A Pickup and Delivery problem with multiple time windows in supply chain industry

Department of Informatics

University of Bergen

Preben Bucher Johannessen

November 11, 2019

abstract

This is a test Shaw (1997). Hopefully it works..

Contents

List of Figures

Chapter 1

Introduction

1.1 4flow, A 4PL planning and strategy company

The automobile industry is one of the most significant industries in Europe today which is facing high cost-reduction pressure from fierce international competition and difficult conditions.4flow AG (2018) The supply chain process in automobile industry requires alot of flexibility in their planning and modelling of transportation which often takes the form some sort of vehicle transportation problem. In planning the most efficient way of organizing your transportation network the classical VRPTW from is likely the first problem that comes to mind.

1.2 Litterature review

Parragh et al. (2008) The Vehicle Routing Problem with Time Window Constraints (VRPTW) is defined as the problem of minimizing costs when a fleet of homogeneus vehicles has to destribute goods from a depot to a set of customers satisfying time windows and capacity constraints Cordeau and Groupe d'études et de recherche en analyse des décisions (Montréal(2000). Another classical problem is the PDPTW, Pickup and Delivery Problem with Time Windows, with multiple vehicles is another problem that has been widely researched and solved for example in Nanry and Barnes (2000).

Cordeau and Groupe d'études et de recherche en analyse des décisions (Montréal(2000) Presents a VRPTW Problem as an extension of the Capacitated Vehicle routing problem, with hard and soft timewindows, where soft can be broken agains a cost and hard time-windows cannot. They present a multi-commodity network flow formulation of the problem and use approximation methods to derive upper bounds and use Lagrangean relaxation and column generation to derive lower bounds. To produce integer solutions they use cutting and branching strategies and finally they also present special cases and extensions. Solomon (1987) formalized an algorithm for general PDP Lu and Dessouky (2006) insertion based heuristic Kalantari et al. (1985) An algorithm for the travelling salesman problem with pickup and delivery of customers Dumas et al. (1991) pickup and delivery problem with time windows Fagerholt et al. (2010) speed optimization and reduction of fuel emissions Berbeglia et al. (2010) dynamic and static pickup and delivery problems survey and classification scheme Berbeglia et al. (2007) static Zhou (2013) Inventory management

and inbound logistics optimization Savelsbergh and Sol (1995) General pickup and delivery problem. Dror et al. (1989) Properties and solution framework

Christiansen and Fagerholt (2002) present a ship scheduling PDP problem of bulk cargoes with multiple time windows. They also handle specific shipping industry challenges regarding ship idle time, transport risks due to weather and unpredictable service time at ports. They use a set partitioning approach to solve the problem and found that you could increase the robustness of the schedules on account of costs. Christiansen et al. (2004) Ship routing and scheduling Hemmati et al. (2014) Consider a class of cargo ship routing and scheduling problems from tramp and industrial shipping industry. They provide solutions to a wide range of benchmark instances both optimal for smaller instances, using a commercial mixed-integer programming solver. And they present adaptice large neighbourhood search heuristics to larger instances. They also provide the benchmark instances and an instance generator. Hemmati et al. (2016) A iterative two-phase hybrid metaheuristic for multi-product short sea inventory routing problem Ferreira et al. (2018) variable neighbourhood search for vehicle routing problem with multiple time windows Desrosiers et al. (1995) Time constrained routing and scheduling

In this paper we will propose a mathematical formulation to a multivehicle PDPTW problem, with certain adjustments to satisfy the needs of an automobile manufacturing company. To these type of companies the inbound production side of the supply chain is more in focus and require ajdustments on the classical PDPTW. In these type of industries companies often facing challenges related to the production side of the problem, which is often referred to as looking at the problem from an inbound perspective. Inbound oriented perspective means that you have less focus on the the supplier side of the of the production, and on the means of transportation as long as what is being transported arrives on time for production. This changes our problem away from the classical problems mentioned above. Instead of having a customer oriented view, we must instead shift our view to focus on getting a set of orders from a set of suppliers to a set of factories where the orders are needed for production.

Being more inbound oriented also leads to having more focus on the problems that sometimes occur within the factory and the delivery part of the problem. Some manufacturers might have alot of traffic on their factories, and want to limit that, saying that a vehicle can only visit a certain amount of docks for each visit. This might lead to more ineffective solutions and more use of vehicles in some cases however, for the inbound oriented producer it is a nessecary limitation. To our knowledge dock constraints has not been researched before in PDPTW problems.

Another result from the inbound thinker is that instead of using your own fleet of vehicles, you hire a logistics carrier to do your transports. The carries might have different cost structures and payment methods which you have no influence over. Leading you to need a model that takes that into account. We focus here on dealing with a carrier that offers a varying cost per kilometer. Meaning that the cost parameter of the standard PDP will change depending on how far a vehicle is travelling. Using a carrier also changes the problem to that you dont care so much where the vehicle is travelling from to its first pickup. The vehicle rather starts at its first pickup point because the cost of the car manufacturer starts only from the time of the first pickup. Before pickup and after delivery the carrier has the costs. To our knowledge very little research has been made into using a logistics carrier as

vehicle fleet.

In car manufacturing business you might have production periods that go over several days with strict opening times on delivery. This leads to the problem of multiple time windows. The manufacturer might only want the parts delivered in the morning on one of three days but have a break in the middle of each time window (lunch). There might also be advantageous to move the delivery of one order to the next day to bundle orders together and we want our model to be able to handle such problems.

In Favaretto et al. (2007) they propose a time window model to handle multiple time windows and multiple visits. We are using a modified version of this model to handle the multiple time windows in this paper to handle the time window constraints. However as they focus on satisfying a set of customers instead of production companies our model differs greatly when it comes to the rest of the constraints.

To make a model that is as realistic as possible it is important to integrate each of the above aspects and take them all into account. One aspect of the model could greatly influence the decision of another so it is important to build a model that satisfies all the aspects above. Research often forcuses on solving single issues, like the multiple time windows, to handle this specific problem for a certain industry. However handeling all of the above mentioned aspects have to our knowledge never been researched before and makes it therefore an important problem to solve.

The goal of this paper is to present a realistic mathematical model that can be further developed into a realistic model solving pickup and delivery problems on a daily/weekly basis for manufacturing companies. The goal is also to have a mathematical basis to be used in a solver such as AMPL to solve a small instance, and finally to make a heuristic model that can solve the problem efficiently and be used by companies in similar sort of industries.

test after file

Chapter 2

Model Description

Like mentioned above to take the perspective of a vehicle manufacturing company, we see that we have a certain set of factories where the products are produced, each with a set of demanded parts/orders for production. The parts, or orders (typically named transport orders but we refer to orders here), can be delivered from a set of suppliers. At a given point in time (could be delivery for one day or a week) we consider a planning problem with a demand for orders that should be satisfied with the given suppliers using vehicles provided by different logistic carriers.

v	-	vehicle
i	-	node
f	-	factory
p	-	timewindow
s	-	stop location
α	-	distance interval in cost structure
β	-	weight interval in cost structure

Table 2.1: Indices

The vehicles have different capacities(kg and volume), cost structures, incompatabilities and start at the first pickup location at the first pickup time (ie. costs to get from start to first pickup are not relevant to the manufacturer). The cost structure depend on the total distance a vehicle is driving, the maximum weight transported by that vehicle, and a fix cost. There is also a cost for stopping at a node.

When a delivery is assigned to a vehicle, the vehicle must load the delivery from the supplier, and deliver the delivery at the factory dock. The docks (each reperesented by a different node) in each factory can differ according to which order is being delivered and there is a limit to how many docks each vehicle can unload at per visit to the factory.

Each delivery/pickup can have several time windows, lapsing sometimes over several days. If a car arrives before a time window it has to wait.

The mathematical formulation of the problem will now follow. The problem can be viewed as a graph $G(E, N)$ where $N = \{0, ..., 2n\}$ are the verticies and n is the number of orders in the problem, and $E = \{(i, j) : i, j \in N, i \neq j\}$ represent the edges in the graph.

	Sets	
N	-	nodes $\{1, .., 2n\}$ where n is number of orders
V	-	vehicles
E	-	edges
E_v	-	edges visitable by vehicle v
N_v	-	nodes visitable by vehicle v
N^P	-	pickup nodes
N^D	-	delivery Nodes
F	-	factories
N_f	-	delivery nodes for factory f
A_v	-	index of elements in the cost where α goes from $(1, .., \gamma_v)$ and β from $(1, .., \mu_v)$
P_i	-	set of time windows at node i, $\{1, .., \pi_i\}$
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 $s \in S$

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}. The set of Edges that each vehicle can traverse is represented by E_v. Since n is the number of orders in the problem, then if i is a specific pickup-node then $i+n$ corresponds to the delivery node for the same order. The set of pickup Nodes (supplier docks) we denote using N^P and each delivery node (Factory dock) is denoted by N^D. 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 wich we denote N_f. Since all factories are delivery nodes these sets only include delivery nodes. 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. The distance and costs from here to the first pickup is zero. The factory docking limit is denoted by H_f.

Parameters

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 $(\alpha,\beta) \in A_v$ for vehicle v
$C_{v\alpha\beta}^{kg}$	-	cost per weight unit (kg) in cost matrix element $(\alpha,\beta) \in A_v$ for vehicle v
$C_{v\alpha\beta}^{fix}$	-	fixed cost in index $(\alpha,\beta) \in A_v$ 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 A_v column index
Z_β	-	weight for interval β in cost matrix A_v row index

Each delivery node has a variable h_i indicating how many docks have been visited including the node i.

Each pickup node has a weight Q_i^{kg} and a volume Q_i^{vol} parameter indicating the weight and volume of the order 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 timewindows. Each node has a current time based on when its being served, denoted by t_i and where $i \in N$. 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}.

Each time a vehicle v makes a stop and a 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 $(1..\gamma_v$ and β is the weight interval ranging from $(1..\mu_v)$. Together these pairs make a matrix we refer to in this paper as a cost matrix. Each type of cost has a matrix, including distance, weight, and fix costs and the cost in a certain interval (α,β) is represented by $C_{v\alpha\beta}^{cost-type}$. The total distance travelled by vehicle v will be denoted by the variables $d_{v\alpha\beta}$ for each $(\alpha,\beta) \in A_v$, 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 $d_{v\alpha\beta}$ and $l_{v\alpha\beta}$ has a value will be determined by the binary variable $b_{v\alpha\beta}$ and the distance insterval parameter B_α and the weight interval parameter Z_β.

Variables

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	-	docking times in factory after visiting node $i \in N_f$
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 $(\alpha, \beta) \in A_v$
$b_{v\alpha\beta}$	-	binary indicating interval $(\alpha, \beta) \in A_v$ for vehicle $v \in V$
$l_{v\alpha\beta}$	-	the highest weight trasported by vehicle $v \in V$ for interval $(\alpha, \beta) \in A_v$

l_{iv}^{kg} is the weight and l_{iv}^{vol} is the volume on the vehicle v leaving node i. x_{ijv} is a binary vairable 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.

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

subject to:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

The objective function (**??**) 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 penilized with costs and the aim is to minimize the sum of all these costs. Constraint 2.2 secures that a load is picked up once and only by one vehicle or not picked up at all. **??** makes certain that if a load at a node j is travelled to it is also left. This is not the case for

origin and final destination node. Then **??** makes sure that origin is only left once by each vehicle, and **??** ensures that desination is only arrived to once. Finally **??** is there to say if a load is picked up it must also be delivered.

Regarding weight, constraint **??** ensures that if a load at node j is picked up, the weight before plus the weight of the load at node j minus the weight leaving the node is equal to zero. If not the summation cannot exceed the capacity of the vehicle. Also **??** makes certain that at delivery of node j the weight before delivery minus the weight of the load at j minus the weight after, is equal to zero, ie. the weight of the vehicle after is lighter than before by exactly the weight of the load at j. The final weight constraint **??** says that weight leaving node i always has to be between 0 and the capacity of the vehicle.

The next constraints **??**-**??**, ensures the same as for the weight but for volume, that each load is increased by the volume of the order, that the volume is decreased once delivered and that the volume at any node is between 0 and the volume capacity of that vehicle.

It follows from constrain 2.13 that within a certain factory, if you travel between two nodes, the amount of stops you have made should always be iterated by one. If you are not travelling between them, the summation cannot be bigger than one plus the limitation corresponding to that factory. The next constraint on factories 2.14, ensures that any node visited in a factory cannot exceed the docking limit. Then 2.15 makes sure that when a vehicle enters a factory from outside, the docking amount gets an initial value of 1, and if one is not travelling from i to j the value has to be greater than or equal to 0.

Constraint 2.16 ensures only one timewindow is used per node, and 2.17 says that the time a node i is visited has to exceed a lower timewindow limit. 2.18 ensures the same upper bound timewindow is used. Then constraint 2.19 ensures that the travel from one node to the next is appropriately increased by the travel time between them and for any other two nodes the values cannot exceed a large constant. Finally 2.20 ensures that the time a vehicle visits the delivery of an order must always be after the pickup of that order.

From 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. Then regarding weight 2.22 ensures that the sum of the max weight of a vehicle v is greater than or equal to the weight at all nodes visited by that vehicle. Constraint 2.23 ensures that for each vehicle v the total distance variable can only exist in the appropriate distance interval. The same is the case in 2.24 for maximum weight in the appropriate weight interval. Finally 2.25 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 dont calculate the fixed costs of said vehicle.

Constraints 2.26 to 2.31 are ensuring positive and binary variables where appropriate.

Chapter 3

Solution Methods

This chapter will go over the most common solution methods in metaheuristics and will be divided up in three parts. The first part, section 3.1 will cover metaheuristics and requirements of a good solution method. The next section 3.2 will present approaches that help intensify the search. The final part section 3.3 will focus on approaches that helps a model to diversify the search for solutions. The chapter will lead us to the solution method we have chosen which is presented in detail in chapter 4.

3.1 Solution method requirements

A metaheuristic 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 metaheuristic is therefore by definition a very open way of approaching a problem and many different approaches have been developed in the last few decades. The most noteable metaheuristics include (adaptive) large neighbourhood search, tabu search, simmulated anealing, genetic algorithm, variable neighbourhood search and ant colonization, among many more. The term metaheuristic was first used by Glover (1986) when he coined it, aswell as tabu search which he combined with artificial intelligence strategies to build a framework for integer programming.

For a metaheuristic 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 metaheuristic approach requires a good balance between two important principles *intensification* and *diversification*. A good balance between these principles will help ensure a near global optimal solution. In the following sections we will go more into details on these two principles.

3.2 Intensification

By intensification we refer to a metaheuristics 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

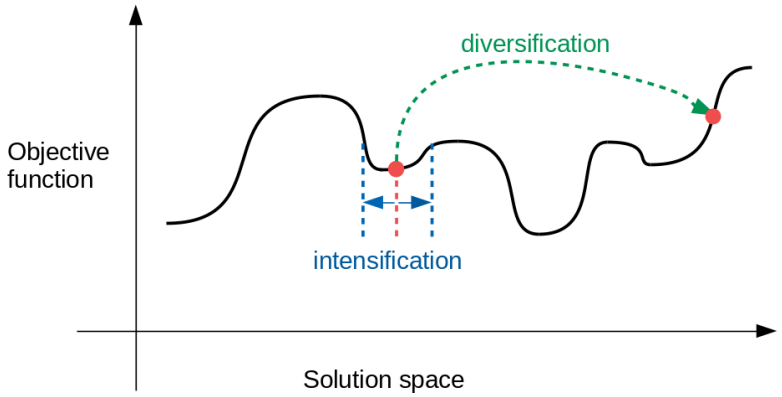


Figure 3.1: The figure illustrates how intensification, in blue, and diversification, in green, works while exploring a solutions space.

optimal solution. In fig. 3.1 we show how intensification might work on a given solution space.

Local search heuristics like the ones implemented by Nanry and Barnes (2000) and Li and Lim (2003) are build using intensification heuristics that perform smaller moves such as moving an order from one vehicle to another, or swapping two orders with eachother. We implemented two similar heuristics in section 4.4.1 and section 4.4.2 which are performing small efficient intensification moves.

Lin (1965) implemented a 2-opt heuristic for travelling salesman problem. The 2-opt heuristic is a very intensification operator as it focuses on one vehicles routes neighbourhood through several iterations and ends up returning the local optimum for that vehicle route. We have implemented a version of the 2-opt heuristic in section 4.4.3.

3.3 Diversification

Diversification refers to a metaheuristics 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. Figure 3.1 illustrates how diversification is working on a solution space.

Cordeau et al. (2001) diversified by relaxing constraints and thereby allowing the search to visit infeasible solutions. In section 4.8 we have implemented a diversifying escape algorithm that successfully allows our algotihm to escape one part of a solution space and move to new unexplored areas, as we show in section 5.3.1

Large neighbourhood search (LNS) was introduced by Shaw (1997) and this metaheuristic diversifies by removing and reinserting a certain amount of orders q in each iteration. Our implementation will be based on a similar type of framework as LNS.

Algorithm 1 shows a pseudocode 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

Algorithm 1 Large neighbourhood search

```
1: function LNS(solution s)
2:     solution s_best = s
3:     repeat
4:         s' = 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 = s'
10:        if accept(s', s) then
11:            s = s'
12:    until stop condition met
13:    return s_best
```

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 implement a different acceptance criteria which we will present below.

Ropke and Pisinger (2006) implemented a version of the LNS called Adaptive large neighbourhood search. He changed the LNS to adapt itsself by keeping track of the performance of each heuristic. In section 4.6 we have used similar adaptive weights to allow our model to automatically keep a good balance between diversification and intensification.

The tabu search 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 behaivior helps the algorithm to diversify and avoid searching in a cycle by performing the same type of move. We have chosen to remember which solutions we have visited and not rewarding this type of behaivior in our adaptive weights.

Kirkpatrick et al. (1983) implemented a simmulated anealing algorithm applied to the travelling salesmen problem. Simmulated anealing helps diversify a search by sometimes accepting worse solutions based on a temperature T. The temperature T is on a cooling down schedule and the longer the search goes on the lower the chances of acceptance becomes. In section 4.7 we have used a similar acceptance criteria in our model.

The proposed solution method, which we call *the fourth party logistics optimizer*, is trying to balance the intensification and diversification to solve the 4PL problem. The complete model is explained in detail in the next chapter.

Chapter 4

A 4th party logistics optimizer

This chapter explains the setup of our model, the fourth party logistics optimizer (4PLO). The first section section 4.1, desribes the metaheuristic 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.

The first of these section 4.2 explain how we have represented a solution in our model and what a vehicle schedule is. The next section 4.3 explain how we have chosen the initial solution. Then section 4.4 present each of the heuristics included in the basic version of our model. The next section 4.5, explains how our model is choosing a heuristic in a given iteration. After which section 4.6 show how we made our model adapt the choice of heuristic throughout the iterations using historical data. Then section 4.7 explain how we accept solutions based on simmulated annealing. Finally section 4.8 present the wild escape algorithm used by our model to diversify away from a local optimum.

4.1 The 4th party logistic optimizer

A brief pseudocode of our implementation of the 4th party logistic optimizer is described in algorithm 2.

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 and decides to accept the newly created solution or not until the stop condition is met. 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.

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

4.2 Solution representation

In order for our model to solve the 4PL problem, choosing a good *solution representation* is an important part of the algorithm. There are many ways to represent a solution like matrix, binary matrix and permutation. It is tempting to represent our solution in a matrix, where each row represent a vehichles schedule. However we will be using a permutation as solution representation in this paper, where each *vehicles schedule* S_v will be separated by a 0.

Algorithm 2 4th party logistic optimizer

```
1:  function 4PLO(heuristics H, wild heuristics W)
2:      solution s = generateInitialSolution()
3:      solution s_best = s
4:      iterations since best solution found i = 0
5:      repeat
6:          if i > escape condition then
7:              s = wildEscape(s, s_best, W)
8:          s' = s
9:          select a heuristic, h ∈ H based on selection parameters
10:         s' = applyHeuristic(h, s')
11:         if f(s') < f(s_best) then
12:             s_best = s'
13:         if accept(s', s) then
14:             s = s'
15:         update selection parameters and escape condition
16:     until stop condition met
17:     return s_best
```

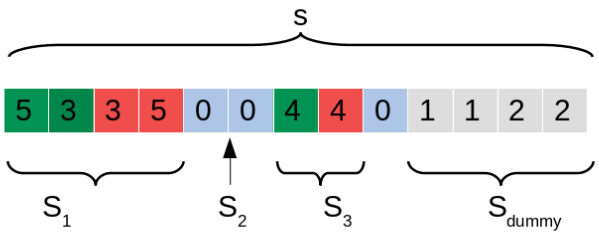


Figure 4.1: Example of a solution representation of a problem with 5 orders and 3 vehicles. A vehicle schedule, represented by S_v, contain information for each vehicle v, on which and in what order a vehicle schould pick up and deliver orders. The dark green tiles represent a pickup, the red tiles represent a delivery. The 0 tiles separate each vehicle schedule. The complete permutaion represent our solution representation s.

Figure 4.1 show an example of a solution s, with 5 orders and 3 vehicles. Each vehicle schedule indicate in which order a pickup and delivery should follow. The first time an order appears has to be the pickup (highlighted as dark green in the figure). In the same way the second time the same order number appears, it has to be the delivery. As an example, S_3 is first picking up the order 4 and then delivering the order right after, as delivering an order before a pickup is not possible. The same goes for S_1, which first picks up order 5, then picks up order 3 and then delivers each of them after. S_2 represent an empty vehicle schedule.

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

4.3 Initial Solution

We mentioned premature solutions in chapter 3, that can lead to a problem of diversifying solutions in a pickup and delivery model. To avoid this problem we have chosen to start with an initial solution s where no orders are assigned to any vehicle, ie. all orders are assigned to the dummy vehicle in the solution representation. Not only does this contribute to avoid the problem of premature solutions, 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

This section presents the heuristics used by our algorithm. The first three heuristics sections 4.4.1 to 4.4.3, we call *shuffeling heuristics*. They try to quickly schuffle a given solution around to do intensification search.

Removal and reinsertion heuristics are well reserached tools used for both intensification and diversification when pickup and delivery problems, as we showed in chapter 3. The last four heuristics, sections 4.4.4 to 4.4.7, we refer to here as *removal and reinsertion heuristics*. Each of them contain one heuristic for removal and one heuristic for reinsertion. These are used partly for diversification, and partly for intensification depending on the amount of solution elements, q, being reinserted.

4.4.1 Swap

This heuristic tries to swap the pickup and delivery of two different orders until the first time it finds a fit. Figure 4.2 illustrates a successful swap between two orders in a vehicle schedule.

This heuristic is very efficient in terms of running time and jumps randomly around the solution space. It is only working on two orders at a time so it is an intensification heuristic.

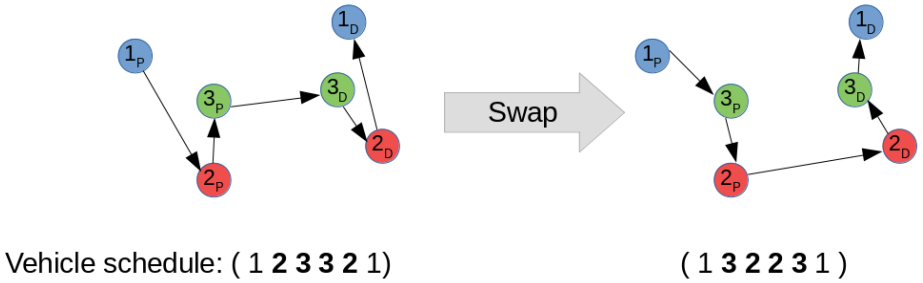


Figure 4.2: A swap heuristic performed on a vehicle schedule with three orders. The numbered nodes indicate an order location and the letters P and D indicate pickup and delivery respectively. Emphazised numbers in the vehicle schedule are selected by the heuristic.

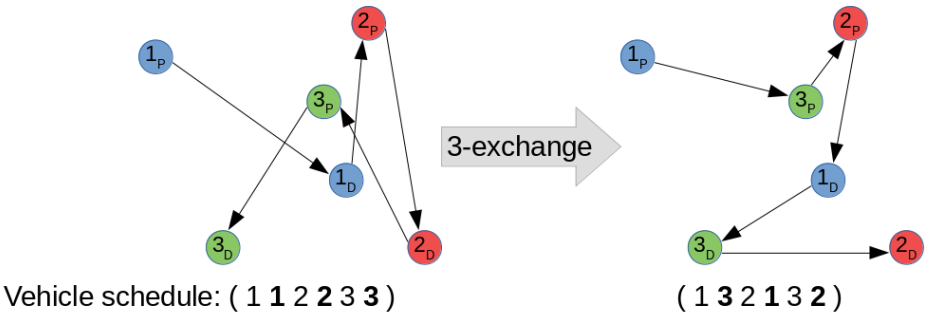


Figure 4.3: A 3-exchange heuristic performed on a vehicle schedule with three orders. The numbered nodes indicate an order location and the letters P and D indicate pickup and delivery respectively. Emphazised numbers in the vehicle schedule are selected by the heuristic.

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 assigned positions until a feasible new combination is found. A 3-exchange of position 2, 4 and 6 in a vehicle schedule is illustrated by fig. 4.3.

This heuristic is very fast, as the exchanges are fast operations and checking if a vehicles schedule is feasible is also a very effective operation. Like the swap heuristic from the previous section this heuristic jumps randomly around the solution space and is an intensification heuristic.

4.4.3 2-opt

The 2-opt heuristic is used in metaheuristics to solve vehicle routing problems. Lin (1965) implemented the 2-opt heuristic for travelling salesmen problem and Bullnheimer et al. (1999) combined 2-opt with the ant system metaheuristic. The 2-opt heuristic used in this paper is similar to the heuristic from Carrabs et al. (2007). Figure 4.4 illustrates one iteration of our 2-opt heuristic performed on a vehicle schedule.

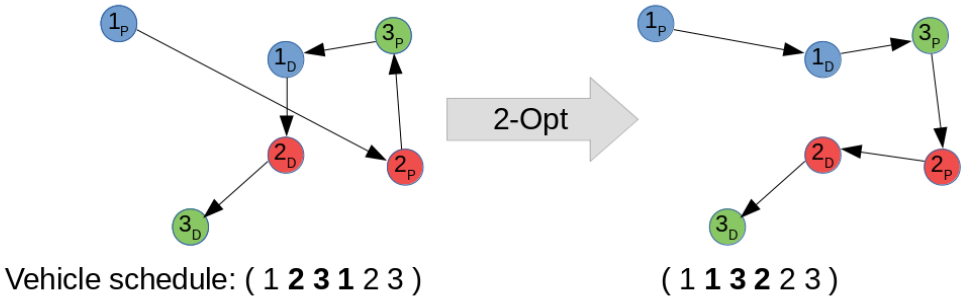


Figure 4.4: One 2-opt heuristic operation performed on a vehicle schedule with three orders. The numbers indicate an order and the letters P and D indicate pickup and delivery respectively. Emphazised numbers indicate the $S_{(i+1)j}$ selected vehicle schedule part to reverse.

Algorithm 3 2-opt heuristic

function 2-OPT
  select random vehicle v with $|v| > 2$
  vehicle schedule S_v
  repeat
    $S_{best} = S_v$
    $n = |S_v|$
    for $i < n$ **do**
      $j = i + 1$
      for $j < n$ **do**
        current vehicle schedule $S'_v = S_{0i} + reverse(S_{(i+1)j}) + S_{(j+1)n}$
        if $f(S'_v) < f(S_{best})$ **then**
          $S_{best} = S'_v$
    $S_v = S_{best}$
  until no further improvement found
  return S_v

The heuristic is described in algorithm 3 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_0i, 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, aswell 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 ie. the best possible schedule for the selected vehicle has been found.

4.4.4 Random fit

In most problems there are solutions that you cannot find using pure logic. Constantly minimizing costs and searching for the most efficient solution possible could lead to a problem of diversification, and not really exploring the whole solution space. We therefore decided to make one remove and reinsert heuristic that does not have any specific priorities but rather moves around the solution space randomly.

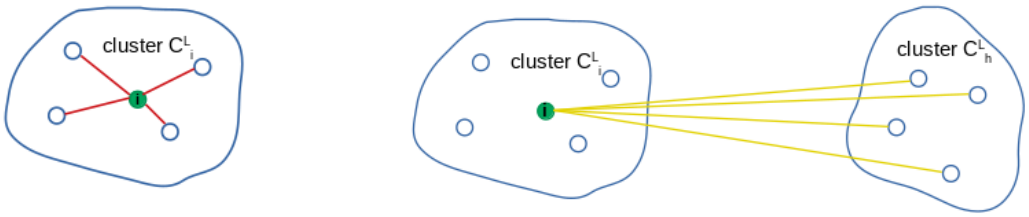
Algorithm 4 Remove random and reinsert first fit heuristic

```
1:  function randomFit(s ∈ {solution)
2:      select the number of orders to reinsert, q
3:      solution s' = s
4:      remove q orders from s'
5:      set I = removed orders
6:      for z ∈ 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 4 shows the pseudocode for our remove random insert first fit heuristic. It starts by selecting a random amount of orders q which it is to 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 amount of orders in the instance. The heuristic then 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.

4.4.5 Clustering

Section 1.1 illustrates how a typical 4PL customer typically have their factories and suppliers structured in clusters around a continent or a country. This led us



(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 C_i^L is its cluster. The cluster C_h^L is the cluster with the minimum average distance to i.

to try to build a heuristic which considers clusters as a factor when removing and reinserting an order in a vehicle schedule. Orders delivered from and too different clusters 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 vehicles.

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

The Siluette 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 the what they call a Siluette coefficient. The coefficient is calculated based on two aspects, cohesion a_i and separation b_i of a node (or in our case a location) i. Together these aspects make out the Siluette u_i.

Figure 4.5 illustrate the cohesion and sparation of a green node i part of cluster C_i^L. The cohesion a_i is the mean of the distances, d_{ij}, from i to all other nodes $j \in C_i^L$. The separation b_i is the minimum of the mean of the distances d_{ij} from node i to the nodes $j \notin C_h^L$ in all other clusters $h \in \Gamma$, where Γ is a set containing all cluters apart form C_i^L. The two effects can be written as follows:

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

$$b_i = \min_{h \neq i} \frac{1}{|C_h^L|} \sum_{i \neq j} d_{ij} \tag{4.2}$$

The Siluette coefficient u_i for location i is then calculated as follows:

$$\begin{aligned} u_i &= \frac{b_i - a_i}{\max a_i, b_i}, && if\ |C_i^L| > 1 \\ u_i &= 0, && if\ |C_i^L| = 1 \end{aligned} \tag{4.3}$$

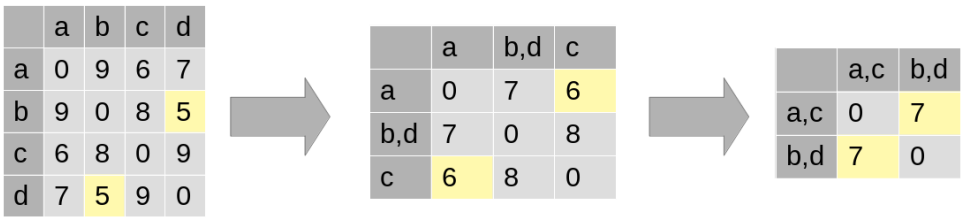


Figure 4.6: Each table represents a distance matrix between pairs of nodes. In the first table, d_{bd} marked in yellow is the shortest distance and therefore chosen to be part of the same cluster. The new node, or cluster, (b, d) contain the shortest distances to the other nodes. In the second table,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

Comparing the Siluette coefficient of one cluster to another requires that the nodes are divided into clusters. The next section will explain how we found the best cluster for a given size k.

Hirarchial single linkage clustering

To find the best possible cluster of a given size k, we used hirarchial single linkage clustering algorithm.

Figure 4.6 illustrates how the algorithm works. 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 amount of nodes.

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 best cluster for a given size. We used this in our model by running a preprocessiong algorithm that assigns the locations to their corresponding clusters of the optimal size using these concepts. This means that before our model starts each location i is assigned a cluster C_i^L.

Algorithm 5 start by selecting the amount of orders to remove. Then it sorts all orders from the given solution s according to the cluster value c_i^L. The cluster value is calculated based on the locations visited by the vehicle, ie. which orders are bundled together on a vehicle schedule. We can define c_i^L as:

$$c_i^L = \frac{|L_i^P| + |L_i^D|}{|S_v| - 2} \tag{4.4}$$

Here the set L_i^P contain the locations from the same cluster C_i^{LP} as the pickup location P of i on the same vehicle schedule. L_i^D is a set of locations from the same cluster C_i^{LD} as the delivery location D of i, visited in the same vehicle schedule. The set S_v is the vehicle schedule for vehicle v, ie. $|S_v|$ is the amount of locations visited.

Algorithm 5 Cluster heuristic

```
1:  function CLUSTER(solution s, p)
2:      select the number of orders to reinsert q
3:      add orders to array A acending based on cluster value c_i^L
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:     for i ∈ I do
11:         find vehicle schedule S_v which maximizes c_i^L
12:         insert i in best possible position in S_v
13:         update s based on S_v
14:     return s
```

To help explain how the c_i^L works we give an example:

- A vehicle schedule $S_1 = (1\ 2\ 1\ 2\ 3\ 3)$
- Order 1 has a pickup location in cluster A and delivery location in cluster B.
- Order 2 has a pickup location in cluster B and delivery location in cluster C.
- Order 3 has a pickup location in cluster C and delivery location in cluster B.

For each location, related to other orders, visited in cluster A, order 1 will be rewarded a point since its pickup is in cluster A. For each location visited in cluster B, related to other orders, order 1 will be rewarded a point. The pickup of order 1 is in cluster A which is unvisited by other orders, so $|L_1^P| = 0$. Therefore c_1^L is not increased. The delivery of order 1 is in cluster B, which is visited during pickup of order 2 and delivery of order 3 and therefore $|L_1^D| = 2$ which increases c_i^L.

With the same logic, for order 2 the $|L_2^P| = 2$ and $|L_2^D| = 1$. Order 3 gets $|L_3^P = 1$ and $|L_3^D| = 2$. To be able to compare these points with orders on other vehicles we divide the points on the maximum possible points possible which is the same as $|S_v| - 2$, or in the example above $6 - 2 = 4$ locations. That leads us to $c_1^L = \frac{2}{4}$ and $c_2^L = c_3^L = \frac{3}{4}$ in our example.

In continuation of explanation of algorithm 5 it uses the cluster ranking c_i^L 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 c_i^L sorted list of orders, where y is a random number between $[0, 1]$. We have chosen $p = 4$ for all our heursitics.

To reinsert the orders we have chosen to try and insert the order by maximizing c_i^L. This means we find the vehicle schedule where the c_i^L rank is the highest, and insert the order in the best possible position in the chosen vehicle.

4.4.6 Greedy

Removing orders in the most costful 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. The C_i is calculated as the increase in a vehicles schedule cost with the chosen order.

Algorithm 6 Greedy heuristic

```
1: function GREEDY(solution s, p)
2:     select the number of orders to reinsert q
3:     add orders in array A decending based on cost C_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:    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
```

We remove orders based on the same randomness factor explained above p. We do this by first sorting the orders decending based on the cost C_i. The cost can be calculated as $C_i = f(S_v) - f_{-i}(S_v)$, for a given order i served by vehicle 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 $c_i = \min(\Delta f_{iv})$, were the Δf_{iv} is the lowest increase in objective function value when inserting i in a vehicle schedule v. The c_i is therefore the minimum of these increases. We reinsert the order in its best possible position in the vehicle schedule s before we update the solution s.

4.4.7 Similar Regret

The removal part of this heuristic is based on Shaw (1997)'s removal heuristic with slight modifications based on our problem. It removes orders that share specific similar qualities. The basic idea is that replacing these orders by eachother 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: a distance property, a weight property, an overlapping timewindow property, a property indicating if the same vehicles can be used to serve each request, and finally if the orders belong to the same factory.

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 paper $\psi = 0.7$, $\omega = 1.0$, $\phi = 0.8$, $\tau = 0.3$, $\chi = 0.3$.

D_{ij}, Q_i, are the same as in chapter 2 and all values have been normalised to result in a value between [0..1]. V_i is the set of 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. U_{ij} is the timewindows 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 inbetween these two. It can be formulated as follows:

$$U_{ij} = 1 - \frac{T_{ij}^O}{T_{ij}^A - T_{ij}^{NO}} \tag{4.6}$$

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

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

T_{ij}^A represent 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:

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

The factor T_{ij}^{NO} is the opposite of the above factor T_{ij}^O and represent the time when neither i nor j have a time window. This can be examplified by night time when no factory is open. It can be formulated as follow:

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

Therefore the dividend in eq. (4.6) correspond 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_{ij}^O T_{ij}^{NO}, T_{ij}^A and $T_{ij}^A - T_{ij}^{NO}$ would be calculated for two locations 1 and 2.

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

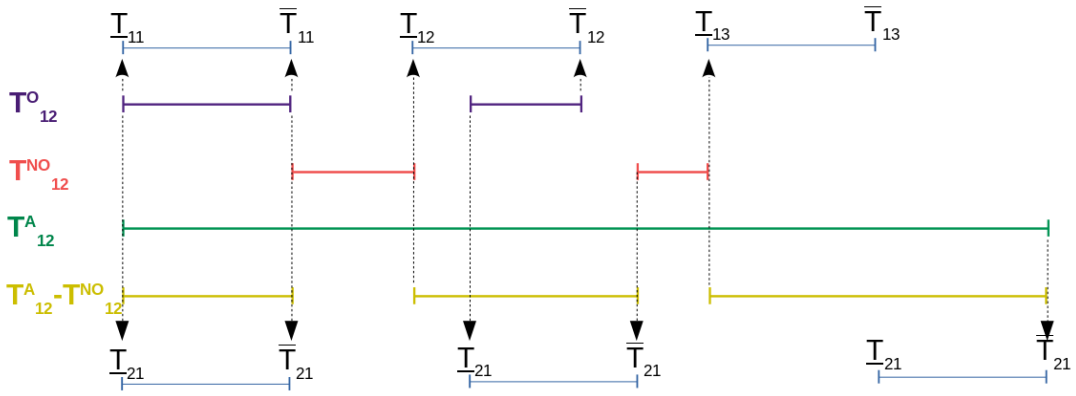


Figure 4.7: The figure shows two locations, 1 and 2, timewindow sets $[\underline{T_{1p}}, \overline{T_{1p}}]$ and $[\underline{T_{2p}}, \overline{T_{2p}}]$. The purple T_{12}^{O} is the overlapping time windows. The red T_{12}^{NO} represent the time when no location has a time window. The green T_{12}^{A}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

Algorithm 7 Similar Regret heuristic

function SIMILARREGRET(solution s, p)
  select the number of orders to reinsert q
  select a random order i from s
  add i to order set I
  repeat
    add all orders $j \notin I$ in array A acending based on relatedness r_{ij}
    choose a random number y between $[0, 1]$
    remove the order in position y^p in A from s
    add removed order to I
  until $|I| = q$
  repeat
    sort I acending based on regret value c_i^*
    insert the first order z from I in its best possible position in S_v
    update s based on S_v
    remove order z from I
  until $|I| = 0$
  return s

Algorithm 7 shows a pseudocode of the complete Similar Regret heursitic. 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 acending 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. That means $\Delta f_{S_{i1}} \leq \Delta f_{S_{i2}} \leq \Delta f_{S_{i3}}$. We can then define the regret value as follows:

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

The $c*_i$ therefore represent 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.

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. The reason for doing so 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 type of heuristics gives us amore robust algorithm. We will however do alot of testing in chapter 5, which will lead us to a more compact version of the model.

To select a heuristic in each iteration of algorithm 2, we have used what can be refered to as a *roulette weel principle*. This means that we represent each heuristic by a number $i \in [1..m]$ where m is the number of heuristics. We select a heuristic with a probability p_i calculated based on each heuristics *weight* w_i as follows:

$$p_i = \frac{w_i}{\sum_{j=1}^{m} w_j} \tag{4.11}$$

These weights could be set fixed per problem but we choose to use an adaptive weight system explained futher in section 4.6.

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 model. The basic idea is to keep track of the perfomance 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 entire algorithm 2 is divided into *segments*, or a numer of iterations. We have used segments here of 100 iterations.

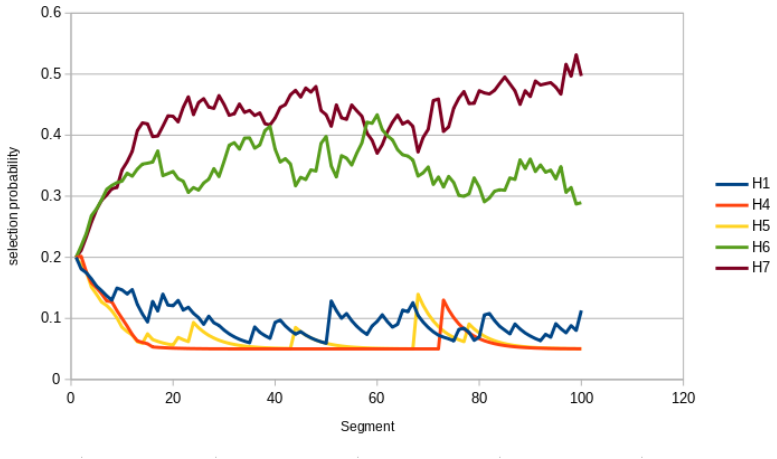


Figure 4.8: The figure illustrates an example of the developement of the weight probability p_i when running our model. The x-axis represent a segment and the y-axis the probability of selecting a heuristic.

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.
- 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_{ij} be the weight of heuristic i in segment j the update would be as follows:

$$w_{ij} = w_{i(j-1)}h + (1-h)\frac{\pi_i}{\lambda_i} \tag{4.12}$$

The h represent here a historical weight factor which we have set to 80% meaning we let the previous weight compose 80% of the new weight. 20% of the current weight w_{ij} is composed of the score π_i divided over the times we have run the heuristic λ_i during the current segment.

Figure 4.8 show how the weight probability p_i developes for five different heuristics in an example from our model. We observe that the probability start out equal but that some heuristics are getting more probability after only a few segments. The figure also shows that we have put a lower limit to the probility 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 only choose to only accept solutions that are better than the current one. However this could lead our model to get problems with diversification and not be able to get explore the entire solution space. We have

therefore chosen the acceptance criteria used in simmulated annealing, mentioned in chapter 3. This acceptance criteria is accepting a solution that is better than the current solution. It also accepts a worse solution with the probability $e^{-|f-f_{new}|/T}$, where $T < 0$ is the temperature. The cooling schedule 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 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:

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

The algorithm 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 aswell as globally. However considering that our model is more focused on instensification, 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 2 we see in **??** 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 8 Wild escape algorithm

```
1: function WILDESCAPE(solution s, s_best, set of heuristics W)
2:     repeat
3:         choose a random heuristic h from W
4:         apply h to s
5:         if f(s) < f(s_best) then
6:             s_best = s
7:     until stop condition met
8:     return s
```

The algorithms pseudocode 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 inspite 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.

Chapter 5

Computational Results

This chapter will present the results from several experiments leading us to the experiments of a final composition of our ALNS model.

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 contain the initial experiments of our model that lead us to the final composition of our model. The first part, section 5.3.1, starts by testing the wild algorithm from section 4.8 and it finds that the wild 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 singificant 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 aswell as the wild 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 a robust and efficient alorithm.

5.1 Experimental Setup

In this section we describe the technical setup of the experiments aswell as the Analytical setup of our eperiments.

Technical Setup

The computational experiments in this paper 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 (Oracle Corporation, 2019). The mathematical model from chapter 2 is setup in AMPL (AMPL Optimization Inc., 2019), using the Gurobi solver (Gurobi Optimization Inc., 2019). We used Java (Oracle Corporation, 2019) to write the program to run the solution method from chapter 4. The ANOVA(III), multiple linear regressions and t-tests are performed in Matlab (The MathWorks Inc., 2019).

Analytics Setup

To test our model from 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 aswell 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 experiements, including ANOVA (III) and multiple linear regression analysis, and pairwise t-tests. For all statistical experiments in this paper 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. Here we then compared the performance of the final composition with solutions found by the mathematical model. 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. Hemmati et al. (2014) made an instance generator for tramp shipping routing and scheduling problems. Our instance generator was created based on real world data from an anonumous 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 compatabilities. Some vehicles might have cooling possibilities, some orders might require special equipment for transport, (like breakable or explosive goods) or there might be equipment required at the pickup or delivery location (like f.eks. a crane to unload goods). We refer to these capabilities as a *vehicle requirement*. Other information aquired by the data was travel distances and cost structure.

In the following we refer to the mathematical symbols from 2. To keep the instances feasible but still as realistic as possbile it makes sense to limit the data

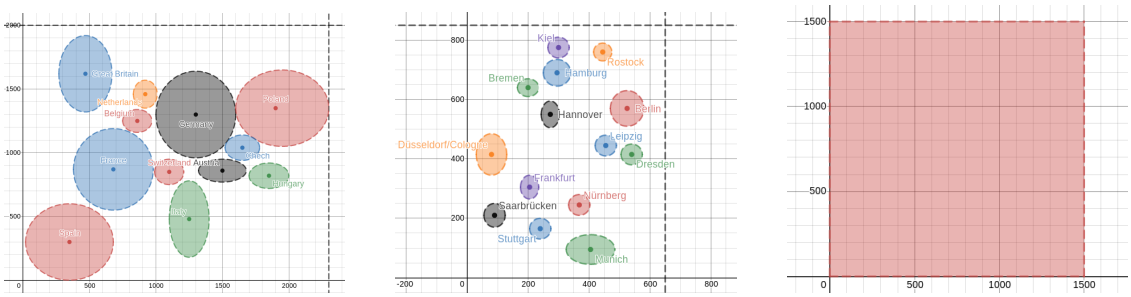
to ranges or predesigned patterns. Our data was generated based on the following possibilities:

- 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 euclidian 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 timewindows per day, and three to sevem days per week scaled based on the instance size.

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. To simplify we have sticked to geographical points with an eliptic uniformly distributed area surrounding the point to represent

Figure 5.1: Area of random point generation



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

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.

a country or a city. fig. 5.1 illustrates the areas of possible locations used in the generator. Larger elipses are more likely to be selected by the generator than smaller elipses.

For the selected elipse a point was selected within the elipse 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 contain 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.

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. **??** contain 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 contain the further evaluation of the heuristics. Afterwards, in section 5.3.5, we will present the final composition of the model.

Table 5.1: ALNS with all heuristics using 3 different reset algorithms

				Average Objective			Best Objective			Running time (sec)		
#Ord	#Veh	#Loc	Initial objective	No escape	Random reset	Wild escape	No escape	Random reset	Wild escape	No escape	Random reset	Wild 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

The columns contains in order: the first three columns show the instance size in number of orders, vehicles and locations. Then follows three columns that contain the average solution from runs with the different escape modifications: no escape, random reset and our wild escape alorithm. 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.

5.3.1 Evaluation of the wild algorithm

To help our algorithm escape from a local optimum, we designed a wild 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, refered 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, refered to as *random reset*. Lastly, we ran our algorithm with our *wild escape algorithm*. Comparing these three options towards eachother will help us evaluate if our wild algorithm is helping us in general and see if our wild algorithm 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 wild escape algorithm

The table 5.1 shows that on average using our wild algorithm clearly outperforms both the restart algorithm and not including a reset algorithm. We see that the average result for all instance sizes are lowest using our wild algorithm. In addition our wild algorithm outperforms the others 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 and 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. This taken into regards it becomes 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 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 differnt 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 analysis more readable.

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

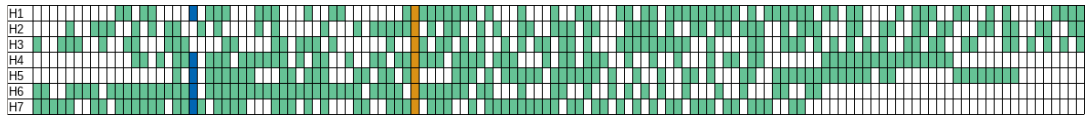
- In the first part we ranked the combinations based on the best improvement and the average improvement.
- In the second part we ran ANOVA and regression analysis to see which heuristics have a significant impact on the result.
- In the third part we run t-tests to see if certain heuristics in combination with others have a positive or a negative impact on the final result.

Ranking the combinations

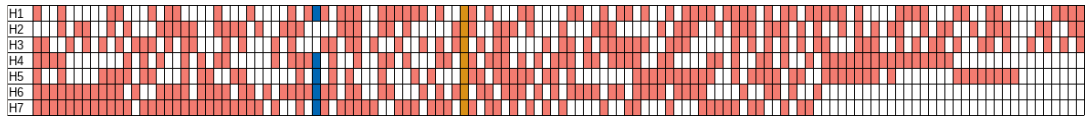
We shorten the names of each heuristic and will refer to them here on as:

- 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.

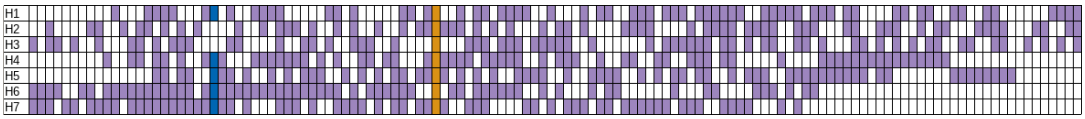
Figure 5.2: Ranking of improvements from initial solution



(a) By average improvement



(b) By best improvement



(c) By best+average improvement

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.

- 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:rank 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 figure 5.2a shows the combination of heuristics ranked based on the average improvment over the 10 runs for that combination. Then figure 5.2b shows the combination of heuristics ranked based on the best improvement overall during the 10 runs. Finally figure 5.2c shows the ranking of the average improvement + the best improvement to see which combinations performs best overall. The blue highlightes 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. However, ANOVA (III) and multiple linear regression analysis can help us evaluate which heuristics have a significant positive impact on the result. tab:anovaNormal 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 figure 5.2a, and one for the best improvement found in figure 5.2b, to see if some heuristics are making the

Table 5.2: Analysis of variance

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

The columns contains in order: the source of the variability and for each source the sum of the sqares, the degrees of freedom, the mean squares, the F-statistic and the p-value.

Table 5.3: Results of multiple linear regression model

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, $R^2 = 0.718$

The columns contains 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

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 regressional analysis gives us insight into if a heuristic is positively or negatively influencing the result, aswell 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 results from table table 5.2a and table 5.2b tells 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 soolutions. 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 which we need to do further testing.

Regarding the remaining heursitics, H1-swap, H2-exchange, and H3-2opt, we can safely conclude that they are not contributing singificantly 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 pooulation 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 on 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 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 G and the significant heuristics could help explain the variations in the

Table 5.4: Results of t-tests on the undecided group mean vs no undecided heuristics

Populations	Improvement type	Tail	H-stat	p-value	tStat	Confidence 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

The columns contain in order: the populations tested agains eachother, 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

Table 5.5: Analysis of variance with the undecided group in combination with the significant heuristics

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

The columns contains in order: the source of the variability and for each source the sum of the sqares, the degrees of freedom, the mean squares, the F-statistic and the p-value.

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 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, aswell as H4-random has a significant impact on the average improvement result. From table 5.6a we also see that the combinations

Table 5.6: Results of multiple linear regression model with the undecided group in combination with the significant heuristics

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, $R^2 = 0.967$

The columns contains 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

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 reagards to the best improvement tabels 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 conmbinations 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 nessecary 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 refering 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 use any combination of heuristics where only undecided heuristics are used, so we remove these observations from the data also in these tests. In our 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.

Table 5.7: Results of t-tests on individual heuristics

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

The columns contain in order: the populations tested agains eachother, 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

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 eachother 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 im-

provement 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% significanse 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 tells 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 tells us that our final model should include our wild 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 algorithm that the running time of H1-swap is significantly lower than our other heuristics **??**. 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. It will as section 5.3.4 also showed have no significant negative effect on the results of our model. H5-cluster is also not effecting the average improvement of our model but it significantly effects the result of the best improvement found. 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 model composition will be the ALNS algorithm with the wild algorithm and heuristics H1-swap, H4-random, H5-cluster, H6-greedy and H7-similar.

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.

Table 5.8: Model performance in european instance set 1

				Mathematical model			SCALNS			
Inst Set	Ord	Veh	Loc	Solution objective	Optimaily gap	Run time(sec)	Average objective	Best objective	Avrg 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

The columns contains 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 follows 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 - SCALNS best objective)/solution objective MM.

5.4.1 Evaluation of final model

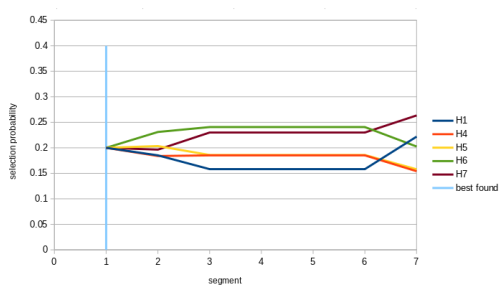
For each instance we used the powerful maschine to let AMPL try for 10 000 seconds to find an optimal solution for each instance. The larger runs resulted in a "out of memory" failure message so these results we simply marked as NA. The results from the runs of our final model composition together with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

Figure 5.3 shows the weight of each heuristic through each segment of our model SCALNS 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.

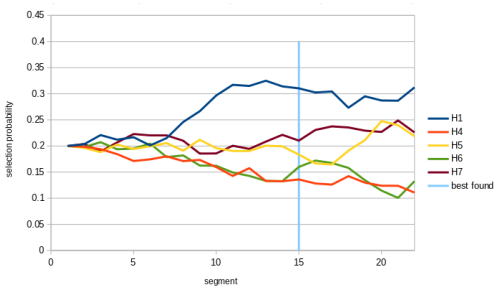
5.4.2 Observations of results the final composition

The first noticeable thing from the results of table 5.8 is that the performance of the mathematical model is quickly deteriarating with the size of the instances. Already at the second instance size the model is struggeling 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 objecteve after

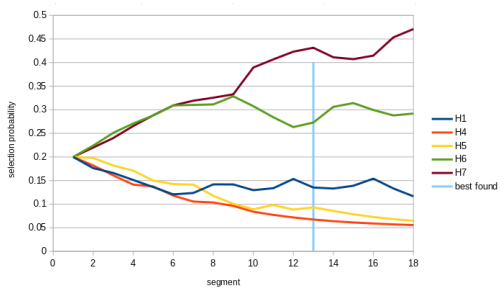
Figure 5.3: Operator weights per segment from instance set 1



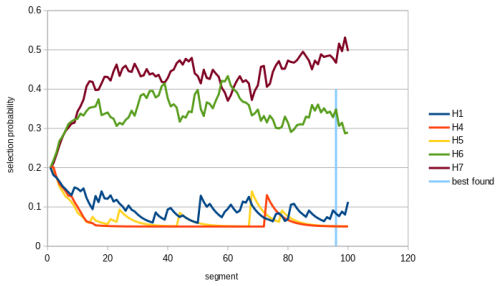
(a) Instance size 4 orders, run 1



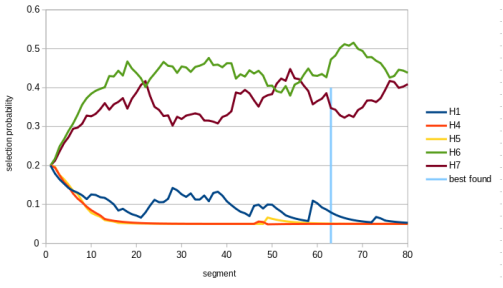
(b) Instance size 12 orders, run 3



(c) Instance size 35 orders, run 5



(d) Instance size 80 orders, run 7



(e) Instance size 150 orders, run 9

The y-axis represent the weight of each heuristic, the x-axis represent a segment run. The blue vertical lines represent the segment where the best solution was found.

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 SCALNS 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 **??** 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 fig. 5.3 show the performance of our heuristics. Figure 5.3b show that for smaller instances the heuristics H1-swap, H5-cluster aswell 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 5.3c, 5.3d and 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 indicate that they are contributing on possibly moving 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 doesnt 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 clearly our adaptive weights are 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

Conclusions

This is a test Shaw (1997). Hopefully it works..

Appendix A

Appendix

This is a test Shaw (1997). Hopefully it works..

Bibliography

4flow AG. Industries & references - automotive manufacturers, www.4flow.com. 2018.

AMPL Optimization Inc. Ampl ide, 2019. URL `http://www.ampl.com`.

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.

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

M. Christiansen, K. Fagerholt, and D. Ronen. Ship routing and scheduling: Status and perspectives. *Transportation science*, 38(1):1–18, 2004.

J.-F. Cordeau and Q. Groupe d'études et de recherche en analyse des décisions (Montréal. *The VRP with time windows*. Montréal: Groupe d'études et de recherche en analyse des décisions, 2000.

J.-F. Cordeau, G. Laporte, and A. Mercier. A unified tabu search heuristic for vehicle routing problems with time windows. *Journal of the Operational research society*, 52(8):928–936, 2001.

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

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.

M. Dror, G. Laporte, and P. Trudeau. Vehicle routing with stochastic demands: Properties and solution frameworks. *Transportation science*, 23(3):166–176, 1989.

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.

K. Fagerholt, G. Laporte, and I. Norstad. Reducing fuel emissions by optimizing speed on shipping routes. *Journal of the Operational Research Society*, 61(3): 523–529, 2010.

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.

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

Gurobi Optimization Inc. Gurobi for ampl, 2019. URL `https://www.gurobi.com/`.

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.

A. Hemmati, L. M. Hvattum, M. Christiansen, and G. Laporte. An iterative two-phase hybrid matheuristic for a multi-product short sea inventory-routing problem. *European Journal of Operational Research*, 252(3):775–788, 2016.

B. Kalantari, A. V. Hill, and S. R. Arora. An algorithm for the traveling salesman problem with pickup and delivery customers. *European Journal of Operational Research*, 22(3):377–386, 1985.

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.

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.

Q. Lu and M. M. Dessouky. A new insertion-based construction heuristic for solving the pickup and delivery problem with time windows. *European Journal of Operational Research*, 175(2):672–687, 2006.

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.

Oracle Corporation. Java se jdk, 2019. URL `https://www.oracle.com/technetwork/java/javase/downloads/jdk11-downloads-5066655.html`.

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.

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. 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.

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.

The MathWorks Inc. Matlab r2019a update 5, 2019. URL `https://mathworks.com/`.

H. Zhou. *Inventory Management and Inbound Logistics Optimization for a Food Processing Company*. PhD thesis, University of Cincinnati, 2013.