is it the transportation company or the third-party logistics provider (3PL). It is managing all these resources to ensure the most efficient way of handling the transportation

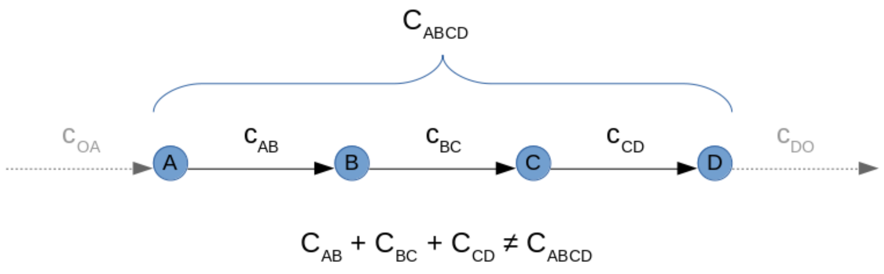


Figure 1.2: The figure illustrates an example of the spot prices when using a transport company to cover different stops in a vehicles schedule. The 4PL company does not pay for getting the vehicle from an origin O and back, C_{OA} and C_{DO}. The costs of the whole vehicle schedule C_{ABCD} is not the same as the sum of C_{AB}, C_{BC} and C_{CD}, but depend on the distance travelled, the weight of the transport and the fixed costs of hiring the transport.

for a manufacturer.

The role of a 4PL planning company shifts the problem away from classical pickup and delivery problems in several ways. Most importantly since a 4PL company is not handling its own fleet of vehicles, it relies on transport companies and their spot prices.

The implications this has for a 4PL company's cost structure have been illustrated in Fig. 1.2. The first implication to notice from the figure, is that picking up an *order* at A and delivering it at B is no longer the cost from origin O to A then to B and back to O. The price is only paid from the pickup at A until delivery at B. Another implication is that the spot price of moving from A to B to C to D is not simply the sum of the costs of each singular transport $C_{AB} + C_{BC} + C_{CD}$. Rather is it calculated based on the total *vehicle schedule* $ABCD$ and can depend on several factors. These factors are most commonly distance, transported weight plus some fixed hiring cost. In addition to this the transport company might charge the manufacturer a *stop cost* fee per location visited. To make it even more complex all these prices can vary between vehicle types and different types of transport carriers. The price will also be different if the selected vehicle is LTL (less than truckload) of FTL (full truckload). We refer to this as the *complex price structure* ahead.

Another implication from Fig. 1.2 is that the problem is not bound to a fixed amount of vehicles, since there is no fleet owned by the 4PL company or the manufacturer its self. Theoretically a 4PL company could send all orders separately, like you would send a package from your local postal office, however this would be very cost inefficient. Planning good routes based on the given spot prices can save a significant amount of costs and have an impact on the climate in the long run and that is why optimization is so important in this industry.

Since the 4PL planning company is dealing with real manufacturers and suppliers with normal working hours and is planning to optimize the pickup and delivery schedule for several days, sometimes weeks, it cannot simply plan that transport arrives at the pickup and delivery location at any given time. It needs to stick to a time window and because the planning might take place over several days it needs to abide to *multiple time windows.*

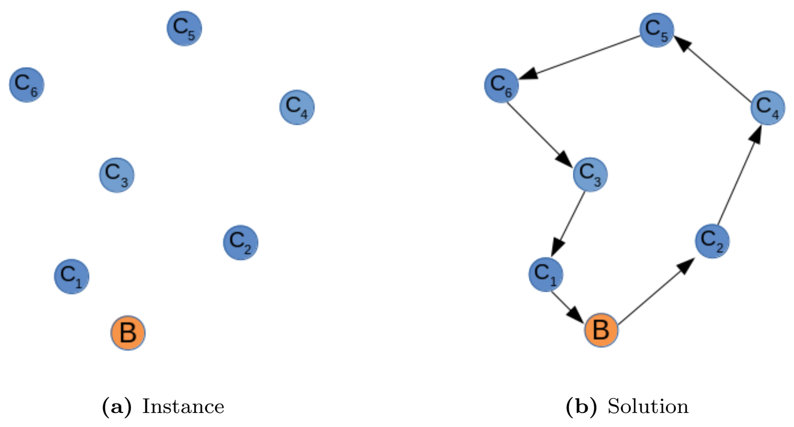


Figure 1.3: The figure illustrates an instance of a Travelling Salesman Problem with a solution. C_i represent a customers location where $i = [1..6]$. B indicates the base or starting location for the salesman.

Another aspect which makes the 4PLP different from other vehicle routing problems is that since the manufacturer is the customer of the 4PL planning company, they sometimes have specific wishes for how the delivery should proceed. Some manufacturers have factories with several docks. Each dock might require specific vehicles with a specific size or special equipment. This indicates that the 4PLP must have *heterogeneous vehicles*.

The manufacturer might also require that each vehicle only visits a certain amount of docks before leaving a factory again. The reason behind this would be to reduce traffic and avoid delivery trucks getting lost inside a factory area. We refer to this as the *factory dock constraint*.

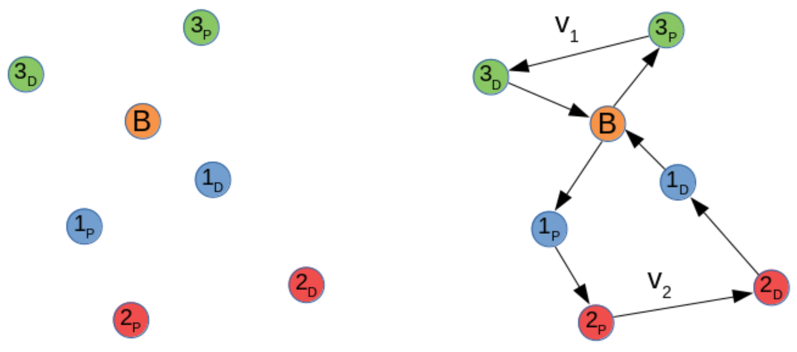
All the aspects mentioned above needs to be considered when solving a 4PLP. We will go into more details and formulate mathematically the differences mentioned above in Chapter 2. In the next section we will take a look at what literature have been published with similar routing problems and how that can benefit us when we solve the 4PLP.

1.2 Literature Review

To understand how to solve the 4PLP we will now look at what research has been made surrounding similar routing problems.

The Travelling Salesman Problem, or TSP, was formulated as the Truck Dispatching Problem by Dantzig and Ramser (1959) and is classified as NP-hard, meaning that by increasing the size of the problem the complexity quickly becomes so high that a computer has difficulty finding an exact solution to the problem. The TSP is the problem a sales person might face when having to visit several different customers on different locations in the most efficient way possible, but as in Dantzig and Ramser (1959) and later Clarke and Wright (1964) it can be modified to apply to multiple demands and trucks with capacities. As in Lin (1965), they implemented an exact approach for smaller instances and applied heuristics to solve larger instances.

Figure 1.3 illustrates the TSP with a solution where each node C_i represents a



(a) Instance **(b)** Solution

Figure 1.4: The figure illustrates an instance of the Pickup and Delivery Problem with a solution. The numbers represent orders and P and D represent the orders pickup and delivery location. B represent base or the starting point of the two vehicles V_1 and V_2.

customer that a salesperson wants to visit in the most efficient way possible. The salesperson starts from his base B and will also return to the same base.

There are many extensions of the TSP problem. One of them is the *Vehicle Routing Problem* (VRP) where each vehicle starts at a base, as in the TSP, but the vehicles have a limited travel length, or time, and must return before they violate this limitation. Lenstra and Kan (1981) determined the VRP, as an extension of TSP, is a NP-hard problem. Later Laporte and Osman (1995) made a survey of research around the TSP and VRP, and Fisher (1995) explored different solution approaches to the VRP with time windows, including several meta-heuristics and artificial intelligence algorithms.

Pickup-and-Delivery Problems (PDPs) constitute an important family of routing problems in which goods or passengers have to be transported from different origins to different destinations, Toth and Vigo (2014). The classical PDP is where a product has one pickup and one delivery location.

Figure 1.4 illustrates a PDP with three orders each with a pickup P and a delivery D location solved by two vehicles V_1 and V_2 leaving from a base B. Several versions of the PDP have been explored since Savelsbergh and Sol (1995) formalized the general pickup and delivery problem and we will look at some versions of the problem here.

Desrosiers et al. (1986) implemented a dynamic programming solution to a version of the PDP called the single-vehicle dial-a-ride problem (DARP) where individuals are being picked up and dropped off. DARP's tend to be of smaller sizes than the PDP problems and can even be solved exactly as in Beck et al. (2003).

Pickup and Delivery Problem with Time Windows (PDPTW) is concerned with the construction of optimal routes to satisfy transportation requests, each requiring both pickup and delivery under capacity, time window and precedence constraints Dumas et al. (1991).

PDPTW has been widely studied in the literature. The first survey on this research topic was done by Mitrovic-Minic (1998) and a decade later more comprehensive reviews were published by Berbeglia et al. (2007), Parragh et al. (2008). They study different classes of the problem and review the solution methods that were developed for each class. Berbeglia et al. (2010) provides a general framework for dynamic PDPTW's where the demand is revealed over time instead of being known from the beginning.

The early mathematical formulation of the PDPTW was proposed in the late 1970s Solomon (1987). The formulation has since changed and improved. Very recently, Furtado et al. (2017) have introduced a new compact two-index formulation for PDPTW. Their proposed formulation shows good overall performance in their experiments.

One of the very first exact methods for solving pickup and delivery problems with time windows were first published in Psaraftis (1983) where a dynamic programming algorithms were developed. Later on, another exact method was developed by Dumas et al. (1991). They aimed to solve a specific class of multi-vehicles pickup and delivery problems using a Dantzig-Wolfe decomposition/column generation scheme with a constrained shortest path as sub-problem. They were able to handle problems with up to 55 requests.

As an extension of the traveling sales man problem, the PDPTW is NP-hard, which has led to a wide range of research done on how to solve this problem. Finding the global optimal solution for a realistic instance size with exact methods is unrealistic in reasonable time, therefore alternative solution methods have been proposed. Among the more successful solution methods are the meta-heuristic's, which are high-level problem-independent algorithmic frameworks. A good meta-heuristic has a good balance between the terms diversification and intensification which Blum and Roli (2003) used to describe the contribution of meta-heuristic components. We will go more into detail on these terms in Section 3.2.

One of the very first meta-heuristic approaches for PDPTW was presented by Nanry and Barnes (2000) and it was based on a reactive tabu search algorithm. They generated instances from the vehicle routing problem with time windows instances proposed by Solomon (1987). Li and Lim (2001) propose a tabu-embedded simulated annealing meta-heuristic to solve the PDPTW. Beside the test instances from Nanry and Barnes (2000), they solved new-generated instances. The new test instances are based on Solomon (1987) and they are now the most well-known benchmark instances for the PDPTW. Lau and Liang (2002) developed a tabu search to PDPTW, and they applied several construction heuristics to generate an initial solution. They also proposed a strategy to generate good problem instances and benchmarking solutions for PDPTW. More recently Hemmati et al. (2014) created benchmark instances for the PDP in tramp shipping and routing problems. Since our problem is different from the standard PDP we will generate our own instances based on real anonymized data from 4flow in Section 5.2.

Bent and Van Hentenryck (2006) present a two-stage hybrid algorithm for PDPTW. The first stage uses a simple simulated annealing algorithm to decrease the number of routes, while the second stage uses *Large Neighborhood Search* (LNS) to decrease total travel cost. They provide many new best solutions for the benchmark instances of Li and Lim (2001).

Ropke and Pisinger (2006) propose an adaptive large neighborhood search (ALNS) heuristic for the problem. The ALNS heuristic is composed of a number of competing sub-heuristics that are used with a frequency corresponding to their historic performance. They have tested the proposed heuristic on more than 350 benchmark instances with up to 500 requests any have been able to improve the best known solutions from the literature for more than 50% of the problems.

Masson et al. (2014) considers the Pickup and Delivery Problem with Shuttle routes (PDPS) which is a special case of the PDPTW where the trips between the pickup points and the delivery points can be decomposed into two legs. The first leg visits only pickup points and ends at some delivery point. The second leg is a direct trip – called a shuttle – between two delivery points. This optimization problem has practical applications in the transportation of people between a large set of pickup points and a restricted set of delivery points. Their paper proposes three mathematical models for the PDPS and a branch-and-cut-and-price algorithm to solve it. The pricing sub-problem, an Elementary Shortest Path Problem with Resource Constraints (ESPPRC), is solved with a labeling algorithm enhanced with efficient dominance rules. Three families of valid inequalities are used to strengthen the quality of linear relaxations. The method is evaluated on generated and real-world instances containing up to 193 transportation requests. Instances with up to 87 customers are solved to optimality within a computation time of one hour.

Beside the classic road-based applications, pickup and delivery problem with time windows has application in maritime transportation Hemmati et al. (2014) and in air cargo Azadian et al. (2017). There are also many extensions to the PDPTW in the literature. Ghilas et al. (2016) consider the pickup and delivery problem with time windows and scheduled lines. The problem concerns scheduling a set of vehicles to serve freight requests such that apart of the journey can be carried out on a scheduled public transportation online. They propose an ALNS heuristic to solve the problem.

As we described in Section 1.1 the 4PLP includes some aspects that differentiate it from standard problems in the literature. We mentioned multiple time windows as one of them. Favaretto et al. (2007) proposed an ant colonization model to solve the VRP with multiple time windows and multiple visits. Later Christiansen and Fagerholt (2002) described a ship scheduling PDP problem of bulk cargoes with multiple time windows. Also Manier et al. (2016) presented an exact model to solve the Pickup and Delivery Problem with Multiple Time Windows (PDPMTW) in the shipping industry. And more recently Ferreira et al. (2018) implemented a variable neighbourhood search for vehicle routing problem with multiple time windows.

We also mentioned in Section 1.1 that our problem has a heterogeneous fleet which means we have different type of vehicles to handle different type of goods. Desrosiers et al. (1995) looked at heterogeneous vehicles when they provide an extensive overview over algorithms for solving vehicle routing and scheduling problems. Savelsbergh and Sol (1998) also had a heterogeneous vehicle fleet for the VRP problem and Xu et al. (2003) did the same for PDP. More recently Masmoudi et al. (2017) implemented a hybrid *Genetic Algorithm* (GA) to solve the heterogeneous Dial-a-ride problem (H-DARP). The results from their computational experiments show that their GA is outperforming the state of the art solution methods. Sun et al. (2019) formulated an exact approach for solving the *Green Pickup and Delivery Problem* (GPDP) which aims to minimize carbon emissions of pickups and deliveries by a fleet of heterogeneous vehicles.

The complex cost matrix and dock constraint has not been widely studied in the literature. Although there has been some research on single parts of the 4PLP, multiple time windows and heterogeneous fleet, the complete 4PLP problem have not been investigated extensively, increasing the need for research in this area.

Chapter 2

Problem Description

In Chapter 1 we introduced the 4th Party Logistic Problem (4PLP). The 4PLP is an extension of the Pickup and Delivery Problem with Multiple Time Windows (PDPMTW). The additional requirements of the problem are the *complex cost structure* and the *factory dock constraint*. In this chapter we will describe the problem in detail and Section 2.1 we will formulate the mathematical model of the 4PLP. Later in Chapter 3 we will describe several solution methods which can be used to solve the 4PLP.

The 4PLP consists of a supply and demand of products, or in our case *orders*. Each order should be picked up from a certain pickup location, which constitutes the *suppliers*, and be delivered to a certain delivery location, which constitutes the manufacturers *factory docks*. To serve the demand of orders exists a set of heterogeneous *vehicles* provided by different logistic carries either directly or through third party logistic carries (3PL), see Section 1.1.

The vehicles have different capacities, incompatibilities, cost structures and start at the first pickup location at the first pickup time, i.e. costs to get from depot to first pickup location are not relevant in the 4PLP, see Section 1.1. The capacities of each vehicle is given in volume and weight which represents the size of the vehicle. When an order is assigned to a vehicle, the vehicle must load the order from the supplier pickup location, and deliver the order at the factory dock location. Some orders are not compatible with the vehicles, if for example the vehicle does not have cooling capabilities. Some vehicles are not compatible with the delivery/pickup locations, if for example the vehicle does not have the equipment required to load the order. At delivery, it could be that a vehicle has to visit several docks within the same factory. The manufacturer might have set a limit to how many docks can be visited each time a vehicle enters the factory area.

Each pickup/delivery location can have multiple time windows, lapsing sometimes over several days. If a vehicle arrives before a time window starts it has to wait. All orders must be served within the given time windows.

The cost paid for a transport depend on the total distance a vehicle is driving and the maximum weight transported by that vehicle. The cost is calculated based on a price per kilometer, a price per kg and a fixed cost, all different within each distance/weight interval. There is also a cost for each time a vehicle has to stop at a location.

In the next section we will present the complete mathematical formulation of the 4PLP problem described here.

Table 2.1: The table show indices used in the 4PLP mathematical model

Notation	Description
v	vehicle
i	node
f	factory
p	time window
s	stop location
α	distance interval in cost structure
β	weight interval in cost structure

2.1 Mathematical Model

In this section we will present the mathematical model, with notations, of the 4th Party Logistics Problem (4PLP). These notations will be used throughout this thesis, unless otherwise specified.

Table 2.2: The table contains sets used in the 4PLP mathematical model.

Notation	Description
N	nodes $\{1, 2, .., 2n\}$ where n is number of orders
V	vehicles
A	arcs
A_v	arcs visitable by vehicle v
N_v	nodes visitable by vehicle v
N^P	pickup nodes $[1, 2, .., n]$ or orders
N^D	delivery Nodes $[n + 1, n + 2, .., 2n]$
F	factories
N_f	delivery nodes for factory f
E_v	index of elements in the cost structure of vehicle v
P_i	set of time windows at node i, $\{1, 2, .., \pi_i\}$
T_i	set of time parameters $[\underline{T_{ip}}, \overline{T_{ip}}]$ at node i where $p \in P_i$
S	set of stops indicating a pickup/delivery location
L_s	Sets of nodes sharing a stop location $s \in S$

We can view the 4PLP as a graph $G(A, N)$ where $N = \{1, 2, .., 2n\}$ are the vertices, n is the number of orders in the problem, and $A = \{(i, j) : i, j \in N, i \neq j\}$ are the arcs in the graph. Since n is the number of orders in the problem, then if i is the orders pickup-node then $i + n$ is its corresponding delivery node.

The set of pickup nodes (suppliers) we denote using $N^P := \{1, 2, .., n\}$ and each delivery node (factory dock) is denoted by $N^D := \{n + 1, n + 2, .., 2n\}$. All nodes are therefore equivalent to $N = N^P \cup N^D$.

Each Factory, $f \in F$, also has a set of nodes belonging to the same factory which we denote N_f. Since all factories are delivery nodes these sets only include delivery nodes. The factory docking limit is denoted by H_f.

Table 2.3: The table contains parameters used in the 4PLP mathematical model.

Notation	Description
n	amount of orders
K_v^{kg}	weight capacity of vehicle $v \in V$
K_v^{vol}	volume capacity of vehicle $v \in V$
$o(v)$	starting node of vehicle v
$d(v)$	ending node of vehicle v
Q_i^{kg}	weight of order at node $i \in N$
Q_i^{vol}	volume of order at node $i \in N$
H_f	docking limit at factory $f \in F$
T_{ijv}	travel time for vehicle $v \in V$ over edge $(i,j) \in E_v$
π_i	amount of time windows at node $i \in N$
$\overline{T_{ip}}$	upper bound time of time window $p \in P_i$ at node $i \in N$
$\underline{T_{ip}}$	lower bound time of time window $p \in P_i$ at node $i \in N$
γ_v	amount of distance intervals for vehicle v
μ_v	amount of weight intervals for vehicle v
$C_{v\alpha\beta}^{km}$	cost per distance unit (km) in cost matrix element $(\alpha,\beta) \in E_V$ for vehicle v
$C_{v\alpha\beta}^{kg}$	cost per weight unit (kg) in cost matrix element $(\alpha,\beta) \in E_V$ for vehicle v
$C_{v\alpha\beta}^{fix}$	fixed cost in index $(\alpha,\beta) \in E_V$ for vehicle v
C_i^{stop}	costs of making a stop at node i
C_i	cost of not transporting order $i \in N^P$
D_{ij}	distance between node $i \in N$ and $j \in N$
B_α	distance for interval α in cost matrix E_V column index
Z_β	weight for interval β in cost matrix E_V row index

The set of vehicles used is denoted by V and weight capacity of each vehicle $v \in V$ is denoted by K_v^{kg} and volume capacity is denoted K_v^{vol}. We also introduce A_v as the set of arcs that each vehicle $v \in V$ can traverse. Each vehicle has a set of Nodes it can travel to represented by N_v. This set also includes an origin node, $o(v)$ and a destination node $d(v)$ which is a fictive start and ending point unique to each vehicle v. As we mentioned in Section 1.1 since a 4PL planning company is not having its own fleet, but rather paying for another company to transport from A to B it will not be charged for the cost of getting to A. For our mathematical model this means that the distance and costs from $o(v)$ to the first pickup and from the last delivery to $d(v)$ is equal to zero, meaning it will not influence any decision variables for a vehicle to travel from start to the first pickup node. To keep the model as general as possible we still include $o(v)$ and $d(v)$ in the problem formulation.

To evaluate the weight and volume constraints we say that each pickup node has a weight Q_i^{kg} and a volume Q_i^{vol} for $i \in N^P$ as parameters indicating the weight and volume of the order getting picked up at that node.

Each node has a set T_i of time windows represented by $[\underline{T_{ip}}, \overline{T_{ip}}] \in [0,T]$ where $p \in P_i = \{0,1,...,\pi_i\}$ and all nodes should be picked up and delivered within given time windows. The distance from node i to node j is denoted by D_{ij} and the time for each vehicle v to travel between them is represented by T_{ijv}.

The cost structure mentioned in Section 1.1 is complex and depend on several of the problems decision variables. Each time a vehicle v makes a stop at node i there will be a stop cost represented by C_i^{stop}. The costs of vehicle v depends on the total distance of that vehicle and the maximum weight transported. Each possible interval of weight and distance is represented by an index pair (α, β), where α is the distance interval index ranging $(1, 2, .., \gamma_v)$ and β is the weight interval ranging from $(1, 2, .., \mu_v)$. Together these pairs make a matrix we refer to as a cost matrix. Each type of cost has a matrix, including distance, weight, and fix costs. The cost in a certain interval (α,β) is therefore represented by $C_{v\alpha\beta}^{cost-type}$.

Table 2.4: The table contains variables used in the 4PLP mathematical model.

Notation	Description
x_{ijv}	binary indicating travel from node $i \in N$ to $j \in N$ of vehicle $v \in V$
y_i	binary indicating that an order $i \in N^P$ is not picked up
l_{iv}^{kg}	weight of vehicle v after visiting node i
l_{iv}^{vol}	volume of vehicle v after visiting node i
h_i	number of times docked inside a factory after visiting node $i \in N^D$
t_i	time after visiting node $i \in N$
u_{ip}	binary indicating usage of time window $p \in P_i$ at node i
$d_{v\alpha\beta}$	total distance travelled of vehicle $v \in V$ if it fits in interval $(\alpha, \beta) \in E_v$
$b_{v\alpha\beta}$	binary indicating interval $(\alpha, \beta) \in E_v$ for vehicle $v \in V$
$l_{v\alpha\beta}$	the highest weight transported by vehicle $v \in V$ for interval $(\alpha, \beta) \in E_v$

The variable t_i denotes the time after node $i \in N$ has been served and each delivery node has a variable h_i indicating how many docks, within the factory, have been visited including the node i. The variable l_{iv}^{kg} is the weight and l_{iv}^{vol} is the volume on the vehicle v leaving node i. The x_{ijv} is a binary variable indicating if vehicle v is travelling between i and j node. The cost of not transporting an order will be represented by C_i for each node i, with a corresponding binary variable y_i, indicating that an order is not picked up.

The total distance travelled by vehicle v will be denoted by the variables $d_{v\alpha\beta}$ for each $(\alpha, \beta) \in E_V$, where only one variable per vehicle will have the value equal to the total distance of that vehicle. The maximum weight transported by a vehicle is represented by $l_{v\alpha\beta}$ and also only one of these variables per vehicle will have a the corresponding value, which is determined by the binary variable $b_{v\alpha\beta}$. Each $b_{v\alpha\beta}$ has a corresponding distance parameter B_α and a weight parameter Z_β which represents the intervals in the cost matrix.

The mathematical formulation of the problem is represented as follows:

$$\min \sum_{v \in V} \sum_{(\alpha,\beta) \in E_V} (C_{v\alpha\beta}^{km} d_{v\alpha\beta} + C_{v\alpha\beta}^{kg} l_{v\alpha\beta} + C_{v\alpha\beta}^{fix} b_{v\alpha\beta}) + \sum_{v \in V} \sum_{s \in S} \sum_{\substack{i \in L_s \\ j \in N_v \notin L_s}} C_i^{stop} x_{ijv} + \sum_{i \in N^P} C_i y_i \tag{2.1}$$

subject to:

$$\sum_{v\in V}\sum_{j\in N_v} x_{ijv} + y_i = 1, \qquad i \in N^P \tag{2.2}$$

$$\sum_{j\in N_v} x_{ijv} - \sum_{j\in N_v} x_{jiv} = 0, \qquad v \in V, i \in N_v \notin \{o(v), d(v)\} \tag{2.3}$$

$$\sum_{j\in N_v} x_{o(v)jv} = 1, \qquad v \in V \tag{2.4}$$

$$\sum_{j\in N_v} x_{jd(v)v} = 1, \qquad v \in V \tag{2.5}$$

$$\sum_{j\in N_v} x_{ijv} - \sum_{j\in N_v} x_{(i+n)jv} = 0, \qquad v \in V, i \in N_v^P \tag{2.6}$$

$$l_{iv}^{kg} + Q_j^{kg} - l_{jv}^{kg} \leq K_v^{kg}(1 - x_{ijv}), \qquad v \in V, j \in N_v^P, (i,j) \in E_v \tag{2.7}$$

$$l_{iv}^{kg} - Q_j^{kg} - l_{(j+n)v}^{kg} \leq K_v^{kg}(1 - x_{i(j+n)v}), \qquad v \in V, j \in N_v^P, (i, n+j) \in E_v \tag{2.8}$$

$$0 \leq l_{iv}^{kg} \leq K_v^{kg}, \qquad v \in V, i \in N_v^P \tag{2.9}$$

$$l_{iv}^{vol} + Q_j^{vol} - l_{jv}^{vol} \leq K_v^{vol}(1 - x_{ijv}), \qquad v \in V, j \in N_v^P, (i,j) \in E_v \tag{2.10}$$

$$l_{iv}^{vol} - Q_j^{vol} - l_{(j+n)v}^{vol} \leq K_v^{vol}(1 - x_{i(j+n)v}), \qquad v \in V, j \in N_v^P, (i, n+j) \in E_v \tag{2.11}$$

$$0 \leq l_{iv}^{vol} \leq K_v^{vol}, \qquad v \in V, i \in N_v^P \tag{2.12}$$

$$h_i + 1 - h_j \leq (H_f + 1)(1 - x_{ijv}), \qquad v \in V, f \in F, i \in N_f, j \in N_f, j \neq i \tag{2.13}$$

$$h_j \leq H_f, \qquad v \in V, f \in F, j \in N_f, \tag{2.14}$$

$$h_j \geq \sum_{\substack{i\in N_v \\ i \notin N_f}} (x_{ijv}) \qquad v \in V, j \in N_f \tag{2.15}$$

$$\sum_{p\in P_i} u_{ip} = 1, \qquad i \in N \tag{2.16}$$

$$\sum_{p\in P_i} u_{ip}\underline{T_{ip}} \leq t_i, \qquad i \in N \tag{2.17}$$

$$\sum_{p\in P_i} u_{ip}\overline{T_{ip}} \geq t_i, \qquad i \in N \tag{2.18}$$

$$t_i + T_{ijv} - t_j \leq (\overline{T_{i\pi_i}} + T_{ijv})(1 - x_{ijv}), \qquad v \in V, (i,j) \in E_v \tag{2.19}$$

$$t_i + T_{i(i+n)v} - t_{(i+n)} \leq 0, \qquad v \in V, i \in N_v^P \tag{2.20}$$

$$\sum_{(\alpha,\beta)\in E_V} d_{v\alpha\beta} = \sum_{(i,j)\in E_v} x_{ijv} D_{ij}, \qquad v \in V \tag{2.21}$$

$$\sum_{(\alpha,\beta)\in E_V} l_{v\alpha\beta} \geq l_{iv}^{kg} \qquad v \in V, i \in N_v \tag{2.22}$$

$$B_{(\alpha-1)} b_{v\alpha\beta} \leq d_{v\alpha\beta} \leq B_\alpha b_{v\alpha\beta}, \qquad v \in V, (\alpha,\beta) \in E_V \tag{2.23}$$

$$Z_{(\beta-1)} b_{v\alpha\beta} \leq l_{v\alpha\beta} \leq Z_\beta b_{v\alpha\beta}, \qquad v \in V, (\alpha,\beta) \in E_V \tag{2.24}$$

$$\sum_{(\alpha,\beta)\in E_V} b_{v\alpha\beta} \leq \sum_{j\in N_v} x_{o(v)jv}, \qquad v \in V \tag{2.25}$$

$$h_i, t_i \geq 0, \qquad i \in N \tag{2.26}$$

$$u_{ip} \in \{0,1\}, \qquad i \in N, p \in P_i \tag{2.27}$$

$$b_{v\alpha\beta} \in \{0,1\}, \qquad v \in V, (\alpha,\beta) \in E_V \tag{2.28}$$

$$d_{v\alpha\beta}, l_{v\alpha\beta} \geq 0 \qquad v \in V, (\alpha,\beta) \in E_V \tag{2.29}$$

$$y_i \in \{0,1\}, \qquad i \in N^P \tag{2.30}$$

$$x_{ijv} \in \{0,1\}, \qquad v \in V, (i,j) \in E_v \tag{2.31}$$

The objective function, Eq. (2.1), sums up to the cost of all vehicles given corresponding costs from their cost matrix. Costs could be variable per distance, weight, fixed and/or related to stops made. Loads not transported will be penalized with costs and the aim is to minimize the sum of all these costs. Eq. (2.2) is a constraint to ensure that a load is picked up once and only by one vehicle or not picked up at all. Eqs. (2.3) to (2.6) govern the flow of the orders served by a vehicle, and takes care that when a node is visited it is also left, controls departure and arrival and ensures delivery of picked up orders respectively.

The weight on a vehicle during pickup and delivery is managed by Eqs. (2.7) to (2.8). The constraint represented by Eq. (2.9) ensures the weight does not exceed the vehicles capacity.

The next constraints Eqs. (2.10) to (2.12), ensures the same as the weight constraints for volume, that each load is increased by the volume of the order at pickup, that the volume is decreased at delivery and that the volume at any node does not exceed the capacity of the vehicle.

It follows from constraint Eq. (2.13) that within a certain factory, if you travel between two nodes, the number of docks you have visited should always be increased by one. The next constraint on factories Eq. (2.14), ensures that any node visited in a factory cannot exceed the docking limit. Then Eq. (2.15) makes sure that the docking number is correct when a vehicle is entering a factory for the first time.

Constraint Eq. (2.16) ensures only one time window is used per node, and Eqs. (2.17) to (2.18) says that the time a node is visited has to be within the lower and upper bound of the time window. Then constraint in Eq. (2.19) ensures that the travel from one node to the next is appropriately increased by the travel time between them. Finally Eq. (2.20) ensures that the delivery is after the pickup of an order.

From Eq. (2.21) we have that for each vehicle, the sum of the total distance variables has to be equal to the total travel distance of that vehicle. The constraint from Eq. (2.22) ensures that the maximum weight variable for a vehicle is corresponding to the highest amount of weight transported by that vehicle. Constraint Eq. (2.23) ensures that for each vehicle the total distance variable can only exist in the appropriate distance interval. The same is the case in Eq. (2.24) for maximum weight in the appropriate weight interval. Finally Eq. (2.28) says that if a vehicle is not leaving its origin node, there cannot be a cost interval binary for that vehicle, which in turn ensures that we do not calculate the fixed costs of said vehicle.

Constraints from Eqs. (2.26) to (2.31) define the type and range of each variable.

Chapter 3

Solution Methods

This chapter will present known solution methods to pickup and delivery problems. In Section 3.1 we present exact approaches commonly used to solve smaller instances of PDP problems. The results from Chapter 5 made it clear that an exact approach would not be a realistic solution approach to our problem. We therefore look at meta-heuristic approaches in Section 3.2 which gives near optimal solutions in shorter times than exact approaches. The model presented in Chapter 4 is a meta-heuristic approach to solve the 4th Party Logistics Problem.

3.1 Exact Approach

An *exact approach* is an algorithm that always return the global optimal solution. To ensure this, the algorithm might have to visit most, or all of the solutions in a solution space. For problems that are NP-hard, this will lead an algorithm to become very slow when increasing the size of the instance.

There exist many different exact approaches, but in this section we will give an overview over the best known approaches, *branch-and-bound*, *branch-and-cut* and *branch-and-price*, Costa et al. (2019). We introduce these approaches to give a broader understanding of how an exact approach work and explain why we have chosen a different solution method, meta-heuristics, in this thesis.

Branch-and-Bound has been utilized as an exact approach when solving NP-hard problems. Among the most general approaches to the solution of constrained optimization problems is that of "branching-and-bounding". Like dynamic programming, branching-and-bounding is an intelligently structured search of the space of all feasible solution Lawler and Wood (1966). The main idea is to traverse branches in a tree and bounding the tree while traversing to avoid visiting the whole solution space.

Figure 3.1 shows the logic of the branch-and-bound algorithm when solving a travelling salesman problem (TSP). The numbers in each node are calculated by traversing the path indicated by the node title and then calculating the minimum spanning tree (MST) of the root node, the leaf node and any unvisited node. This ensures that no shorter complete route can be found when traversing this path. As an example, traversing *abd* gives the cost 11. The MST of the nodes *a* (root), *c*(unvisited) and *d*(leaf) is 7 which gives the node value of 18. The lower bound from traversing the first leg in this position is 21, meaning the algorithm moves

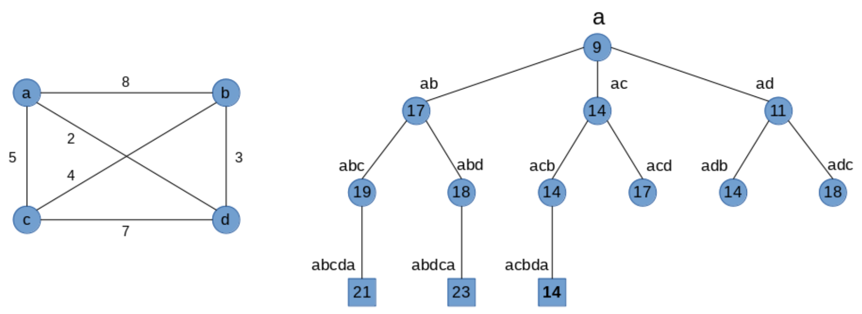


Figure 3.1: The figure shows the branch and bound algorithm on a TSP problem. The TSP instance is the left part of the figure and the tree represent the branch and bound logic while traversing a tree. The algorithm gets the bounds 21 after traversing the first leg and then 14, after visiting the third branch. The bound is calculated using MST on leaf and root node with the remaining nodes. The remaining branches, ac and ad, reaches the bound and are therefore not explored further.

on to calculate the full route *abdca*. The whole solution space in this case is 16 nodes, however, as we see from Fig. 3.1, branch-and-bound only visits 12. Branch-and-bound therefore guarantees an optimal solution without traversing the whole solution space. For more details on the branch-and-bound solution approach we refer to Morrison et al. (2016) who did a survey of recent research advances in the design of branch-and-bound algorithms.

Branch-and-cut refers to a combination of the branch-and-bound and *cutting planes* method. As we have just seen, branch-and-bound is checking every solution in a tree unless a bound is reached. The cutting planes method helps by tightening the linear programming relaxations. This means that it tries to cut off any infeasible solutions from the solution space.

Figure 3.2 represent the graph from solving the following linear program:

$$\min x + y \tag{3.1}$$

$$\text{s.t.}$$

$$2x + y \geq 2 \tag{3.2}$$

$$-x + 2y \geq -2 \tag{3.3}$$

$$x, y \in \mathbb{Z} \tag{3.4}$$

The orange and purple lines represent the constraints. The blue area represent the solution space satisfying Eq. (3.2) and Eq. (3.3). The black dots in Fig. 3.2 represent feasible integer solutions, and the red line represent the optimal objective value. The green line shows a plane cut given by adding Eq. (3.2) to two times Eq. (3.3):

$$\begin{aligned} 2x + y + 2(-x + 2y) &\geq 2 + 2(-2) \\ 5y &\geq -2 \\ y &\geq -\frac{2}{5} \\ \text{since y is integer} \rightarrow y &\geq 0 \end{aligned} \tag{3.5}$$

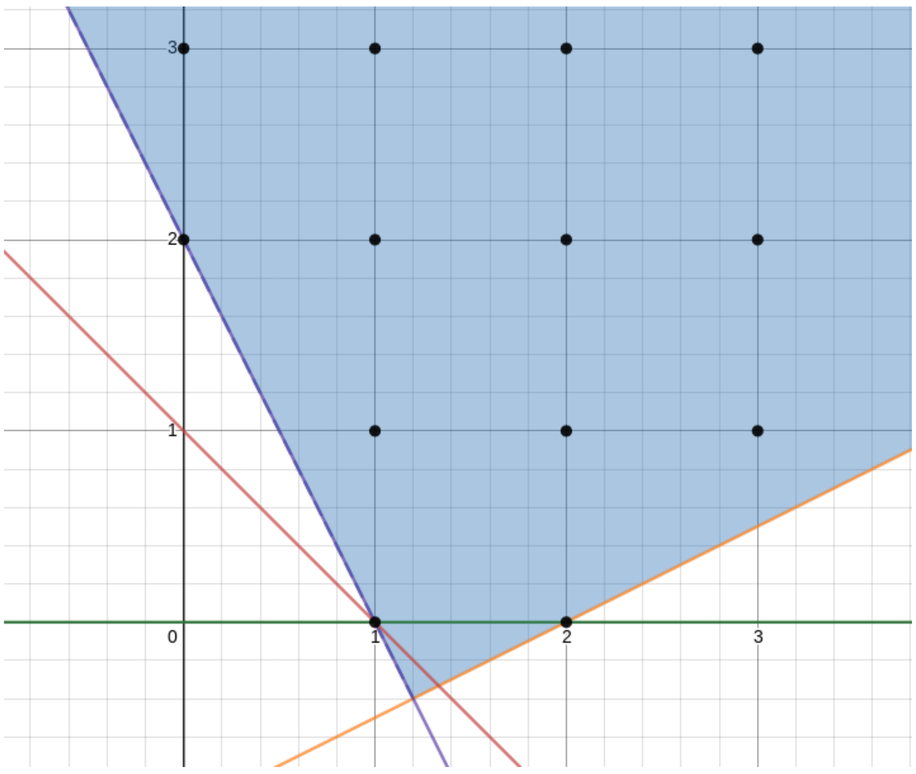


Figure 3.2: The figure shows how the cutting plane algorithm works for Eq. (3.1). The red line represent the optimal value. The orange and purple lines represent the constraints. The blue shaded area represent the solution space under the constraints. The green line represent a cut.

This means that we can cut away a part of the solution space because values of $y < 0$ are not feasible in our problem. To translate this to our tree from Fig. 3.1, this would mean that even though a branch would be within the bound, we would still not visit it because it would be infeasible. For further details on the branch-and-cut approach we refer to Qiu et al. (2018) who present a mixed integer linear programming model of the multi-product multi-vehicle production routing problem.

Branch-and-price is another technique that combines branch-and-bound with *column generation*. Ford Jr and Fulkerson (1958) used this technique to solve a multi commodity maximum flow problem. Branch-and-price is trying to split the problem into a master and a sub problem. The idea is to identify which variables have a negative reducing cost, assuming that most of the variables will be set to 0.

Figure 3.3 illustrates the branch-and-price algorithm. It shows how we go from an original problem to a reformulated master problem, typically by using what is called the Dantzig–Wolfe decomposition. Then a restricted version of the master problem (RPM) is solved with linear programming relaxation. The duals from the RMP is passed to the sub-problem, often referred to as the price-problem, and if any columns (column generation) with negative reduced cost are found it will be added to the RPM again and repeated until no negative reduces costs can be found. If the solution is integral, the minimum is found. If not, branching will occur. Further details on the branch-and-price algorithm can be found in Casazza et al. (2018) who implemented a branch-and-price algorithm exploiting column generation procedures to solve a pickup and delivery vehicle routing problem with split pickup and delivery.

In Chapter 2 we formulated a mathematical model of our 4PLP. Mathematical

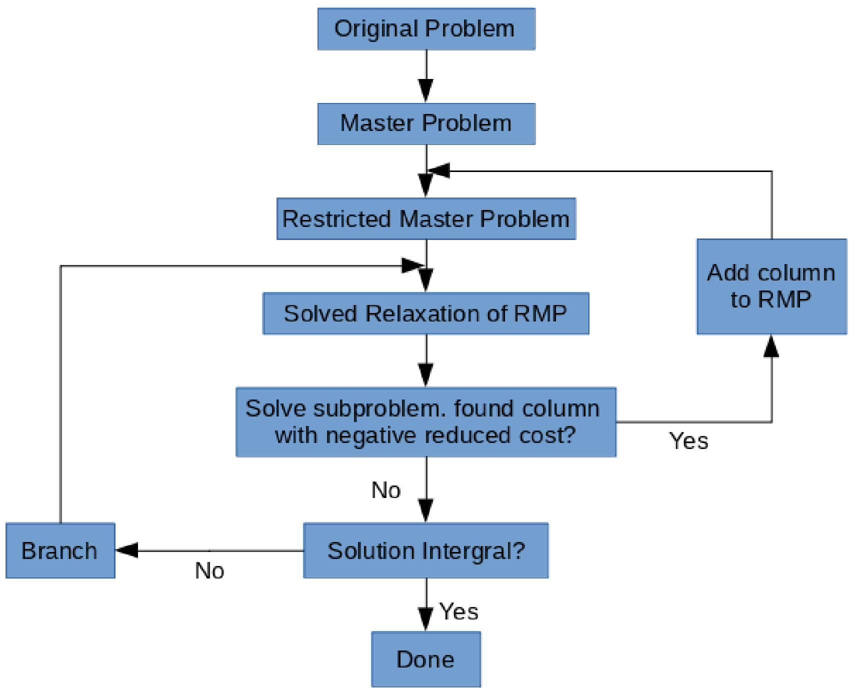


Figure 3.3: The figure shows the branch-and-price algorithm.

models can be given to modeling tools like AMPL (A Mathematical Programming Language), which utilizes commercial solvers to find the global optimal solution to our problem. AMPL is a program that solves general mathematical problems, and therefore it is not as efficient as an implementation of an exact approach. But it gives us an idea of how quickly the running time will increase with the instance size in an exact approach. We used AMPL to solve the mathematical formulation of our problem in Section 5.4.1. For instances larger than 12 orders we were not able to find optimal solutions in ten thousand seconds. This indicated to us that implementing an exact approach to our problem will not be able to solve realistic sized instances. This has lead us to search for a (meta) heuristic approach to solve our 4PLP. The next section will present different meta-heuristic approaches used to solve VRP.

3.2 (Meta)-Heuristic Approach

Definition 3.1. A meta-heuristic is a high-level problem-independent algorithmic framework that provides a set of guidelines or strategies to develop heuristic optimization algorithms Sörensen and Glover (2013).

A meta-heuristic is therefore by definition a very open way of approaching a problem and many different methods have been developed in the last few decades. The most notable meta-heuristics include (adaptive) large neighbourhood search, Tabu Search, Simulated Annealing, Genetic Algorithm, Variable Neighbourhood Search and ant colonization, among many more. The term meta-heuristic was first used by Glover (1986) when he coined it, as well as Tabu Search which he combined with artificial intelligence strategies to build a framework for integer programming.

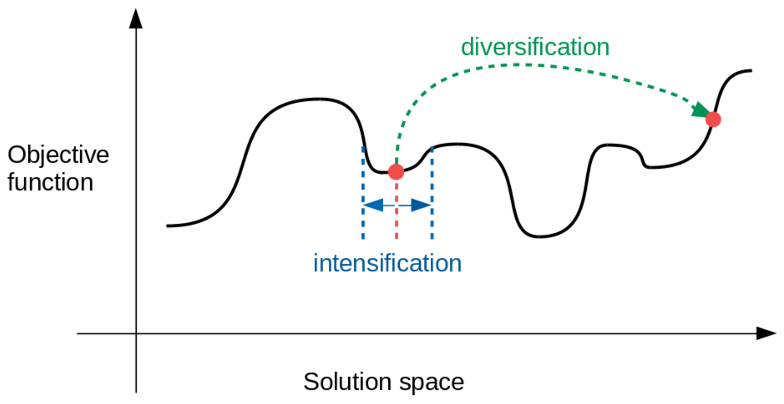


Figure 3.4: The figure illustrates how intensification, in blue, and diversification, in green, works while exploring a solutions space.

For a meta-heuristic algorithm to be efficient it has to be able to generate new solutions that are likely to improve on previous/existing solutions. At the same time it has to be able to cover the most important parts of the solution space where the global optimum may be found as well as being able to escape a local optimum. In other words a good meta-heuristic approach requires a good balance between two important principles *intensification* and *diversification*, Blum and Roli (2003). A good balance between these principles will help ensure a near global optimal solution.

Figure 3.4 illustrates how diversification and intensification are working on a solution space. By intensification we refer to a meta-heuristics ability to focus the search on a local region with a good solution in it. Too much intensification will lead the algorithm to be trapped in a local optima and will make it nearly impossible to find a global optimal solution.

Diversification refers to a meta-heuristics ability to explore the solution space on a global scale by generating diverse solutions.

Too much diversification will lead to a random search that will make it hard for the algorithm to converge to a local optima, and it will slow down the overall search performance. Too little diversification could cause a problem of *premature convergence*. Premature convergence means that the heuristics are converging to quickly on a certain part of the solution space and therefore getting stuck in a local optimum. Pandey et al. (2014), Rocha and Neves (1999) looked at different ways of preventing premature convergence in Genetic Algorithms. We will use what we refer to as a Wild Escape Algorithm in Section 4.8 to avoid getting stuck in a part of the solution space, and we will start our model by generating an empty initial solution in Section 4.3 also to avoid this problem.

Local search heuristics or *hill climbing* are heuristics which involve applying a simple neighbourhood heuristic to a given solution a repeated amount of times until no improvement can be found. By neighbourhood we refer to the solutions reachable by applying the given heuristic to a solution.

Algorithm 1 shows an implementation of the local search or hill climbing algorithm. The algorithm is given a starting solution s_0 and a neighbourhood heuristic h as input. Applying the neighbourhood heuristic h to a solution s i.e. $h(s)$ gives all the neighbouring solutions s^N to s. The heuristic starts by going into a loop

Algorithm 1 Local Search

```
1: function LS(solution $s_0$, Neighbourhood heuristic $h$)
2:     solution $s \leftarrow s_0$
3:     $done \leftarrow false$
4:     while $done = false$ do
5:         solution $s^N_{best} \leftarrow s$
6:         for $s^N \in h(s)$ do
7:             if $f(s^N) < f(s^N_{best})$ then
8:                 $s^N_{best} \leftarrow s^N$
9:         if $s = s^N_{best}$ then
10:            $done \leftarrow true$
11:        else
12:            $s \leftarrow s^N_{best}$
13:    return $s$
```

and for each neighbor of s it checks if it has found a new best neighbor s^N_{best}. As long as it can find a new s^N_{best} it can move "up-hill". It will continue to do so until no improvement can be found, i.e. $s = s^N_{best}$. This means we have found a local optimum.

Nanry and Barnes (2000) and Li and Lim (2003) implemented local search algorithms that performed simple neighbourhood operations such as moving an order from one vehicle to another, or swapping two orders with each other. We implemented two heuristics in Section 4.4.1 and Section 4.4.2 which are performing small efficient intensification moves inspired by these heuristics.

Lin (1965) implemented a 2-opt local search heuristic for the travelling salesman problem. The 2-opt heuristic can generate any solution in a TSP including the optimal solution. This inspired us to implement our own 2-opt heuristic in Section 4.4.3. The 2-opt heuristic focuses on one vehicles routes neighbourhood through several iterations and ends up returning the local optimum for that vehicle route.

The local search heuristic only focuses on intensification. This leads it to get stuck in a local optimum without the possibility of getting away. The solution to this problem will be to introduce diversification.

Kirkpatrick et al. (1983) implemented a ***Simulated Annealing*** (SA) algorithm applied to the travelling salesmen problem. Simulated annealing helps to diversify a local search by sometimes accepting worse solutions based on a temperature T.

Algorithm 2 shows how SA extends the local search by not only accepting the neighbouring solutions that have a better objective function $f(s^N) < f(s_{best})$, but also accepting any solution as a current solution s' with a probability $e^{\frac{-\Delta E}{T}}$. The temperature T is on a *cooling down schedule* and the longer the search goes on the lower the chances of acceptance becomes. The algorithm finishes when the temperature has cooled down from T_0 to T_f. Before a cool down proceeds, the algorithm searches for m iterations. To diversify our search algorithm the SA algorithm has inspired us to use an acceptance criteria based on a cooling schedule in Section 4.7.

The ***Tabu Search*** (TS) implemented by Glover (1986) uses a memory, or a tabu list, which remembers parts of the search which are then being cut off, or tabu'ed. This type of behavior helps the algorithm to diversify and avoid searching in a cycle by performing the same type of moves again and again.

Algorithm 2 Simulated Annealing

```
1: function SA(solution $s$, $h$, $T_0$, $T_f$,cooling $c$, iterations $m_i$)
2:     solution $s_{best} \leftarrow s$
3:     solution $s' \leftarrow s$
4:     temperature $T \leftarrow T_0$
5:     repeat
6:         repeat
7:             pick a neighbour solution $s^N$ from $h(s')$
8:             calculate $\Delta E = f(s^N) - f(s_{best})$
9:             if $\Delta E < 0$ then
10:                $s_{best} \leftarrow s^N$
11:                $s' \leftarrow s^N$
12:            else
13:                Accept $s' \leftarrow s^N$ with probability $p = e^{\frac{-\Delta E}{T}}$
14:        until $m_i$ iterations reached
15:        Decrease $T$ according to cooling schedule
16:    until $T = T_f$
17:    return $s_{best}$
```

The Tabu Search algorithm, described in Algorithm 3, has a list T_L which elements are tabu. The algorithm searches the neighbourhood of the best solution s_{best} and keeps the best element not contained in the tabu list. It then updates the s_{best} and the T_L with s^N_{best}. This procedure continues until a stop condition is met. We have chosen to remember which solutions we have visited and not rewarding this type of behavior in our adaptive weights in Section 4.6.

Large neighbourhood search (LNS) was introduced by Shaw (1997) and this meta-heuristic diversifies by removing and reinserting a certain amount of orders q in each iteration. Our implementation is inspired by the framework used by LNS.

Algorithm 4 shows a pseudo-code of the LNS algorithm which is given a starting solution s. First the algorithm updates the best solution s_{best} as the given s. It then goes into a loop which runs until a stop condition is met. Here it selects a number of orders q to remove from the current solution s'. The algorithm then inserts the removed orders back into the current solution s'. The removal and insertion procedures are selected separately. The amount of orders q is influencing if the algorithm will perform a very diversified action or a very intensified operation. In our model we have chosen to let the heuristics choose how many orders to operate on between a certain interval $[1, ..m]$ where m is 10% of the total amount of orders. We believe this will let our heuristics have a good balance between intensification and diversification.

After removing and inserting orders, the s_{best} is updated if the new solution results in a better objective value. Then the solution s is updated if the solution is accepted. Shaw (1997) chose to accept solutions that are better than the current one. We will use the SA acceptance criteria mentioned above.

Ropke and Pisinger (2006) implemented a version of the LNS called ***Adaptive large neighbourhood search***. He changed the LNS to adapt its-self by keeping track of the performance of each heuristic.

The ALNS algorithm psuedo-code is summarized in Algorithm 5. The difference

Algorithm 3 Tabu Search

```
1: function TS(solution $s$, $h$, max size $m_s$)
2:     solution $s_{best} \leftarrow s$
3:     empty tabu list $T_L$
4:     add $s$ to $T_L$
5:     repeat
6:         $S^N \leftarrow h(s_{best})$
7:         $s^N_{best} \leftarrow$ first element in $S^N$
8:         for $s' \in S^N$ do
9:             if not $T_L$.contains($s'$) and $f(s') < f(s^N_{best})$ then
10:                $s^N_{best} \leftarrow s'$
11:        if $f(s^N_{best}) < f(s_{best})$ then
12:            $s_{best} \leftarrow s^N_{best}$
           $T_L$.push($s^N_{best}$)
13:        if $|T_L| > m_s$ then
14:            remove first element of $T_L$
15:    until stop condition met
16:    return $s_{best}$
```

Algorithm 4 Large Neighbourhood Search

```
1: function LNS(solution $s$)
2:     solution $s_{best} \leftarrow s$
3:     repeat
4:         $s' \leftarrow s$
5:         select number of orders to remove $q$
6:         remove $q$ orders from $s'$
7:         insert removed orders in $s'$
8:         if $f(s') < f(s_{best})$ then
9:             $s_{best} \leftarrow s'$
10:        if $accept(s', s)$ then
11:            $s \leftarrow s'$
12:    until stop condition met
13:    return $s_{best}$
```

between ALNS and LNS is that the ALNS uses *selection parameters* with adaptive weights to let the algorithm decide its self which of the heuristics to use for removal and insertion of the orders. ALNS have inspired us in Section 4.6 to use adaptive weights to allow our model to automatically keep a good balance between which heuristic it uses.

This chapter illustrates the importance of a good balance between intensification and diversification in the chosen solution method. The proposed solution method, which we call *the 4th Party Logistics Optimizer*, is using a combination of these concepts to solve the 4PLP. The complete model is explained in detail in the next chapter.

Algorithm 5 ALNS

```
1: function ALNS(solution $s$, removal heuristics, insertion heuristics)
2:     solution $s_{best} \leftarrow s$
3:     repeat
4:         $s' \leftarrow s$
5:         select removal and insertion heuristics based on selection parameters
6:         select a number of orders to remove, $q$
7:         remove $q$ orders from $s'$
8:         insert the removed orders in $s'$
9:         if $f(s') < f(s_{best})$ then
10:            $s_{best} \leftarrow s'$
11:        if $accept(s', s)$ then
12:            $s \leftarrow s'$
13:        update selection parameters
14:    until stop condition met
15:    return $s_{best}$
```

Chapter 4

A 4th Party Logistics Optimizer

This chapter explains the setup of our model, the fourth party logistics optimizer (4PLO). The first Section 4.1, describes the meta-heuristic approach and explains the overall algorithm of our optimization model. The following Sections 4.2 to 4.8 go more into detail on the different parts of our 4PLO model.

Section 4.2 explains how we have represented a solution in our model and what a vehicle schedule is. Section 4.3 explains how we have chosen the initial solution. Section 4.4 presents each of the heuristics included in the basic version of our model. Section 4.5 shows how our model is choosing a heuristic in a given iteration. Section 4.6 explains how we made our model adapt the choice of heuristic throughout the iterations using historical data. Section 4.7 shows how we accept solutions, even when they are not improving the current objective function. Section 4.8 presents the *wild escape algorithm* developed for our model to diversify the search for a global optimum.

4.1 Model Overview

A brief pseudo-code of our implementation of the 4th party logistic optimizer is described in Algorithm 6.

The algorithm starts by picking an initial solution s, and then moves into a *loop* where it picks a heuristic h and applies it to to the current solution, in each iteration. It then updates the best solution by checking if the objective function value $f(s')$ is better than then current best $f(s_{best})$. After which it updates the current solution based on the result of an accept function, $accept(s', s)$, which evaluates the acceptance criteria described in Section 4.7. The algorithm stops when the stop condition is reached. We have used a stop condition of ten thousand iterations in this thesis. At the start of the *loop* it checks if an escape condition is met where it will run the algorithm from Section 4.8 on our current solution s. The escape condition is true if we have not found a new best solution s_{best} in five hundred iterations.

In the following sections we will go more into detail on each part of the model.

Algorithm 6 4th Party Logistic Optimizer

```
1: function 4PLO(A set of heuristics $\mathbb{H}$)
2:     generate initial solution,$s$
3:     solution $s_{best} \leftarrow s$
4:     iterations since best solution found $i \leftarrow 0$
5:     repeat
6:         if $i >$ escape condition then
7:             apply wild escape algorithm from Section 4.8 to $s$
8:         $s' \leftarrow s$
9:         select a heuristic, $h \in \mathbb{H}$ based on selection parameters
10:        apply heuristic $h$ to $s'$
11:        if $f(s') < f(s_{best})$ then
12:            $s_{best} \leftarrow s'$
13:        if $accept(s', s)$ then, see Section 4.7
14:            $s \leftarrow s'$
15:        update selection parameters and escape condition
16:    until stop condition met
17:    return $s_{best}$
```

4.2 Solution Representation

In order for our model to be able to apply (sub-)heuristics on a solution, we need to decide on how we present a solution so that the model can interpret what a solution is and how to differentiate it from another solution. Choosing a good *solution representation* is therefore an important part of the algorithm. There are many ways to represent a solution, matrix, binary matrix and permutation. A way to represent our solution could be in a matrix, where each row represents a *vehicles schedule* (a vehicle schedule is defined in Chapter 2). However this would require more memory and could lead to more iterations than necessary when checking a solution. To save memory and efficiency in our model we will be using a permutation as a solution representation in this thesis, where each *vehicles schedule* S_v will be separated by a 0. We separate with a 0 as each orders pickup and delivery will be represented by a real number.

Figure 4.1 shows an example of a solution s, with 5 orders and 3 vehicles. Each vehicle schedule indicates in which order a pickup and delivery should follow. The first time an order number appears refers to the pickup (highlighted as dark green and marked with * in the figure). The second time the same order number appears is referring the delivery, red tiles marked with ’ in the figure. As an example, vehicle 3’s vehicle schedule S_3 indicates it should first pick up the order 4 and then delivering the order right after. The same goes for vehicle 1 where S_1 indicates first the pickup of order 5, then the pickup of order 3 and then the delivery of order 3 and 5 respectively. S_2 is an example of an empty vehicle schedule.

The final part of the solution representation, S_{dummy}, shows a vehicle schedule that represents the orders that have not yet been assigned to a vehicle. In this chapter we will continue to refer to s as a *solution* and S_v as a *vehicle schedule* represented as above.

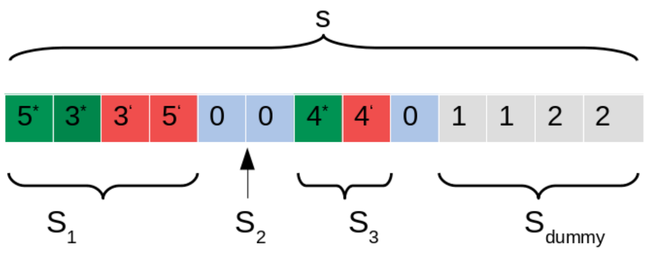


Figure 4.1: Example of a solution representation of an instance with 5 orders and 3 vehicles. A vehicle schedule, represented by S_v, contain information for each vehicle v, on which and in what order a vehicle should pick up and deliver orders. The dark green tiles, also indicated with a *, represent a pickup, the red tiles, also indicated with a ', represent a delivery. The 0 tiles separate each vehicle schedule. The complete permutation represent our solution representation s. The last schedule S_{dummy}, represent the orders that have not yet been assigned to a vehicle.

4.3 Initial Solution

We mentioned premature convergence in Chapter 3, that can lead to a problem of getting stuck in a certain part of the solution space. To avoid this problem we have chosen to start with an initial solution s where no orders are assigned to any of the vehicles, i.e. all orders are assigned to the dummy vehicle S_{dummy}. Not only does this contribute in avoiding the problem of premature convergence, it is also efficient in terms of running time. On top of that we design our model to be able to adapt to the problem on its own, regardless of the initial solution.

4.4 Heuristics

In each subsection of this section, we present the (sub-)heuristics used by our model. The first two heuristics Sections 4.4.1 to 4.4.2, are focusing on diversification. They try to shuffle a random part of the solution, to search for new solutions regardless of their objective value. The third heuristic is focusing on intensification and searches for improvements in the solution.

Removal and reinsertion heuristics are well researched tools when solving PDP's, Korsvik et al. (2011), Ropke and Pisinger (2006), Shaw (1997), Sze et al. (2016). The last four heuristics, Sections 4.4.4 to 4.4.7, are heuristics partly inspired by known removal and reinsertion heuristics and partly developed in this thesis. Each of them contain one heuristic for removing elements from a solution s and one heuristic for reinserting them back into the solution. They are not chosen separately as in Ropke and Pisinger (2006), Shaw (1997). These heuristics are used partly for diversification, and partly for intensification depending on the number of solution elements, q, being reinserted.

4.4.1 Swap

This *Swap* heuristic tries to exchange the pickup and delivery of two randomly selected orders until it finds a feasible solution. Figure 4.2 illustrates a successful swap between two orders in a vehicle schedule.

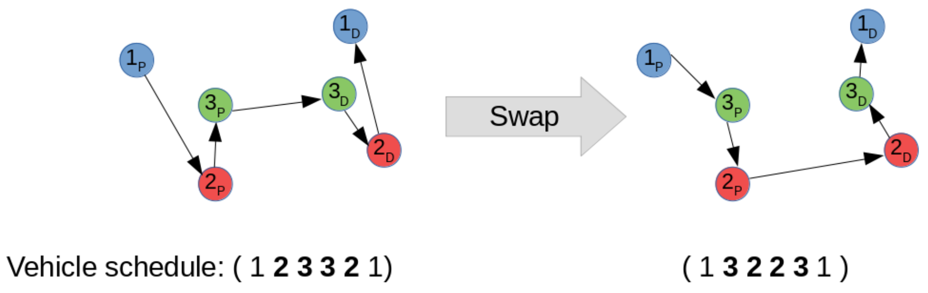


Figure 4.2: A swap heuristic performed on a vehicle schedule with three orders. The numbered nodes indicate the location of an order and the letters P and D indicate pickup and delivery respectively, 1_P is therefore the pickup location of order 1. Emphasized numbers in the vehicle schedule are selected by the heuristic.

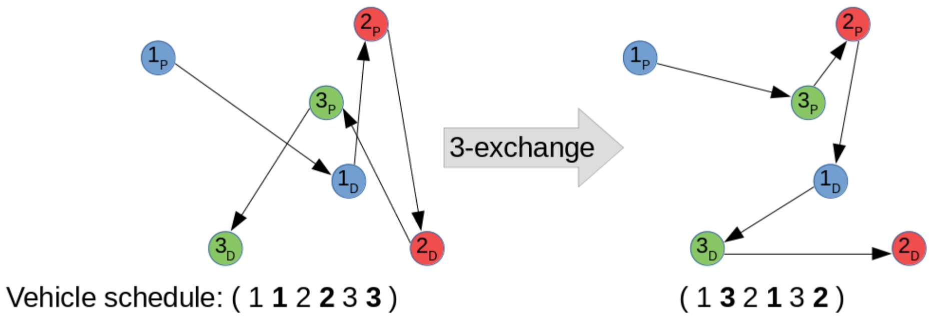


Figure 4.3: A 3-exchange heuristic performed on a vehicle schedule with three orders. The numbered nodes indicate the location of an order and the letters P and D indicate pickup and delivery respectively, 1_P is therefore the pickup location of order 1. Emphasized numbers in the vehicle schedule are selected by the heuristic for the exchange.

This heuristic is efficient in terms of running time and jumps randomly around the solution space. It is only working on two orders at a time but it is helping to diversify the search since it is not trying to find any improvement in the solution.

4.4.2 3-Exchange

The 3-exchange heuristic selects a random vehicle with at least two assigned orders, and performs an exchange of 3 randomly selected indices in the vehicle schedule until a feasible new combination is found or a certain number of iterations has passed. A 3-exchange of index position 2, 4 and 6 in a vehicle schedule is illustrated by Fig. 4.3.

This heuristic is efficient in terms of running time, as the exchanges are fast operations and checking if a vehicles schedule is feasible, compared to a whole solution, is an effective operation. Like the swap heuristic from the previous section this heuristic jumps randomly around the solution space and is therefore helping to diversify the search.

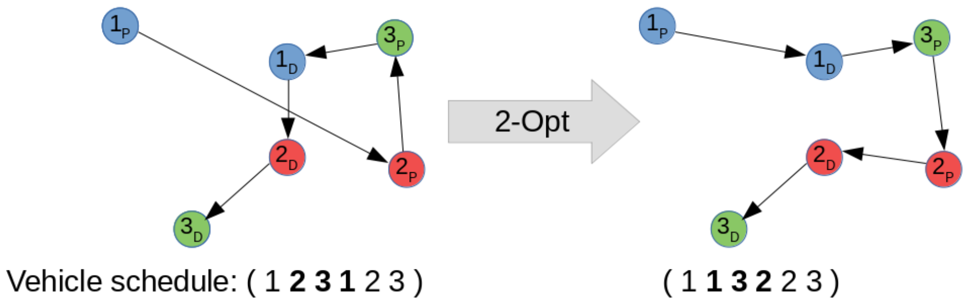


Figure 4.4: One 2-opt heuristic operation performed on a vehicle schedule with three orders. The numbered nodes indicate the location of an order and the letters P and D indicate pickup and delivery respectively, 1_P is therefore the pickup location of order 1. Emphasized numbers indicate the sub part of the vehicle schedule $S_{v,(j+1),k}$, from index $(j+1) tok of vehicle$v$, being reversed.$

4.4.3 2-Opt

The 2-opt heuristic is used in meta-heuristics to solve vehicle routing problems. Englert et al. (2007, 2014) and Lin (1965) implemented the 2-opt heuristic for travelling salesmen problem and Bullnheimer et al. (1999) combined 2-opt with the ant system meta-heuristic. Also Barthélemy et al. (2010) implemented a meta-heuristic approach for the clustered VRP using the 2-opt heuristic to improve a local search. The 2-opt heuristic used in this thesis is inspired by the heuristic from Carrabs et al. (2007). Figure 4.4 illustrates one iteration of our 2-opt heuristic performed on a vehicle schedule.

Algorithm 7 2-opt Heuristic

```
1:  function 2-OPT
2:      select random vehicle $v$ with $|S_v| > 2$
3:      vehicle schedule $S_v$
4:      repeat
5:          $S_{best} \leftarrow S_v$
6:          $n \leftarrow |S_v|$
7:          for $j = [1, 2, .., n]$ do
8:              $k \leftarrow j + 1$
9:              for $k = [j+1, j+2, .., n]$ do
10:                 current vehicle schedule $S'_v \leftarrow S'_{v,0,j} + reverse(S'_{v,(j+1),k}) + S'_{v,(k+1),n}$
11:                 if $f(S'_v) < f(S_{best})$ then
12:                     $S_{best} \leftarrow S'_v$
13:         $S_v \leftarrow S_{best}$
14:     until no further improvement found
15:     return $S_v$
```

The heuristic is described in Algorithm 7 and it starts by selecting a random vehicle with more than 2 orders. For the selected vehicle it divides up the vehicle schedule S_v in 3 parts, $S_0 i$, S_{ij} and S_{jn}, where $n = |S_v|$. It does this for all possible combinations of $0 < i < n$ and $i < j \leq n$. All orders up until the index i, as well as orders from the index $j+1$ until the end of the vehicle schedule, are left unchanged.

Then the orders from the index $i+1$ until index j are inserted in reverse order. If the new vehicle schedule S'_v has a smaller cost than the best vehicle schedule S_{best}, it is remembered as the new best vehicle schedule. After all iterations the best vehicle schedule S_{best} is selected as the new S_v. This operation is continued until no improvement can be made for S_v i.e. the best possible schedule for the selected vehicle has been found. This heuristic is not as efficient, in terms of running time, as Swap and 3-exchange, but it improves a given vehicle schedule as much as possible with 2-opt operations.

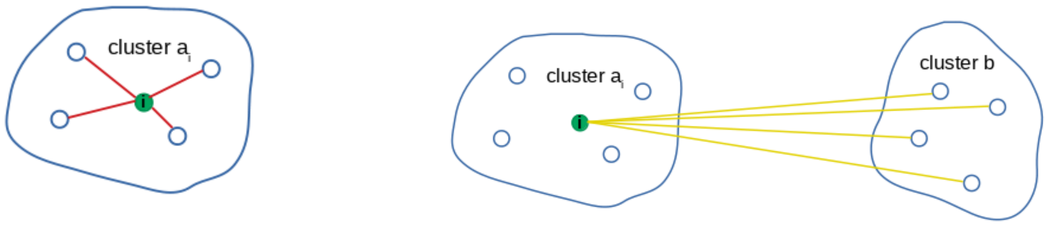
4.4.4 Random Fit

A certain degree of "randomness" is important in heuristics, as shown by Drias (1999) who experimented with degrees of random noise in different meta-heuristics. Only focusing on improving a given solution could lead our algorithm to always explore similar parts of a solution space. This would result in a less robust algorithm. Therefore we decided to make one remove and reinsert heuristic that does not have any specific priorities but rather moves around the solution space randomly.

Algorithm 8 Random Fit Heuristic

```
1: function RandomFit(solution $s$)
2:     select the number of orders to reinsert, $q$
3:     solution $s' \leftarrow s$
4:     remove $q$ orders from $s'$
5:     set $I \leftarrow$ removed orders
6:     for $z \in I$ do
7:         repeat
8:             choose random vehicle $v$
9:             choose random position in vehicle schedule $S_v$
10:            insert $z$ in $S_v$
11:        until feasible vehicle schedule found
12:        update s'
13:    return s'
```

Algorithm 8 shows the pseudo-code for our Random Fit heuristic. It starts by selecting a random number of orders q, which it will remove and reinsert. Then it selects q random orders and removes them from the solution. We have chosen q between 2 orders and 10% of the number of orders in the instance. The heuristic selects a random vehicle v and inserts the removed orders in its vehicle schedule S_v randomly until a feasible vehicle schedule is found. It then updates the solution s with the S_v. This heuristic is used for diversifying the solution and is trying to search for possible solutions regardless of the cost they produce. The remove and reinsertion heuristics are not expected to be as efficient as the heuristics presented in Sections 4.4.1 to 4.4.3 in terms of running time. But in comparison to other remove and reinsertion heuristics, this one should be more efficient as it is not searching for a specific improvement of the solution, but rather returns the first feasible.



(a) Cohesion **(b)** Separation

Figure 4.5: The figures show the cohesion and separation effect illustrated respectively by the red and yellow lines. The green node represent location i and a is its cluster. The cluster b is the cluster with the minimum average distance to i.

4.4.5 Clustering

Section 1.1 illustrates how a typical 4PL customer typically has its factories and suppliers structured in clusters around a city or in a country. We refer to a cluster here as a certain set of locations in close proximity to one another in relation to other locations. Since we suspected that locations are likely to form clusters, this led us to try to build a heuristic which considers clusters as a factor when removing and reinserting an order in a vehicle schedule. Orders with delivery and pickup locations from different clusters, served by the same vehicle, should be removed. In the same way orders that are being delivered from and to similar clusters should be bundled together on the same vehicles.

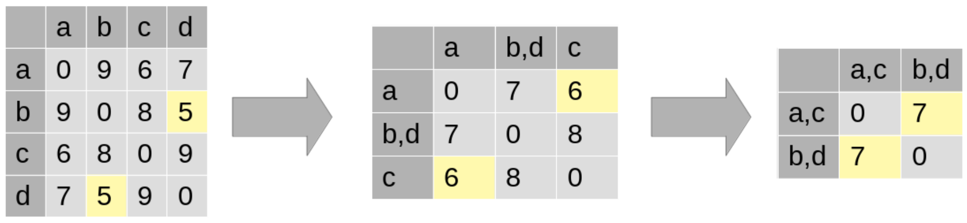
To build a heuristic which takes advantage of this we have to determine the size of the clusters in an instance and divide up the locations in their corresponding clusters. The next two paragraphs will explain the methods we have used to solve this problem, before we go into detail on how our Clustering heuristic is working.

The Silhouette Coefficient

To decide which pickup and delivery locations belong to which cluster we first needed to find how many clusters we should have. Kaufman and Rousseeuw (1990) introduced an efficient way to compare clusters of different sizes by calculating what they call a *Silhouette Coefficient*. Reddy and Vinzamuri (2018) did a survey of partitional and hierarchical clustering algorithms and considered the Silhouette Coefficient as a viable method to estimate the number of clusters. The coefficient is calculated based on two aspects, cohesion ι_i and separation η_i of a node (or in our case a location) i. Together these aspects make out the Silhouette ν_i.

Figure 4.5 illustrates the cohesion and separation of a green node i part of cluster $a_i \in [1, 2, .., k]$ where k is the number of clusters, a_i represent the index of the cluster for node i. The cohesion ι_i is the mean of the distances, d_{ij}, from i to all other nodes $j \in \kappa_{a_i}$ where κ_{a_i} is a set containing all nodes in cluster a_i. The separation η_i is the minimum of the mean of the distances, d_{ij} from node i to all the nodes $j \in \kappa_{a_j}$ in cluster a_j, of all other clusters $a_j \neq a_i$. The two effects can be written as follows:

$$\iota_i = \frac{1}{|\kappa_{a_i}| - 1} \sum_{j \in \kappa_{a_i}, i \neq j} d_{ij} \tag{4.1}$$



	a	b	c	d
a	0	9	6	7
b	9	0	8	5
c	6	8	0	9
d	7	5	9	0

	a	b,d	c
a	0	7	6
b,d	7	0	8
c	6	8	0

	a,c	b,d
a,c	0	7
b,d	7	0

Figure 4.6: Each table represents a distance matrix between pairs of nodes, a, b, c and d. In the first table, d_{bd} marked in yellow is the shortest distance and therefore chosen to be part of the same cluster. The new node, or cluster, (b, d) contain the shortest distances to the other nodes. In the second table, d_{ac} is shortest and therefore put together in a cluster.

$$\eta_i = \min_{\substack{b \in [1,2,..,k] \\ b \neq a_i}} \frac{1}{|\kappa_b|} \sum_{j \in \kappa_b} d_{ij} \tag{4.2}$$

The Silhouette Coefficient ν_i for node i is then calculated as follows:

$$\begin{aligned} \nu_i = \frac{\eta_i - \iota_i}{\max \iota_i, \eta_i}, \qquad & if\ |\kappa_{a_i}| > 1 \\ \nu_i = 0, \qquad & if\ |\kappa_{a_i}| = 1 \end{aligned} \tag{4.3}$$

Comparing the Silhouette Coefficient of one clustering of size k to another requires that the nodes are divided into clusters. When comparing the Silhouette Coefficient we are calculating the average ν_i for all $i \in N$. The cluster size k with the smallest value of average ν_i is the chosen cluster size by our algorithm. The next section will explain how we decided the clustering for a given size k.

Hierarchical single linkage clustering

To find the clustering of a given size k, we used hierarchical single linkage clustering algorithm Reddy and Vinzamuri (2018).

Figure 4.6 illustrates how the algorithm works for a set of nodes a, b, c and d. In each iteration we choose the pair of nodes, or locations, with the smallest distance, d_{ij} to be a part of a new cluster. The new distance matrix contain the shortest distance from the merged nodes to any other node. The merging of nodes goes on until we reach k number of nodes which then represent the clusters.

Removing and Inserting Elements based on Clusters

In the previous sections we described how we can compare clusters of different sizes and how to find the clustering for a given size. We used this in our model by running a preprocessing algorithm that assigns each location $i \in N$ to their corresponding clusters where the size k is decided by calculating the average Silhouette Coefficient, ν_i, for all $i \in N^P$ and keeping the clusters with the lowest value of average ν_i. This means that before our model starts, each location $i \in N$ is assigned a cluster index a_i and is contained in the cluster set κ_{a_i}.

Algorithm 9 Cluster Heuristic

```
 1: function CLUSTER(solution $s$, random selection degree $p$)
 2:     select the number of orders to reinsert, $q$
 3:     add orders to array $A$ ascending based on cluster value $\sigma_i$
 4:     empty order set $I$
 5:     repeat
 6:         choose a random number $y$ between $[0, 1]$
 7:         remove the order in position $y^p$ in $A$ from $s$
 8:         add removed order to $I$
 9:     until $|I| = q$
10:     while $|I| > 0$ do
11:         sort $I$ descending based on best possible $\sigma_{iv}$
12:         remove first $i$ from $I$
13:         insert $i$ in its best possible position in $s$
14:         return $s$
15:
```

Algorithm 9 starts by selecting the number of orders to remove q. Then it sorts all orders from the given solution s according to the cluster value σ_i where i is the pickup location of the order. The cluster value of an order is calculated based on the locations visited by the vehicle, that is which orders are bundled together on a vehicle schedule. The idea of the cluster value is to evaluate how much an order i is "clustered" with other orders on a vehicle schedule. We can define σ_i as:

$$\sigma_i = \frac{|\ell_{iv}| + |\ell_{(i+n),v}|}{2(|S_v| - 2)} \;\forall\; i \in N^P \tag{4.4}$$

Here the set ℓ_{iv} contains all the locations visited by the vehicle v from the same cluster κ_i as the pickup location i, excluding i and/or $i+n$. The $\ell_{(i+n),v}$ is a set of all locations visited by the vehicle v from the same cluster κ_{i+n} as the delivery location $i+n$, excluding $i+n$ and/or i. The set S_v is the vehicle schedule for vehicle v, i.e. $|S_v|$ is the number of locations visited and $2(|S_v|-2)$ is the maximum possible "common" cluster locations that $|\ell_{iv}| + |\ell_{(i+n),v}|$ can contain. The resulting σ_i will therefore be a value between $[0, 1]$ where a high value indicates that orders are highly "clustered", that is the locations of other orders on the same vehicle schedule are within the same cluster. A value close to 0 indicate order locations are from different clusters.

To help explain how the σ_i works we refer to Example 4.4.1.

Example 4.4.1. A cluster value σ_i example.

- A vehicle schedule $S_1 = (1\ 2\ 1\ 2\ 3\ 3)$ for vehicle 1
- Order 1 ($i = 1$) has $a_1 = 1$ and $a_4 = 2$.
- Order 2 ($i = 2$) has $a_2 = 2$ and $a_5 = 3$.
- Order 3 ($i = 3$) has $a_3 = 3$ and $a_6 = 2$.

For order 1 ($i = 1$) each location, related to other orders ($i = 2, 3$), visited in cluster $a_1 = 1$, order 1 will be rewarded a point since its pickup is in cluster 1. Order 1 will also be rewarded 1 point for each location the vehicle visits in cluster $a_4 = 2$, related to other orders, since its delivery is in cluster 2. The pickup of order 1 is in cluster 1 which contains no pickup or delivery location of other orders, leading to $|\ell_{1,1}| = 0$. The delivery of order 1 is in cluster 2, which is the same cluster as pickup location of order 2 ($a_2 = 2$) and delivery location of order 3 ($a_6 = 2$) and therefore $|\ell_{4,1}| = 2$. This leads to a cluster value $\sigma_1 = \frac{0+2}{2(4)} = \frac{2}{8}$. With the same logic, for order 2 the $|\ell_{2,1}| = 2$ and $|\ell_{5,1}| = 1$ and a cluster value of $\sigma_2 = \frac{3}{8}$. Order 3 gets $|\ell_{3,1}| = 1$ and $|\ell_{6,1}| = 2$ and a cluster value of $\sigma_3 = \frac{3}{8}$.

We divide by $2(|S_v| - 2)$ to be able to compare the cluster value with orders on other vehicle schedules.

In continuation of the explanation of Algorithm 9 it uses the cluster value σ_i first to decide which orders to remove. It will remove the orders with the lowest ranking, using some randomization, based on the parameter p. We choose to remove the order on the position y^p in the ascending σ_i sorted list of orders, where y is a random number between $[0, 1]$. We have chosen $p = 4$ for all our heuristics.

To reinsert the orders, the algorithm sorts the removed orders based on σ_i. It then inserts the order i with the highest cluster value σ_{iv} in the best possible position in s.

This algorithm is expected to perform quite similarly, in terms of running time, to the remove and reinsert heuristics from Sections 4.4.6 and 4.4.7, meaning it will be one of the more demanding heuristics used by our model.

4.4.6 Greedy

Removing orders in the most costly positions and reinserting it in its cheapest (greedy) position seems to be a reasonable way of moving towards a better solution. We therefore propose a heuristic that removes the orders with the highest cost $C_i^S \; \forall \; i \in N_i^P$. The C_i^S is calculated as the increase in a vehicles schedule cost with the chosen order.

We remove orders based on the same randomness factor explained above p. We do this by first sorting the orders descending based on the cost C_i^S. The cost can be calculated as $C_i^S = f(S_v) - f_{-i}(S_v)$, for a given order i served by vehicle v, where $f(S_v)$ is the objective value, or cost, of vehicle schedule S_v. Here the $-i$ indicate that we calculate the cost of vehicle schedule S_v without the order i. We then choose the order in the y^p position.

To reinsert an order we sort the removed orders based on their minimum increase in the objective value c_i. The $c_i = \min_{v \in V}(\Delta f_{iv})$, were the Δf_{iv} represents the difference (*Delta*) in objective function value f_{iv} by inserting order i in the position in v with the lowest increase in objective function value. The c_i is therefore the minimum of these increases. We insert the order in its best possible position in the solution s.

This algorithm is expected to perform similarly with the heuristics from Sections 4.4.5 and 4.4.7, in terms of running time. This means it will be one of the more demanding heuristics when it comes to running time. The greedy heuristic is

Algorithm 10 Greedy Heuristic

```
1: function GREEDY(solution $s$, random selection degree $p$)
2:     select the number of orders to reinsert, $q$
3:     add orders $i \in N^P$ in array $A$ descending based on cost $C_i^S$
4:     empty order set $I$
5:     repeat
6:         choose a random number $y$ between $[0, 1]$
7:         remove the order in position $y^p$ in $A$ from $s$
8:         add removed order to $I$
9:     until $|I| = q$
10:    repeat
11:        sort $I$ based on each orders minimum increase in objective value $c_i$
12:        insert the first order $i$ from $I$ in its best possible position in $s$
13:        remove order $i$ from $I$
14:    until $|I| = 0$
15:    return $s$
   =0
```

an intensification search heuristic that tries to search for a local or global optimal solution.

4.4.7 Similar Regret

The removal part of this heuristic is inspired by Shaw (1998)'s removal heuristic, which is often used in removal and reinsertion heuristics Curtois et al. (2018), Korsvik et al. (2011), Ropke and Pisinger (2006). The heuristic removes orders that share specific similar qualities. The basic idea is that replacing these orders by each-other will find new, hopefully better, solutions. We define a *relatedness factor* r_{ij} which represents how much the order i is related to the order j. The lower the value of r_{ij} the more the two orders i and j are related. The relatedness of two orders were based on the following properties in this thesis: a distance property, a weight property, a property indicating if the same vehicles can be used to serve each request, a property indicating if the orders belong to the same factory and finally an overlapping time window property.

The relatedness factor is given by the following equation:

$$r_{ij} = \psi(D_{ij}+D_{(i+n)(j+n)})+\omega|Q_i-Q_j|+\phi(1-\frac{|V_i \cap V_j|}{max(|V_i|, |V_j|)})+\tau G_{ij}+\chi(U_{ij}+U_{(i+n)(j+n)}) \tag{4.5}$$

We have chosen the following values in this thesis $\psi = 0.7$, $\omega = 1.0$, $\phi = 0.8$, $\tau = 0.3$, $\chi = 0.3$, which represent how much we value each property.

In Eq. (4.5) D_{ij} and Q_i are the same as in Chapter 2 and all values have been normalised to result in a value between [0..1]. The set V_i contain the vehicles that can serve order i. The parameter G_{ij} is 1 if i belong to another factory than j and 0 if they belong to the same factory. The U_{ij} is the time windows at the pickup and delivery location corresponds to the portion of overlapping time windows divided by the total span of the two time window sets. This resulting value will be in the interval $[0, 1]$. Here a value of 0 would indicate no overlapping time window and 1 would mean identical time windows and some overlapping time windows is scaled in between these two. It can be formulated as follows:

$$U_{ij} = 1 - \frac{T^O_{ij}}{T^A_{ij} - T^{NO}_{ij}} \tag{4.6}$$

Here $\overline{T_{ip}}$ and $\underline{T_{ip}}$ are the upper and lower time windows defined in Section 2.1. The factor T^O_{ij} consists of all the time windows where order i overlaps with the time windows from order j. It can be written mathematically as follows:

$$T^O_{ij} = \sum_{\substack{p\in\pi_i \\ o\in\pi_j \\ \underline{T_{ip}}\le\overline{T_{jo}} \\ \underline{T_{jo}}\le\overline{T_{ip}}}} (\min(\overline{T_{ip}}, \overline{T_{jo}}) - \max(\underline{T_{ip}}, \underline{T_{jo}})) \tag{4.7}$$

Here T^A_{ij} represents the total span of the time window sets of location i and j. It starts from the first lower time window and ends on the last upper time window of these locations. It can be formulated as:

$$T^A_{ij} = \max\left(\max_{p\in\pi_i}\overline{T_{ip}}, \max_{o\in\pi_j}\overline{T_{jo}}\right) - \min\left(\min_{p\in\pi_i}\underline{T_{ip}}, \min_{o\in\pi_j}\underline{T_{jo}}\right) \tag{4.8}$$

The factor T^{NO}_{ij} is the opposite of the above factor T^O_{ij} and represents the time when neither i nor j have a time window. This can be exemplified by night time when no factory is open. It can be formulated as follows:

$$T^{NO}_{ij} = \sum_{\substack{p\in\pi_i \\ o\in\pi:q_j \\ \underline{T_{ip}}\ge\overline{T_{j(o-1)}} \\ \underline{T_{jo}}\ge\overline{T_{i(p-1)}}}} (\min(\underline{T_{ip}}, \underline{T_{jo}}) - \max(\overline{T_{i(p-1)}}, \overline{T_{j(o-1)}})) \tag{4.9}$$

Therefore the dividend in Eq. (4.6) corresponds the intersection between the two time window sets i and j. This ensures that U_{ij} always stays between $[0, 1]$.

Figure 4.7 illustrates how T^O_{ij} T^{NO}_{ij}, T^A_{ij} and $T^A_{ij} - T^{NO}_{ij}$ would be calculated for two locations.

Thus the relatedness measure r_{ij} is given a value $0 \le r_{ij} \le 2\psi + \omega + \phi + \tau + \chi$.

Algorithm 11 shows a pseudo-code of the complete Similar Regret heuristic. It starts by removing a random order i from the solution and adding it to a set I. It then creates an array and adds all orders $j \notin I$ to this array, before it sorts it ascending based on r_{ij}. It then selects the order with the y^p highest relatedness to remove from s and add to I. This continues until q orders have been removed. The insertion part of this heuristic tries to improve on insertion algorithm from Section 4.4.6 by calculating a regret value, c^*_i. The regret value tries to predict the "what if I insert later" value of an order i. If we let S_{i1}, S_{i2} and S_{i3} represent the vehicle schedules with respectively first, second and third lowest insertion cost for an order $i \in N^P$. That means $\Delta f_{S_{i1}} \le \Delta f_{S_{i2}} \le \Delta f_{S_{i3}}$, where Δ represent the difference in objective value $f_{S_{ik}}$ by inserting order i in its $k \in [1, 2, 3]$ best vehicle schedule. We can then define the regret value as follows:

$$c^*_i = \Delta f_{S_{i2}} - \Delta f_{S_{i1}} + \Delta f_{S_{i3}} - \Delta f_{S_{i2}} \tag{4.10}$$

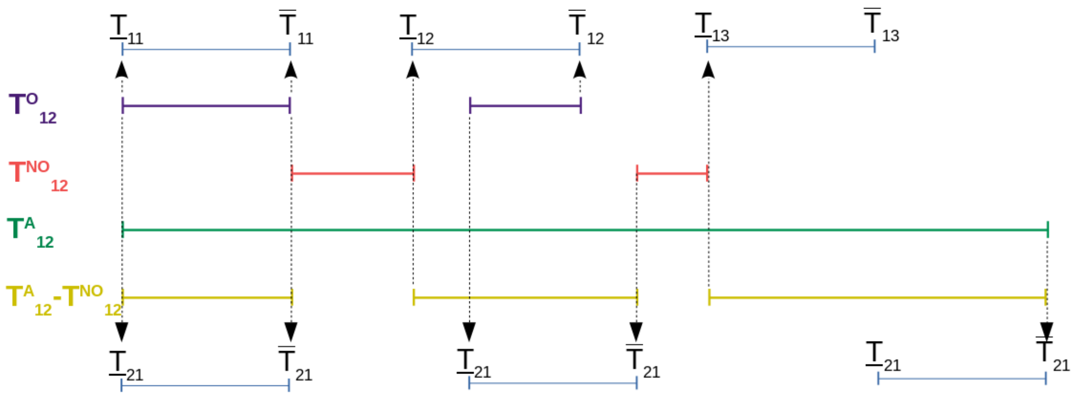


Figure 4.7: The figure shows two locations, 1 and 2, time window sets $[\underline{T_{1p}}, \overline{T_{1p}}]$ and $[\underline{T_{2p}}, \overline{T_{2p}}]$. The purple T^{O}_{12} is the overlapping time windows. The red T^{NO}_{12} represents the time when no location has a time window. The green T^{A}_{12} represents the whole span of both time windows and the yellow $T^{A}_{12} - T^{NO}_{12}$ represents the intersection of the two time window sets.

Algorithm 11 Similar Regret Heuristic

```
1: function SIMILARREGRET(solution $s$, $p$)
2:     select the number of orders to reinsert, $q$
3:     select a random order $i$ from $s$
4:     add $i$ to order set $I$
5:     repeat
6:         add all orders $j \notin I$ in array $A$ ascending based on relatedness $r_{ij}$
7:         choose a random number $y$ between $[0, 1]$
8:         remove the order in position $y^p$ in $A$ from $s$
9:         add removed order to $I$
10:    until $|I| = q$
11:    repeat
12:        sort $I$ ascending based on regret value $c_i^*$
13:        insert the first order $z$ from $I$ in its best possible position in $S_v$
14:        update $s$ based on $S_v$
15:        remove order $z$ from $I$
16:    until $|I| = 0$
17:    return $s$
```

The c_i^* therefore represents the difference in inserting order i in its best position and its second best position plus the difference in inserting it in its second best position and its third best position. In each iteration the heuristic chooses to insert the order with the highest c_i^*. The order will be inserted in its best possible position. Ties were broken by choosing the order with the lowest insertion cost c_i defined in Section 4.4.6.

This algorithm is expected to perform similarly with the heuristics from Sections 4.4.5 and 4.4.6, in terms of running time. This means it will be one of the more demanding heuristics when it comes to running time.

4.5 Choosing a Heuristic

We proposed several heuristics of different classes in the previous sections, and one could choose one of them and use them throughout the search. However we propose to use all the presented heuristics, for now. In Chapter 5 we will reduce the number of heuristics based on their performances in statistical experiments. The reason for including several heuristics is that the swap heuristic from Section 4.4.1 could be good for one type of instance, while the Similar Regret heuristic from Section 4.4.2 might be good for another type of instance. We think that alternating between several types of heuristics gives us a more robust algorithm.

To select a heuristic in each iteration of Algorithm 6, we use a *roulette wheel principle*. This means that we represent each heuristic by an index $h \in [1, 2, .., m]$ where m is the number of heuristics. We select a heuristic with a probability p_h calculated based on each heuristics *weight* w_h as follows:

$$p_h = \frac{w_h}{\sum_{g=1}^{m} w_g} \tag{4.11}$$

These weights could be set fixed per problem but we choose to use an adaptive weight system explained further in the next section.

4.6 Adaptive Weight Adjustment

The weights from Section 4.5 can be adapted automatically by the algorithm. Ropke and Pisinger (2006) implemented an adaptive weight system in a Large Neighbourhood Search algorithm. The basic idea is to keep track of the performance of each heuristic through a scoring system. A heuristic is given a higher score for a better performance and a low score for low performance. The loop in Line 5 in Algorithm 6 is divided into *segments*, or a number of iterations. Each segment in our algorithm has 100 iterations. At the beginning of the algorithm, each heuristic is given the same weights, resulting in equal probability in selecting each heuristic for the first segment. Throughout a segment, each heuristic is rewarded points based on the following system:

- Finding a new global best solution is given a high score to the heuristic for that iteration.

- Finding a new solution that is better than the current solution gives a medium score to the heuristic for that iteration.

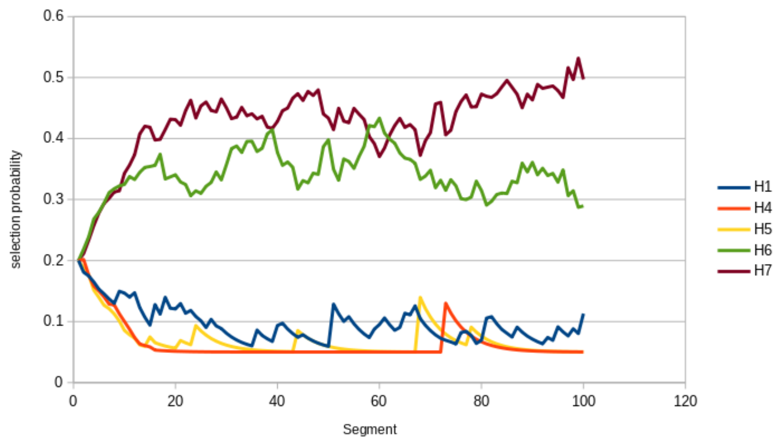


Figure 4.8: The figure illustrates an example of the development of the heuristics weight probability p_h when running our model. The x-axis represents a segment and the y-axis the probability of selecting a heuristic.

- Finding a new solution that has not been found before is rewarded a small score to the heuristic for that iteration.

After a segment, the sum of the scores for each heuristic are used to update the weights. If we let w_{hg} be the weight of heuristic h in segment g, as well as π_h and λ_h be the score and number of times the heuristic h was run in the current segment, then the update would be as follows:

$$w_{hg} = w_{h(g-1)}\Upsilon + (1-\Upsilon)\frac{\pi_h}{\lambda_h} \tag{4.12}$$

The Υ represents here a historical weight factor which we have set to 80% meaning we let the previous weight compose 80% of the new weight. The new weight is then also composed of $1-\Upsilon = 20\%$ of the weight from the current segment $\frac{\pi_h}{\lambda_h}$.

Figure 4.8 shows how the weight probability p_i develops for five different heuristics in an example from our model. We observe that the probability starts out equal but that some heuristics are getting higher weights after only a few segments. The figure also shows that we have put a lower limit to the probability to make certain that every heuristic will be selected at least a few times during a segment.

4.7 Acceptance Criteria and Stopping Condition

When searching for a new solution we could choose to only accept solutions that are better than the current one in Line 13 in Algorithm 6. By accepting we mean that our model chooses the new solution as its current solution and continues the search from there. However this could lead our model to get stuck and not be able to explore the entire solution space Blum and Roli (2003). We have therefore chosen the acceptance criteria used in simulated annealing, mentioned in Section 3.2. This acceptance criteria is accepting a solution that is better than the current solution. Furthermore it accepts a worse solution with the probability $e^{-|f-f_{new}|/T}$, where $T < 0$ is the temperature, f is the objective value of the current solution and f_{new} is the objective value with the new solution. The *cooling schedule*, which means the way at which T is decreasing, we use here was implemented by Crama and Schyns (2003). That sets a certain starting temperature T_{start} that decreases per iteration with a certain cooling rate $0 < c < 1$. We wanted to let T_{start} depend on the problem instance our model is trying to solve. Therefore we run 100 iterations with a fixed acceptance rate of $a = 0.8$. We calculate the average objective of all worse solutions that are accepted over these iterations $f^T_{average}$. Then we use this to calculate the fitting starting temperature as follows:

$$T_{start} = \frac{f^T_{average}}{ln(a)} \tag{4.13}$$

The Algorithm 6 stops when a specified number of iterations are reached which we specify here as 10 thousand iterations.

4.8 Wild Escape Algorithm

Algorithms containing large neighbourhood search heuristics, such as our model, are known to be good at searching locally as well as globally. However considering that our model is more focused on intensification, it could be that we end up getting stuck in a part of the solution space. It is important that our algorithm is able to react in these situations. In Algorithm 6 we see in Line 6 that if an escape condition is fulfilled, we will run a Wild Escape function. We set the criteria that if we, for 500 iterations, do not find an improvement in s_{best}, we will perform this action.

Algorithm 12 Wild Escape Algorithm

```
1: function WILDESCAPE(solution $s$, $s_{best}$, set of heuristics $\mathbb{H}$)
2:     repeat
3:         choose a random heuristic $h$ from $\mathbb{H}$
4:         apply $h$ to $s$
5:         if $f(s) < f(s_{best})$ then
6:             $s_{best} \leftarrow s$
7:     until stop condition met
8:     return $s$
```

The algorithms pseudo-code is described in Section 4.8. The stopping condition, in Line 7, is 20 iterations. The algorithm accepts any new solution found regardless of the objective value to move as far away from the current solution as possible. Lines 5 to 6 in Section 4.8 are saying that if we happen to find a better solution s_{best} on our way out of the current solution space, we should remember it. We still continue moving away from the current solution s in spite of finding a new s_{best}. The heuristics used by the escape algorithm are the heuristics random fit from Section 4.4.4, 3-exchange from Section 4.4.2 and swap Section 4.4.1. The heuristics have an increased size of q to increase diversification. We chose these heuristics because they are not trying to improve the solution in any specific way and select targeted solutions, but rather moves randomly around the solution space for increased diversification.

Chapter 5

Experimental Results

This chapter presents the results from several experiments leading us to the final composition of our 4th party logistics optimizer (4PLO). The final experiments will compare the performance of our 4PLO model to a benchmark created by the mathematical model from Chapter 2 implemented in AMPL .

We start by explaining the experimental setup of our testing in Section 5.1. Then we will explain the instances we used in our experiments in Section 5.2.

Section 5.3 contains the initial experiments of our model which lead us to the final composition of our model. The first part, Section 5.3.1, starts by testing the Wild Escape Algorithm from Section 4.8 and it finds that the Wild Escape Algorithm outperforms a random restart and not using any escape algorithm. Then in Section 5.3.2 we rank the heuristics performance and do ANOVA(III) and multiple linear regressions of the heuristics described in Chapter 4. We find here that three heuristics have a significant positive influence on the result, the Random Rit algorithm from Section 4.4.4, the Greedy heuristic from Section 4.4.6, and Similar Regret heuristic from Section 4.4.7.

We continue our initial experiments in Section 5.3.3 with extended ANOVA(III), multiple linear regression models and pairwise t-tests to evaluate if combining the not significant heuristics with the significant ones from Section 5.3.2, can have a significant positive impact on the result. Here we find that the cluster heuristic from Section 4.4.5 has a significant positive influence on the best solution found. We also find that the Swap heuristic from Section 4.4.1 has no significant influence on the result.

In Section 5.3.5 we present the final composition of our model which contain the heuristics mentioned above as well as the Wild Escape algorithm.

The final part of this chapter Section 5.4 contains the evaluation of our final model composition and compares the results from the mathematical model from Chapter 2 with the results of our model. It also presents an evaluation of the performance of the included heuristics. The results show that the proposed algorithm is robust and efficient.

5.1 Experimental Setup

In this section we describe the technical setup of the experiments as well as the analytical setup of our experiments.

Technical Setup

The computational experiments in this thesis are run on a 64-bit Ubuntu 18.04 computer with a 1.8 Ghz quad core i7-8550u processor and 16GB RAM. In Section 5.2 we will describe an instance generator and this was implemented using Java SE JDK version 11.0.4. The mathematical model from Chapter 2 is setup in AMPL IDE version 3.5.0.201802211231, using the Gurobi solver version 8.1. Our proposed solution method from Chapter 4 was implemented using Java SE JDK version 11.0.4. In Sections 5.3.2 to 5.3.4 we will perform ANOVA(III), multiple linear regressions and t-tests, which are performed in Matlab R2019a version 9.6.0.1174912.

Analytics Setup

To test our model from Chapter 4 we generated five instance sets, each containing five instances of different sizes, totally 25 instance. Each instance set is described in Section 5.2. While testing the algorithm from Section 4.8, we used one of these instance sets, ie. 5 instances. All tests were run 10 times and results are given as an average and best objective value over the 10 runs as well as an average running time. To analyse the performance of each of the heuristics in Section 5.3.2 and Section 5.3.3, 5 reasonably sized instances (80 orders) were solved 10 times using each of the $2^7 = 128$ combinations of heuristics. Here we also try to determine which of the heuristics influence the result by performing several statistical experiments, including ANOVA (III) and multiple linear regression analysis, and pairwise t-tests. For all statistical experiments in this thesis we have used a 95% confidence interval. After the analysis of the heuristics, a final composition was chosen for further testing in Section 5.4. We then compared the performance of the final composition with solutions found by the mathematical model in AMPL. We also present figures which show the performance of our selected heuristics in arbitrarily selected runs.

5.2 Instances

Many instance generators have been created for pickup and delivery problems. An instance generator for the classical Pickup and Delivery Problem was implemented by Li and Lim (2001). Later Hemmati et al. (2014) made an instance generator for tramp shipping routing and scheduling problems. Our instance generator is created based on real world data from an anonymous customer of 4flow. In the following, we will describe how we designed the generator and how we generated the instances used in the analytical part of this thesis.

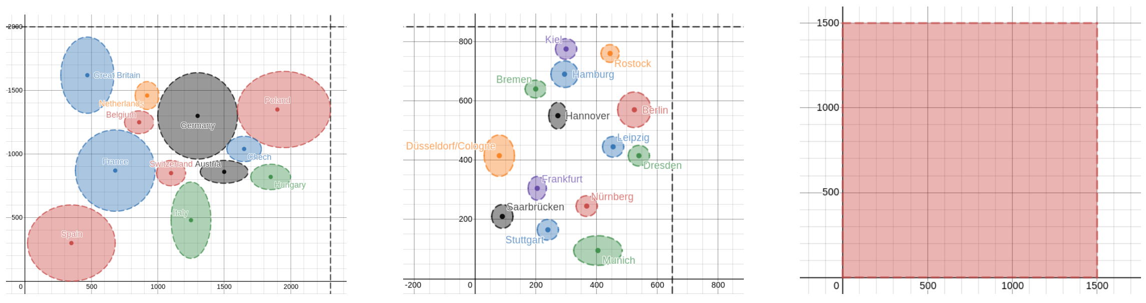
5.2.1 Generate Instances Based on Real World Data

The 4flow data gives information about the number of orders $|N|$, locations $|L|$, factories $|F|$. The amount of vehicles $|V|$ are selected based on 4flow data and are in the bound between $|N|/2 \leq |V| \leq 2/3|N|$. We give $|N|, |V|, |L|, |F|$ as input to the generator.

In addition the data from 4flow gave us information about the size of a vehicle and its compatibilities. Some vehicles might have cooling possibilities, some orders might require special equipment for transport, (for example breakable or explosive goods) or there might be equipment required at the pickup or delivery location (for example a crane to unload goods). We refer to these capabilities as a *vehicle requirement*. Other information acquired by the data was travel distances and cost structure.

In the following we refer to the mathematical symbols from Chapter 2. To keep the instances feasible but still as realistic as possible it makes sense to limit the data to ranges or pre-designed patterns. Our data was generated based on the following possibilities:

- Orders are assigned to pickup and delivery locations randomly. Orders assigned to the same location are given the same stop L_s.
- Each delivery location is independently assigned to a factory at random N_f.
- 5% of each order and location were on average given a vehicle requirement. This will decide which vehicle can pickup which order, N_v^P and N_v^D.
- We let the vehicles types be split up in 3 different vehicle types, small, medium, large, each with expected capacities and capabilities.
  - Large vehicle: slow speed vehicle compatible with all locations and orders, with $Q_v^{kg} = 24k$ and $Q_v^{vol} = 102$.
  - Medium vehicle: medium speed vehicle and compatible with all locations but not orders with vehicle requirements, with $Q_v^{kg} = 18k$ and $Q_v^{vol} = 71$.
  - Small vehicle: fast speed vehicle, not compatible with any vehicle requirements, with $Q_v^{kg} = 12k$ and $Q_v^{vol} = 55$.
- The distances d_{ij} are euclidean distances between the randomly generated points described in the next paragraph.
- The travel time T_{ijv} were scaled with 60% of the travel distance, added with a random variation of $+/-10\%$ of the travel distance. The travel time also depends on the speed of the vehicle mentioned above. Slow speed vehicles had a 5% increase in travel time while medium speed had a 2.5% increase and fast speed vehicles had no increase in travel time.
- The cost matrices $C_{v\alpha\beta}^{km}$, $C_{v\alpha\beta}^{kg}$, $C_{v\alpha\beta}^{fix}$ were based on a real 4flow cost matrix, scaled to the size of the instance and to the size of the vehicle.
- The cost of not transporting an order C_i was set to a minimum lower bound (the most expensive transport) and scaled based on the weight, volume and distance of the order.
- The stop costs C_{vi}^{stop} were calculated relative to the size of the vehicle and the cost data from 4flow.
- Time windows $[\underline{T_{ip}}, \overline{T_{ip}}]$ were generated randomly based on typical factory opening hours. Meaning, one to two time windows per day, and three to seven days per week scaled based on the instance size.



(a) Europe **(b)** Germany **(c)** Uniform

Figure 5.1: The figure shows the area of random point generation for our instance generator. For Europe and Germany, shaded areas indicate a possible location generation. Larger areas are more likely to generate a point. Uniform means location points were generated using a uniform distribution within the greyed area.

Random Locations Based on Real Geographical Data

Most of 4flow's customers are based either in Germany or in Europe. To make the instance generator as realistic as possible we have decided to split the instances into 3 geographical types; European, German and uniform geographically distributed locations. We made 2 maps based on real scale approximations of geographical data from National Geographics, in km. We have used to geographical points with an elliptic uniformly distributed area surrounding the point to represent a country or a city. Fig. 5.1 illustrates the areas of possible locations used in the generator. Larger ellipses are more likely to be selected by the generator than smaller ellipses.

For the selected ellipse a point was selected within the ellipse at random with a uniform distribution. For Fig. 5.1c points were generated at random within the limits shown. From our 5 instance sets, two were generated using Fig. 5.1a, two with Fig. 5.1b and one with the uniform distribution from Fig. 5.1c. If a point belong in the same factory as a previously generated point, that point was generated within a reasonable radius of three kilometer.

5.2.2 Generated Instances

For testing our algorithm, 5 instance sets of each 5 instances of varying sizes were generated. We have numbered the sets as follows:

- Set 1 and Set 2 are generated based on the European map from Fig. 5.1a.
- Set 3 and Set 4 are generated on the German map from Fig. 5.1b.
- Set 5 is generated on the uniform distribution from Fig. 5.1c.

Each instance set contains representative instance sizes of our problem based on data from 4flow, $4, 12, 35, 80$ and 150 orders using respectively $3, 7, 20, 45$ and 80 vehicles and containing $7, 9, 22, 45$ and 85 locations. The sizes of the instances and the instance set number are shown for each result presented in the following sections.

Table 5.1: The table shows the results of running the 4PLO model, with all heuristics, using 3 different reset algorithms. The columns contains in order: the first three columns show the instance size in number of orders, vehicles and locations. Three columns that contain the average solution from runs with the different escape modifications follow: no escape, random reset and our Wild Escape Algorithm. The next three columns contain the best solution found for the same three escape modifications, and the final three columns contain the average running time in seconds for the three escape modifications.

				Average Objective			Best Objective			Running time (sec)		
			Initial	No	Random	Wild	No	Random	Wild	No	Random	Wild
#Ord	#Veh	#Loc	objective	escape	reset	escape	escape	reset	escape	escape	reset	escape
4	3	7	609 680.3	3 444.7	3 444.7	3 444.7	3 444.7	3 444.7	3 444.7	0.18	0.20	0.20
12	7	9	1 023 745.5	149 692.6	154 832.9	149 692.4	149 692.4	149 692.4	149 692.4	0.39	0.51	0.46
35	20	22	2 682 067.9	10 639.1	10 849.5	10 350.9	10 404.9	10 358.6	10 025.1	2.31	1.61	2.49
80	45	45	6 422 128.6	22 262.2	25 802.9	21 377.4	20 761.2	21 777.4	20 831.3	15.89	8.05	14.97
150	80	85	12 059 380.3	40 667.2	38 313.0	35 705.7	34 316.0	34 345.0	34 282.3	88.21	48.92	77.78

5.3 Initial Results

In this section we present the results and observations from our initial experiments. This data lead us to the final composition of our model. We have done this in 3 parts. Section 5.3.1 contains the evaluation of our Wild Escape Algorithm. Then follows Section 5.3.2 which is the initial evaluation of the heuristics. Finally Section 5.3.3 contains the further evaluation of the heuristics. Afterwards, in Section 5.3.5, we will present the final composition of the model.

5.3.1 Evaluation of the Wild Escape Algorithm

To help our algorithm escape from a local optimum, we designed a Wild Escape Algorithm described in section Section 4.8. To evaluate the implementation of this algorithm we have decided to compare the result of running the complete model from Chapter 4 including all heuristics, with three different modifications. We ran all modifications on instance Set 1. First modification was that we ran the algorithm without any escape algorithm, referred to as *no escape*. Secondly we ran the algorithm with a modification that resets the algorithm each time we get stuck in a local optimum and then start from a new random solution, referred to as *random reset*. Lastly, we ran our algorithm with our *Wild Escape Algorithm*. Comparing these three options towards each other will help us evaluate if our Wild Escape Algorithm is helping us in general and see if it is different than doing a reset and just starting from a new solution somewhere.

Results of Escape Modifications

We ran our algorithm 10x3 times, ten for each escape adjustment, on instance set 1 from Section 5.2. We modified our escape algorithm on this run to ignore if a best solution is found while moving from one neighbourhood to the next. This way the extra iterations explained in Section 4.8, does not give any unfair advantage to our Wild Escape Algorithm. We will simply be moving from one neighbourhood to the next without checking if we find a better solution on the way.

The result from running our algorithm with these three adjustments are summarized in Table 5.1. It shows the objective values found on average during each of the 10 runs, the best solution found overall and the average running time, for each escape adjustment.

Observations of Results from the Escape Modifications

The Table 5.1 shows that on average our Wild Escape Algorithm clearly outperforms both the restart algorithm and not including an escape algorithm. We see that the average result for all instance sizes are lowest using our Wild Escape Algorithm. In addition our algorithm outperforms the alternatives for all instances in finding the best solution, except for the instance with 80 orders, where not using an escape algorithm ended up finding a better best solution. This is however probably due to the algorithm starting on one run in a good neighbourhood. Since the escape algorithm is significantly outperforming it on average we do not analyse this any further.

With regards to the running time we observe the following pattern for the largest two instances: the random restart is the fastest, followed by our Wild Escape Algorithm, and lastly not using any escape algorithm. It is expected that the random reset here outperforms the other modification as we have generated the random solutions before starting our algorithm. This gives random reset an unfair advantage due to running time. Taking this into regards makes it clear that our Wild Escape Algorithm outperforms the alternative modifications and should be included in a final composition.

5.3.2 Initial Evaluation of Heuristics

Different heuristics have different strengths and weaknesses. Some heuristics are not performing well on their own, but work very well in combination with other heuristics. Other heuristics are strong on their own, and their performance is prohibited by other heuristics, or not getting used often enough when too many heuristics are included.

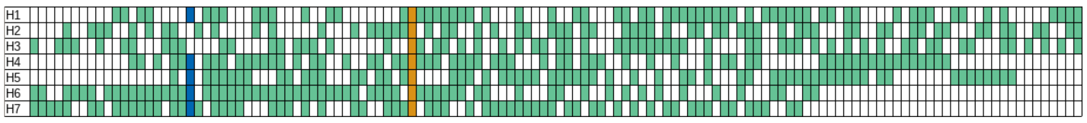
To find the right combination of heuristics we have done an evaluation of their performance to search for the best possible combination of heuristics. To do this, we selected all the five instances with a representative size of 80 orders and ran our algorithm, without the Wild Escape Algorithm adaptation, with each possible combination of heuristics. This results in $2^7 = 128$ algorithm runs, each running 10 times with a different random seed. To be able to compare the results from different instances we calculated the improvements in percent from the initial solution. We present the improvement of the best solution found, during the 10 runs, and an average improvement from the initial solution from the 10 runs. Each percentage was also multiplied by 1000 to make the results more readable.

We evaluated the data resulting from the different combination in three steps:

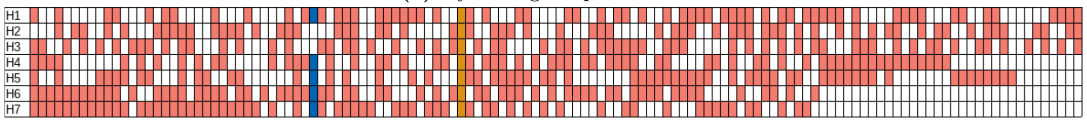
- In the first part we ranked the combinations based on the best improvement and the average improvement.

- In the second part we ran ANOVA and regression analysis to see which heuristics have a significant impact on the result.

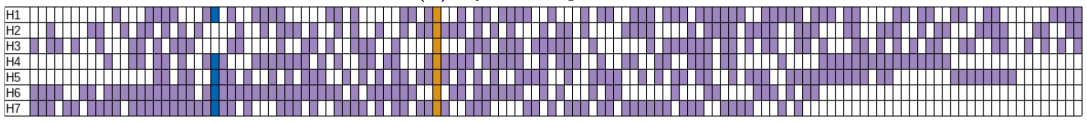
- In the third part we run t-tests to see if certain heuristics in combination with others have a positive or a negative impact on the final result.



(a) By average improvement



(b) By best improvement



(c) By best+average improvement

Figure 5.2: Figure shows the ranking of improvements from initial solution. Highlighted tiles indicate use of a heuristic. The combinations with better results are to the left. The blue highlighted tiles show the final composition from Section 5.3.5. Yellow highlighted tiles show the use of all heuristics.

Ranking the Combinations

We shorten the names of each heuristic and will refer to them here on as:

- H1-swap: the Swap heuristic described in Section 4.4.1.
- H2-exchange: the Exchange heuristic from Section 4.4.2.
- H3-2opt: the 2-opt heuristic in Section 4.4.3.
- H4-random: the Random Fit heuristic from Section 4.4.4.
- H5-cluster: the Clustering heuristic described in Section 4.4.5.
- H6-greedy: the Greedy heuristic in Section 4.4.6.
- H7-similar: Similar Regret heuristic described in Section 4.4.7.

Fig. 5.2 shows the ranking of each combination of heuristics from left to right. A colored tile indicates that the current heuristic is in use. The further a combination is to the left, the higher improvement was from the initial solution compared to the other combinations.

The Fig. 5.2a shows the combination of heuristics ranked based on the average improvement over the 10 runs for that combination. Then Fig. 5.2b shows the combination of heuristics ranked based on the best improvement overall during the 10 runs. Finally Fig. 5.2c shows the ranking of the average improvement + the best improvement to see which combinations performs best overall. The blue highlights tiles indicates the final composition from Section 5.3.5 and the yellow highlighted tiles show the use of all heuristics.

ANOVA and Regression Analysis

The ranking gives us an overview over which heuristics are working and is always part of a good combination. It also gives us information on which heuristics are not performing well overall and which combination of heuristics are not working well.

Table 5.2: The table shows an Analysis of variance (ANOVA III) on the result performance while including the listed heuristic. The columns contains in order: the source of the variability and for each source the sum of the squares, the degrees of freedom, the mean squares, the F-statistic and the p-value.

Source	Sum sq.	df	Mean sq.	F	P>F
H1-swap	0.052	1	0.052	0.09	0.7682
H2-exchange	0.748	1	0.748	1.26	0.2628
H3-2opt	0.038	1	0.038	0.06	0.8017
H4-random	22.432	1	22.432	37.66	0
H5-cluster	12.734	1	12.734	21.38	0
H6-greedy	117.945	1	117.945	198	0
H7-similar	112.406	1	112.406	188.7	0
Instance	371.704	4	92.926	156	0
Error	350.257	588	0.596		
Total	975.207	599			

(a) Average improvement statistics

Source	Sum sq.	df	Mean sq.	F	P>F
H1-swap	0.075	1	0.0745	0.23	0.6306
H2-exchange	0.195	1	0.1949	0.61	0.4369
H3-2opt	0	1	0.0001	0	0.988
H4-random	8.372	1	8.3718	26	0
H5-cluster	3.679	1	3.6789	11.43	0.0008
H6-greedy	66.236	1	66.2357	205.71	0
H7-similar	67.081	1	67.0813	208.33	0
Instance	344.971	4	86.2428	267.84	0
Error	189.331	588	0.322		
Total	671.378	599			

(b) Best improvement statistics

Table 5.3: The table shows results of multiple linear regression model on the performance result of including the listed heuristic. The columns contain the following in this order: the term and for each term the coefficient estimate, the standard error of the coefficients, t-statistics to test if the term is significant, and the p-value.

Term	Estimate	SE	tStat	pValue
Intercept	994.58	0.11713	8491.5	0
H1-swap	-0.018582	0.063017	-0.29487	0.7682
H2-exchange	-0.07063	0.063017	1.1208	0.26283
H3-2opt	-0.01583	0.063017	-0.2512	0.80175
H4-random	0.39108	0.063729	6.1366	1.5502e-09
H5-cluster	-0.29466	0.063729	-4.6236	4.6407e-06
H6-greedy	0.89676	0.063729	14.071	5.7098e-39
H7-similar	0.87544	0.063729	13.737	1.923e-37
Inst1	0.50928	0.099639	5.1113	4.3337e-07
Inst2	-0.051052	0.099639	-0.51237	0.60859
Inst3	1.6601	0.099639	16.662	2.4375e-51
Inst4	1.755	0.099639	17.614	4.4128e-56

(a) Average improvement statistics, $R^2 = 0.641$

Term	Estimate	SE	tStat	pValue
Intercept	994.97	0.086114	11554	0
H1-swap	0.022291	0.046332	0.48112	0.63061
H2-exchange	-0.036047	0.046332	-0.77801	0.43687
H3-2opt	-0.00069693	0.046332	-0.015042	0.988
H4-random	0.23891	0.046855	5.099	4.6112e-07
H5-cluster	-0.15838	0.046855	-3.3801	0.00077249
H6-greedy	0.67202	0.046855	14.342	3.2e-40
H7-similar	0.67629	0.046855	14.434	1.2062e-40
Inst1	0.64176	0.073257	8.7605	2.0732e-17
Inst2	0.16783	0.073257	2.291	0.022316
Inst3	1.837	0.073257	25.077	6.2887e-95
Inst4	1.6686	0.073257	22.778	8.2582e-83

(b) Best improvement statistics, $R^2 = 0.718$

However, ANOVA (III) and multiple linear regression analysis can help us evaluate which heuristics have a significant positive impact on the result. Table 5.2 shows the results of ANOVA (III) analysis performed using each heuristic as a source of variance. We extended the model with the instances as random effects to give us a better explanation of the result and less noise in the model. We did one ANOVA analysis for the average improvement over 10 runs shown in Table 5.2a, and one for the best improvement found in Table 5.2b, to see if some heuristics are making the model more robust (average) and if some heuristics are good at finding best known solutions (best).

In addition to this we performed a multiple linear regression in Table 5.3 on the same data as the ANOVA. The multiple linear regression analysis gives us insight into if a heuristic is positively or negatively influencing the result, as well as how well the heuristics are explaining the result through the R^2. Also here we used the instances as a random effect.

Observations from Initial Evaluation of Heuristics

Ranking the combinations in Fig. 5.2 does indicate that using all heuristics are giving a performance in the middle of all rankings which indicates we should use a combination that excludes some heuristics. It also shows that H6-greedy and H7-similar are usually included in the better performing combinations.

The results from Table 5.2 and Table 5.3 split our heuristics into two groups. The first obvious group are the significant heuristics. The ANOVA (III) results from table Table 5.2a and Table 5.2b tell us that four heuristics, H4-H7, have a significant impact on the result using a 95% confidence interval. However from Table 5.3a and Table 5.3b we see that only 3 heuristics, H4-random, H6-greedy and H7-similar, have a positive estimated coefficient, ie. a positive significant influence, for both average and best improvement (H1-swap is positive for best but negative for average improvement). From this we can safely conclude that H4-random, H6-greedy and H7-similar are significantly, positively improving the result of our model on average and they are good at finding good solutions. H5-cluster is significantly influencing the result but with a negative coefficient. This leads us to conclude that this heuristic either performs poorly alone, or has a negative influence on the model. To figure out more we need to do further testing.

Regarding the remaining heuristics, H1-swap, H2-exchange, and H3-2opt, we can safely conclude that they are not contributing significantly on the result on their own, ie. an algorithm containing only these heuristics are not having a significantly positive impact on the results using a 95% confidence interval. However it could be that they have a positive effect in on the result if they are used in combination with the significant heuristics. We therefore observe that we need further testing to know if these heuristics have a positive or negative influence on the result in combination with other heuristics. We refer to the heuristics H1-swap, H2-exchange, H3-2opt and H5-cluster as the undecided group, or G in our further evaluation of the heuristics.

5.3.3 Further Evaluation of Heuristics

To further analyse the performance of the heuristics we want to analyse how the heuristics are performing as an undecided group (G), see Section 5.3.2, to see if they have an effect on the results. The result from Section 5.3.2 tells us that it is out of the question to use any combination of heuristics where only heuristics from the undecided group are used. These heuristics will alone have a negative impact on the result but it is still possible that using them combined with the significant heuristics could have a positive impact on the result We have therefore removed the observations where the undecided group appear alone in the following testing. We want to test if using one or several of the G heuristics in combination with other heuristics have a positive impact on the result. We have done this in two parts. We first wanted to see if the heuristics as a group, G, has a positive or negative influence on the result. To test this we have done 2-sample t-tests to compare the mean of the population where we combine the G heuristics with some significant heuristic, to the mean of the population when we are not using G. Then we wanted to see if using G with specific combinations of the significant heuristics have significant impact on the model. We did this using further ANOVA (III) statistical analysis and multi linear regression model.

Evaluation of Undecided Heuristics as a Group

The first thing we did was to run a t-test that compares the mean of the population which include some combination of the G heuristics and the significant heuristics, towards the population that does not contain any heuristic from G. To represent the population without any heuristics from G we have the used the parameter P_N

Table 5.4: The results show t-tests on the undecided group mean results versus no undecided heuristics mean result. The columns contain in order: the populations tested against each other, type of data tested(average or best improvement), tail which determines the alternative hypothesis, h-value (1 rejects the null hypothesis, 0 failure to reject), p-value of the test, t-Statistic of the test, confidence interval for the true population mean.

	Improvement					Confidence
Populations	type	Tail	H-stat	p-value	tStat	interval
$P_H - P_N$	Average	both	0	0.5187	-0.6458	-0.3785 – 0.1912
$P_H - P_N$	Average	right	0	0.7407	-0.6458	-0.3326 – inf
$P_H - P_N$	Average	left	0	0.2593	-0.6458	-inf – 0.1453
$P_H - P_N$	Best	both	0	0.6903	0.3986	-0.2174 – 0.3281
$P_H - P_N$	Best	right	0	0.3452	0.3986	-0.1734 – inf
$P_H - P_N$	Best	left	0	0.6548	0.3986	-inf – 0.2841

Table 5.5: The table shows the analysis of variance with the undecided group in combination with the significant heuristics. The columns contain in order: the source of the variability and for each source the sum of the squares, the degrees of freedom, the mean squares, the F-statistic and the p-value.

Source	Sum sq.	df	Mean sq.	F	P>F
G+H4	19.024	1	19.0236	538.34	0
G+H6	0	1	0	0	0.9859
G+H7	0.005	1	0.0045	0.13	0.7204
G+H4+H6	0.059	1	0.0589	1.67	0.1973
G+H4+H7	0.001	1	0.0009	0.03	0.8722
G+H6+H7	0.22	1	0.2202	6.23	0.0128
G+H4+H6+H7	0.166	1	0.1657	4.69	0.0308
Inst	309.022	4	77.2555	2186.21	0
Error	19.365	548	0.0353		
Total	385.274	559			

(a) Average improvement statistics

Source	Sum sq.	df	Mean sq.	F	P>F
G+H4	12.879	1	12.8793	609.42	0
G+H6	0.131	1	0.1307	6.19	0.0132
G+H7	0.239	1	0.2389	11.31	0.0008
G+H4+H6	0.223	1	0.2227	10.54	0.0012
G+H4+H7	0.153	1	0.1525	7.22	0.0074
G+H6+H7	0.42	1	0.4195	19.85	0
G+H4+H6+H7	0.395	1	0.395	18.69	0
Inst	295.738	4	73.9346	3498.45	0
Error	11.581	548	0.0211		
Total	352.547	559			

(b) Best improvement statistics

and to represent the population with some combination of the G-heuristics and the significant heuristics we have used the parameter P_H. The results from the t-test are summarized in Table 5.4.

We continued the testing of G by performing ANOVA (III) analysis on different combinations of the G and the significant heuristics. We did this to see if a combination of the undecided group G and the significant heuristics could help explain the variations in the result and to see if a combination of some or all of the significant heuristics work better than others.

The results are summarized in Table 5.5 and Table 5.6. As an example a source of G+H4, contains all observations where at least one of the undecided heuristics from G are combined exclusively with H4-random.

Observations from Results of the Heuristics Further Evaluation

The results from Table 5.4 tells us that we cannot reject the null hypothesis, for all populations, that the mean of the two populations are different using a 95% confidence interval. This is the case for both, average and best improvement. This tells us that the heuristics from the undecided group (G) either have no significant impact on the result, or that the effect from some are nulling out the others. Further testing is needed to make any further conclusions.

Table 5.5 and Table 5.6 gives us further insight into our model. First of all the result from Table 5.5a tells us that only two combinations of the undecided group

Table 5.6: The table shows the results of multiple linear regression model with the undecided group heuristics in combination with the significant heuristics. The columns contain in order: the term and for each term the coefficient estimate, the standard error of the coefficients, t-statistics to test if the term is significant, and the p-value.

Term	Estimate	SE	tStat	pValue
Intercept	995.85	0.035525	28032	0
G+H4	-0.89285	0.038481	-23.202	1.8003e-83
G+H6	0.00068095	0.038481	0.017696	0.98589
G+H7	0.013782	0.038481	0.35815	0.72037
G+H4+H6	0.049672	0.038481	1.2908	0.19731
G+H4+H7	-0.0061945	0.038481	-0.16097	0.87217
G+H6+H7	0.096062	0.038481	2.4963	0.012841
G+H4+H6+H7	0.083317	0.038481	2.1651	0.030808
Inst1	0.53458	0.02512	21.281	1.0604e-73
Inst2	0.031671	0.02512	1.2608	0.20793
Inst3	1.71	0.02512	68.073	1.6221e-269
Inst4	1.5959	0.02512	63.531	6.3233e-255

(a) Average improvement statistics, $R^2 = 0.95$

Term	Estimate	SE	tStat	pValue
Intercept	995.88	0.027473	36249	0
G+H4	-0.73464	0.029759	-24.686	5.012e-91
G+H6	0.07402	0.029759	2.4873	0.013167
G+H7	0.10006	0.029759	3.3625	0.00082635
G+H4+H6	0.096599	0.029759	3.246	0.0012417
G+H4+H7	0.079948	0.029759	2.6865	0.0074396
G+H6+H7	0.13259	0.029759	4.4554	1.0157e-05
G+H4+H6+H7	0.12865	0.029759	4.3232	1.8269e-05
Inst1	0.70773	0.019426	36.432	1.6371e-148
Inst2	0.27728	0.019426	14.273	1.5823e-39
Inst3	1.873	0.019426	96.415	0
Inst4	1.5781	0.019426	81.237	8.6749e-308

(b) Best improvement statistics, $R^2 = 0.967$

and the significant variables have a significant impact on the result using a 95% confidence interval. Using some heuristics from the undecided group combined with heuristic H6-greedy and H7-similar, as well as H4-random has a significant impact on the average improvement result. From Table 5.6a we also see that the combinations are also getting a positive coefficient. We also observe that the $R^2 = 0.95$ is very high so this model explains the results very well and this supports the use of these heuristic combination in regards to the average improvement.

The same combinations are significant and positive in regards to the best improvement tables Table 5.5b and Table 5.6b. And even though more combinations are significant for best improvement, the combinations with highest positive estimated coefficients are still including H6-greedy and H7-similar, or H6-greedy H7-similar and H4-random. We also observe here the increased $R^2 = 0.967$ which indicates that this model is explaining the result from best and average improvement very well. This could indicate that the combinations with H6-greedy, H7-similar and H4-random are consistently contributing to the same positive results.

The results from the further testing tells us that there are combinations of the undecided group and the significant heuristics that have a positive significant impact on the result of both average and best improvement. It supports our results from Section 5.3.2 of significant heuristics H6-greedy, H7-similar and H4-random and leads us to conclude that there might be a positive influence from the undecided group heuristics, however further testing is necessary to determine which heuristics should be included.

5.3.4 Evaluation of Individual Heuristics

Until now, the heuristic H4-random, H6-greedy and H7-similar have been proven significant. The heuristics H1-swap, H2-exchange, H3-2opt and H5-cluster have been proven significant in combination with the significant heuristics but not alone. We continue referring to them as the undecided group heuristics or G.

We want to figure out which of the heuristics in the undecided group, if any, have a positive, or negative, influence on the result. To do this we performed pairwise t-tests to check if an undecided heuristic is significantly improving or decreasing the best and average improvement. Like in Section 5.3.3 it is out of the question to

Table 5.7: The table shows results of t-tests on individual heuristics. The columns contain in order: the populations tested against each other, type of data tested(average or best improvement), tail which determines the alternative hypothesis, h-value (1 rejects the null hypothesis, 0 failure to reject), p-value of the test, t-Statistic of the test, confidence interval for the true population mean

Populations	Tail	H-stat	p-value	tStat	conf-int
$P_{HA}^{H_1} - P_{NA}^{H_1}$	both	0	0.5427	-0.6090	-0.1580 – 0.0727
$P_{HA}^{H_1} - P_{NA}^{H_1}$	right	0	0.7286	-0.6090	-0.1325 – inf
$P_{HA}^{H_1} - P_{NA}^{H_1}$	left	0	0.2714	-0.6090	-inf – 0.0472
$P_{HB}^{H_1} - P_{NB}^{H_1}$	both	0	0.9240	-0.0955	-0.1428 – 0.1296
$P_{HB}^{H_1} - P_{NB}^{H_1}$	right	0	0.5380	-0.0955	-0.1208 – inf
$P_{HB}^{H_1} - P_{NB}^{H_1}$	left	0	0.1076	-0.0955	-inf – 0.1076
$P_{HA}^{H_2} - P_{NA}^{H_2}$	both	1	3.4853e-10	-6.5098	-0.0692 – 0.0412
$P_{HA}^{H_2} - P_{NA}^{H_2}$	right	0	1.0000	-6.5098	-0.0661 – inf
$P_{HA}^{H_2} - P_{NA}^{H_2}$	left	1	1.7426e-10	-6.5098	-inf – 0.0443
$P_{HB}^{H_2} - P_{NB}^{H_2}$	both	1	4.5425e-10	-3.5486	-0.0370 – 0.0106
$P_{HB}^{H_2} - P_{NB}^{H_2}$	right	0	0.9998	-3.5486	-0.0349 – inf
$P_{HB}^{H_2} - P_{NB}^{H_2}$	left	1	2.2712e-04	-3.5486	-inf – 0.0127
$P_{HA}^{H_3} - P_{NA}^{H_3}$	both	1	0.0244	-2.2635	-0.0276 – -0.0043
$P_{HA}^{H_3} - P_{NA}^{H_3}$	right	0	0.9878	-2.2635	-0.0250 – inf
$P_{HA}^{H_3} - P_{NA}^{H_3}$	left	1	0.0122	-2.2635	-inf – -0.0069
$P_{HB}^{H_3} - P_{NB}^{H_3}$	both	0	0.6271	-0.4864	-0.0134 – 0.0081
$P_{HB}^{H_3} - P_{NB}^{H_3}$	right	0	0.6865	-0.4864	-0.0116 – inf
$P_{HB}^{H_3} - P_{NB}^{H_3}$	left	0	0.3135	-0.4864	-inf – 0.0063
$P_{HA}^{H_5} - P_{NA}^{H_5}$	both	0	0.4702	-0.7231	-0.0301 – 0.0118
$P_{HA}^{H_5} - P_{NA}^{H_5}$	right	0	0.7649	-0.7231	-0.0255 – inf
$P_{HA}^{H_5} - P_{NA}^{H_5}$	left	0	0.2351	-0.7231	-inf – 0.0071
$P_{HB}^{H_5} - P_{NB}^{H_5}$	both	1	1.7956e-04	-3.7972	-0.0167 – 0.0528
$P_{HB}^{H_5} - P_{NB}^{H_5}$	right	1	8.9779e-05	-3.7972	-0.0197 – inf
$P_{HB}^{H_5} - P_{NB}^{H_5}$	left	0	0.9999	-3.7972	-inf – 0.0499

use any combination of heuristics where only undecided heuristics are used, so we remove these observations from the data also in these tests. We test if the mean of the populations, where we combine the undecided heuristics with some significant heuristic, is significantly different than the population when we are not using an undecided heuristic.

T-tests of Individual Heuristics

Similar to the previous section we use the parameter $P_{NA}^{H_i}$ to refer to the population without a specific heuristic H_i for the average improvement, indicated by A, and the parameter $P_{HB}^{H_i}$ as the population including the heuristic H_i for the best improvement, indicated by B. Here H_i represents one of the heuristics described in the previous section. We did the test for all 4 of the undecided heuristics from the previous section H_1, H_2, H_3 and H_5. The results from the pairwise t-tests are summarized in Table 5.7.

Observations from the Results of the Individual Heuristic Evaluations

The first thing we see from the t-tests is that the effect of the heuristics are cancelling each other out, as some are left significant, and some are right significant. We go through each of the heuristics results from Table 5.7 here.

For the heuristic H1-swap the population mean of both best and average improvement is not significantly different using a 95% confidence interval. It follows then that the tails are also not significantly different.

Regarding the results for heuristic H2-exchange we reject the null hypothesis that the means are equal using a 95% significance interval for the average and best improvement regarding this heuristic. The left tail alternative hypothesis' that the means of $P_{HA}^{H_2}$ and $P_{HB}^{H_2}$ is lower than $P_{NA}^{H_2}$ and $P_{NB}^{H_2}$ are accepted, while the right tail alternative hypothesis is rejected for both best and average improvement.

The results of H3-2opt show that the null hypothesis that the population mean of $P_{HA}^{H_3}$ is equal to the population mean of $P_{NA}^{H_3}$ is rejected and accepted for $P_{HB}^{H_3}$ and $P_{NB}^{H_3}$. The left tail alternative hypothesis that the means of $P_{HA}^{H_3}$ is with 95% confidence lower than $P_{NA}^{H_3}$, is accepted.

For H5-cluster the null hypothesis is rejected for $P_{HB}^{H_5}$ and $P_{NB}^{H_5}$ and accepted for $P_{HA}^{H_5}$ and $P_{NA}^{H_5}$. The alternative hypothesis of the right tail of the best improvement, that the mean is significantly higher in $P_{HB}^{H_5}$ than $P_{NB}^{H_5}$ is accepted.

The results summarized above tell us that using H1-swap will have no effect on the outcome of the result. Also H2-exchange and H3-2opt are significantly decreasing the average for both heuristics and also for best for H2-exchange. This indicates that it is not beneficial to include these heuristics in a model. Finally H5-cluster is not affecting the average improvement however it is positively effecting the best improvement, indicating that it could be beneficial to include this heuristic.

5.3.5 Deciding on the Final Model Composition

The results from Section 4.8 tell us that our final model should include our Wild Escape Algorithm. This will influence our running time a little but give us much more reliable results.

As for selecting heuristics to include, the results from section Section 5.3.2 and Section 5.3.3, indicates that we need to include H4-random, H6-greedy and H7-similar in our final model. We observed from our testing of the Wild Escape Algorithm that the running time of H1-swap is significantly lower than our other heuristics Table A.1. Even though Section 5.3.4 showed us that H1-swap had no significant effect on the model, the low running time is leading us to rather include this heuristic than not, to lower the running time of our algorithm overall. As Section 5.3.4 showed it will have no significant negative effect on the results of our model. H5-cluster is also not effecting the average improvement of our model. It is though significantly effecting the result of the best improvement found positively. This tells us that we should include this heuristic in our final composition.

The results from Section 5.3.3 indicated that a combination of H4-random, H6-greedy and H7-similar + some of combination of the undecided group heuristics is significantly effecting both the average and best improvement. Therefore our final 4PLO model composition will be the 4PLO algorithm with the Wild Escape Algorithm and heuristics H1-swap, H4-random, H5-cluster, H6-greedy and H7-similar.

Table 5.8: The table shows the 4PLO model performance on instance sets 1-5. The columns contain in order: Instance set, number of orders, number of vehicles and number of locations in the instances. The next three columns contain the result from running our mathematical model (MM) run in AMPL; solution objective, optimality gap and the running time. Then follow 3 columns with the results of our model, average objective found, best objective found and average running time. The final column contains a delta which is calculated as follows: (solution objective MM - 4PLO best objective)/solution objective MM.

				Mathematical model			4PLO			
Inst Set	#Ord	#Veh	#Loc	Solution objective	Optimality gap	Run time(sec)	Average objective	Best objective	Average run time	delta
Set 1	4	3	7	3 444.7	0.00%	0.1	3 444.7	3 444.7	0.1	0.00%
	12	7	9	149 692.3	0.00%	9963.7	149 692.4	149 692.3	0.5	0.00%
	35	20	22	2 091 776.5	99.91%	10000.2	10 323.2	9 997.9	2.8	99.52%
	80	45	45	6 422 128.6	99.99%	10000.0	21 170.5	20 911.5	21.1	99.67%
	150	80	85	NA	NA	NA	34 479.1	32 798.0	100.8	NA
Set 2	4	3	7	2 501.0	0.00%	0.1	2 501.0	2 501.0	0.1	0.00%
	12	7	9	5 987.8	0.00%	1041.8	5 987.8	5 987.8	0.6	0.00%
	35	20	22	1 985 165.5	99.90%	10000.3	14 382.4	14 272.1	2.6	99.28%
	80	45	45	6 809 899.5	99.99%	10000.0	25 736.6	24 760.8	16.8	99.64%
	150	80	85	NA	NA	NA	36 927.2	35 932.1	112.1	NA
Set 3	4	3	7	1 404.0	0.00%	0.3	1 404.0	1 404.0	0.1	0.00%
	12	7	9	5 862.5	33.16%	10000.0	5 862.5	5 862.5	0.9	0.00%
	35	20	22	679 594.8	99.71%	10000.4	6 334.7	6 267.7	3.7	99.08%
	80	45	45	5 748 613.6	99.99%	10000.5	12 609.0	12 347.6	32.2	99.79%
	150	80	85	NA	NA	NA	19 771.9	19 149.8	126.1	NA
Set 4	4	3	7	1 696.1	0.00%	0.7	1 696.1	1 696.1	0.1	0.00%
	12	7	9	3 285.2	21.45%	10000.0	3 109.6	3 109.6	1.5	5.35%
	35	20	22	547 881.6	99.72%	10000.2	4 652.8	4 494.5	5.1	99.18%
	80	45	45	6 201 301.5	99.99%	10000.1	15 540.0	15 290.6	21.4	99.75%
	150	80	85	NA	NA	NA	22 508.6	22 252.6	103.4	NA
Set 5	4	3	7	5 154.9	0.00%	0.4	5 154.9	5 154.9	0.1	0.00%
	12	7	9	3 716.2	22.93%	10000.2	3 716.2	3 716.2	0.6	0.00%
	35	20	22	1 757 079.6	99.90%	10000.2	13 138.9	12 944.2	2.1	99.26%
	80	45	45	5 909 616.9	99.99%	10000.0	24 034.0	23 855.5	9.1	99.60%
	150	80	85	NA	NA	NA	41 343.4	39 847.0	82.6	NA

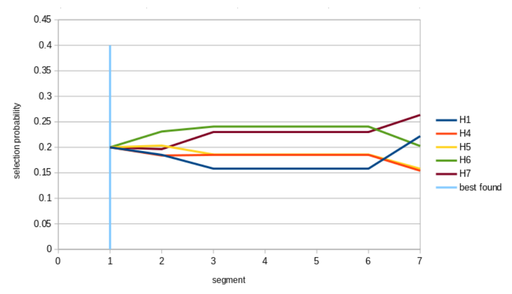
5.4 Final Results

After deciding on a model best suited to our problem we did a final run of our algorithm using composition described in the previous section. The algorithm was run using 5 instance sets of each 5 representative sizes.

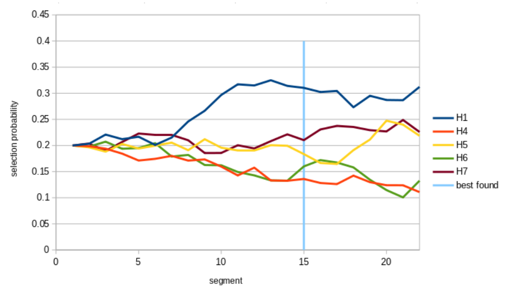
5.4.1 Evaluation of the Final Model

For each instance we used the powerful machine to let AMPL try for 10 000 seconds to find an optimal solution for each instance. The larger runs resulted in a “out of memory” failure message so these results we simply marked as NA. The results from the runs of our final model composition together with the mathematical model runs in AMPL are summarized in Table 5.8.

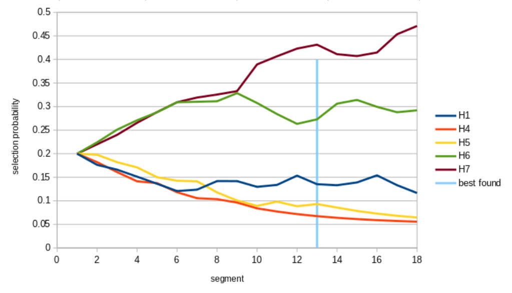
Section 5.4.1 shows the weight of each heuristic through each segment of our 4PLO model segments for arbitrarily selected runs. The vertical blue lines indicate which segment the best solution for the current run was found. The most relevant sections are the ones before this blue lines.



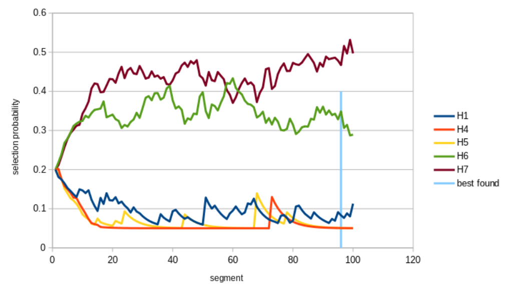
(a) Instance size 4 orders, run 1



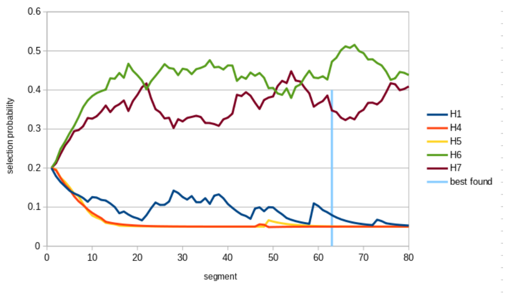
(b) Instance size 12 orders, run 3



(c) Instance size 35 orders, run 5



(d) Instance size 80 orders, run 7



(e) Instance size 150 orders, run 9

Figure 5.3: The figure illustrates heuristic weights per segment from instance set 1. The y-axis represents the weight of each heuristic, the x-axis represents a segment run. The blue vertical lines represents the segment where the best solution was found.

5.4.2 Observations of Results from the Final Composition

The first noticeable thing from the results of Table 5.8 is that the performance of the mathematical model is quickly deteriorating with the size of the instances. Already at the second instance size the model is struggling to finish on time, and for set 3 and set 5 there is still an optimality gap. For the third instance size (35 orders) the model can barely start iterating through solutions and the solution objective after 10 000 seconds has a very large optimality gap. We observe that the optimality gap is very large for the third and fourth instance size (35 orders and 80 orders) which means that the AMPL would need substantially more that 10 000 seconds to search for any type of real solution. This tells us that our problem is quickly getting very complex and that building a good model that can quickly find close to optimal solutions would be very important for this problem.

Moving on to the results of our 4PLO model, there are several observations that are important here. We see from the smallest instance size (4 orders) that our model is performing equally with the Mathematical model in terms of optimal solution but is already beating it in terms of running time. This is not unexpected as the smallest problem does not require a lot of iterations but it is unexpected that our model beats the mathematical model quite consistently in terms of running time.

As we increase the instance size to 12 orders, we see that our model consistently finds the same solution as AMPL, apart for in one case (instance 4) where AMPL was not able to find the optimal solution in 10 000 seconds. This combined with the fact that our model is doing this in around 1 second for this instances of this size tells us that our model is performing very well. It is also showing that our model is a robust model as the average and best solutions found are all the same for instances of size orders ≤ 12.

For the larger instance sizes (orders ≥ 35) AMPL was not able to find any solutions remotely close to the optimal value. The delta from Table 5.8 is higher than 99% for all these instances, which indicates that our model is significantly faster and better at finding good solutions. Another observation is how tight the best solution is overall with the average solution, also for the larger instance sets. This further indicates that our model is very robust and consistently finding similar solutions. A best solution much lower than the average could indicate that our model cannot move away from a local optimum in some cases. However our average solutions are within a 5% range of the best solutions. This means we are consistently hitting solutions close to the best known.

The Section 5.4.1 shows the performance of our heuristics. Figure 5.3b shows that for smaller instances the heuristics H1-swap, H5-cluster as well as H7-similar are getting higher weights before we find the optimal solution. This indicates that they are important for the smaller instances. The remaining figures Fig. 5.3c, Fig. 5.3d and Fig. 5.3e indicate that H6-greedy and H7-similar are the most important heuristics that our algorithm are giving more weight than the others and that they are consistently outperforming the other heuristics in regards to finding new and better solutions. We do however see several spikes from the heuristic H5-cluster and H1-swap up until the best solution is found (blue line). Which indicates that they are contributing to move to new and/or better solutions from time to time. The H1-swap, H5-cluster, and H4-random are doing this less consistently than H6-greedy an H7-similar for larger instance sets, but that does not mean that they are not important for the final result.

These observations also support our observations from Section 5.3.2 and Section 5.3.3, where H6-greedy and H7-similar are outperforming the other heuristics. Another observation from Table 5.8 is that our adaptive weights are clearly important for our search as size and problem constraints influence how the heuristics are performing and thereby changing how our model gives them weight.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

This thesis has illustrated the importance of a robust solution model for the 4th party logistics planning company. A thorough literature review illustrated how important further research around this problem and its solution method is. The development of a good algorithm to efficiently solve the transportation problem the 4PL companies are facing could give them a competitive advantage.

We have formulated the 4th Party Logistics Problem (4PLP) mathematically and we used commercial solvers to solve the model. As expected the solvers were not able to handle the realistic instances of the problem and therefore, it was necessary to look at alternative solution methods such as meta-heuristics. We have proposed a solution method we call the 4th party logistics optimizer which uses a framework inspired by Adaptive Large Neighbourhood Search. Within our solution method we designed seven heuristics which changes and improves a given current solution. These heuristics include a Swap heuristic, a 3-exchange heuristic, a 2-Opt heuristic, a Random Fit heuristic, a Clustering heuristic, a Greedy heuristic and a Similar Regret heuristic. Achieving a good balance between intensification and diversification in our search were important factors when deciding on the final composition of our solution method. We also proposed an algorithm that lets the model make large jumps to new parts of the solution space referred to as the Wild Escape Algorithm.

To determine the final composition of our solution method we decided to run several experiments. To test the algorithm we created an instance generator based on real anonymized data from a 4PL company. We started by testing how to jump from one part of the solution space to another when our algorithm gets stuck. Here we found that our proposed Wild Escape Algorithm vastly outperforms, a Random Reset adaptation, and Not Escaping, in spite of a small increase in running time.

Later we focused on deciding which heuristics our model should include by performing a series of test runs on a representative instance size including all heuristics. We then performed a ranking on the results of these results and statistical analysis of the data, including ANOVA(III), T-tests and multiple linear regression analysis. The result of these tests showed that the heuristics Random Fit, Greedy and Similar Regret were significant, while the other heuristics were undecided for the chosen instance size.

This led us to do further testing of the data by focusing on the combinations of the significant heuristics with insignificant ones. We found that combinations of decided and undecided heuristics exist that have significant positive influence on the result. This led us to do testing on individual heuristic basis.

The evaluation of the individual heuristics showed that the 3-exchange and the 2-opt heuristics had a significant negative effect on the results. The Clustering heuristic had a significant positive influence on the result in combination with other significant heuristics. The Swap heuristic had no effect on the result.

On the basis of the complete evaluation of our heuristics we chose a final composition of our model which included the Wild Escape Algorithm and the heuristics Swap, Random Fit, Clustering, Greedy and Similar Regret.

To compare the results of our final 4PLO composition, we used the results from running the mathematical model in AMPL as a benchmark. Our 4PLO model is outperforming the mathematical model implementation already on medium sized instances both in terms of the best solution found and in running time. The average solution was close to the best solution found over the course of 10 runs for all instances and this shows the robustness of our model. The final results also confirm the results from the evaluation of the individual heuristics that the Greedy and Similar Regret are the best performing heuristics. It also showed that the Clustering as well as the Swap heuristic is important for smaller sized instances. Overall the results illustrate how the 4th Party Logistics Optimizer is a robust and efficient algorithm that can be used to find near optimal solutions for a 4th party logistics provider company.

6.2 Future Work

To continue research surrounding the 4th Party Logistic Problem and our proposed model we suggest the following.

One could extend the computational experiments based solely on real data. The instance generator proposed in this thesis is based on real data, however it could be advantageous to test the model on actual customer data of different sizes.

We proposed completely new heuristics such as the Clustering heuristic in Section 4.4.5, and implemented our version of known heuristics such as Similar Regret in Section 4.4.7. We would encourage the further research and development of heuristics to help the 4PLO to perform even better, if possible.

The 4PLO and its components where tailored for the 4th Party Logistics industry, however it could be advantageous to test our implementation in other industries.

Appendix A

Appendix

Table A.1: The table shows the heuristic results from running the Wild Escape algorithm from Section 4.8 on instance Set 1. The fist column show the name of each heuristic. The next columns contain for each instance size $[4, 12, 35, 80, 150]$ the average running time of the heuristic and the times the heuristic was selected.

Instance size	4 Order		12 Order		35 Order		80 Order		150 Order	
Heuristic	Average run time	Times selected	Average run time	Times selected	Average run time	Times selected	Average run time	Times selected	Average run time	Times selected
H1-swap	2,49E-05	16292	9,97E-06	17657	1,01E-05	9717	2,56E-05	7400	8,17E-05	7554
H2-exchange	8,47E-06	14242	5,03E-06	22551	4,13E-06	11420	6,01E-06	9271	9,08E-06	10690
H3-2opt	5,87E-06	11561	5,01E-06	8190	1,01E-05	7687	2,81E-05	10328	5,43E-05	10477
H4-random	1,39E-05	14419	6,90E-05	10045	4,43E-05	5872	1,48E-04	5694	7,57E-04	5469
H5-cluster	2,31E-05	13538	8,08E-05	14887	3,62E-04	6697	0,001820441	5918	0,009244	5771
H6-greedy	1,88E-05	13620	8,51E-05	11590	3,27E-04	24352	0,001849877	25856	0,010488626	30845
H7-similar	2,16E-05	16328	6,43E-05	15080	3,87E-04	34255	0,002473415	35533	0,013395134	29194

Bibliography

4flow AG. Industries & references - automotive manufacturers, 2019. URL `www.4flow.com`.

F. Azadian, A. Murat, and R. B. Chinnam. An unpaired pickup and delivery problem with time dependent assignment costs: Application in air cargo transportation. *European Journal of Operational Research*, 263(1):188–202, 2017.

T. Barthélemy, A. Rossi, M. Sevaux, K. Sörensen, et al. Metaheuristic approach for the clustered vrp. In *EU/MEeting: 10th Anniversary of the Metaheuristics Community-Université de Bretagne Sud, France*, 2010.

J. C. Beck, P. Prosser, and E. Selensky. Vehicle routing and job shop scheduling: What's the difference? In *ICAPS*, pages 267–276, 2003.

R. Bent and P. Van Hentenryck. A two-stage hybrid algorithm for pickup and delivery vehicle routing problems with time windows. *Computers & Operations Research*, 33(4):875–893, 2006.

G. Berbeglia, J.-F. Cordeau, I. Gribkovskaia, and G. Laporte. Static pickup and delivery problems: a classification scheme and survey. *Top*, 15(1):1–31, 2007.

G. Berbeglia, J.-F. Cordeau, and G. Laporte. Dynamic pickup and delivery problems. *European journal of operational research*, 202(1):8–15, 2010.

C. Blum and A. Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM computing surveys (CSUR)*, 35(3):268–308, 2003.

B. Bullnheimer, R. F. Hartl, and C. Strauss. Applying the ant system to the vehicle routing problem. In *Meta-heuristics*, pages 285–296. Springer, 1999.

F. Carrabs, J.-F. Cordeau, and G. Laporte. Variable neighborhood search for the pickup and delivery traveling salesman problem with lifo loading. *INFORMS Journal on Computing*, 19(4):618–632, 2007.

M. Casazza, A. Ceselli, and R. W. Calvo. A branch and price approach for the split pickup and split delivery vrp. *Electronic Notes in Discrete Mathematics*, 69: 189–196, 2018.

M. Christiansen and K. Fagerholt. Robust ship scheduling with multiple time windows. *Naval Research Logistics (NRL)*, 49(6):611–625, 2002.

G. Clarke and J. W. Wright. Scheduling of vehicles from a central depot to a number of delivery points. *Operations research*, 12(4):568–581, 1964.

L. Costa, C. Contardo, and G. Desaulniers. Exact branch-price-and-cut algorithms for vehicle routing. *Transportation Science*, 53(4):946–985, 2019.

Y. Crama and M. Schyns. Simulated annealing for complex portfolio selection problems. *European Journal of operational research*, 150(3):546–571, 2003.

T. Curtois, D. Landa-Silva, Y. Qu, and W. Laesanklang. Large neighbourhood search with adaptive guided ejection search for the pickup and delivery problem with time windows. *EURO Journal on Transportation and Logistics*, 7(2):151–192, 2018.

G. B. Dantzig and J. H. Ramser. The truck dispatching problem. *Management science*, 6(1):80–91, 1959.

J. Desrosiers, Y. Dumas, and F. 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.

J. Desrosiers, Y. Dumas, M. M. Solomon, and F. Soumis. Time constrained routing and scheduling. *Handbooks in operations research and management science*, 8: 35–139, 1995.

H. Drias. Randomness in heuristics: An experimental investigation for the maximum satisfiability problem. In *International Work-Conference on Artificial Neural Networks*, pages 700–708. Springer, 1999.

Y. Dumas, J. Desrosiers, and F. Soumis. The pickup and delivery problem with time windows. *European journal of operational research*, 54(1):7–22, 1991.

M. Englert, H. Röglin, and B. Vöcking. Worst case and probabilistic analysis of the 2-opt algorithm for the tsp. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1295–1304. Society for Industrial and Applied Mathematics, 2007.

M. Englert, H. Röglin, and B. Vöcking. Worst case and probabilistic analysis of the 2-opt algorithm for the tsp. *Algorithmica*, 68(1):190–264, 2014.

D. Favaretto, E. Moretti, and P. Pellegrini. Ant colony system for a vrp with multiple time windows and multiple visits. *Journal of Interdisciplinary Mathematics*, 10 (2):263–284, 2007.

H. S. Ferreira, E. T. Bogue, T. F. Noronha, S. Belhaiza, and C. Prins. Variable neighborhood search for vehicle routing problem with multiple time windows. *Electronic Notes in Discrete Mathematics*, 66:207–214, 2018.

M. Fisher. Vehicle routing. *Handbooks in operations research and management science*, 8:1–33, 1995.

L. R. Ford Jr and D. R. Fulkerson. Constructing maximal dynamic flows from static flows. *Operations research*, 6(3):419–433, 1958.

M. G. S. Furtado, P. Munari, and R. Morabito. Pickup and delivery problem with time windows: a new compact two-index formulation. *Operations Research Letters*, 45(4):334–341, 2017.

V. Ghilas, E. Demir, and T. Van Woensel. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows and scheduled lines. *Computers & Operations Research*, 72:12–30, 2016.

F. Glover. Future paths for integer programming and links to artificial intelligence. *Computers & operations research*, 13(5):533–549, 1986.

A. Hemmati, L. M. Hvattum, K. Fagerholt, and I. Norstad. Benchmark suite for industrial and tramp ship routing and scheduling problems. *INFOR: Information Systems and Operational Research*, 52(1):28–38, 2014.

HM Group. Logistics glossary - 4pl, 2019. URL `https://www.logisticsglossary.com/term/4pl/`.

L. Kaufman and P. Rousseeuw. Finding groups in data; an introduction to cluster analysis. Technical report, J. Wiley, 1990.

S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *science*, 220(4598):671–680, 1983.

J. E. Korsvik, K. Fagerholt, and G. Laporte. A large neighbourhood search heuristic for ship routing and scheduling with split loads. *Computers & Operations Research*, 38(2):474–483, 2011.

G. Laporte and I. H. Osman. Routing problems: A bibliography. *Annals of operations research*, 61(1):227–262, 1995.

H. C. Lau and Z. Liang. Pickup and delivery with time windows: Algorithms and test case generation. *International Journal on Artificial Intelligence Tools*, 11(03): 455–472, 2002.

E. L. Lawler and D. E. Wood. Branch-and-bound methods: A survey. *Operations research*, 14(4):699–719, 1966.

J. K. Lenstra and A. R. Kan. Complexity of vehicle routing and scheduling problems. *Networks*, 11(2):221–227, 1981.

H. Li and A. Lim. Large scale time-constrained vehicle routing problems: A general metaheuristic framework with extensive experimental results. *AI Review*, 2001.

H. Li and A. Lim. A metaheuristic for the pickup and delivery problem with time windows. *International Journal on Artificial Intelligence Tools*, 12(02):173–186, 2003.

S. Lin. Computer solutions of the traveling salesman problem. *Bell System Technical Journal*, 44(10):2245–2269, 1965.

H. Manier, M.-A. Manier, and Z. Al Chami. Shippers' collaboration in city logistics. *IFAC-PapersOnLine*, 49(12):1880–1885, 2016.

M. A. Masmoudi, K. Braekers, M. Masmoudi, and A. Dammak. A hybrid genetic algorithm for the heterogeneous dial-a-ride problem. *Computers & operations research*, 81:1–13, 2017.

R. Masson, S. Ropke, F. Lehuédé, and O. Péton. A branch-and-cut-and-price approach for the pickup and delivery problem with shuttle routes. *European Journal of Operational Research*, 236(3):849–862, 2014.

S. Mitrovic-Minic. Pickup and delivery problem with time windows: A survey. *SFU CMPT TR*, 12(1-43):38, 1998.

D. R. Morrison, S. H. Jacobson, J. J. Sauppe, and E. C. Sewell. Branch-and-bound algorithms: A survey of recent advances in searching, branching, and pruning. *Discrete Optimization*, 19:79–102, 2016.

W. P. Nanry and J. W. Barnes. Solving the pickup and delivery problem with time windows using reactive tabu search. *Transportation Research Part B: Methodological*, 34(2):107–121, 2000.

H. M. Pandey, A. Chaudhary, and D. Mehrotra. A comparative review of approaches to prevent premature convergence in ga. *Applied Soft Computing*, 24:1047–1077, 2014.

S. N. Parragh, K. F. Doerner, and R. F. Hartl. A survey on pickup and delivery models part ii: Transportation between pickup and delivery locations. *Journal für Betriebswirtschaft*, 58(2):81–117, 2008.

H. 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.

Y. Qiu, L. Wang, X. Xu, X. Fang, and P. M. Pardalos. Formulations and branch-and-cut algorithms for multi-product multi-vehicle production routing problems with startup cost. *Expert Systems with Applications*, 98:1–10, 2018.

C. K. Reddy and B. Vinzamuri. A survey of partitional and hierarchical clustering algorithms. In *Data Clustering*, pages 87–110. Chapman and Hall/CRC, 2018.

M. Rocha and J. Neves. Preventing premature convergence to local optima in genetic algorithms via random offspring generation. In *International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems*, pages 127–136. Springer, 1999.

S. Ropke and D. Pisinger. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation science*, 40(4): 455–472, 2006.

M. Savelsbergh and M. Sol. Drive: Dynamic routing of independent vehicles. *Operations Research*, 46(4):474–490, 1998.

M. W. Savelsbergh and M. Sol. The general pickup and delivery problem. *Transportation science*, 29(1):17–29, 1995.

P. Shaw. A new local search algorithm providing high quality solutions to vehicle routing problems. *APES Group, Dept of Computer Science, University of Strathclyde, Glasgow, Scotland, UK*, 1997.

P. Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In *International conference on principles and practice of constraint programming*, pages 417–431. Springer, 1998.

M. M. Solomon. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations research*, 35(2):254–265, 1987.

K. Sörensen and F. W. Glover. Metaheuristics. *Encyclopedia of operations research and management science*, pages 960–970, 2013.

W. Sun, Y. Yu, and J. Wang. Heterogeneous vehicle pickup and delivery problems: Formulation and exact solution. *Transportation Research Part E: Logistics and Transportation Review*, 125:181–202, 2019.

J. F. Sze, S. Salhi, and N. Wassan. A hybridisation of adaptive variable neighbourhood search and large neighbourhood search: Application to the vehicle routing problem. *Expert Systems with Applications*, 65:383–397, 2016.

The Port Arthur News. Meryll frost - 'most courageous athlete of 1945'. 1946.

P. Toth and D. Vigo. *Vehicle routing: problems, methods, and applications.* SIAM, 2014.

H. Xu, Z.-L. Chen, S. Rajagopal, and S. Arunapuram. Solving a practical pickup and delivery problem. *Transportation science*, 37(3):347–364, 2003.