Truck Routing for an Online Grocer

Solving a Pickup and Delivery Problem with Resource Constraints

Master Thesis Computer Science

Thomas Barendse

Truck Routing for an Online Grocer

Solving a Pickup and Delivery Problem with Resource Constraints

by

Thomas Barendse

To obtain the degree of Master of Science
at Delft University of Technology
to be publicly defended on April 21st, 2022 at 10:00 a.m.

Committee

Dr. N. Yorke-Smith	Supervisor TU Delft
G.C. Konijnendijk, MSc	Supervisor Picnic
Dr. ir. J.T. van Essen	Committee member
Dr. J. Alonso-Mora	Committee member

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

TU Delft

Abstract

Demand for online grocer Picnic has increased exponentially over the past years, and their truck transport operation must scale with it. Given the resource constraints at all warehouses, as well as other specific restrictions, this poses a Multi Depot Pickup and Delivery Problem with Resource Constraints, for which no good solutions are found to date. The goal of this thesis is to develop an algorithm that provides good solutions to this problem within 30 minutes.

This thesis presents three solution methods to solve this problem. The first is an Adaptive Large Neighbourhood Search (ALNS) algorithm closely related to earlier research. We propose a mechanism with only linear time additional complexity to impose the resource constraint. We also demonstrate ways to take additional constraints into account, such as minimum route duration and driver switches. For the second method, dubbed ALNS+LS, we extend the ALNS with local search heuristics to enhance its performance. As a third method, we propose a matheuristic novel to this specific problem class. This consists of the ALNS+LS algorithm applied to the problem without resource constraints in the first phase, and imposing the constraints using a Constraint Programming model in the second phase. We show that the ALNS+LS outperforms the other two algorithms on a real-life-inspired benchmark set, and that the matheuristic comes close to the ALNS+LS for small instances. We finally show that the full problem is best solved by decomposing it into four parts and solving these separately.

Acknowledgements

I am thrilled to present to you the work that marks the end of 7,5 years of studying at TU Delft. These years have brought me from a BSc in Mechanical Engineering to a year of building a hydrogen-powered race car to an MSc in Computer Science, which I now conclude with a thesis aimed at improving truck planning at Picnic.

In light of the latter, I first want to thank my TU Delft supervisor Neil Yorke-Smith. You have given me the freedom to shape this project, and always provided me with valuable ideas, pointers and insights. I look back fondly on our collaboration for the past 9 months.

A second word of gratitude goes out to the people at Picnic, the company who gave me the opportunity to work on this project. I want to thank in particular my Picnic supervisor Geert Konijnendijk. You proved to be a great sparring partner, a calm, observant and smart person excellent at giving feedback. I truly enjoyed working with you. Besides that, I want to thank the DIST team for embracing me as a true team member. You, along with so many more people at Picnic, have made the past 9 months a lot of fun.

A final thank-you goes out to my family and friends. To my parents in particular for their unconditional pride, for always keeping me sharp to obtain the best results and for their continuous support.

All I now have left to say is: have fun reading.

Thomas Barendse
Delft, April 2022

Contents

Abstract iii

Acknowledgements v

1 Introduction 1
- 1.1 Problem statement . . . 1
- 1.2 Goal and research questions . . . 2
- 1.3 Data anonimisation . . . 3
- 1.4 Outline of the thesis . . . 3

2 Background and problem statement 5
- 2.1 Picnic's supply chain . . . 5
  - 2.1.1 Size of the operation . . . 6
  - 2.1.2 Regular and Morning Outbounds . . . 6
  - 2.1.3 The Outbound picking line . . . 7
  - 2.1.4 Shipment characteristics . . . 7
  - 2.1.5 Hub-FC combinations . . . 8
  - 2.1.6 Site-Located Trucks and associated costs . . . 8
- 2.2 Additional restrictions . . . 8
  - 2.2.1 Capacity constraints and combining shipments . . . 8
  - 2.2.2 Time Windows . . . 9
  - 2.2.3 Route duration . . . 9
  - 2.2.4 Driver switching . . . 9
  - 2.2.5 Synchronising trucks visiting the same location . . . 10
  - 2.2.6 Implementation restrictions . . . 10
- 2.3 Optimisation objective . . . 10
- 2.4 The problem in a CP model . . . 10
  - 2.4.1 Solving the CP model . . . 13
- 2.5 Current practice, limitations and potential . . . 13
  - 2.5.1 Planning Outbounds: FMS . . . 13
  - 2.5.2 Planning Inbounds . . . 13
  - 2.5.3 Limitations of the current practice . . . 13
  - 2.5.4 Expected areas of potential . . . 14
- 2.6 Data(sets) and adaptations . . . 14
  - 2.6.1 Outbound data . . . 14
  - 2.6.2 Inbound data . . . 15
  - 2.6.3 Representative instances . . . 15

3 Related work 17
- 3.1 Routing problems . . . 17
  - 3.1.1 TSP, VRP and PDP . . . 17
  - 3.1.2 Common extensions to routing problems . . . 18
  - 3.1.3 Resource constraints in routing problems . . . 19
- 3.2 Constraint Programming . . . 19
  - 3.2.1 Constraint Programming for VRPs . . . 20
  - 3.2.2 MiniZinc and OR-Tools . . . 20

3.3 Heuristic solution methods . . . 20
3.3.1 Local search . . . 20
3.3.2 Large neighbourhood search . . . 22
3.4 Matheuristics . . . 24
3.5 Most relevant papers . . . 24

4 An ALNS approach **25**
4.1 Adaptations to the resource constraint . . . 25
4.1.1 Theoretical comparison to Grimault's procedure . . . 28
4.2 ALNS heuristics . . . 30
4.2.1 Removal heuristics . . . 30
4.2.2 Insertion heuristics . . . 30
4.3 Initial solution and vehicle minimisation . . . 31
4.4 Further adaptations . . . 31
4.4.1 Minimum route duration incorporated in the cost function . . . 31
4.4.2 Driver switching . . . 32
4.4.3 Dealing with multiple depots and empty routes . . . 32
4.4.4 Adaptive parameters . . . 33
4.5 Summarising the ALNS . . . 34
4.6 Running experiments . . . 34
4.7 Evaluation . . . 35
4.7.1 Tuning relevant parameters . . . 35
4.7.2 Partitioning instances . . . 37
4.7.3 Concluding remarks . . . 38

5 Extending the ALNS with local search **39**
5.1 Local search heuristics . . . 39
5.1.1 Relocate . . . 40
5.1.2 Exchange . . . 40
5.1.3 Crossover . . . 40
5.1.4 Excluded heuristics . . . 40
5.2 Local search and the resource constraint . . . 41
5.3 General framework . . . 41
5.4 Evaluation . . . 42
5.5 Concluding remarks . . . 44

6 A Matheuristic approach **45**
6.1 The CP model . . . 45
6.2 The matheuristic framework . . . 46
6.2.1 Reducing the number of CP evaluations . . . 46
6.3 First evaluation . . . 47
6.4 Promoting feasibility . . . 47
6.4.1 First ideas: inter-route penalties . . . 48
6.4.2 Promoting slack . . . 48
6.4.3 Enforcing more routes . . . 49
6.4.4 Evaluation after applying feasibility promotion . . . 49
6.5 Concluding remarks . . . 50

7 Final evaluation **51**
7.1 Comparing the best versions of the different algorithms . . . 51
7.2 Five full problem instances . . . 52
7.3 Solving small instances using the CP solver . . . 54
7.4 Comparison to FMS . . . 54

7.5 Full instances and decompositions . . . 55
7.5.1 Incorporating Inbounds vs. keeping them separate . . . 57
7.6 Concluding remarks . . . 59

8 Discussion **61**
8.1 ALNS . . . 61
8.2 ALNS+LS . . . 62
8.3 Matheuristic . . . 62
8.4 Final evaluation . . . 63

9 Conclusions and recommendations **65**
9.1 Conclusions . . . 65
9.2 Recommendations to Picnic . . . 66
9.3 Future research . . . 66

References **71**

A Paper outline **73**
A.1 Introduction . . . 73
A.2 Related work . . . 74
A.3 Problem formulation . . . 74
A.4 Solution method . . . 74
A.5 Computational results . . . 74
A.6 Conclusion . . . 74

B Tables and figures **75**

1 Introduction

Transport problems come in all flavours, and while they all trace back to the same core problems, each has its own characteristics and subtleties that make it different from ones studied before. Picnic's truck planning problem is no exception, as this introduction will clarify.

Picnic is the fastest growing online(-only) supermarket in the Netherlands [52], and possibly beyond those borders. Picnic also is a quintessential example of a company with an intricate logistical operation. Their proposition to the customer is to deliver groceries to your doorstep within 24 hours, guaranteeing lowest prices and free delivery. First of all, Picnic has no physical stores to pay for, and secondly, they have set up a distribution model unhindered by existing store-oriented infrastructure. This distribution network is split into two stages: the last-mile delivery, i.e. the delivery to the customer, and the truck transport, which covers everything before that. This thesis takes the latter as a use case for solving a variant of the vehicle routing problem with a few specific constraints.

1.1. Problem statement

Picnic's truck planning problem specifically entails creating a truck schedule such that all shipments between its warehouses on a given day are shipped in time. Shipments have an origin warehouse, a destination warehouse and a time window in which they must be shipped. Since a substantial part of these time windows are tight compared to the time horizon (a few hours at most on a horizon of a full day), and because shipments are large compared to truck capacity (often almost full, sometimes less than half full), a basic Vehicle Routing Problem [10] does not capture the essence of the problem. Rather, we cast the problem as a Pickup and Delivery Problem (PDP) [14]. In this model, shipments are defined by two nodes: a pickup node and a delivery node. Both of these nodes have a location and a service (loading/unloading) time, and a time window ([earliest start service time, latest start service time]). Hundreds of shipments are to be shipped on any given day, which is reasonably large compared to benchmark instances of PDPs in the literature. The variation in this number is sizable, as it is strongly influenced by how many people order groceries on that day. One can imagine that some days of the week are preferred over others, and that peaks arise around e.g. national holidays.

The objective function that we aim to minimise is the monetary costs associated with the generated schedule. Costs in this case are made up of distance costs and driving hour costs. Distance costs are the costs per driven kilometre, which naturally consists of fuel costs, truck depreciation, etc. Driving hour costs are the wages paid to the truck drivers. Since both costs can be expressed in Euros directly, they can be added up and hence we end up with a single objective.

The major aspect that sets this problem apart from others is the resource constraint we are dealing with at the warehouses. Since the delivery locations generally have a single dock, only one truck can be serviced at a time. Pickup locations have to deal with the fact that no two trucks can depart right after one another for operational reasons. Both of these restrictions lead to the fact that routes become dependent on one another, which adds considerably to the complexity of the problem. Most other constraints are simpler of nature, and include lower and upper bounds on working hours, switching drivers on a truck halfway through the day, and more.

What makes this problem somewhat more specific is that pickups and deliveries appear somewhat clustered. The majority of the shipments consists of groceries that are picked at fulfillment centres (FCs, large central warehouses), and delivered to hubs (local transfer warehouses). Shipments are composed such that a hub is usually supplied from its closest fulfillment centre. An important question that arises is whether it is better to consider all shipments together, or a decomposition of the shipments by fulfillment centre. This trade-off is one between complexity and potential synergy: planning shipments with close by pickup locations together may be beneficial from combinatorial perspective. However, planning shipments from FCs that lie far apart will hardly be beneficial, and will suffer from increased complexity due to the increase in size.

Finally, it must be noted that this truck planning problem is one of many links in Picnic's daily planning process of the entire supply chain. Therefore, the time we have to solve this problem is limited. For this thesis, also considering the implementation specifics, we limit ourselves to a runtime of 30 minutes.

The scientific contribution of this thesis lies in the fact that the pickup and delivery problem we are dealing with combined with the location constraint has not been subject to a lot of research, but is likely to be encountered more often. It is interesting to see how existing solution methods are applicable to this problem, and what adaptations have to be made for this problem's specific needs and to better solve it. The limited runtime is also a significant factor in this regard. The business contribution of this project is also clear, in the sense that we aim to improve on Picnic's current planning methods in order to improve efficiency.

1.2. Goal and research questions

Having set the scene for the problem we are looking at, we can define the goal of this thesis:

Develop an algorithm that finds a solution to Picnic's truck planning problem with the lowest possible costs within 30 minutes

To this end, we intend to answer the following research questions:

- Which vehicle routing/scheduling model suits Picnic's use case best, and how should such existing models be extended to account for specific constraints from Picnic's operations?
- What exact solving technique, if any, can be used to obtain a good solution to the problem in the specified time?
- What inexact solving technique(s) can be used to obtain a good solution to the problem in the specified time?
- Can we solve full instances effectively, or does it pay off to decompose the problem into subproblems and solve those separately?

1.3. Data anonimisation

For confidentiality reasons, some of the data used throughout this thesis has been obfuscated. All objective values are multiplied by a constant such that true costs are not made clear. Also, the fulfillment centers are characterised by a letter, while Picnic numbers the FCs, in a different order. Also, two sections have been moved to a separate document as they contained too much sensitive information.

1.4. Outline of the thesis

The remainder of this thesis is structured as follows: Chapter 2 provides a detailed background of Picnic's supply chain, problem specifics and current solution methods. We also formalise the problem in a mathematical model in this chapter. We explore the state-of-the-art regarding similar routing problems in Chapter 3. Chapters 4 through 6 describe the three algorithmic approaches we have developed to solve the problem, and evaluate their performances. Chapter 7 compares the best configurations of the three algorithms and presents experimental results on full day instances. In Chapter 8 we discuss our work, and in Chapter 9 we draw conclusions and provide recommendations for future work.

2

Background and problem statement

To obtain a complete understanding of the problem at hand, we dedicate this chapter to all background and context related to it. Section 2.1 describes the relevant parts of Picnic's supply chain in detail, and introduces all relevant concepts and terms. Section 2.2 lists all restrictions implicated by other parts of the operation and section 2.3 introduces the optimisation objective. Section 2.4 formally defines a mathematical model for the problem, whereas section 2.5 outlines Picnic's current methods for solving it. Finally, section 2.6 discusses the data we use, the assumptions and adaptations we make to that, and a set of instances we use throughout this thesis to assess the performance of our algorithms.

2.1. Picnic's supply chain

In essence, Picnic collects groceries from its *suppliers*, and delivers these to their *customers*. To make this flow of goods work efficiently, Picnic employs mainly three different types of warehouses: Distribution Centres (*DC*s), Fulfillment Centres (*FC*s) and *Hub*s. Some products are shipped in bulk directly from suppliers to FCs. Others, usually those that go in smaller quantities, are first shipped to DCs, and later distributed over FCs. Order picking for the customer is done centrally at the FC: products are put in bags which go in totes, such that a bag will always contain groceries belonging to a single customer. The totes are then placed in *Dispatch Frame*s, such that a frame contains deliveries destined to a single Hub. In fact, all totes that go in the same dispatch frame will be delivered in the same last-mile delivery trip. The frames are hence distributed by trucks over the Hubs, from where the orders get delivered to the customers.

This thesis considers all truck transport between (see Figure 2.1):

- Suppliers and FCs
- Suppliers and DCs
- FCs and DCs
- FCs and Hubs

An important note regarding suppliers is that Picnic does not plan all transport belonging to those. Most suppliers have their own distribution network, trucks and schedules, and Picnic is often bound by this. There are a few exceptions where Picnic closely collaborates with its suppliers, and the transport of which it has in its own hands. Part of the transport is also the (back)flow of empty load carriers such as pallets and roll containers, packaging materials, etc. This also explains why there is a need to plan trips from FCs back to DCs.

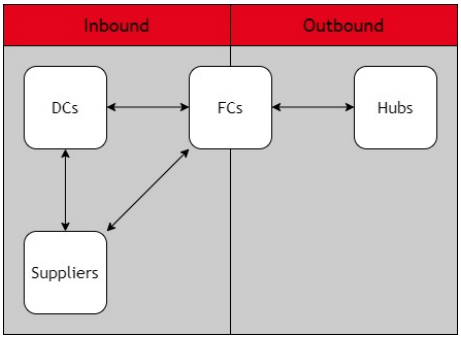


Figure 2.1: Picnic's transport flows

Since the FCs lie at the heart of Picnic's supply chain, all truck transport is considered from an FC's perspective. All transport between FCs and Hubs, i.e., picked orders in dispatch frames destined for hubs and possible backflow, are categorised as *FC Outbound* or *Outbound* shipments. All other transport, i.e. between any two locations excluding Hubs, is considered *FC Inbound* or *Inbound* shipments. As Figure 2.1 points out, this also includes a few shipments between DCs and suppliers. Throughout this thesis, we will use these terms consistently with this definition to make a distinction between transport flow types.

2.1.1. Size of the operation

Picnic's operation is growing at a high pace. At the start of writing this thesis, the numbers of the different warehouse types are as follows:

- 7 FCs
- 2 DCs
- 53 Hubs
- 5 Other

The geographical locations are shown on the map in Figure 2.2. Other locations include suppliers, truck depots, etc. The number of shipments can vary significantly on a daily basis. For instance, the day of the week on its own has a noticeable impact.

The complexity of vehicle routing problems, as with many optimisation problems, grows exponentially in the number of shipments. Generally this number is referred to as the instance size, where the instance is defined as the set of shipments we are looking at at any point.

2.1.2. Regular and Morning Outbounds

The Outbounds can be specified further. This is not so much a specification by definition, but rather follows from how Picnic operates. Originally, Picnic would deliver groceries only in the afternoon or evening, which would allow them to do order picking at FCs in the morning or early afternoon. As Picnic grew along with its customer base, the operation had to scale up. Picnic therefore decided to start delivering in the morning also. This required them to do order picking and Outbound transport on the evening before, or in the early morning. To this day, these shipments are planned separately and are known as *Morning Outbounds*, as opposed to the *Regular Outbounds*. Throughout this thesis we use this terms when referring to either of the two, whereas we use *Outbouds* as an overarching term.

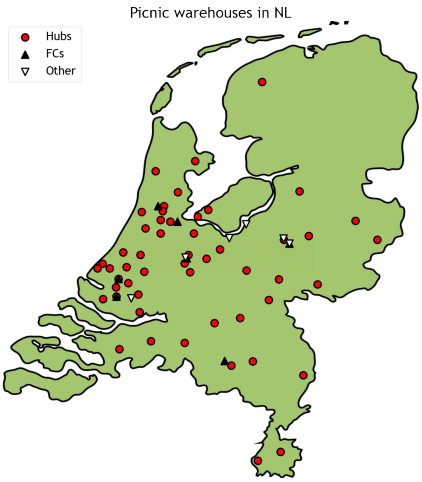


Figure 2.2: Picnic warehouses in NL

2.1.3. The Outbound picking line

When an Outbound shipment is ready to be picked up is determined by when it is picked, i.e., by the order of the picking line. The picking of shipments is almost a sequential process, and for this thesis we assume that it is. This means that throughout the day new shipments are ready to be picked up, and Picnic can freely determine this order, as long as the shipment can be transported to the hub in time. Picnic's current planning algorithm (see section 2.5) actually integrates picking line scheduling with truck planning.

2.1.4. Shipment characteristics

For planning purposes, any shipment in this context can be fully described by the following attributes:

- **Pickup and delivery location**
  The locations where a shipment is to be picked up and delivered respectively.
- **Earliest pickup time and latest delivery time**
  These times are determined by other parts of the operation (see also the previous subsection). For the scope of this thesis, we separate these processes from the routing, so we assume that these times are fixed.
- **Size**
  The size defines what part of a truck is filled by the shipment. Shipments may be combined to create a more efficient schedule, provided that they fit together in one truck.
- **Pickup and delivery service time**
  These are equivalent to the loading and unloading times of a shipment at its pickup and delivery location respectively. Service times consist of a fixed component, and a component variable in terms of shipment size. This can be explained most easily by the fact that a truck will need time for parking, docking, etc. regardless of the size of its load, and time per unit of load for loading/unloading.

Throughout this thesis, we may use the terms pickup time and departure time interchangeably. The same goes for the terms delivery time and arrival time.

2.1.5. Hub-FC combinations

To prevent having to transport groceries from one end of the country to the other, Picnic strategically positions their FCs evenly across it. As a natural consequence, Hubs can often be supplied from the FC closest to it. This creates a rather decentralized network of Hubs and FCs, in which it makes sense to plan truck schedules per FC: it will hardly ever be beneficial to have a truck transporting shipments from an FC on one end of the country, and then drive to an FC on the other end to pick up and deliver a shipment. On the other hand, some FCs are relatively close to one another, and it may be advantageous to plan their shipments jointly.

2.1.6. Site-Located Trucks and associated costs

The trucks that Picnic uses for this transport operation are owned by multiple third party logistics (3PL) providers, henceforth referred to as *the 3PLs*. The situation is such that the 3PLs station an agreed upon number of vehicles at each of Picnic's FCs and DCs. Picnic refers to these trucks as *Site-Located Trucks*, or *SLTs*. The cost that Picnic incurs for the truck use is in reality a non-linear function of the number of SLTs used and driven kilometres. But while it is desirable to keep this number low, it does not pay off to minimise this on a daily basis given the number of available SLTs. For simplicity's sake, we reduced these costs to a fixed price per kilometre. In spite of the actual complex cost function, this fixed price reasonably reflects the true price.

There is an important side benefit to having trucks stationed at FCs and DCs. Since most truck routes will start with a pickup at their depot, a truck driver can start his schedule when the truck is ready to leave, i.e., after loading the shipment. In case a truck is not stationed at its first pickup location, it should first drive to that location, and then wait until loading is finished, adding significantly to the cost of this routing. Throughout the rest of the thesis, we will use the term *truck* instead of SLT for simplicity's sake.

2.2. Additional restrictions

Besides the problem characteristics that follow from the previous sections, there are a few extra constraints that are to be taken into account. They are described in this section.

2.2.1. Capacity constraints and combining shipments

In some vehicle routing problems, there is no limit on the number of deliveries a vehicle can do before having to return to its depot. When maximum capacity is infinite, or in practice never reached, this is something that does not even have to be taken into consideration. An example could be a parcel delivery service.

With our truck planning problem, we find ourselves almost on the other side of the spectrum. Currently, Picnic generally picks up a single truckload, delivers it to its destination, picks up a new load, and so on. Most truckloads are at least half full, such that putting multiple shipments together will hardly be possible in practice. Moreover, Picnic implicitly assumes that a truck transports backflow (empty dispatch frames or other load carriers) back to its origin location. For our problem, we will allow combining at most two shipments as long as capacity constraints allow it. As we demonstrate later, this will also decrease the computational complexity of our problem.

How combining shipments affects service times

When combining two shipments with same origin, i.e., performing two subsequent pickups at the same location, we can disregard the fixed service time of the second pickup. Naturally, a truck does not have to dock twice in this case. The total pickup time becomes the sum of once the fixed service time of the origin and the variable service times of both pickups.

2.2.2. Time Windows

We already discussed the concepts of earliest pickup time and latest departure time, but we formalise this further in terms of time windows. Warehouse operations and planning determine when a shipment is ready to be picked up, i.e., its earliest pickup time. Similarly, a shipment has a latest delivery time. Latest pickup time l_{pickup} follows from the latest delivery time minus the pickup service time minus the driving time. Earliest delivery time $e_{delivery}$ follows from earliest pickup time along the same lines:

$$l_{pickup} = l_{delivery} - st_{delivery} - dt_{pickup-delivery} \tag{2.1}$$

$$e_{delivery} = e_{pickup} + st_{pickup} + dt_{pickup-delivery} \tag{2.2}$$

A time window (*TW*) is then defined as the interval $[e_{pickup}, l_{pickup}]$ for a pickup, and, not surprisingly, $[e_{delivery}, l_{delivery}]$ for a delivery. Throughout this thesis, whenever we talk about time windows and optimal times, we refer to start service times, unless otherwise specified.

2.2.3. Route duration

The number of hours a driver can work each day, and each week, are bounded by law. Laws also exist for the number and duration of breaks that a driver should take. Finally, a driver is only allowed to work so many hours on a weekly basis. Picnic naturally has to abide by these laws, but the nature of the driver schedules allows us to disregard break times in this case. This is because trucks spend a relatively big part of their time standing still (un)loading at warehouses. Also, we will not consider weekly limits, assuming that the 3PLs can always schedule drivers to trucks such that these laws are respected.
The rules for daily working hours are as follows:

- Case 1: Schedule starts at or after 5:15H.
  - The total schedule duration must be $\leq$ 15 hours
  - The total driving duration must be $\leq$ 9 hours
- Case 2: Schedule starts before 5:15H
  - The total schedule duration must be $\leq$ 13.5 hours
  - The total driving duration must be $\leq$ 9 hours

Again, due to the many stops, the 9 hour driving time limit is never reached in practice, and hence disregarded. Picnic does not plan schedules of duration close to the upper bound, as delays may make drives exceed these bounds. We therefore adopt upper limits of 13.5 and 11 hours for Case 1 and 2 respectively.
On the other hand, drivers have to work a minimum number of hours per day. We take this minimum to be 8. Important to note here is that this is not a hard constraint. A schedule may be shorter than 8 hours, but then the driver is still paid for 8 hours of work. While this is not desirable in general, there may be cases where this is preferred.

2.2.4. Driver switching

Picnic's transport is currently not a 24/7 operation, but every day the earliest trucks will depart around 4 a.m. and the latest will return by midnight. This means that a truck schedule could be as long as 20 hours, but this would be rendered infeasible by the maximum schedule duration described above. To make optimal use of the available trucks however, we can plan two drivers to a single truck. A truck schedule is eligible for a driver switch under the condition that the truck visits its depot during the day such that switching drivers at this visit splits the schedule into two sub-schedules which both have a duration below the maximum duration. Again, this may yield one or even two sub-schedules that last shorter than the soft minimum, but this is not forbidden.

2.2.5. Synchronising trucks visiting the same location

Finally, there are restrictions regarding trucks that visit the same locations applying to both Hubs and FCs/DCs. Even though they differ from an operational perspective, they are conceptually surprisingly equivalent.

- **Hub restrictions**
  Many Hubs can only service one truck at a time, because Hubs generally only have one dock. Should a truck arrive while another one is being serviced, it simply has to wait. There are exceptions to this, but for generality's sake, we assume this is always the case.
- **FC/DC constraints**
  At FCs and DCs, the number of docks is assumed to never be limiting, so arriving trucks should never have to wait. However, operations require that there is at least 10 minutes between any two consecutive departing trucks.

This means that for Hubs, the arrival times of two consecutive trucks should be at least the service time of the first truck apart. For FCs/DCs the departure times (=pickup time + service duration) of two consecutive trucks should be at least 10 minutes apart. As we later demonstrate, this restriction has a major impact on building truck schedules. Because of this restriction, truck schedules visiting the same locations become dependent on one another. All other restrictions mentioned before did not introduce this phenomenon. From now on, we will refer to this restriction as the *resource constraint*, in line with existing literature.

2.2.6. Implementation restrictions

For the algorithm to fit in Picnic's existing planning procedure, we require it to run in no more than 30 minutes. This time restriction has an impact on the parameter configuration of the algorithm, as we later demonstrate.

2.3. Optimisation objective

Some vehicle routing problems require the total distance travelled to be minimized. Others intend to minimize travel time, and yet others aim to find the smallest number of vehicles required. Combinations of the above also exist, resulting in a multi-objective optimisation problem.

We are looking at minimizing total cost, expressed in terms of both distance and time. The cost per driven kilometre is assumed to be known, as well as the cost per working hour for drivers. Together these form a single cost objective. In addition to that, we incorporate the desired minimum route duration into the objective. This essentially penalises schedules that are shorter than this duration, and hence functions as a soft constraint. Also, this does truly represent the cost function, as Picnic is obliged to pay truck drivers for this minimum duration. As mentioned, we are indifferent to the number of trucks used given the number of available trucks. Under the reasonable assumption of a fixed price per kilometre, and no fixed costs per truck used, the cost function for a route therefore is expressed as:

$$c_{tot} = r_d * d + r_t * max(t, t_{min}) \tag{2.3}$$

where d is distance, r_d is the rate per distance unit, t is route duration, t_{min} is the minimum route duration, r_t is rate per time unit and c_{tot} is total route cost.

2.4. The problem in a CP model

We formally define the problem in a Constraint Programming (CP) model (see section 3.2 for background on CP). We describe the model in terms of input parameters, decision variables, the objective function and the constraints. CP models for VRPs have been defined before, among others by Kilby [28]. We expand upon this model given our problem specifics.

Parameters

P	Set of pickup nodes, each of which corresponds to exactly one shipment ($\|P\| = x$)
D	Set of delivery nodes, each of which corresponds to exactly one shipment ($\|D\| = x$)
E_s	Set of depot start nodes, each of which corresponds to exactly one truck ($\|E_s\| = m$)
E_e	Set of depot end nodes, each of which corresponds to exactly one truck ($\|E_e\| = m$)
N_s	Nodes having a successor: union of sets ***P***, ***D*** and E_s
N_p	Nodes having a predecessor: union of sets ***P***, ***D*** and E_e
N	All nodes: union of sets ***P***, ***D***, E_s and E_e
M	Set of trucks ($\|M| = m$)
H	Set of hub locations. Many pickup/delivery node actually coincide in the same location
H_j	Set of nodes belong to hub location j
F	Set of non-hub locations
F_j	Set of nodes belong to non-hub location j
t_{ij}	Driving time from node ***i*** to node ***j***
$dist_{ij}$	Driving time from node ***i*** to node ***j***
st_i^{fix}	Fixed service time at node ***i***
st_i^{var}	Variable service time at node ***i***
(a_i, b_i)	Time window for pickup/delivery at node ***i***
C	Truck capacity. Since we assume a homogeneous fleet, this is the same for all trucks
q_i	Size of the request of node i. Positive for pickup, negative for delivery
u_i	Location index of node i
i_l	Idle time between two departures at location l
t_{min}	Minimal driver day duration
$t_{max,j}(t_{2*x+j})$	Maximal driver day duration for truck j, depending on start time t_{2*x+j}
t_{max}	Maximal driver day duration for truck j, independent of start time.
km_rate	Cost for driving one distance unit
$hour\_rate$	Cost for employing a driver for one time unit

Decision Variables

s	Successor node for each node in N_s
p	Predecessor node for each node in N_p
r	Array indicating for each node in ***N*** which truck visits that node
t	Visiting (arrival) time for each node in ***N***
c	Load of vehicle after visiting each node in ***N***
d	Node index where truck switches driver, zero if no switch

Constraints

alldifferent(s)	Ensure each shipment is shipped exactly once
alldifferent(p)	Ensure each shipment is shipped exactly once
$s_{p_i} = i \; \forall i \in N_s$	Consistency between s and p
$p_{s_i} = i \; \forall i \in N_p$	Consistency between s and p
$s_i = i + x \vee s_{s_i} = i + x$	Ensure that at most two shipments are shipped simultaneously
$p_i = i - x \vee p_{p_i} = i - x$	Ensure that at most two shipments are shipped simultaneously
$r_i = r_{s_i} \; \forall i \in N_s$	Successive visits must be done by same truck
$r_i = r_{p_i} \; \forall i \in N_p$	Successive visits must be done by same truck
$r_{n-k} = rn - m - k = k \; \forall k \in M$	Fix truck to depot
$r_i = r_{i+x} \; \forall i \in P$	Pickup truck and delivery truck must be the same
$t_i + st_i + t_{i,i+x} <= t_{i+x} \; \forall i \in P$	Pickup must be done before delivery
if $u_i = u_{s_i}$ then $t_i + st_i^{fix} + t_{i,s_i} <= t_{s_i}$ else $t_i + st_i^{fix} + st_i^{var} + t_{i,s_i} <= t_{s_i} \; \forall i \in N_s$	Successive visits must be time consistent. Variable service time is not considered when truck stays at the same location.
if $u_i = u_{p_i}$ then $t_{p_i} + st_{p_i}^{fix} + t_{p_i,i} <= t_i$ else $t_{p_i} + st_{p_i}^{fix} + st_{p_i}^{var} + t_{p_i,i} <= t_i \; \forall i \in N_s$	Successive visits must be time consistent. Variable service time is not considered when truck stays at the same location.
$a_i \leq t_i \leq b_i \; \forall i \in N$	Time windows must be respected
if $d_i = 0$ then $t_{min} <= t_{2*x+m+j} - t_{2*x+j} <= t_{max,j}$ else $t_{min} <= t_{d_j} - t_{2*x+j} <= t_{max,j} \wedge t_{min} <= t_{2*x+m+j} t_{d_j} <= t_{max}$	Schedule length restrictions
$0 <= c_i + q_{s_i} = c_{s_i} <= C \; \forall i \in N_s$	Loads must be consistent and within bounds
$c_i = 0 \; \forall i \in E_s \cup E_e$	Loads at depots must be zero
$l_{d_k} == l_{2*x+k}$ if $d_k > 0 \; \forall k \in M$	Driver switches must take place at depot
$r_{d_k} == k$ if $d_k > 0 \; \forall k \in M$	Driver switch node must be on trucks route
disjunctive($[t_i \; \forall i \in H_j \mid s_i \; \forall i \in H_j]$) $\forall j \in H$ [1]	A hub can service one truck at a time
disjunctive($[t_i + s_i \; \forall i \in H_j \mid i_i \; \forall i \in F_j]$) $\forall j \in F$ [1]	Any two trucks cannot leave the same non-hub location within the required idle time

Objective

min $\sum_{i \in E_s \cup P \cup D} dist_{i,s_i} * km\_rate + \sum_{j \in M} (t_{2*x+m+j} - t_{2*x+j}) * hour\_rate$

[01] The *disjunctive(*$x_i \mid y_i \; \forall i \in I$) predicate ensures that no $x_j \; \forall j \in I \setminus \{i\}$ lies in the interval $[x_i, x_i + y_i]$.

2.4.1. Solving the CP model

We attempted to solve this model using an exact solver. The model was coded in the MiniZinc language [38], and the Google OR-Tools solver [21] was used for solving (see section 3.2). We found that this would not yield competitive results quickly, due to the problem size and the several non-linearities in the problem. We did manage to solve very small instances of the problem though, which we come back to in Chapter 7. For solving the full problem, we proceeded with different solving techniques.

2.5. Current practice, limitations and potential

To further motivate this thesis, we outline Picnic's current practice regarding truck planning, demonstrate how it is limiting both practically and in terms of efficiency, and highlight potential gains. As we touched upon before, all shipments are categorised in three groups: Day Outbound, Morning Outbound and Inbound. For each of these three groups, Picnic has a separate planning procedure. These are put together to form one grand daily planning.

2.5.1. Planning Outbounds: FMS

Picnic's current planning procedure for Outbound shipments is the Freight Management System (FMS). FMS takes as input the set of daily Dispatch Frames (DFs) and outputs a feasible truck planning. In fact, FMS is a two-stage procedure which first composes truck loads (shipments) from these DFs, and then schedules these shipments to trucks. FMS operates per FC, and runs separately for Morning Outbounds and Regular Outbounds. The two resulting plannings do not communicate and are thus very unlikely to form nicely joint schedules by themselves. Moreover, FMS does not take into account warehouse opening times. This poses a problem for Morning Outbounds: FMS will schedule these shipments throughout the night, which is operationally infeasible. This is not a problem for Regular Outbounds, which are by construction scheduled during the day.

2.5.2. Planning Inbounds

At the time of writing this thesis, Inbound planning is done manually. The planners have the freedom to plan an Inbound shipment to an Outbound schedule when they see fit, but generally this is a separate planning altogether.

2.5.3. Limitations of the current practice

The current planning procedure has a number of drawbacks:

- Planning per FC is sensible from operational perspective, as many hubs are supplied from the FC closest to it. However, this drastically reduces the combinations of trips that can be made compared to the case of planning all FCs combined.
- Taking the Day Outbounds as a given before scheduling Morning Outbounds is also limiting on the possible combination. Some manual rearranging may occur in practice, but is still limited.
- The number of shipments planned per FC per flow type is small, so chances are that trucks are used inefficiently to make up for some 'remnant' shipment. For instance, when the shipments just cannot be scheduled to three trucks, a fourth is added, but the resulting schedule becomes inefficient.
- Inbounds and Outbounds are planned in separate processes, so there is little synergy among these schedules. Operationally however, there is no reason to separate these flows.
- The large amount of manual planning is labour-intensive and therefore costly.

2.5.4. Expected areas of potential

The presented drawbacks lead us to believe that there is potential in developing an overarching algorithm that can plan all shipments regardless of the flow type they belong to. The gains logically follow from the drawbacks of the current practice presented above:

- Putting Morning and Regular Outbounds and Inbounds together in a single planning instance could result in a much more synergised planning.
- The same goes for putting multiple FCs together in a single instance. Note however that this effect may be limited, since far away FCs are highly unlikely to yield a more efficient planning when put together.
- The need for manual planning should decrease significantly

As with all combinatorial optimisation problems, the increased number of possible combinations comes at a cost. The complexity of the problem grows exponentially in instance size, meaning that it quickly becomes very hard to find good solutions. Add to that the resource constraint we have to take into account, and the problem becomes one that has not been solved before as such in existing literature.

2.6. Data(sets) and adaptations

For running experiments on our algorithms, we have Picnic's data readily available. Outbound shipments follow from FMS, and Inbound data from the 3PLs and suppliers. This data requires some tweaking where information is missing, which we will elaborate on in this section. We then present a set of representative instances we will use to run experiments on, before running our algorithms on full day instances.

2.6.1. Outbound data

The astute reader may have noticed that we did not consider composing shipments before the previous section, but that it is in fact an integral part of the current FMS planning procedure. Not only does FMS put single Dispatch Frames together in shipments, it can also determine the picking line order to optimise for earliest departure time. Shipment picking is assumed to be done sequentially and is limited by FC workforce. In case we want to pick a shipment earlier, another has to be picked at a later time. For the scope of this thesis however, we separate shipment composition and shipment planning for two reasons:

1. We intend to build a general planning algorithm regardless of whether a shipment is Inbound or Outbound. Since composing Inbounds is also done by a separate process, it makes sense to separate Outbound shipment composition from planning.
2. By considering significantly bigger instances than FMS, the complexity of the problem becomes such that integrating shipment composition into the grand scheme will make it very unlikely to solve such instances (close to) optimally.

This means that we have to determine the shipment composition and earliest departure times a priori. We take shipment composition directly from FMS, but FMS does not explicitly give us earliest departure times. Instead, we have to come up with a rule to do so. This goes as follows for Regular Outbounds:

1. We know for each shipments the latest arrival time at the Hub. By subtracting the driving time from FC to Hub, we know the latest departure time from the FC. We order all shipments by increasing latest departure time, such that the shipment with the earliest *earliest departure time* is picked first.
2. From FC operations, we know that all orders are picked between $picking\_start\_time$ and $picking\_end\_time$. This gives an average picking time per DF of

$$picking\_time_{avg} = (picking\_end\_time - picking\_start\_time)/\#DFs$$

3. Given the order of shipments in step 1, we can determine the time required to pick each shipment. Since we assume shipments are picked sequentially, this gives us an earliest departure time for each shipment.
4. For operational reasons, a shipment cannot depart earlier 8:30H. In case step 3 yields a departure time earlier than that, it is overridden to 8:30H.

Ordering on latest departure time leads to rather evenly distributed time window lengths. In fact, this guarantees that the minimum time window size among the shipments is maximised. This is does not say anything about how easy it is to thereafter schedule the shipments, but seems sensible when we have no further knowledge on how the time window length affects the scheduling process.
Since Morning Outbounds are picked in the late afternoon of the day before, we do not have to consider such a rule regarding earliest departure times. We assume that these will always be ready early enough. On the other hand, the earliest delivery time, and hence earliest departure time, is limited by the time that the Hub opens in the morning, which is usually around 6:00H.

2.6.2. Inbound data

The Inbound data requires less pre-processing. We can use the time windows provided to us as is. It does however occur that two shipments with the same origin have the same fixed departure times, which is in direct violation with the resource constraint. In these cases, we give the shipments a 30 minute time window so that the algorithm can still impose the constraint. This is acceptable from a business perspective.

A second point is that shipment sizes are often undefined. Picnic assumes that Inbound shipments will never be combined with other shipments, in which case size is not an issue. Only the number of shipments going from location A to location B is known. We on the other hand do want to find out if combining shipments is beneficial when the opportunity is there. From a business perspective, it is interesting to see what routes are created when we do allow this. We therefore defined the following rule:

1. When there are n shipments with the same origin A and destination B on a given day, we assume that $n-1$ shipments are 100% full. Otherwise it is very likely that a more efficient allocation of the cargo exists.
2. The n^{th} shipment can be of any size. We make the simple assumption that it is 50% full.

A final remark has to be made about partitioning Inbounds over FCs. When an Inbound shipment is between an FC and another non-FC location, we assign it to the set of shipments of that FC. When an Inbound shipment is between FCs (usually around 2 per day), we simply assign it to the lexicographically lowest FC. When an Inbound is between two non-FCs, we assign it to FC-E, which makes most sense from a geographical perspective.

2.6.3. Representative instances

To be able to show robustness in our results, we put together a diverse set of test instances derived from operational data. These instances are always a subset of a set of shipments from a particular day, such that we choose a subset of FCs, and take all shipments belonging to those FCs on that day. The choices for these instances were made based on the following rules:

- Diversity in terms of instance size
- Diversity in the FCs
- Diversity in flow type (= Morning Outbounds / Regular Outbounds / Inbounds)

Table 2.1: Test instances. M=Morning Outbound, R=Regular Outbound, I=Inbound.

Name	FCs	Flows	#shipments
fcG_r_18	G	R	18
fcG_mr_26	G	MR	26
fcFG_r_37	F, G	R	37
fcFG_mr_56	F, G	MR	56
fcACE_r_50	A, C, E	R	50
fcACE_mr_67	A, C, E	MR	67
fcACEFG_r_87	A, C, E, F, G	R	87
fcACEFG_mr_123	A, C, E, F, G	MR	123
fcB_ri_32	B	RI	32
fcB_mri_44	B	MRI	44
fcDE_ri_49	D, E	RI	49
fcDE_mri_61	D, E	MRI	61
fcBDE_ri_81	B, D, E	RI	81
fcBDE_mri_105	B, D, E	MRI	105

We derived these instances from Picnic data from relatively busy days. Hence we expect more shipments to the same Hubs, and thus a stronger influence of the resource constraint. The instances are displayed in Table 2.1. The instance names are all in the format *fc[fc_letters]_[flows]_[instance_size]*. Note that some instances are subsets of other instances,e.g. instances *fcFG_mr_56* and *fcACE_mr_87* together form instance *fcACEFG_mr_123*. This allows us to observe how two sub-instances are solved compared to their union.

3

Related work

This chapter provides a comprehensive study of related work. Section 3.1 walks through the historical literature of routing problems relevant to our case. We start at the very beginning with the Travelling Salesman Problem, moving on to the famous Vehicle Routing Problem and then the more general Pickup and Delivery Problem. We then explore literature related to the specific constraints we encounter in our problem. We find examples of what aspects of our problem have been solved before, and what aspects are new or unsolved.
In the second part, we dive into the relevant solution methods, both exact and heuristic. We explore the paradigm of Constraint Programming as an exact solving method in section 3.2, then continue to inexact methods such as (Adaptive) Large Neighbourhood Search and Local Search in 3.3, and discuss the realm of matheuristics in section 3.4. We end the chapter by summarising the most relevant papers in section 3.5.

3.1. Routing problems

3.1.1. TSP, VRP and PDP

Routing problems are as old as the road to Rome, as the Dutch would say. Along those lines, the Travelling Salesman Problem (*TSP*) is the Romulus of them all. Conceived a little later, in the 19th century, and formalised in 1930 [34], the TSP laid the foundation of what would become one of the most studied problems in combinatorial optimisation. The goal is for a salesman to travel to all cities to sell his goods, then return home, covering the smallest possible distance. As elegantly simple as this sounds, solving this problem is not trivial. In fact, given that there are n cities, the number of possible solutions is in the order of $n!$. The TSP is NP-complete, meaning that it is believed to be unsolvable in polynomial time.

Hardly 30 years later, Dantzig and Ramser [10] extended the TSP to what they called the Truck Dispatching Problem, later dubbed Vehicle Routing Problem (*VRP*). Rather than a single salesman, or vehicle for that matter, they generalise to a multi-vehicle setting. How can multiple trucks be routed such that all locations (cities) are visited, and distance is minimized? TSP is reduced to VRP when only one truck is used, and therefore VRP is NP-hard.

The VRP is among others applicable to settings where one needs to deliver goods to customers after doing one single pickup up front, which can be disregarded when solving the problem. An example could be a parcel delivery service. A subsequent generalisation extends VRP to the setting where we do not just visit locations to deliver goods after a single pickup, but integrate the pickups per delivery in the problem. This may be necessary for a variety of reasons, such as capacity constraints

or the lack of a single pickup location. In this case, for each pickup-and-delivery, or *request*, both the pickup location and the delivery location are taken into account, effectively doubling the instance size. Constraints that follow naturally from this definition is that a pickup node must be visited before its corresponding delivery node, and both nodes must be visited by the same vehicle. This Pickup and Delivery Problem (*PDP*) was first introduced by Dumas et al. [14], and generalised further by Savelsbergh and Sol (1995) [49].

3.1.2. Common extensions to routing problems

Braekers et al. (2016) [4] and later Elshaer et al. [16] have surveyed the large realm of VRPs and their solution methods. The best-known survey on PDPs dates back further to Parragh et al. (2008) [42]. This section describes the most common extensions to these problems and refers to relevant papers. Some of these extensions turn out to be relevant to our problem also.

Time Windows

An issue frequently encountered in delivering goods, or any routing problem for that matter, is that the delivery cannot be done just at anytime. Often, delivery (and pickup) times are bounded by other parts of the logistic operation, or customers can specify a time window (TW) in which they want to be visited. A time window is specified by an earliest and latest pickup/delivery time. Essentially, the visit needs to be performed somewhere in this range. Cordeau et al. [7] surveyed VRPs with time windows extensively. Most research treats problems with hard time windows, meaning that it is impossible to service customers outside of their time windows. Others concern themselves with soft time windows, where a penalty is incurred for service outside of the time window [31].

Minimizing duration

As discussed, the original routing problems aim to minimize just the distance covered by the vehicles. Arguably more important is the time spent by the drivers. Minimizing total route duration was first introduced by Savelsbergh [48]. While this would introduce more complexity to the problem, Savelsbergh presents the concept of *forward slack* to efficiently search the solution space and keep track of minimum duration.

Capacity constraints

As soon as the delivered cargo becomes of considerable size compared to the transporting vehicle, capacity constraints become relevant. Capacitated VRPs, or CVRPs, are among the oldest extensions to VRPs since their introduction [43] For VRPs, this influences which/how many deliveries a vehicle can carry upon leaving its depot. For PDPs, this affects which requests may possibly be picked up in succession, and when a request must first be delivered in order to make room for new pickups. On the other side of the spectrum we find full-truckload PDPs, where vehicles only have unit capacity, and no two shipments can be combined. Desrosiers et al. [12] were among the first to introduce this problem formally.

Multiple depots

Classical VRP assumes that there exists a single depot where all trucks depart at the start of the day, and return at the end. In many cases however, one is dealing with multiple depots. Sometimes, this means that each depot has a finite number of trucks that all depart from and return to this depot [53, 33, 8]. It could also occur that vehicles can be assigned to depots arbitrarily, that vehicles can return to any depot, or do not have to return to a depot at all (the Open VRP) [47].

Driver regulations

For routing problems with a time span of at least one day, driver regulations may factor into the equation. This entails ensuring that maximum driving times, working times and break times are respected. Among others, Goel and Gruhn [19] and Kok et al. [30] look at such problems extensively, in relation to international regulations.

Heterogeneous fleet
A final extension we discuss in this paragraph is that of the heterogeneous fleet. At times, the vehicles at hand may differ in terms of size, range or some other attribute. Some vehicles may only transport a specific type of good, or may be unsuitable for some good. In terms of Picnic, one could consider trucks designated to transport cooled or frozen products only. Heterogeneous fleets for VRPs were introduced by Golden et al. [20].

3.1.3. Resource constraints in routing problems

Apart from the extensions treated in the previous section, we discuss resource constraints in routing problems. We devote a separate section to this constraint, because in contrast to those mentioned before, this constraint has a major impact on problem complexity. Also, as pointed out in Chapter 2, this constraint plays a central role in our research.

A resource constraint implies the presence of some entity that can only be used by a limited number of trucks at any given time, or a central consumable entity of which a limited quantity is available. Fundamental to resource constraints is that it makes the vehicles and their routes dependent on one another. In many cases, resource constraints induce temporal constraints. No two vehicles can dock to the same dock at the same time, no two electric vehicles can charge at the same charging station simultaneously, and so on. This means that service times of one vehicle are affected by those of others. In fact, the validity and/or cost associated with a vehicle can no longer be determined from its route itself, but is rather a function of (a subset of) all routes. Drexl [13] has extensively surveyed synchronisation, or inter-route constraints, in routing problems, of which resource synchronisation is a specific variant. Other variants include task, operation, load and movement synchronisation. The reader is deferred to this survey for more details on those variants. In an earlier paper, Hempsch and Irnich [25] define *resource extension functions* as a general framework for dealing with resource constraints for VRPs. They provide a local search procedure to solve their problem.

Hachemi et al. [15] and Grimault et al. [24] investigate the log truck scheduling problem with resource constraints on all locations, such that only one truck can be serviced at any point in time per location. Hachemi et al. employ a constraint programming model and a constraint-based local search, whereas Grimault et al. develop an ALNS procedure. Hojabri et al. [26] investigate a delivery problem with synchronised visits and employ an ALNS also, but perform insertion using a CP model. Sarasola and Doerner [46] also treat a PDP with resource synchronisation at the customer, and solve this using an ALNS. Grangier et al. [22] and Grangier et al. [23] treat a VRP with cross-docking, where the cross dock is resource constrained. They solve a relaxed version of the problem, without the constraint, and later impose the constraint using a CP model. They also add a second phase where they construct solutions from earlier obtained routes using a set partitioning and matching procedure.

3.2. Constraint Programming

Constraint programming (CP) is, according to [45], “a powerful paradigm for solving combinatorial search problems that draws on a wide range of techniques from artificial intelligence, computer science, and operations research”. The user defines a set of variables, domains and constraints to describe a problem and the characteristics of its feasible solutions. The user does not define the solution method: this is left to solvers specifically designed to solve problems written in a CP language. Such solvers generally rely on techniques such as backtracking, constraint propagation and branch-and-bound.

CP can be compared to (M)ILP [27], in the sense that the user defines the problem, and solving it is left to a (black-box) solver. But while MILPs are formulated purely mathematically, the CP language is more high-level, and better readable. Decision variables can be (arrays of) any values, often bounded by corresponding user-defined domains. While constraints can be arithmetic, they can also be expressed using logical predicates. A well-known example is the *AllDifferent* constraint,

which dictates that each element of an array should be unique. Because of this high-level language, complex models can usually be described much more concisely than MILPs, and are also easier to understand.

3.2.1. Constraint Programming for VRPs

Since VRPs are quintessential combinatorial optimisation problems, they are very suitable to be written in a CP language. Shaw (1998) [51] was the first to formally define such a model. Kilby provided a very extensive tutorial on CP for VRPs in 2013 [28], which discusses CP for basic VRP and all its extensions, as well as those for which CP is not suitable. The core idea revolves around defining an array of length equal to the number of nodes. Each index represents a node, and the value at the index represents the index of the node succeeding this node in its route. Observe that this representation is much more concise (linear in the number of nodes) compared to an edge-based representation often encountered in ILPs (quadratic in the number of nodes). We refer the reader to this tutorial for further details on this formulation.

3.2.2. MiniZinc and OR-Tools

Multiple CP languages exist, of which the MiniZinc [38] language is an actively-developed open source modelling language. According to their own annual challenge [39], the state-of-the-art solver for problems in this language is the Google OR-Tools solver [21]. The solver is especially effective on multiple cores, and is able to find effective search strategies on its own by enabling *free search*. For this reason, we attempted to solve problem instances with this language, solver and settings as part of this thesis.

3.3. Heuristic solution methods

Over the decades, these many VRP descendants have also brought about a large variety of solution methods. The two main flavours are exact and heuristic solution methods. The survey papers mentioned before treat all of these methods extensively. This section goes over the most relevant heuristic methods.

3.3.1. Local search

Local search operators are perhaps the most common type of heuristics used in non-exact algorithms for solving VRPs. Braysy and Gendreau [5] have extensively surveyed the realm of such heuristics. Local search is based on the concepts of *neighbourhoods*: Given a solution and a local search operator, the neighbourhood of a solution is the set of solutions that can be obtained by applying the given operator in all possible ways. A strategy could then be to look at all possible solutions in the set, choose the best one, and then apply the next operator. A quicker, but less exact strategy could be to accept the first improving move once it is found. These strategies are known as *best-accept* and *first-accept* respectively. At some point, one will run into the situation where no improving moves can be found. A so-called *local optimum* is reached. Generally, algorithms will have a mechanism (a *metaheuristic*) to escape local optima, for instance by perturbing the solution (often making it worse).

We will first demonstrate the most well-known local search operators, and then go over a few common local search metaheuristics.

Local search operators

We will describe the relocate, exchange, crossover, k-Opt and Or-Opt operators, since these are the most common. We illustrate the moves from both edge-exchange and node-exchange perspective. The former is more widely accepted, while the latter is at times more suitable to our problem. Chapter 5 describes these operators in context of our problem.

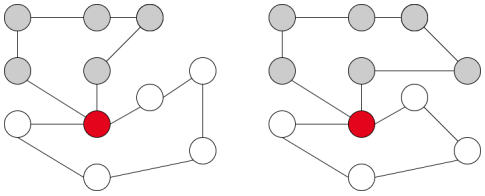


Figure 3.1: Relocate

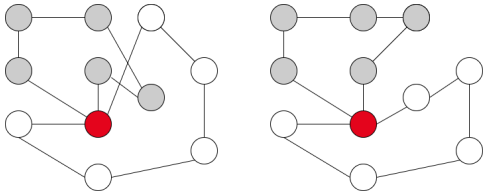


Figure 3.2: Exchange

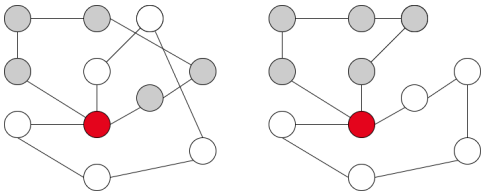


Figure 3.3: 2-opt*

- **Relocate**
  The *relocate* operator attempts to move a node from its current route and position to another one. It may be moved to another position in the same route, in which case *relocate* works as an *intra-route* operator. Otherwise, when moved to a different route, the operation is of *inter-route* nature. *Relocate* causes a total of three edges to be replaced (Figure 3.1).
- **Exchange**
  Similar to *relocate* is the *exchange* operator. In fact, *exchange* performs two *relocates* simultaneously. Two nodes are removed from their respective routes (possibly the same route), and reinserted in the position of the other node. This could be beneficial over two consecutive *relocate*s when neither *relocate* move is improving, or even feasible, by itself. *Exchange* effectively replaces four edges, as can be seen in Figure 3.2.
- **k-Opt and Or-Opt**
  The *k-Opt* operator is an intra-route operator that replaces k edges in a route in order to find a better one. *2-opt* is the simplest form, where two edges are replaced. The sequence of nodes between these two edges will consequently be reversed in the route. *k-opt* with $k > 2$ gives rise to more complex restructurings, which often also contain such reversals. Evaluating all *k-Opt* moves takes $O(n_r^k)$ time and hence scales badly in size, especially for longer routes. *Or-Opt* [41] is an inter-route variation to *2-Opt*, where a sequence of nodes is removed from one route and inserted into another. The order of the sequence is preserved, not reversed as with *2-Opt*. Note that relocate is a special case of *Or-Opt* where the length of the moved sequence is 1.
- **Crossover and 2-Opt***
  Combining *Or-Opt* and *exchange* yields the *crossover* operator, where two sequences of nodes are swapped between their routes. When a route contains n_r nodes, it naturally has n_r^2 sequences of nodes (n_r possible start nodes and n_r possible end nodes). Consequently, *crossover* scales more badly than both *Or-Opt* and *exchange*. A special case is the *2-Opt** operator, which exchanges the tails of two routes. The number of 'swappable' sequences reduces to n_r again, and this heuristic is a lot more manageable because of this. This *2-Opt** is illustrated in Figure 3.3, where only two edges are replaced.

LS metaheuristics

The grand procedure in which local search operators, or any operators for that matter, are applied, is usually referred to as a metaheuristic. These dictate the operator choice for each iteration, when to accept a newly proposed solution, how to escape local minima and so on. We highlight Tabu Search, Simulated Annealing and Guided Local Search in this section.

- **Tabu Search**
  Tabu search essentially forms a set of rules by which local minima are escaped and a larger part of the search space can be explored. As with straightforward local search strategies, improving moves are always accepted by *first-accept* or *best-accept* strategies. However, when no improving moves exist, tabu search allows to accept a worsening move to escape the local optimum. Next to that, a list of recently explored solutions is maintained. This prevents the algorithm to revisit such solutions. Without this mechanism, the algorithm may instantly move back to the local optimum it just moved out of. The length of said list, or in other words, the size of the memory, influences this procedure. A larger memory usually leads to more diverse search.
- **Guided Local Search**
  Guided local search (GLS) [54] also attempts to escape local minima, but rather does so by changing the cost function. Whenever a local optimum is reached, GLS penalises features of this solution such that they are less likely to appear in subsequently found solutions. For VRPs, these features are usually the edges present in the solution. Features that remain present in locally optimal solutions (and are likely to be desirable) are penalised less heavily upon each appearance. Newly found solutions under the penalty function should be evaluated without penalty to see if these are actually improved solutions.
- **Simulated Annealing**
  A final well-known metaheuristic is Simulated Annealing (SA) [29]. It is inspired by the process of heating and then cooling a material in a controlled manner (annealing) in order to alter its physical characteristics. Similarly, the algorithm starts with a high 'temperature', meaning that worse solutions are often accepted. Rather than always finding a local optimum first, any worsening move will be accepted with a certain probability. The probability is usually dependent on how much worse the proposed solution is compared to the current one. A slightly worse solution has a higher probability of being accepted than a much worse one. As time progresses, the algorithm is 'cooled down' by some *cooling rate*, such that fewer and fewer worsening solutions are accepted. The idea is that the algorithm has moved to the best parts of the search space by this time.

3.3.2. Large neighbourhood search

Shaw [50] introduced Large Neighbourhood Search (LNS) for VRPs for the first time in 1997. He considered this a variant of local search, but nowadays LNS is viewed as an entirely separate group of algorithms. Especially over the last years, LNS has received increasing attention for solving VRPs [16].

Given a feasible solution to start with, LNS iteratively removes and reinserts requests from and into this solution in order to obtain new, and hopefully better ones. More specifically, one defines so-called removal and insertion heuristics, or operators, which heuristically determine which requests to remove/insert. A multitude of such heuristics have been crafted over the years, the most important of which we outline below:

Removal heuristics

Along with the concept of LNS, Shaw was the first to introduce removal heuristics. Ever since, many more have been developed, often aimed at specific characteristics of a problem. We list the most common ones below.

- **Random Removal**. The simplest of all removal heuristics, where requests are removed at random. This is especially effective from a diversity point of view.
- **Worst Removal**. Worst removal [44] checks for each request by how much the objective decreases upon removing it. It will then remove the one which does so the most. It continues in

this fashion until a satisfactory number of requests are removed.

- **Shaw Removal**. Named after its founding father, Shaw removal tries to remove similar requests. Requests can be similar in terms of geographical location, time window or size. The similarity function is usually a weighted function of the three. For details, the reader is referred to the original paper.
- **Related Route Removal**. Attempts to remove routes that are related. This relation is usually in terms of geographical proximity. Removing routes lying close together may allow for more possibilities upon recombination. A random route is removed first, and while more requests should be removed, a route close to it is removed. (Related) route removal is frequently used in other research [11, 35, 24].
- **Related Location Removal**. Rather than removing routes, a location removal heuristic removes all requests with the same origin or destination. A subsequent nearby location can be selected when more nodes need to be removed. A variant of (related) location removal is presented among others in [11, 35, 24] also.

Insertion heuristics
Early LNS papers such as Shaw and also Bent and Van Hentenryck [2] employ relatively complicated branch-and-bound techniques. Ropke and Pisinger [44] opt for simpler heuristics, which are outlined below. The list of insertion heuristics has grown through subsequent research.

- **Cheapest (Greedy) Insertion**. Attempts to insert each removed request in every route, and inserts the one which makes the objective increase by the least amount. Some of these values will need to be recalculated before the next insertion, since they may have become outdated. An insertion may even no longer be possible after the previous one.
- **Regret Insertion**. Regret insertion is essentially a greedy strategy also, but does look ahead. It compares the cost of inserting a request in the best route to the cost of inserting it in the next-best route. The request for which this difference is largest will be inserted first. We refer the reader to Ropke and Pisinger (2006) for details. Again, values have to be recalculated after each insert.
- **Insertion with noise**. As an extension to other insertion heuristics, noise may be added to the insertion costs determined by the heuristic. This should favour diversity in the search by preventing the same requests to be inserted first all the time.

Whenever one uses multiple heuristics in either category, one of either heuristics is picked at random for each iteration, in order to promote diversity in the search.

Another important parameter to set is the number of requests to remove in each iteration. This is usually a fixed percentage of the total number of requests, or a distribution (usually uniform) from which a percentage is drawn. Several papers that implement some form of an LNS investigate the optimal value for this parameter in their experiments [44, 24].

A major development in the LNS area was made by Ropke and Pisinger, who introduced Adaptive Large Neighbourhood Search (ALNS). Rather than selecting a heuristic using a fixed probability distribution, they came up with a procedure to adjust the probabilities per heuristics based on how successful they turn out to be. For each heuristic, a score is maintained which indicates its performance. Heuristics finding new and better solutions have their scores increased, which translate to higher weights and hence a higher probability of being selected in subsequent iterations. To date, ALNS is seen as one of the most successful heuristic algorithms to solve all kinds of VRPs. A state-of-the-art algorithm for solving PDPs is given by Curtois et al. [9]. They combine local search and LNS procedures, together with a guided ejection search and hold several best-known solutions to benchmark instances at the moment of writing.

3.4. Matheuristics

A final type of algorithms we discuss are so-called *matheuristics* [3]. This class of algorithms incorporate an exact solving component in a larger heuristic scheme, and has gained popularity in recent years. The exact component can be an algorithm, or even a solver solving a subproblem formulated as a MIP or a CP. In fact, the LNS algorithms of both Shaw and Bent and Van Hentenryck can be considered matheuristics of the former type, as their reinsertion is based on a branch-and-bound algorithm. In a similar vein, Hojabri at el. perform reinsertion using a CP model [26]. Grangier et al. [23] use a CP for imposing the resource constraint, after solving a relaxed version of their problem with an LNS procedure. They follow this up by a set partitioning and matching procedure to further improve their solution, which is a second exact component in the algorithm. Kramer et al. [32] employ a similar procedure for their pollution routing problem. Froger et al. [17] also first solve a relaxed version of their electric VRP with capacitated charging stations, and then use a CP model to find solution from routes generated by the previous step, while taking the resource constraint into account. For further research on matheuristics for VRPs we refer the reader to the survey by Archetti et al. [1].

3.5. Most relevant papers

We end this chapter by highlighting aforementioned papers which are most relevant to our problem. Where applicable, we compare characteristics of their problems to ours.

- **Ropke and Pisinger (2006)** lay the foundation for Adaptive Large Neighbourhood Search, building on Shaw's LNS procedure. They introduce insertion heuristics and the adaptive heuristic selection that have been used widely since.
- **Grimault et al. (2017)** treat a problem most similar to ours in literature at the moment of writing. They consider a PDP where no two trucks can visit any location simultaneously, and provide a graph-based ALNS to solve this. Shipment combining is left out altogether, so they do not have to consider capacity constraints. They show that the increased complexity makes this problem substantially harder to solve than a PDP without such an inter-route constraint. The largest instance they solve is of size 85, which is significantly smaller than our instances. They have runtimes up to an hour and implemented the algorithm in C++.
- **Curtois et al. (2018)** hold at the moment of writing this thesis several records for benchmark instances of the PDP. They propose a multi-stage LNS+LS+Guided Ejection Search by which they effectively obtain low-cost routes with a low number of vehicles.
- **Sarasola and Doerner (2020)** also treat a PDP with resource constraints, though only at the customer. Next to preventing subsequent deliveries from overlapping, they also set an upper bound on the time between such deliveries. Customers have 2-3 deliveries on average, which is slightly less than our hubs, but far less than our FCs/DCs.
- **Grangier et al. (2021)** treat a PDP with cross-docking, which is a resource constraint also, but at a single location only. They solve the problem without the dock constraint using an LNS, and then check feasibility with respect to the constraint using either a CP model or a scheduling heuristic. This allows for the LNS to be faster compared to an LNS designed on the constrained problem.

4

An ALNS approach

In section 3.3.2 we discussed that ALNS techniques have shown to be effective when solving pickup and delivery problems, which is why we choose to apply them in this thesis. Having explained the general concept of ALNS for vehicle routing problems in Chapter 3, we dive into the adaptations required to make this work for our problem.

As explained, ALNS takes a feasible solution and iteratively partly destroys and repairs it. In the case of VRPs, this means removing a set of requests such that a partial solution is left, and removed requests are stored in the request bank. These requests are then inserted back into the partial solution to obtain a new feasible solution. Of course, the goal is to find better solutions by removing requests that are costly, and reinserting these in a more efficient manner. It is crucial to this procedure that the (partial) solution is feasible at all times. Whenever a (partial) solution becomes infeasible, there is no guarantee that a subsequent removal/insertion will render it feasible again.

We first devote specific attention to how we impose the resource constraint in section 4.1, and the global effect this constraint has on guaranteeing feasibility. We discuss the destroy and repair heuristics we employ in section 4.2. Constructing initial solutions is discussed in section 4.3, while section 4.4 elaborates on adaptations we made to the cost function, for dealing with multiple depots and driver switches, and some adaptations to the meta-heuristics described by Ropke and Pisinger (2006). We will refer to this paper as *R&P* for brevity throughout the rest of this thesis. Section 4.5 summarises the earlier sections, and in section 4.7 we evaluate the performance of the ALNS.

4.1. Adaptations to the resource constraint

Since we intend to minimise route duration, we have to keep track of it. Savelsbergh [48] showed how one can keep track of earliest start service times, latest start service times and slack times to achieve this, while ensuring that checking feasibility is efficient. He also showed how shortest route duration can be derived from these variables without increasing computational complexity. Also, without resource constraints, inserting/removing a request in one route will leave all other routes unaffected. Therefore, updating the costs of subsequent insertions requires only recalculating those for the affected route. Masson [37] and later Grimault et al. [24] (referred to as *Grimault* throughout this chapter) have extended this model to the PDPTW with resource constraints, where these variables are maintained for both routes and resources. Requests are represented by their two nodes in a graph, with edges connecting nodes that are subsequent either in a route, or on a resource (Figure 4.1). This usually leads to a connected graph where changing said variables for one node impacts an arbitrary number of other nodes. In other words, inserting or removing a node

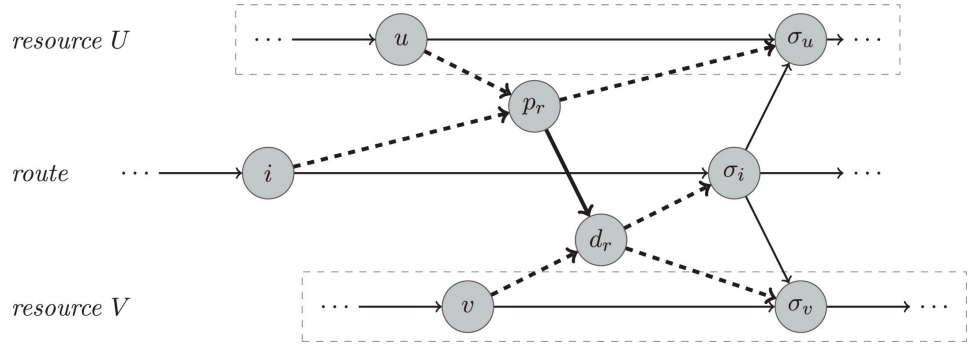


Figure 4.1: The graph structure used by Grimault for efficient feasibility checking (from Grimault et al. [24])

requires us to recalculate the values of these variables for all other nodes.

To circumvent this problem, we opt for an alternative approach. We developed this procedure ourselves, but found later that Sarasola and Doerner [46] employ one that looks rather similar. We return to this in the discussion in Chapter 8. Upon inserting a node in a route, we recalculate the optimal start service time for each node in this route. This is done as follows (see also Algorithm 1):

1. Iteratively schedule each node as late as possible, starting from the last (delivery) node in the route.

2. Fix the first (pickup) node after the depot at this latest time

3. Iteratively schedule each node after the first as early as possible

Algorithm 1 Calculating start service times

function CalculateStartServiceTimes(r)
  for each node n in route r, in reverse order **do**
    $s_n \leftarrow$ succeeding node of n in r
    $st_n \leftarrow$ service time of n
    $t_{n,s_n} \leftarrow$ driving time from n to s_n
    $l_n \leftarrow$ latest service time of n (=upper bound TW)
    $t_n \leftarrow \min(t_{s_n} - st_n - t_{n,s_n}\ ,\ l_n)$
    if `GetStartServiceTimeUnderRC(`n`, asap=False)` returns False **then**
      Revert latest insertion in r
      return
  for each node n in r except the first, in regular order **do** ▷ By not considering the first node, we keep it fixed at its latest possible start time
    $p_n \leftarrow$ preceding node of n in r
    $st_{p_n} \leftarrow$ service time of p_n
    $t_{p_n,n} \leftarrow$ driving time from n to s_n
    $e_n \leftarrow$ earliest service time of n (=lower bound TW)
    $t_n \leftarrow \max(e_n, t_{p_n} + st_{p_n} + t_{p_n,n})$
    if `GetStartServiceTimeUnderRC(`n`, asap=True)` returns False **then**
      Revert latest insertion in r
      return
  return

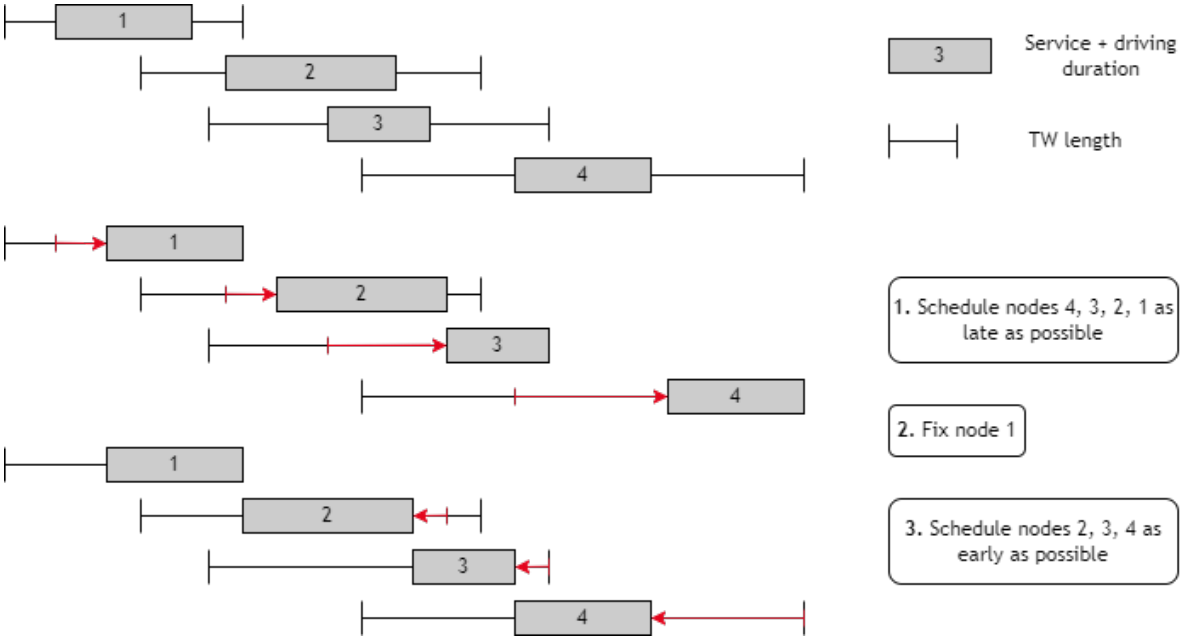


Figure 4.2: An example of calculating start service times for a route of 4 nodes

Figure 4.2 shows a graphical example. This procedure guarantees two properties:

1. The resulting schedule is as short as possible

2. The resulting schedule starts as late as possible

The second property follows directly from the fact that all nodes are scheduled as late as possible initially. The first property follows from the fact that the first node is scheduled as late as possible, and the last node as early as possible given the as-early-as-possible scheduling of all but the first node. The first guarantee is important because we want to minimize costs, which follows directly from minimizing duration. The second guarantee is desirable because it maximizes the chances of a schedule starting later than 5:15H, hence allowing a longer maximum duration.

While this procedure has a higher runtime complexity than Savelsbergh's [48], it sets us up to incorporate the resource constraint. Upon inserting a node in a route, we have a start service time value for all nodes in all other routes, which we keep fixed. At the same time, we keep for each location (resource) a list of its corresponding nodes sorted by their start service time. Upon calculating start service times for the new route, the `GetStartServiceTimeUnderRC` function (Algorithm 2) is incorporated in Algorithm 1 (lines 8 and 17). This function takes as an input node n along with its proposed start (end) service time, and checks for each other node n' on the same resource in turn whether the proposed time for n interferes with the scheduling of n'. If it does, node n is scheduled earlier (later) such that it no longer interferes with n'. Figure 4.3 illustrates this procedure for a single node and resource. Next, the subsequent node on the resource is checked, until either a feasible time for n is found, or it is determined that no feasible time exists. In the latter case, the pending insertion has to be reverted. The function also takes a boolean $asap$ that states whether the start service time may only be decreased or increased. Algorithm 2 describes this procedure for $asap = False$, but the procedure for $asap = True$ follows naturally from this.

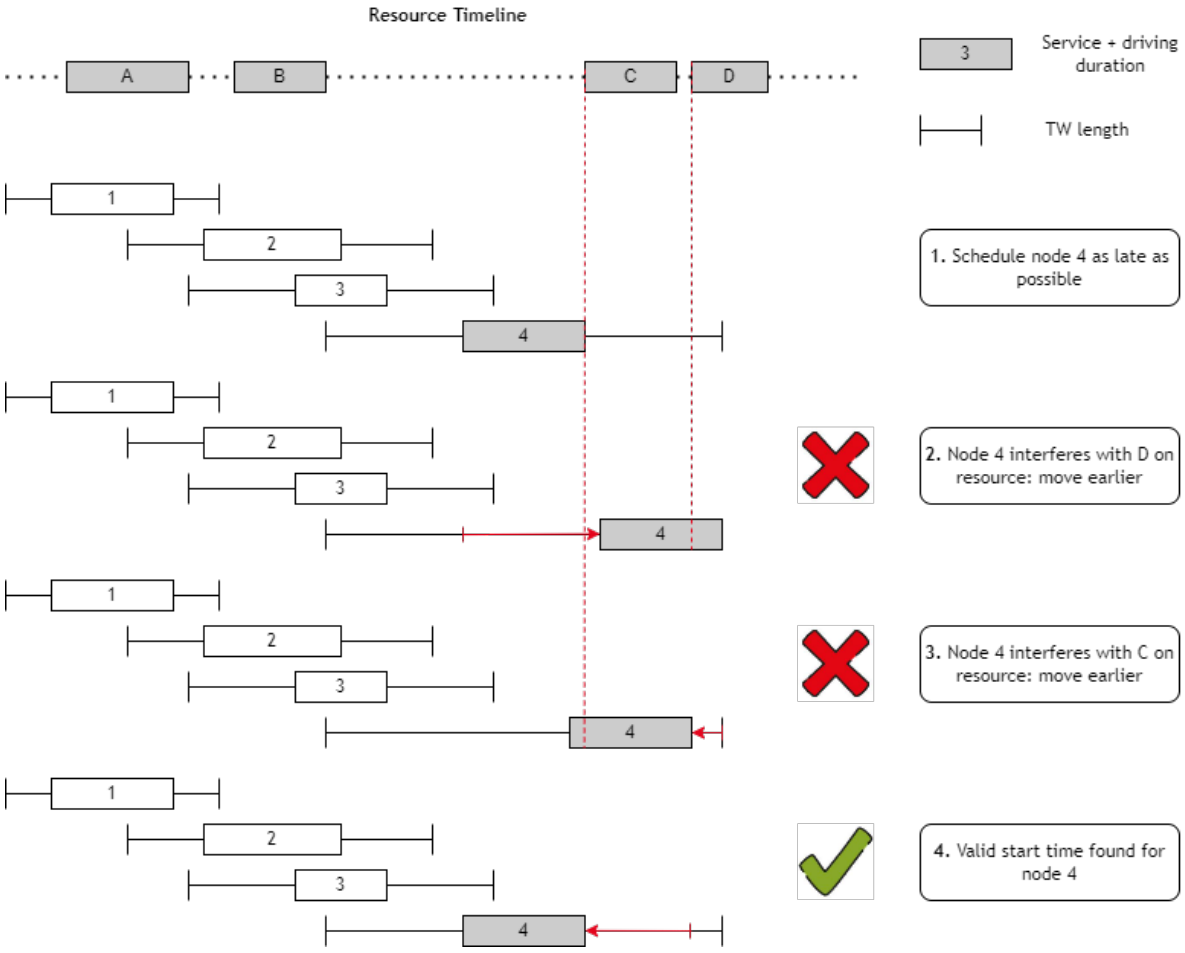


Figure 4.3: An example of adjusting the start service time for a node under the resource constraint.

Upon the introduction of the resource constraint, we need to reconsider the properties we guaranteed before. The second guarantee still holds, as all nodes are still scheduled as late as possible in step 1. Step 3 moves nodes as far back in time as possible, but may be hindered by other nodes scheduled on the same resource. Implicitly, this yields a scheduling of nodes on the resources. The first property still holds given the schedule we obtained, but a shorter route duration might be possible if we move the first node backwards in time and reschedule some nodes on their resource. Since checking all such options comes at the cost of increased complexity, we do not consider such possibilities.

4.1.1. Theoretical comparison to Grimault's procedure

This thesis does not empirically compare our procedure to Grimault's, but below we argue why we favour ours on four relevant parts. The actual comparison is left for future work.

- **Checking insertion feasibility**
  Extending Masson's approach [37], Grimault checks feasibility of inserting a request in a specific position in a route *and* as specific position on a resource in constant time. We can check feasibility for a position in a route also in constant time, but when this check passes, we require worst case $O(k * |r|)$ time to check resource feasibility. Here, k denotes the maximum number of requests per resource, and $|r|$ denotes the route length, i.e., the number of nodes in the route. Note that this does give us the best position on the resource for the request. Grimault does need to check all resource positions (linear time) to obtain this result.

Algorithm 2 Adjusting start service times under the resource constraint

```
1: function GetStartServiceTimeUnderRC(n, asap=False)   ▷ if asap is True, we may only increase t_n, otherwise we may only decrease it
2:     res ← resource of n
3:     δ ← unavailability duration
4:     for node n' on res in reversed order do
5:         if t_n > l_n then
6:             return False                              ▷ n cannot be scheduled on res within its TW, so route is invalid
7:         else if t_n + δ ≤ t_n' then
8:             continue                                  ▷ n' is scheduled after n and does not interfere, continue to next node
9:         else if t_n + δ > t_n' then
10:            break                                     ▷ n' is scheduled before n and does not interfere, t_n is feasible
11:        else
12:            t_n ← t_n' − δ                            ▷ Make sure n fits right before n' on res
13:    return True                                       ▷ A valid t_n was found
```

- **Calculating insertion cost**
  Having established insertion feasibility, Grimault requires $O(n^3)$ time to calculate the new solution cost, required to know insertion cost. In our procedure, the insertion feasibility check, while more computationally expensive by itself, already yields the insertion cost. This part of our procedure is therefore much less time consuming. One question that stands, is how often an insertion turns out infeasible on a resource, when deemed feasible from the route perspective. While we have not done an extensive analysis, we have seen that this happens far less than 50% of the time. Note that this may be highly use case-specific.
- **Recalculating obsolete insertion costs**
  Very often, a repair heuristic is cost-based, e.g. doing the cheapest insert possible. For each route and unplaced request, the value for this metric is stored in a matrix. After each insertion, this matrix has to be updated as some insertions that were possible before now have a different cost, or may no longer be possible at all. Consider the route that most recently had a request inserted as the *last-changed route*. Without inter-route constraints, only the matrix entries belonging to the last-changed route have to be updated, since routes are not interdependent. This changes when an inter-route constraint is introduced. Grimault's graph structure (Figure 4.1) requires the entire matrix to be recalculated. We only need to recalculate the values for those routes, which share at least one common resource with the route that was inserted into. This may be all routes in the worst case, but is often fewer. Recall that recalculating a value refers to recalculating costs as described above.
- **Order of inserting requests**
  By Grimault's graph structure and global variables, a solution is unique by its routes and resource schedule, regardless of the order in which requests were inserted. Since we fix our start service times during the insertion process and do not allow rescheduling of already inserted nodes, insertion order does affect our final solution. This is a compromise we make when opting for the speed-up of our procedure.

Note that all of the above also holds for removing a request from a route. While an insertion may turn out to be infeasible under the resource constraint, a removal is always guaranteed to be feasible. This is because all remaining nodes in a route may keep their old start service times, which were known to be feasible. Often however, a better set of times exists, and these will be found using this same procedure.

4.2. ALNS heuristics

Below we outline the removal and insertion heuristics we used. Most of these are taken from previous work outlined in Chapter 3 or inspired by those.

4.2.1. Removal heuristics

- **Random removal**
  Remove requests at random.

- **Worst removal**
  Remove the most expensive requests

- **Relative worst removal**
  Extends *worst removal* by normalising the cost for the *default cost*. This cost is defined as the cost of picking up the request at the origin, driving to the destination, delivering the request and driving back to the origin.

- **Shaw removal**
  Removes related requests as explained in Chapter 3.

- **Distance related removal**. Like Shaw removal, but only considering the distance component. Also employed by Grimault.

- **Time related removal**
  Like Shaw removal, but only considering the time component. Also employed by Grimault.

- **Related route removal**
  Selects a first request at random and removes the route this request is in from the solution. As long as more requests should be removed, a next route is picked at random from the set of routes with at least one common location, and is also fully removed.

- **Location removal**
  Selects a location at random, and removes all requests with a pickup or delivery at this location. Continues until enough requests are removed.

The rationale behind *relative worst removal* is that trucks often do a single delivery and then move on to the next pickup. This heuristic tends to remove requests which are followed by another request relatively far away. Conversely, it tends to leave request with a subsequent nearby pickup in place, as well as request that are part of a combined request.

4.2.2. Insertion heuristics

- **Cheapest insertion**
  Inserts the request with the lowest insertion cost at that moment.

- **Relative cheapest insertion**
  Similarly to *relative worst removal*, insertion costs are normalised by the default cost. This should prevent lengthy trips from being inserted last, which are usually harder to fit into routes than short trips.

- **k-regret insertion**
  As explained in Chapter 3, regret insertion compares insertion cost in the best route to the cost of inserting in up to the k^{th} best route. This favours hard-to-insert requests earlier on, to prevent inefficient routing towards the end of the insertion procedure. Following R&P and subsequent research, we opt for values of k of 2, 3 and 4.

4.3. Initial solution and vehicle minimisation

Several VRP algorithms start by building a quick initial solution, followed by some vehicle minimisation phase, and finally the actual objective optimisation [2, 6, 9, 44]. Oftentimes, these algorithms are applied to problems where the number of vehicles is part of the objective function. R&P for example define a route minimisation stage where the ALNS procedure is applied given n available vehicles. As soon as a solution is found, one vehicle is removed, ALNS is applied again until all requests are served by $n-1$ vehicles, and so on. Since one can never know when the minimum is reached, the only stopping criterion possible is time or number of iterations.

In contrast to much of the literature, we are not interested in the lowest number of vehicles per se. However, as discussed in section 2.3 we do take minimum route duration into account. Section 4.4 further elaborates on this. When the number of routes in a solution is well above the minimum, we likely end up with many routes with too short a duration.

Since finding the minimum number of routes is not crucial, and the route minimisation phase may take up a considerable amount of the available runtime, we opt for a different strategy. We simply apply the regret heuristic (with $k = 4$) described above on the empty solution. This may be slower in finding an initial solution than a quick greedy construction, but already gives us a solution with a reasonably low number of vehicles from the start. Not employing a vehicle minimisation phase more than compensates for that somewhat higher runtime.

4.4. Further adaptations

This section lists additional adaptations we made to the original ALNS procedure. These include the cost function, driver switching, depot switching and making some parameters adaptive to runtime.

4.4.1. Minimum route duration incorporated in the cost function

As stated in Chapter 2, we employ a non-linear cost function (Equation 4.1), which incorporates the minimum route duration. We can directly transfer this cost function to our ALNS implementation. As Shaw [50] stated, the LNS procedure eminently allows more complex cost functions or constraints.

$$c_{tot} = r_d * d + r_t * max(t, t_{min}) \tag{4.1}$$

Besides the fact that this actually represents the cost function, this has the following benefits over enforcing minimum route duration as a constraint:

- **Placing a first request into a route becomes very expensive**
  Empty routes, i.e. unused vehicles, naturally add no cost to the total costs. Given our non-linear cost function, it becomes very expensive to start using a truck by adding a first request to its schedule, since we immediately have to pay for the full minimum duration. As the marginal cost of adding a request to an empty route is so high, the algorithm will almost exclusively do so when the request cannot be placed in any of the non-empty routes. This keeps the number of vehicles low, which is generally desirable, even though we do not optimise for the number of trucks.
- **Placing requests in routes with a duration lower than the minimum is cheap**
  Conversely, when the minimum duration is not reached, the marginal cost of adding a request to a route is low. It only consists of the distance costs, and at most part of the time costs in case adding requests makes the route duration exceed the minimum duration. The algorithm will therefore favour adding requests to routes with short duration, before those with longer ones.
- **Minimum duration remains a soft constraint**
  Even though we do not want to pay for unused time, there may be cases where a route with duration below the minimum is part of the optimal solution. We therefore do not want to

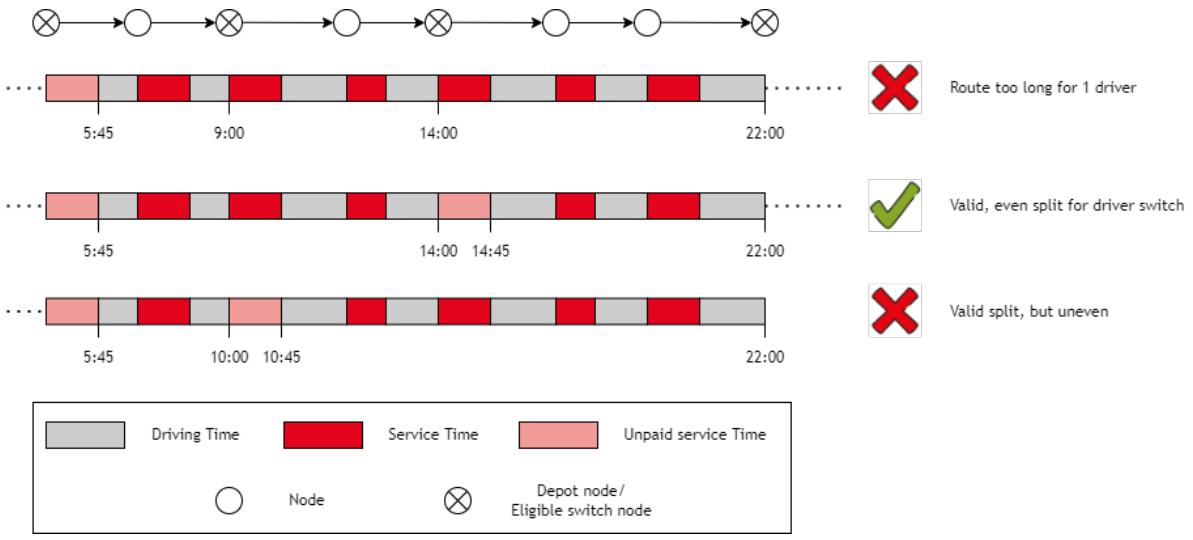


Figure 4.4: Schematic representation of including a driver switch

restrict ourselves exclusively to routes with duration greater than the soft minimum. The fact that the constraint is soft ensures this.

4.4.2. Driver switching

We discussed in section 2.2.4 how we want to incorporate driver switching when checking for route feasibility. Upon checking whether a route is feasible, we first check whether the route is feasible for a single driver. If it is not, because the maximum route duration is exceeded, we have to check whether it is feasible for two drivers. This depends solely on whether the route contains at least one *eligible switch node*, a node with the same location as the depot, such that the resulting routes are both below the maximum route duration. If it does, and there are at least two of such nodes, we choose to make the "cut" where the difference between the duration of two resulting routes is smallest. Figure 4.4 shows this schematically. Note how the first driver ends their shift after driving to the switch node, but the second driver only starts at departure. The service time in between is "saved". The same goes for the service time at the start of the route, in case the first pickup is at the depot (see section 2.1.6).

4.4.3. Dealing with multiple depots and empty routes

Recalculating insertion costs during the insertion phase is the most costly part of the algorithm. Since a route is unique as soon as it contains at least one request, this cannot be circumvented for non-empty routes. For empty routes, however, this is a different story. We introduce the concepts of a route pool and depot switching to prevent doing unnecessary calculations on empty routes, and to efficiently assign the best depot to a route.

The truck pool

When building an initial solution using the regret heuristic, it would be very costly to calculate insertion costs for each empty route belonging to the same depot. After all, assigning a first request to any truck belonging to the same depot will be equally costly. Instead, we start by only calculating costs for one truck per depot, while storing the remaining trucks in the truck pool. The truck pool is a set of empty routes for each depot. As soon as a truck from a depot gets its first request assigned, we move an empty route from the truck pool to the set of active trucks, keeping the number of empty routes per depot at 1 at all times, unless all truck from that depot are already in use. After constructing the initial solution, we remove empty trucks from the set of active routes until less than 10% of the trucks in the active set are empty. This showed to be a good trade off between keeping

the number of empty routes low and giving the algorithm the flexibility to assign requests to empty routes when necessary.

Depot switching
As mentioned in section 2.2, there is a substantial benefit to a truck doing its first pickup at its depot. During the repair phase however, it may occur that a request is inserted at the start or at the end of a route, such that the depot location and the location of the first pickup are no longer the same. When that happens, we allow the algorithm to check whether it is advantageous to switch depot to the new first pickup location, provided that we have an empty truck at that depot still. If this is the case, we always make the switch. In case the first pickup location is not a depot, we check if we can switch to the closest depot. Even though we cannot circumvent waiting on the loading, we may still cut driving time.
Note that depot switching cannot render a route infeasible, provided that the depot nodes have no time windows associated with them. All other nodes will simply retain the same start service times as before.

4.4.4. Adaptive parameters

R&P show that most of the parameter values they use are robust to different problem instances. Subsequent research generally confirms this [24]. We therefore choose not to do an extensive research on all of these parameters, but instead put our focus on other aspects. We take many of their values as a starting point as much as possible. We do make two noteworthy alterations though, based on the fact that as a result of runtime limits, we often do not reach the 25,000 iterations they use.

Segment length
Assuming 25,000 iterations, R&P introduce segments of 100 iterations after which they update ALNS parameters. For larger instances, we may run significantly fewer iterations, in which case the algorithm hardly updates the parameters and the adaptiveness will likely lose its power. To counteract this, we evaluate the expected number of iterations after each segment, and set segment length to 1/250th of this value. We take an initial segment length of 10.

Cooling rate
R&P propose a simulated annealing (SA) procedure for solution acceptance. We also adopt this procedure, but with an adaptation to the cooling rate c. This parameter dictates how quickly the temperature of the SA decreases, i.e. how quickly worse solutions are no longer accepted. The value they use was based on performing 25,000 iterations also. When running fewer iterations while keeping the cooling rate the same, the algorithm will not have 'cooled' enough upon termination. This means that it is still accepting many worse solutions while it should be converging. We adjust for this by also updating the cooling rate after each segment, again based on the expected number of iterations. We intend to keep the *relative cooling rate* the same, such that the probability of accepting a worse solution towards the end is similar compared to a run of 25,000 iterations with the original cooling rate.
Given a start temperature T and cooling rate c, the temperature after 25,000 iterations equals $T * c^{25000}$. With a default of $c = 0.99975$ by R&P, this yields $T * 0.99975^{25000} \approx 0.0019 * T$. This is the temperature we aim to obtain by the end of our runs, regardless of the number of iterations. Algorithm 3 demonstrates how this is done, and Table **??** explains the variables in question.

Algorithm 3 Updating the cooling rate c after each segment

$t_{avg} \leftarrow$runtime so far / #iterations so far
$i_{exp} \leftarrow$ min(maximum runtime / $t_{avg}, 25000$)
$c \leftarrow 0.0019^{(1\ /\ i_{exp})}$

4.5. Summarising the ALNS

We summarise our ALNS choices and adaptations concisely below:

- **ALNS heuristics**. We opt mainly for proven heuristics from literature, especially from Grimault because of its close relation to our problem. We came up with the *relative worst removal* and *relative cheapest insertion* as these are expected to suit our problem well.
- **Resource constraint**. We developed our own procedure for dealing with the resource constraint. Upon inserting a request into a route, we fix the start service times of all other requests, and plan requests on resources accordingly. This might lead to rejecting insertions that would be possible when rescheduling is allowed, but such rescheduling would greatly increase complexity, which is why we choose not to allow this. We show that route duration is optimal given the fixed resource schedules.
- **Initial solution**. We simply obtain the initial solution by applying the regret heuristic on the empty solution. We are not interested in the minimum number of vehicles, so we do not spend excessive time on finding this number.
- **Route pool**. We maintain a pool of 'idle' vehicles such that we do not excessively compute costs for identical empty vehicles. This pool also allows for greedy depot switching of routes.
- **Adaptive parameters**. Since we expect to do significantly fewer iterations compared to related research, we made the segment length of the ALNS and cooling rate of the SA adaptive to the expected number of iterations, such that the parameters they control behave accordingly in a relative sense.

To see how the resource constraint affects the complexity of the problem, we ran the algorithm with and without the constraint implemented on the benchmark set. Figure 4.5 shows that the relative complexity increase seems to be linear in the number of nodes. This supports the observation that e.g. Grimault had difficulty solving larger instances.

4.6. Running experiments

We will now start to evaluate the performance of our algorithm. All algorithms were written in Python and run on a single core. All runs were performed on Linux using 8 Intel Xeon Gold 6148 CPU @ 2.40GHz cores. All runs were capped by 30 minutes of runtime or 25,000 iterations, as per R&P. We did not employ a number of iterations without improvement as another stopping criterion, because this does not influence the results in the end and allows for fairer comparison. Each run of each

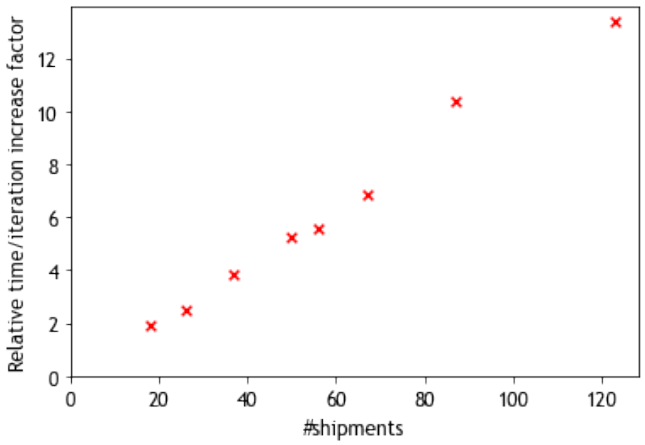


Figure 4.5: Average factor of runtime increase for resource constrained ALNS compared to unconstrained ALNS, for different instance sizes.

instance is performed 10 times. This is because the algorithm is non-deterministic, and we want to be able to draw conclusions with some certainty. All of the above holds for all following experiments, unless otherwise specified.

4.7. Evaluation

In this section we evaluate the performance of the ALNS with the resource constraint and other adaptations described in this chapter. As a starting point, we take the parameters as defined by Ropke and Pisinger, with three exceptions:

- We disregard the loads in Shaw removal since our capacity constraints are so tight, so $\psi = 0$.
- We make the cooling rate c adaptive as explained in Section 4.4.4, but we do take their default of 0.99975 as a starting value.
- We do not apply noise, so $\eta = 0$. Grimault also found that adding noise has little effect.

This leads to the set of parameters as defined in Table 4.1. The reader is referred to the R&P paper for their exact definitions for as far as we do not repeat them here.

Table 4.1: The parameters for the ALNS

ϕ	χ	ψ	ω	p	p_{worst}	w	c	σ_1	σ_2	σ_3	r	η	ξ
9	3	0	5	6	3	0.05	0.99975	33	9	13	0.1	0	0.4

4.7.1. Tuning relevant parameters

We investigate robustness against the destroy percentage ξ of the ALNS, and simulated annealing parameter w. These two parameters could influence the performance of the algorithm, according to initial experiments.

- ξ defines what part of the requests is removed in the destroy phase of the ALNS. It is therefore a value in the interval [0, 1]. Later papers, such as Grimault, experiment with ranges of values. In this case, on each iteration a value is chosen randomly from a uniform distribution in the given range.
- w dictates worse solution acceptance in the simulated annealing procedure. More specifically, it states that at the start of a run, a solution with an objective value that is a factor $1 + w$ worse than the current solution's is accepted with 0.5 probability. This probability decreases because of the cooling rate c described before as time goes by.

We start by investigating the influence of ξ, setting $w = 0.05$ by R&P's default. The results are presented in Table 4.2. This shows that values of 0.1, [0.1, 0.2] and [0.1, 0.3] are generally most favourable in this setting. This is lower than the value of $\xi = 0.4$ that R&P report, and more in line with the results of Grimault. We explain this by noting that setting $\xi = 0.4$ will make a single iteration take much longer, hence we can do fewer iterations in the given time. Also, removing such a big portion of the requests means making big leaps throughout the search space, possibly not converging to local minima. Finally, since Grimault treats a problem more similar to ours than R&P, it feels intuitive to observe that the best values correspond.

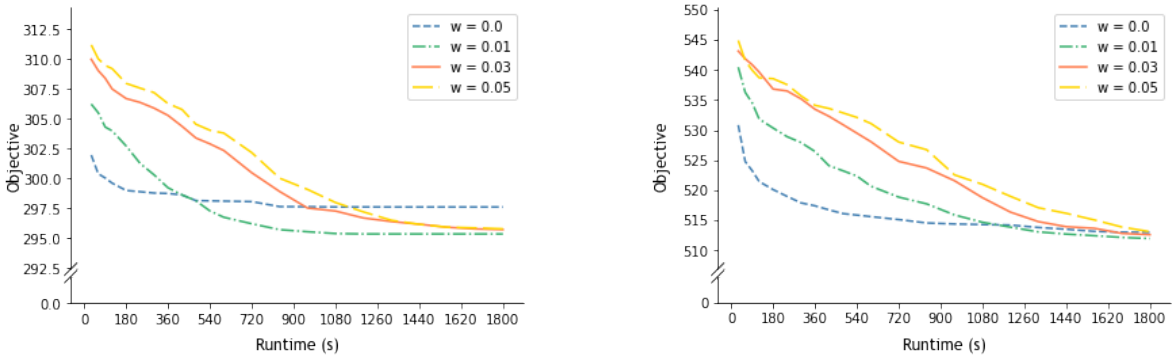
We take these values to our next experiments, where we look at the influence of w. Recall that setting $w = 0.05$ means that initially the algorithm may accept solution that are 5% worse cost-wise with 0.5 probability. Hence the algorithm will spend a lot of time accepting bad solutions, and it may be more favourable to move to good solutions more quickly. We experiment with values for w of 0.05, 0.03, 0.01 and 0. Note that $w = 0$ is equivalent to not using simulated annealing at all, but simply accepting improving solutions only. The final results are presented in Table 4.3, for

Table 4.2: Varying ξ for different instances. $w = 0.05$. Results are averages of 10 independent runs. Best results per instance are in **bold**.

Instance	0.1	[0.1, 0.2]	[0.1, 0.3]	0.2	0.3	0.4
fc6_r_18	116.27	115.57	**115.45**	115.57	**115.45**	**115.45**
fc6_mr_26	170.29	169.77	169.63	169.59	**169.58**	169.77
fc46_r_37	213.25	212.93	**212.91**	212.93	213.48	214.51
fc127_r_50	296.13	**295.79**	296.15	296.18	298.15	301.86
fc46_mr_56	324.45	**324.34**	324.72	325.42	327.43	329.76
fc127_mr_67	**399.57**	399.96	401.88	400.71	403.69	407.43
fc12467_r_87	**511.38**	513.13	515.99	518.72	529.38	534.83
fc12467_mr_123	**732.42**	738.34	744.76	745.72	757.85	785.94
fc3_ri_32	239.83	239.10	**238.71**	239.23	239.27	239.25
fc3_mri_44	345.61	344.41	**343.50**	343.86	345.23	348.27
fc15_ri_49	272.93	**271.70**	271.82	271.91	272.53	274.58
fc15_mri_61	338.88	339.98	**338.70**	342.22	341.88	345.12
fc135_ri_81	513.36	**513.11**	515.56	516.21	526.16	537.48
fc135_mri_105	**688.99**	693.57	700.09	701.50	715.93	730.15

$\xi = [0.1, 0.2]$, which gave the best results over 0.1, [0.1, 0.2] and [0.1, 0.3]. Although the trend is not very pronounced, we do see better results on average for lower values of w, which we attribute to our assumption. More interesting are the curves presented in Figure 4.6, displaying how the best solution improves over time on average. These show that a lower value for w makes the average objective decrease more quickly. This is most logically explained by realising that fewer worsening solutions are accepted for lower w, especially early on. If a relatively good solution is desired in short time, then setting $w = 0$ (disabling SA) is the way to go. We do see, especially in Figure 4.6a, that one runs a higher risk of getting stuck in local optima. The curve plateaus, but clearly not for the optimal value.

We do not investigate different combinations of ALNS heuristics. Such experiments can scale enormously given the number of heuristics we have, and the rationale behind the adaptive component of ALNS, is that the algorithm will use the better performing heuristics more often automatically. Also, several papers indicate that omitting heuristics will hardly affect performance [24, 44].



(a) Instance fc127_r_50

(b) Instance fc12467_r_87

Figure 4.6: Best found solution vs. runtime for two of the instances, for different values of w. Graphs represent averages over 10 independent runs.

Table 4.3: Varying w for $\xi = [0.1, 0.2]$ for different instances. Results are averages of 10 independent runs. Best results per instance are in **bold**.

Instance	0.00	0.01	0.03	0.05
fc6_r_18	115.70	115.63	115.62	**115.58**
fc6_mr_26	170.42	**169.56**	169.70	169.78
fc46_r_37	213.14	**212.77**	213.00	212.93
fc127_r_50	297.61	**295.36**	295.73	295.79
fc46_mr_56	324.90	**323.65**	323.86	324.34
fc127_mr_67	400.29	399.54	**399.27**	399.96
fc12467_r_87	513.01	**511.99**	512.62	513.13
fc12467_mr_123	**732.13**	733.22	737.58	738.34
fc3_ri_32	240.21	239.14	**238.79**	239.10
fc3_mri_44	346.33	344.68	**344.23**	344.41
fc15_ri_49	273.76	**271.68**	272.32	271.70
fc15_mri_61	341.38	339.37	**338.70**	339.98
fc135_ri_81	517.01	**512.12**	512.87	513.11
fc135_mri_105	695.99	**690.11**	694.17	693.57

4.7.2. Partitioning instances

As noted in section 2.6, some of these instances are the union of two others. Table 4.5 shows how well each of these instances are solved jointly versus when split up and solved separately. Note that this split is somewhat arbitrary, so we refrain from drawing conclusions from this observation. It does however suggest that the potential of solving the joint instance is outweighed by the increased complexity. For this set of four instances, the difference grows with instance size, which suggests that runtime complexity in relation to instance size plays a role. Chapter 7 goes into more detail on partitioning instances.

Table 4.4: Comparing results of joint vs decomposed instances. $\xi = [0.1, 0.2]$ and $w = 0.01$

Joint instance	Decomposed instances	Joint objective	Sum of separate objectives	Diff (%)
fc12467_r_87	fc46_r_37 fc127_r_50	511.99	508.12	-0.76
fc12467_mr_123	fc46_mr_56 fc127_mr_67	733.22	723.18	-1.37
fc135_ri_81	fc15_ri_49 fc3_ri_32	512.12	510.82	-0.23
fc135_mri_105	fc15_mri_61 fc3_mri_44	690.11	684.05	-0.88

4.7.3. Concluding remarks

From these results, we list some takeaways given our instances and runtime limit:

- Smaller values for the destroy percentage parameter ξ are generally more favourable. This leads us to believe that making many small steps through the search space is better than making fewer large ones.
- The value for SA parameter w has little influence on the final results. It turns out that smaller values for w, especially $w = 0$, lead to finding relatively good solutions quickly. When time limits are even tighter, this can be taken into consideration.
- The results indicate that there may be cases when partitioning instances into smaller ones is favourable for the final solution. In such cases it seems that the potential of solving the joint instance is outweighed by the increased complexity.

5

Extending the ALNS with local search

In the previous chapter we demonstrated the ALNS to solve our problem. In section 4.7 we showed however that larger instances were better solved when partitioned and solved separately. This indicates that the algorithm does not solve these larger instances optimally. What is more, we cannot know if the smaller instances themselves are solved close to optimality.

In a search for improvements, we suggest the use of local search. Local search heuristics, or operators, were among the first non-exact solution methods for routing problems [18]. Yet they are used less frequently on pickup and delivery problems, relatively speaking. A likely explanation is that many edge-based local search methods cannot be directly applied to these problems, as they will break the pickup-and-delivery constraint (a pickup and delivery must be performed by the same vehicle). Essentially, an edge exchange can only be performed when the vehicles in question are empty at the nodes before the to-be-exchanged edges. In many cases, e.g. when capacity is not so limiting, this is not often the case. When possible edge exchanges are limited, the power of local search quickly decreases. In our highly capacity constrained case however, vehicles are frequently emptied completely, and this leaves enough space for edge exchanges. Also, some local search operators can be applied to PDP with only small alterations, as is done in earlier research by e.g. Lim et al. [36]. Given the success of local search methods and the apparent applicability to our problem, we choose to apply them.

In this chapter, we demonstrate how we expand upon the ALNS presented in the previous chapter. In section 5.1 we introduce three local search heuristics which we apply to our problem and argue why we chose these heuristics, and chose not to implement others. We show how local search is affected by the resource constraint in section 5.2, and how the chosen heuristics are applied in the overall procedure in section 5.3. Finally, we perform a new set of experiments building on the ones from the previous chapter in section 5.4. We again investigate the destroy percentage, and perform an ablation study on the local search heuristics introduced. We end the chapter with some concluding remarks in section 5.5.

5.1. Local search heuristics

This section describes the three local search heuristics (relocate, exchange and crossover) we employ in our new algorithm, why we chose them, and how they were tweaked to suit our specific problem. We end this section with a paragraph arguing why we chose not to include some other popular heuristics.

5.1.1. Relocate

The relocate heuristic tries to move a node from one route to another. In the case of our PDP, this means that both the pickup and delivery node have to be moved, or the pickup-and-delivery constraint is violated. Where this heuristic can be viewed as an edge exchange in VRP, we view it purely as a node exchange. Upon trying to move a request from one route to another, we test all potentially valid positions of the two nodes in the new route. The move to the best possible pair of positions is executed, when at least one exists. The relocate heuristic scales roughly quadratically in the number of nodes, since every request can be placed after every other request in the solution. Because the relocate heuristic is relatively fast, we employ a best-accept strategy to find better solutions more quickly.

5.1.2. Exchange

Similar to relocate, exchange tries to move single requests to a different route. However, it considers swapping a pair of nodes simultaneously, rather than relocating a single node. Both nodes of a tested pair are removed from their current routes, and tried being inserted in the other route. Since the operator considers pairs of nodes ($O(n^2)$), the neighbourhood of this operator scales faster than that of relocate ($O(n)$). Note further that this exchange operator examines more options than the regular edge-based exchange. The latter is limited to swapping nodes such that the new node takes the position of the old node in its route. This inherently limits the number of node pair that can be exchanged. Our procedure provides greater flexibility in this regard. This does mean that our exchange operator has an even larger neighbourhood compared to other operators, and it is therefore expensive in terms of runtime to exhaust it. For this reason, we accept solutions on a first-accept basis for this operator.

5.1.3. Crossover

The crossover operator is purely an edge-based one, making it generally hard to adapt to PDPs as a vehicle has to be empty at the crossover point. As mentioned before, our trucks are often empty throughout their routes, giving us plenty of crossover potential. Note that the crossover we describe here is in fact the 2-Opt* operator as described in Chapter 3. We prefer to use the term *crossover* as it falls in line more nicely with *relocate* and *exchange*.

Crossover works by cutting two routes in half (in places where the trucks are empty), conversely combining the two halves, and evaluating whether this yields routes with a combined lower cost. The nodes in a route after which the truck is empty are considered *eligible crossover nodes*. The number of eligible crossover nodes is at most half the number of nodes in a route, since this can only occur after a delivery node. Given two routes and two cut nodes, there is only a single way to recombine the four resulting half-routes. Hence, the crossover neighbourhood is significantly smaller than e.g. that of exchange, and therefore more quickly exhausted. We employ a best-accept strategy because of this.

5.1.4. Excluded heuristics

We finally go over some common heuristics that we did not implement, and argue why.

Intra-route heuristics

One may have observed that all local search heuristics we implemented are inter-route heuristics that involve two routes. Common heuristics like intra-route 2-opt, k-opt and Or-opt are not explicitly implemented, the reason being that we believe them not to be effective in many cases. First of all, *relocate* partly covers these cases when it tries to relocate a node in its current route. As for more complex intra-route operations, such as reversing part of the routes or recombining sequences of nodes, this will be often invalid because of the tightness of the time windows. While implementing these operators anyway might not have hurt, we chose to focus on the more promising ones.

'Full' crossover
The main reason to exclude crossover as described in Chapter 3 is its runtime complexity. As pointed out earlier, it replaces four edges rather than the two in our crossover (2-Opt*) operator. The latter, as well as the exchange operator, are special cases of this operator, and we believe that many of its moves are covered by the operators we do use.

5.2. Local search and the resource constraint

Savelsbergh [48] demonstrated how local search can be done efficiently for VRPs with route duration minimisation. The foundation of his approach lies in the notion of slack times, which can be efficiently kept track of when exhausting a neighbourhood in lexicographical order. It is built on the assumption that start service times are not fixed. Therefore, transferring this procedure to our setting with the resource constraint and fixed start service times is not trivial. We are bound to recalculate new start service times for any route that changes as a result of a local search move, making the time complexity of calculating the cost for such a move linear in route length rather than constant time as Savelsbergh achieves. This is an inherent drawback of our method, and investigating ways to overcome this would be a very interesting avenue of research. This also supports our argument to not implement the more complex local search operators described in the previous section.

5.3. General framework

We now set out to find a way to incorporate the local search heuristics into a grander scheme. Such heuristics require a framework in which an initial solution is constructed, as well as a way to escape local optima. It turns out that our ALNS algorithm can serve both these purposes. The initial solution is again obtained by applying the regret heuristic on the empty solution. It may occur that a single pass cannot place all shipments in the solution, and we are left with a partial (feasible) solution. Our local search heuristics are not designed to turn such solutions into feasible ones, so instead we apply ALNS iterations until we obtain a feasible solution. As soon as we obtain a feasible solution, we move to the local search phase. When this phase is finished, we can apply another iteration of ALNS to perturb the solution. We then move to the local search phase again, and so on. Figure 5.1 presents a schematic view of the entire framework, dubbed the *ALNS+LS*.

We now only need a procedure for the LS phase itself. One of the most common procedures is a Variable Neighbourhood Search [40] (VNS), where one LS operator is applied until no improvement can be made using this operator. It has reached a local optimum with respect to this operator. The next operator, which has a different neighbourhood, may be applied to find a new local optimum, until no operator can improve the solution anymore. We apply this procedure, with the exception that we do not exhaust an LS operator, but rather apply it once, and then move on to the next, until none of the operators can improve the solution. Preliminary experiments showed that this was a better trade-off between runtime and performance. Algorithm 4 describes this procedure.

Algorithm 4 The LS procedure

```
1: h ← list of LS operators
2: i, fail_count ← 0
3: while fail_count < len(h) do
4:     execute h_i
5:     fail_count ← fail_count +1
6:     if h_i improved solution then
7:         fail_count ← 0
8:     i ← (i + 1) mod len(h)                ▷ To select next operator
```

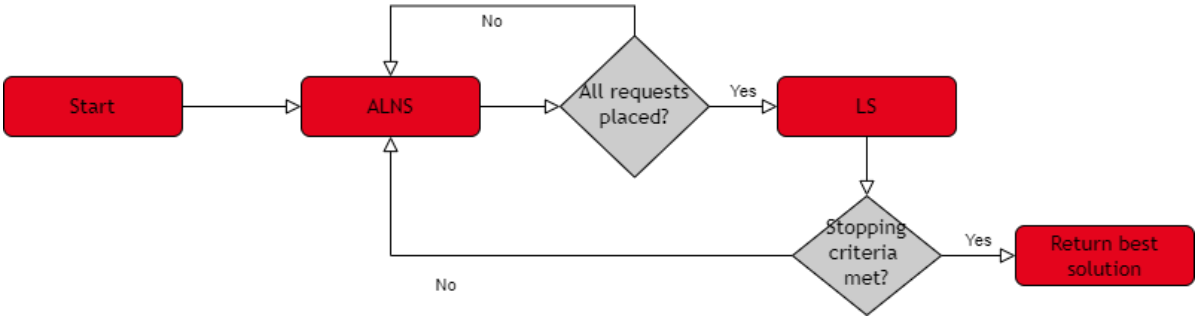


Figure 5.1: Flowchart of the ALNS+LS framework

5.4. Evaluation

As for the ALNS algorithm, we intend to find out which configuration of the ALNS+LS algorithm performs best. An important thing to note beforehand is that an iteration as defined in the previous section takes significantly more time. Besides the ALNS step, we do a number of LS moves until convergence, which is unknown in advance. Recall from the previous chapter that a higher value for SA parameter w would lead to slower convergence of the objective. Since we now expect fewer iterations, we do not investigate the influence of this parameter on the performance, but instead take $w = 0.01$. We do not set $w = 0$ since preliminary experiments showed that this would still limit search diversity too much.

We do investigate destroy parameter ξ again. We argued before that a higher value for ξ leads to larger steps through the search space. We expect therefore more LS steps per iteration also, which would weigh even heavier on the time per iteration. On the other hand, the combination between larger perturbations and local search could be more effective than making smaller steps.

The results for the analysis on ξ are presented in Table 5.1. Clearly, the results lie rather close together over the investigated values, suggesting that this algorithm is actually robust against this parameter, in contrast to the ALNS algorithm. This could be because the local search phase finds similar local optima regardless of how 'big' the ALNS move is. It just may take more LS steps to find it.

In addition to the values of the parameters, we are interested in the effectiveness of the LS operators. Since runtime is limiting, we would want to exclude operators that do not contribute towards a better objective. To that end, we perform an ablation study on the three operators for each of the destroy parameter settings. Table 5.2 presents the results for $\xi = [0.1, 0.2]$, which most often gave the best result over the benchmark instances. For each of the 4 configurations (all three heuristics (RCE), and the three combinations of two heuristics, (RE, RC and CE)), we present the average objective over 10 runs together with the coefficient of variation CV. The CV is defined in Equation 5.1 where μ denotes the mean of the objectives of the 10 runs, and σ the standard deviation.

$$CV = \frac{\sigma}{\mu} \tag{5.1}$$

The CV was chosen as it normalises for the mean, which makes it easier and fairer to compare across instances. A CV value of 1.00 in the table should be interpreted as the standard deviation being 1.00% of the mean.

The best values per instance are marked in **bold**. Across all instances, the RCE configuration most often yields the best results. For instances where this is not the case, RCE is often only slightly worse than the best one. For all instances, the CV is (much) less than 1% of the average, which demonstrates the robustness of the algorithm.

Table 5.1: Varying ξ for the ALNS+LS algorithm

Instance	[0.1, 0.1]	[0.1, 0.2]	[0.1, 0.3]	[0.2, 0.2]	[0.3, 0.3]
fc6_r_18	115.58	**115.45**	**115.45**	**115.45**	**115.45**
fc6_mr_26	169.31	**169.23**	169.23	**169.23**	**169.23**
fc46_r_37	212.29	**212.21**	212.28	212.23	212.26
fc127_r_50	295.70	**294.93**	294.95	295.05	295.07
fc46_mr_56	321.76	**321.64**	321.92	322.15	322.52
fc127_mr_67	397.42	397.05	**396.99**	397.33	397.97
fc12467_r_87	513.19	510.31	510.69	510.69	**509.93**
fc12467_mr_123	724.55	**723.81**	724.99	727.15	726.78
fc3_ri_32	239.15	238.82	**238.50**	238.60	238.45
fc3_mri_44	342.61	**342.36**	342.75	342.49	342.76
fc15_ri_49	270.37	269.93	**269.01**	269.80	270.07
fc15_mri_61	336.80	**336.04**	336.10	336.29	337.24
fc135_ri_81	511.18	**509.69**	510.39	511.04	511.99
fc135_mri_105	**685.41**	686.08	685.79	687.54	689.18

Table 5.2: Ablation study on the local search heuristics. Best averages and standard deviations per configuration are presented in **bold**. $\xi = [0.1, 0.2]$.

	ls	RCE		RE		RC		CE	
Instance		avg	CV (x100)	avg	CV (x100)	avg	CV (x100)	avg	CV (x100)
fc6_r_18		**115.45**	**0.00**	**115.45**	**0.00**	115.56	0.20	**115.45**	**0.00**
fc6_mr_26		**169.23**	**0.00**	169.30	0.07	169.50	0.11	169.24	0.02
fc46_r_37		**212.21**	0.05	212.34	0.05	212.34	0.05	212.22	**0.02**
fc127_r_50		**294.93**	0.08	295.31	0.12	295.11	**0.07**	295.00	0.08
fc46_mr_56		**321.64**	**0.09**	322.36	0.13	322.24	0.12	321.99	0.12
fc127_mr_67		**397.05**	**0.14**	397.72	0.21	397.76	0.31	397.32	0.15
fc12467_r_87		510.31	0.70	511.21	0.35	**509.70**	**0.28**	510.55	0.33
fc12467_mr_123		**723.81**	**0.22**	726.07	0.25	727.00	0.26	726.99	0.41
fc3_ri_32		238.82	0.21	238.82	0.21	**238.71**	**0.18**	238.97	0.24
fc3_mri_44		**342.36**	**0.16**	342.70	0.19	342.39	0.18	342.72	0.24
fc15_ri_49		**269.93**	0.26	271.12	0.56	269.97	0.34	270.34	**0.23**
fc15_mri_61		**336.04**	**0.25**	337.13	0.29	336.12	0.28	338.46	0.61
fc135_ri_81		**509.69**	0.39	513.29	0.48	509.89	**0.27**	514.75	0.44
fc135_mri_105		686.08	0.42	688.65	0.56	**685.94**	0.50	689.49	**0.25**

5.5. Concluding remarks

In this chapter we implemented local search into the ALNS algorithm from the previous chapter. In earlier research to closely related problems, local search was often not used. We however demonstrate how local search methods impact the performance of our algorithm. We summarise our findings below:

- We chose to implement three LS heuristics, the neighbourhoods of which do not scale too badly in instance size. We looked at how the three heuristics contributed to the performance, and found that putting all three together yields the best results.
- Omitting any of the LS heuristics gave results that are close to the configuration with all three operators. While we may lose a set of good neighbourhood moves by doing so, we apparently achieve similar results by being able to apply the other two operators more often.
- We showed how the ALNS+LS algorithm was more robust to the destroy parameter ξ.
- A closer look at Tables 5.1 and 5.2 suggests that large instances are still better solved when decomposed.

6

A Matheuristic approach

Since we still do not solve larger instances optimally, we continue to look for solution methods for our problem. Figure 4.5 demonstrated how runtime complexity scales with instance size as a result of the resource constraint. We therefore present a matheuristic in this chapter that relaxes the resource constraint initially in the ALNS+LS procedure, and later enforces it using a CP model. Grangier et al. [23] did this for a different routing problem with resource constraints at a single transfer location. Relaxing the constraint is expected to increase the possible number of iterations given our runtime limits. We should only make sure that enforcing the constraint can be done quickly such that we keep this advantage in the end. Note that instead of a CP model one can also employ a scheduling heuristic. We chose not to take this path as developing such a heuristic is not trivial, and naturally suffers from the fact that it may reject feasible solutions by its heuristic nature. On the other hand, the CP model we developed is concise and solved relatively quickly using the right solver and settings. The model is presented in the first section of this chapter.

We present the CP model in section 6.1. Section 6.2 describes the matheuristic framework, and we do a first evaluation in section 6.3. It is important to note that a *candidate solution* obtained by the ALNS+LS phase is not guaranteed to be feasible. As it turns out, it often is not, especially when instance size and solution intricacy increases. Therefore, section 6.4 presents two adjustments to the ALNS+LS phase that aim to guide the candidate solutions towards being feasible. This section also evaluates the matheuristic with these adjustments. The chapter is concluded in section 6.5.

6.1. The CP model

As discussed, the CP model imposes the resource constraint on a candidate solution produced by the ALNS+LS step. The model takes as input the routes of the solution, and outputs an optimal scheduling to this solution if and only if one exists. The array of start service times for all nodes is the only decision variable of the problem. This means that the routes themselves remain fixed, i.e., no nodes can be swapped within a route or between routes. In fact, the fixed routes serve as precedence constraints on the nodes: each node must be serviced at a time such that its succeeding node can also be serviced in time. This highly constrains the problem, making it very suitable for a CP solver to solve [45]. Time windows impose another constraint on this service time. Finally and most importantly, the resource constraint is imposed by a disjunctive constraint, similar to the one in the model described in section 2.4. The model is presented below.

Parameters

N	Set of all pickup/delivery nodes
F	Set of first pickup nodes for each route
D	Set of last delivery nodes for each route
L	Set of locations (resources)
(a_i, b_i)	Time window for node *i*
s_i	Duration for which node i occupies its resource
r_i	Service time plus driving time to successive node for node i
δ_i	Service time to add for non-hub nodes, for which departure times should be disjoint. Zero for hub nodes
L_j	Set of nodes with location j
t_{min}	Minimal driver day duration

Decision Variables

t_i	Start service time for node i

Constraints

$a_i \leq t_i \leq b_i \; \forall i \in N$	Node i must be serviced within its time window
$t_i \leq t_{p_i} + r_{p_i} \; \forall i \in N \setminus F$	Precedence constraints on node i. Not applicable to first pickup nodes
disjunctive($[t_i + \delta_i \; \forall i \in L_j \mid s_i \; \forall i \in L_j]$*)* $\; \forall j \in L$	Resource constraint

Objective

min $\sum_{i \in D} \; t_i - \sum_{j \in F} \; t_j$	Minimize schedule duration, which equates to minimizing the summed differences between times of last delivery and first pickup.

6.2. The matheuristic framework

We leave the first part of the algorithm, the ALNS+LS procedure, mostly untouched. We remove the mechanism introduced in section 4.1, and after inserting a request into a route we only have to re-evaluate insertion costs for that route. This is the major time-saver in this step. Figure 6.1 outlines the full procedure schematically.

6.2.1. Reducing the number of CP evaluations

Since CP evaluations do not come for free, we want to avoid unnecessary evaluations where possible. We propose two ways by which we can do so.

Note that enforcing the constraint will never decrease the total solution cost. In case we obtain a candidate solution after the ALNS+LS step with a cost higher than the best known feasible solution, we already know that this candidate solution can never improve on the best solution. In that case, we simply discard this solution. We skip the CP step and immediately move on to the next iteration. This speedup becomes more significant as our solution improves, since more and more candidate solutions will be rejected.

Secondly, recall that the CP model can freely reschedule node visits on resources as long as it preserves the routes themselves. Thus, a resource-unconstrained schedule given as input will always yield the same optimal resource-constrained schedule as output. This means that there is no need to evaluate a schedule more than once. We therefore store representations of evaluated schedules in a set, and check before a CP evaluation whether this schedule was evaluated before. If so, then we skip the CP step and move back to the ALNS+LS procedure.

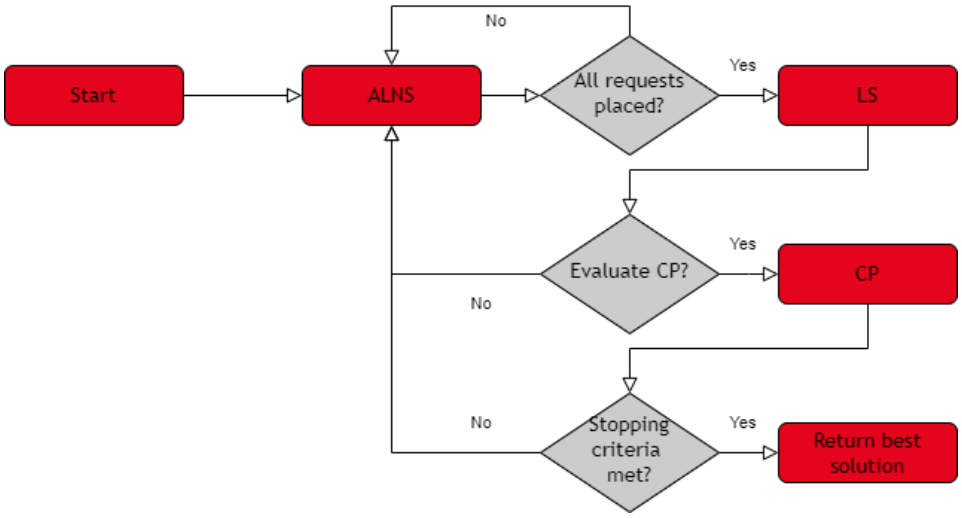


Figure 6.1: Flowchart of the matheuristic framework

6.3. First evaluation

Table 6.1 presents the results of the matheuristic on the benchmark set. While we leave a full comparison between the different algorithms for Chapter 7, we do present the ALNS+LS results here for comparing the number of trucks used. We return to this in the next section. The table also presents the number of failed runs for each instance. We say that a run has failed when no feasible solution has been found at all, which is highly undesirable. For instances *fcACEFG_r_87* and *fcBDE_mri_105*, none of the 10 runs yield a feasible solution. The apparent difficulty in finding (good) solutions for larger instances is most likely explained by the larger number of resources that is to be accounted for, and the more constrained resource scheduling as a result of many interdependent routes. Besides the fact that we are not guaranteed to find solutions, the solutions we do find are not competitive with those of the ALNS+LS algorithm. This suggests that we cannot naively leave out the resource constraint in the first place and hope to obtain good candidate solutions.

6.4. Promoting feasibility

Given that our first results are unsatisfactory, we make an important observation. Table 6.1 shows us that the solutions provided by the matheuristic use fewer trucks than those given by ALNS+LS on average. Apparently the matheuristic attempts, and succeeds, to put more shipments in a route. This can be explained by noting that under the resource constraint, it occurs more often that a shipment cannot be inserted in a route that already contains many shipments. After all, the resource constraint limits the insertion options. The ALNS+LS with resource constraints will more often require to use an extra route to plan all shipments. Put differently, routes with fewer shipments should have more slack. Recall that slack is expressed as the time by which a node can be shifted in time without making the route duration longer.

Conversely, a candidate solution with fewer routes and less slack may be less likely to be feasible than one with more routes and slack. It may therefore be advantageous to guide the candidate solutions

Table 6.1: Results for the basic matheuristic. $\xi = [0.1, 0.2]$, next to the best ALNS+LS results. '-' indicates when no feasible solution was found in 10 runs.

Instance	Matheuristic			ALNS+LS	
	avg	#fails	avg #trucks	avg	avg #trucks
fcG_r_18	115.45	0	6.0	115.45	6.0
fcG_mr_26	170.31	0	6.0	169.23	6.0
fcFG_r_37	212.21	0	11.0	212.21	11.0
fcACE_r_50	296.76	4	14.0	294.93	15.0
fcFG_mr_56	323.91	0	11.0	321.64	13.6
fcACE_mr_67	401.37	0	15.0	397.05	16.4
fcACEFG_r_87	-	10	-	510.31	26.5
fcACEFG_mr_123	739.19	4	26.0	723.81	1.22
fcB_ri_32	237.97	0	10.0	238.82	10.1
fcB_mri_44	342.96	0	11.0	342.36	12.7
fcDE_ri_49	274.65	0	12.5	269.93	12.9
fcDE_mri_61	339.37	0	12.8	336.04	14.4
fcBDE_ri_81	520.03	0	22.0	509.69	23.5
fcBDE_mri_105	-	10	-	686.08	26.4

to being more feasible somehow, i.e., have more routes and more slack. This section describes the two adjustments we made to the ALNS+LS step to try to achieve this. First however, we discuss two ideas that felt intuitive but did not work well.

6.4.1. First ideas: inter-route penalties

Recall that the resource constraint effectively disallows nodes to overlap at a location. In order to minimise this potential overlap, we tried adding a weighted penalty to the objective function. Shipments that overlap at a resource could be penalised for the time that they overlap. While this may have less overhead than the procedure as explained in Chapter 4, we would again introduce interdependence among the routes. Inserting or removing a shipment would require reassessing the penalty function for all affected routes and resources, and updating insertion cost for the remaining uninserted shipments for all these routes before the next insertion. Overall, we would not gain enough complexity decrease and speedup.

A second idea was to create for each resource a discrete timeline, and allow only one shipment per time step. We attempted to implement this, but this suffers from the very same issue. We concluded that the adjustments that we make should not add complexity to the problem, or we would gain too little from it compared to the extra overhead of enforcing the constraint.

6.4.2. Promoting slack

Adding a penalty to the objective function can be a good strategy to promote certain solution characteristics. We only must take care that we penalise attributes belonging to a single route, rather than global attributes, or ones that depend on other routes. This limits the possibilities, but one attribute that could be of interest is slack, discussed in Chapter 3. Recall that the forward slack of a route defines how much we can shift a route forward in time, without making its duration longer. Since we schedule our routes as late as possible as shown in section 4.1, and thus may only move it earlier, we will consider backward slack. The intuition behind favouring more slack is as follows: When two

routes without slack visit the same resource simultaneously, there is no possibility of moving either route backward in time. The overlap cannot be adjusted for, and the solution will be infeasible. When slack in routes is large, adjusting for overlap is expected to be easier. We therefore introduce the slack penalty into the objective function, which now is:

$$c_{tot} = r_d * max(d, d_{min}) + r_t * t + w_s * s \tag{6.1}$$

Rather than simply subtracting the slack value of the route, we define s as

$$s = 1 - min_{n \in R} \frac{atw_{end} - atw_{start}}{tw_{end} - tw_{start}} \tag{6.2}$$

where tw denotes the original time window of the node, wheres atw denotes the adjusted (shortened) time window as a consequence of the other nodes in the route. In words, we take the minimal ratio of actual slack over maximum slack over all nodes as a metric. Normalising for maximum slack makes us indifferent to the fact that some routes simply have little slack because some node it contains has a small time window from itself. The value for w_s has to be determined empirically.

6.4.3. Enforcing more routes

A second observation regarding candidate solutions is that they tend to use fewer routes on average than solutions obtained by the ALNS+LS algorithms. While this may seem beneficial, it could also mean that the relaxed problem can be solved with fewer routes than the constrained problem. This may lead to many infeasible candidate solutions. To counteract this, we introduce a route splitting step between the ALNS and LS steps. This splits the longest route in the solution into two routes. However, it may vary from one instance to another whether fewer routes is better. In order to promote diversity in this search, we decided to only apply this step with a 0.5 probability.

6.4.4. Evaluation after applying feasibility promotion

We evaluated the performance of the matheuristic with the presented adjustments, and present the results in Table 6.2. After some initial runs, we used weights of 0 (no penalty), 10 and 100 for our experiments, all with truck splitting. Note that the performance of the penalty weights depends on how we defined the slack penalty function in Equation 6.2.

Let us first compare the results for $w_s = 0$ without route splits (Figure 6.1) and with route splits (Figure 6.2). We see that the average objective decreased for the nine largest instances. For the smaller ones, they are mostly only insignificantly worse. Interestingly, the number of trucks used on average is clearly bigger, and much more in line with the ALNS+LS results in Figure 6.1. This suggests that the route splitting achieves what it was meant to do. Also, the total number of failed runs decreased from 28 to 15. Instance 87 is still never solved.

Next, we take a look at what increasing the penalty value gives us. First of all, the number of failed runs decreases further, to 4 for $w_s = 10$ and 0 for $w_s = 100$. This suggests that higher slack penalties indeed better guide candidate solutions towards a feasible solution. Further, we observe that the number of routes for each solution has increased on average to values much closer to those of the ALNS+LS results presented in Table 6.1. In terms of average objective, we see that the results improve up to 1% across the benchmark set compared to the ‘naive’ matheuristic. We do observe slightly worse average objectives for $w_s = 100$. When the penalty component weighs heavier on the objective function, its focus shifts away from the true routing objective. This could explain this observation, but more experiments would be required to draw conclusions.

Table 6.2: Results for the matheuristic for different penalty values. With route splitting. $\xi = [0.1, 0.2]$. Instance names are abbreviated to their sizes. '-' indicates when no feasible solution was found in 10 runs.

	w_s	0			10			100		
Instance		avg	#fails	#trucks	avg	#fails	#trucks	avg	#fails	#trucks
fc6_r_18		**115.45**	0	6.0	**115.45**	0	6.0	**115.45**	0	6.0
fc6_mr_26		170.39	0	6.0	**170.31**	0	6.0	170.36	0	6.0
fc46_r_37		**212.18**	0	11.0	212.20	0	11.0	212.32	0	11.0
fc127_r_50		295.87	0	15.1	295.26	0	15.0	**295.21**	0	15.0
fc46_mr_56		322.01	0	13.6	**321.74**	0	13.7	322.81	0	13.2
fc127_mr_67		398.95	0	17.1	**398.76**	0	16.7	398.85	0	16.3
fc12467_r_87		-	10	-	512.16	3	26.3	**510.00**	0	25.9
fc12467_mr_123		734.63	1	28.6	735.61	1	29.3	**733.57**	0	29.5
fc3_ri_32		238.26	0	10.1	**237.94**	0	10.0	238.26	0	10.1
fc3_mri_44		343.47	0	11.2	**342.92**	0	11.0	343.96	0	11.2
fc15_ri_49		**269.54**	0	13.0	270.90	0	13	271.85	0	12.9
fc15_mri_61		338.51	0	14.8	**337.98**	0	14.3	340.28	0	14.2
fc135_ri_81		518.58	0	23.8	**515.12**	0	23.4	516.40	0	23.2
fc135_mri_105		698.55	4	25.7	**695.77**	0	26.1	700.01	0	27.1

6.5. Concluding remarks

We have presented a matheuristic for our problem which has to the best of our knowledge not been presented in this form in earlier research. We address the most important points below.

- Simply splitting up the algorithm in an unconstrained routing phase, and a resource constraint imposing CP phase, yields results that are not competitive with our ALNS+LS algorithm. Without any guidance towards the resource constraint, the routing phase does not provide good quality candidate solutions. Sometimes it fails to find a solution at all. Often, this is because the routing phase returns candidate solutions with too few routes. For some instances, it does not find solutions at all, which is very undesirable.
- The slack penalties and route splitting we introduce improve the performance of the matheuristic noticeably, such that it approaches the performance of the ALNS+LS algorithm on smaller instances. On larger instances, the matheuristic solution quality degrades, likely because the resource constraint becomes more constraining when the number of routes and requests grows.
- While we show the use of slack penalties, the best weight of the penalty is hard to determine. There are signs that a higher penalty makes the algorithm more robust against failure, but at the cost of a higher objective. More research can be put into this, and possibly this weight can be a function of problem characteristics (e.g., problem size). Alternatively, the weight could be made adaptive: depending on how well the algorithm finds feasible solutions, it can be decreased or increased dynamically.
- In terms of guaranteeing finding solutions, an interesting direction would be to hybridise the matheuristic with the constrained ALNS+LS procedure. Besides finding feasible solutions in the first place, we might be able to combine the best of both algorithms and increase performance further.

7

Final evaluation

The ultimate goal of this thesis is to find out how to best solve the problem at hand. In this chapter we take the best version of our algorithms and investigate its performance on a dataset of five full days of shipments.

First, we compare the best versions of our algorithms on the benchmark set in section 7.1. Section 7.2 presents the datasets of the five days we take for empirical study. Section 7.4 makes an as-good-as-possible comparison between the existing Picnic FMS and our best algorithm. Lastly, section 7.5 aims to find out whether we should decompose the problem to best solve it, and if so, how.

7.1. Comparing the best versions of the different algorithms

In this section we take the best configurations of the three presented algorithms in terms of average objective and robustness across the different instances, and compare their performance in more detail.

The algorithm configurations are presented in Table 7.1, and follow from the results in the previous chapters. Table 7.2 shows the average objective, the average gap to the best found solution (BFS, second column), and the coefficient of variation (CV, as defined in Chapter 5) for these configurations as a means to compare the three. Interestingly, but in line with earlier resutls, the BFSs of the three largest instances are actually found by solving two parts of a partition separately.

On all instances except *fcB_ri_32* the ALNS+LS outperforms the other two. Furthermore, we observe that ALNS outperforms the matheuristic on the four largest instances, but the matheuristic outperforms ALNS on the smaller ones. On average, we see that ALNS+LS clearly outperforms the other two on the benchmark set, both on average gap and CV. This makes ALNS+LS both the best performing and most consistent and robust algorithm to solve these instances.

Table 7.1: The three algorithm configurations

Algorithm	ξ	w	LS heur.	w_s	route split
ALNS	[0.1, 0.2]	0.01	n/a	n/a	n/a
ALNS+LS	[0.1, 0.2]	0.01	RCE	n/a	n/a
Matheuristic	[0.1, 0.2]	0.00	RCE	10	yes

Table 7.2: Comparing averages of preferred configuration per algorithm. Best found solutions (BFS) across all simulations are presented in the rightmost column. In case the best solution was found by solving a partition of the instance, it is marked with an *. Average gaps to BFS and coefficients of variation (CV) over the 10 runs are also given. Best results per instance are marked **bold**. Objectives are rounded to integer values for readability. Instance names are abbreviated to their size.

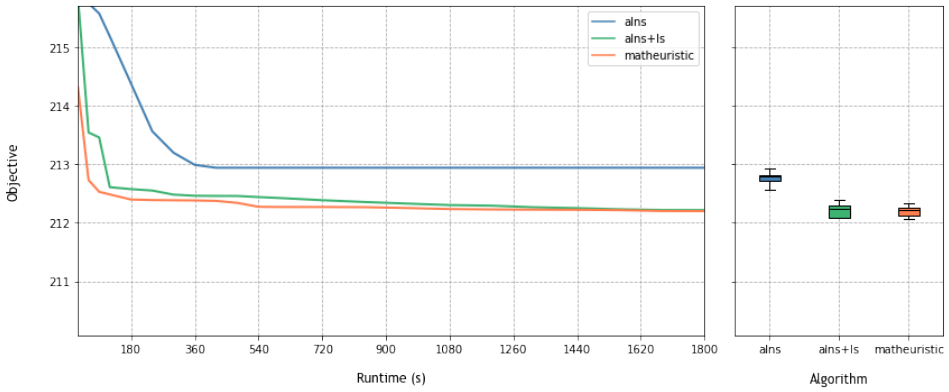
Instance	BFS	ALNS			ALNS+LS			Matheuristic		
		Obj.	CV (x100)	Gap (%)	Obj.	CV (x100)	Gap (%)	Obj.	CV (x100)	Gap (%)
18	115.45	115.64	0.15	0.16	**115.45**	**0.00**	**0.00**	**115.45**	**0.00**	**0.00**
26	169.24	169.56	0.20	0.10	**169.24**	**0.00**	**0.00**	170.32	0.64	0.17
37	212.08	212.76	0.32	0.05	**212.20**	**0.06**	0.05	**212.20**	**0.06**	**0.04**
50	294.68	295.36	0.23	0.09	**294.92**	**0.08**	**0.08**	295.24	0.19	0.14
56	320.88	323.64	0.86	0.15	**321.64**	**0.24**	**0.09**	321.72	0.26	0.16
67	396.00	399.52	0.87	0.27	**397.04**	**0.26**	**0.14**	398.76	0.70	0.16
87	506.76*	511.96	1.01	0.42	**510.28**	**0.69**	0.70	512.16	1.07	**0.30**
123	716.88*	733.20	2.28	0.35	**723.80**	**0.97**	**0.22**	735.60	2.61	1.67
32	237.68	239.16	0.62	0.16	238.80	0.47	0.21	**237.96**	**0.12**	**0.02**
44	341.24	344.68	1.01	0.39	**342.36**	**0.47**	**0.16**	342.92	0.49	0.23
49	267.08	271.68	1.72	0.38	**269.92**	**1.06**	**0.26**	270.88	1.57	1.07
61	334.00	339.36	1.63	0.39	**336.04**	**0.60**	**0.25**	337.96	1.16	0.52
81	505.68	512.12	1.27	0.56	**509.68**	**0.79**	**0.39**	515.12	1.87	0.80
105	675.24*	693.56	2.71	**0.40**	**686.08**	**1.61**	0.42	695.76	3.04	1.07
avg.	-	-	1.06	0.28	-	**0.52**	**0.21**	-	0.98	0.45

We dive into the results further by analysing the average objective throughout time for some of the instances in Figure 7.1. We observe that the matheuristic performs even slightly better on the *fc46_r_37* instance than the ALNS+LS, whereas ALNS lags behind. This is most likely because the resource constraint weighs less heavily on such a small instance. For the two larger instances in Figures 7.1b and 7.1c, the matheuristic performs increasingly worse. The schedules obtained by the routing step are increasingly less likely to be feasible, which takes its toll on performance. The boxplots show that the variation for the solutions by the matheuristic increases with instance size, which shows less robustness for such instances. The algorithm still seems to have trouble guiding towards good feasible solutions consistently, likely because the scheduling on resources becomes more intricate and the constraint weighs heavier. Finally, it must be noted that these results exclude the failed runs for the matheuristic.

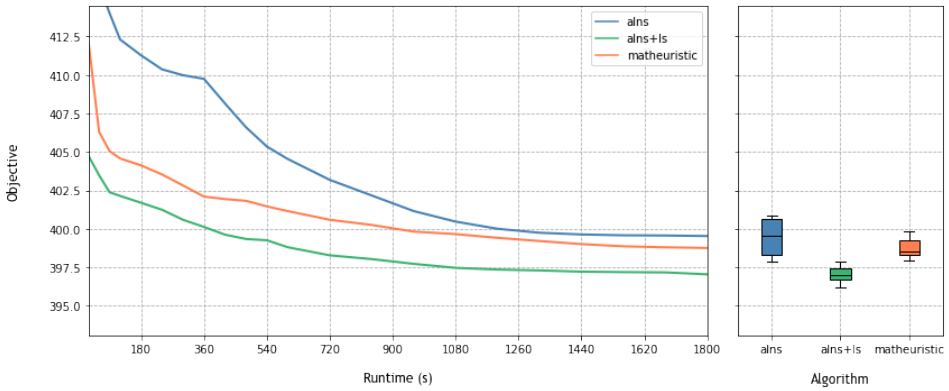
The performances of ALNS and ALNS+LS are rather consistent across the instances, with ALNS+LS steadily outperforming ALNS. Overall, the ALNS+LS with this configuration gives the best and most consistent results on the benchmark set, so we choose to perform the final set of experiments with this algorithm. For the remainder of this chapter, the ALNS+LS algorithm is referred to as *the algorithm*.

7.2. Five full problem instances

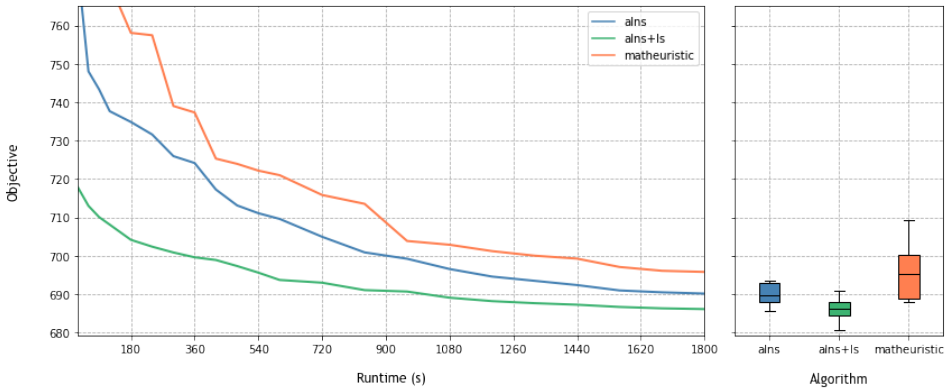
The reader is referred to the Thesis Confidential Supplement *for this section, which presents the instances for Day 1 through Day 5.*



(a) fc46_r_37



(b) fc127_mr_67



(c) fc135_mri_105

Figure 7.1: Average cumulative objective and boxplots for the three algorithms on three instances

Table 7.3: Single FC, Regular Outbound, Day 1. Worst solutions across 4 runs. The last column states whether the solver found the same solution. This is marked **bold** when the solution is proven optimal.

Instance	FMS	ALNS+LS without MH	Gap (%)	ALNS+LS with MH	Gap (%)	Equal to BFS CP?
Day 1 FCA	96.78	95.71	-1.1	93.36	-3.5	**yes**
Day 1 FCB	193.24	182.80	-5.4	179.01	-7.4	yes
Day 1 FCC	143.14	134.90	-5.8	134.90	-5.8	yes
Day 1 FCD	133.84	131.75	-1.6	128.22	-4.2	yes
Day 1 FCE	75.76	66.83	-12.4	66.83	-12.4	**yes**
Day 1 FCF	109.10	103.08	-5.5	101.32	-7.1	yes
Day 1 FCG	120.28	115.45	-4.0	115.45	-4.0	yes
Total	872.13	830.52	-4.8	819.09	-6.1	-

7.3. Solving small instances using the CP solver

In the upcoming sections, small instances are also evaluated using the CP solver. As mentioned in section 3.2, the model was coded in MiniZinc and solved using the state-of-the-art Google OR-Tools solver. Free search was enabled as this is recommended by Google, and the model was run from the same machine as the other experiments, using 8 cores in parallel. The results for the solver in this chapter were obtained after 600s.

7.4. Comparison to FMS

To make the comparison between FMS and our algorithm as fair as possible, we need to take the following into account:

- FMS only generates feasible schedules for Regular Outbounds, so we run our algorithm on Regular Outbounds only too.
- FMS does not take minimum route duration into account, but this is an integral part of our objective function. We run our algorithm with this objective function, but evaluate the final cost without taking the minimum route duration into account
- FMS can only combine shipments for a few predefined pairs of hubs, whereas our algorithm can do so for any two hubs. For this comparison, we disable such *multihub shipments (MH)* for FMS, and run our algorithm with and without multihub shipments enabled.

We ran our ALNS+LS algorithm on these instances in the configuration presented in section 7.1, but with $\xi = [0.1, 0.4]$, since a larger value for ξ is beneficial for small instances as previous experiments showed. We did four runs of the algorithm for 300s each, on each instance. For each instance, with an without multihubs, we observed very consistent results. These are shown in Table 7.3 for Day 1, where we report the worst result across the four runs. The results for the other days can be found in Table B.1. Moreover, CP solver finds the same results in 600s for all instances. Some of these are also proven optimal. We observe that our algorithm outperforms FMS by 3.7-5.8% without multihub shipments, and 5.6-10.4% with multihub shipments. As discussed in section 2.6, a few notions have to be made regarding this comparison:

- FMS has the freedom to change the picking line order to favour the routing process. We use a rule of thumb to determine the picking line order (see section 2.6) which we keep fixed. All in all, we have one fewer degree of freedom, which speaks in our favour.
- FMS discretises time to 5-minute time steps. However, it rounds both down and up, so we cannot say for sure how this influences the results.

Table 7.4: Single FC, Regular + Morning Outbound, Day 1, 10 runs each. BFS CP is marked **bold** if proven optimal.

Instance	n	ALNS+LS			BFS CP
		Avg	Best	Worst	
Day 1 FCA	15	119.38	119.38	119.38	119.38
Day 1 FCB	35	284.75	284.69	284.82	285.38
Day 1 FCC	31	177.59	177.39	177.95	178.33
Day 1 FCD	20	153.48	153.48	153.48	153.48
Day 1 FCE	17	105.35	105.35	105.35	**105.35**
Day 1 FCF	30	160.58	160.41	160.73	161.26
Day 1 FCG	26	169.23	169.23	169.29	169.29

- FMS applies longer driving durations during busy e.g. rush hour, which may skew the results in our favour. We do not evaluate the full impact, but we estimate that correcting for this would result in a gap which is 1 or 2 percentage points smaller.

7.5. Full instances and decompositions

Now that we have shown that we outperform FMS on single-FC Regular Outbound instances, we intend to find out how we can best solve the full instances. The first step we take is to add Morning Outbounds to the Regular Outbounds. Picnic currently does this by hand, and unfortunately we cannot compare this planning to the solutions we obtain. Instead, we run our algorithm on single FC, Morning + Regular Outbound instances to see how consistent our results are there. We also run the CP solver on these instances. Table 7.4 shows these results for Day 1. Note that from now on, we always enable multihub shipments, and all runs of our algorithm are limited at 30 minutes again. We also evaluate all results **with** the minimum route duration again.

We see that the three smallest instances (FCs A, D and E) are solved to the same result 10 out of 10 times, and also equals the BFS found by the CP solver. All other instances are solved to within at most 0.32% between the best and worst solution, showing consistent performance across the different runs. They also outperform the CP solver 10 out of 10 times.

We will now consider combining single-FC instances. We argued before how combining instances could give more combinatorial opportunities, but could also make the problem (too) complex to solve. We consider partitions across FCs of the full instance of different sizes, based on what is geographically sensible. The partitions we consider are presented in Table 7.5. We consider three combinations of flows, namely Morning + Regular Outbound **(MR)**, Inbound only **(I)**, and all three together **(MRI)**. It is interesting to see whether it is beneficial to combine Inbounds and Outbounds, or keep them separate. Separating Morning from Regular Outbounds makes no sense operationally, regardless of whether we add Inbounds to that. We consider a subset of the partitions for each combination of flows, which we present in Table 7.6.

Table 7.5: Partitions over FCs

	FC partitions	Explanation
I	**{A, B, C, D, E, F, G}**	The full instance
II	**{A, C, D, E}, {B, F, G}**	Split into a northern and southern half
III	**{A, C, F, G}, {B, D, E}**	Split into an eastern and western half
IV	**{D, E}, {A, C}, {F, G}, {B}**	A geographically sensible clustering into 4 parts
V	**{A}, {B}, {C}, {D}, {E}, {F}, {G}**	All separate

For each partition, we run the algorithm on each subset for 30 minutes. We then add the results per partition to see which yields the best total result. Even though we effectively give more computation time to the partitions with more subsets, this is the most relevant comparison for Picnic: the different parts can be run in parallel, while a joint instance cannot be parallelised.

We present the average cumulative objective throughout the 30 minutes for all flows and corresponding partitions of Day 1 and Day 3 in Figures 7.2a through 7.2f. The results for the other days can be found in Figures B.1a - B.1i. Table 7.7 summarises the 30 minute final results for all days, flows and partitions. We walk through the three flow combinations in turn.

- **MR**: Figures 7.2a and 7.2b show the performance on the different partitions through time for the Outbound instances. Clearly, evaluating single FCs (partition V) is not the way to go. The instances are too small to be able to find efficient schedules. We observed in the generated schedules that the minimum duration is often not reached, which creates additional costs. Table 7.7 shows how partition III is always (close to) the best partition for MR. This east-west split is apparently beneficial over the north-south split.
- **I**: Table 7.7 suggests that the three partitions evaluated for Inbounds yield results that lie close together. Figures 7.2c and 7.2d do show that partitions II and III yield better solutions more quickly than partition I. This is most likely because partition I is harder to solve because it is generally twice as large.
- **MRI**: The integrated instances show a very clear trend, stating that partition IV outperforms the others 4 out of 5 times, and is very close on Day 3. Furthermore, Figures 7.2e, 7.2f and Appendix figures show how partition IV outperforms the other regardless of how long the algorithm is running. In other words, they are very robust with respect to runtime. Apparently, larger subsets of FCs become too hard to solve, whereas single FCs suffer from too few combinatorial possibilities.

Table 7.6: Considered partitions per flow

Flows	Acronym	Considered partitions	Remarks
Morning + Regular Outbounds	MR	I, II, III, IV, V	-
Inbounds	I	I, II, III	Partitions IV and V will generally give too small instances to efficiently solve
All Outbounds + Inbounds	MRI	II, III, IV, V	The full instance proved to be too hard to solve and gave no meaningful results. A single iteration generally takes over 60s.

Table 7.7: Relative performance after 30min of each partition, for MOB/OB, IB and MOB/OB/IB, for the 5 days. Best partition is marked with a -. Values indicate percentual difference compared to best partition.

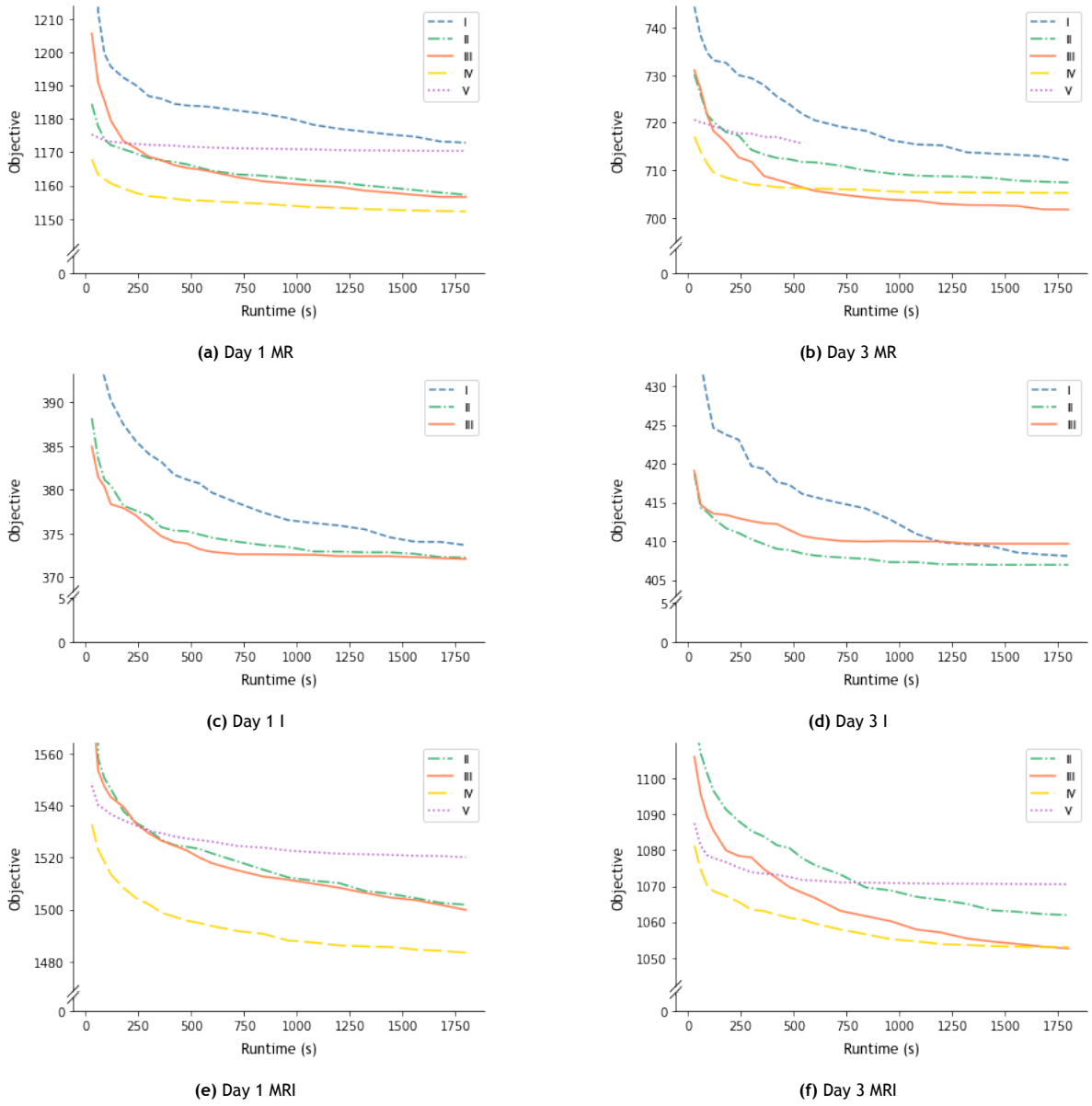
Date	MR					I			MRI			
	I	II	III	IV	V	I	II	III	II	III	IV	V
Day 1	1.79	0.43	0.37	-	1.57	0.43	0.05	-	1.24	1.11	-	2.47
Day 2	0.90	0.84	-	2.17	4.21	1.88	-	0.62	1.25	0.13	-	1.54
Day 3	1.48	0.81	-	0.50	1.99	0.28	-	0.67	0.89	-	0.04	1.70
Day 4	0.82	0.54	-	1.40	2.58	-	0.62	1.63	1.58	0.40	-	1.35
Day 5	1.66	0.40	0.08	-	1.50	0.23	-	0.69	1.28	0.36	-	1.19

7.5.1. Incorporating Inbounds vs. keeping them separate

As a final step in our evaluation, we want to see whether it pays off to solve Inbound instances separately, or integrate them in the Outbound instances. We take for each day the best strategy (i.e. partition(s)) for MR + I, and MRI, and see how the summed objectives relate to one another. Table 7.8 shows the best strategy per day for both combinations, the (summed) objectives and the percentual difference, with the separate instance as a base. MRI clearly outperforms MR + I on all days. Interestingly, there appears to be a positive correlation between the instance size and this difference. This suggests that integrating Inbounds when the number of MR shipments per FC is lower pays off more. Also, a lack of MOB shipments could cause more schedules to be shorter than the minimum route duration.

Table 7.8: Comparing Inbound and Outbound separated vs. combined

Date	MR + I		MRI		Diff (%)
	Best strategy	Obj.	Best strategy	Obj.	
Day 1	IV + III	1524.28	IV	1483.40	-2.68
Day 2	III + II	1372.08	IV	1307.32	-4.72
Day 3	III + II	1108.68	III	1052.56	-5.06
Day 4	III + I	1476.56	IV	1410.96	-4.44
Day 5	IV + II	1649.16	IV	1597.80	-3.11



(a) Day 1 MR

(b) Day 3 MR

(c) Day 1 I

(d) Day 3 I

(e) Day 1 MRI

(f) Day 3 MRI

Figure 7.2: Average cumulative objective for partitions, Day 1 and Day 3

7.6. Concluding remarks

This chapter examined the multiple algorithmic approaches developed by means of benchmarking on actual data from Picnic. We paid attention to also making a comparison with the existing Picnic algorithm as fairly as possible.

- Section 7.1 shows that the ALNS+LS algorithm outperforms the ALNS and the matheuristic on almost all benchmark set instances. We therefore use this algorithm in our final analysis.
- Section 7.3 recalls that the exact CP approach does not scale.
- Section 7.4 shows that the ALNS+LS algorithm outperforms Picnic's current FMS by 4.8%. We do present a few reasons as to why this comparison is not watertight.
- Section 7.5 shows that the ALNS+LS algorithm can solve single FC MR instances very consistently, showing that we do not seem to lose much performance as a result of the instance size growing. We argued before why combining MOB with OB makes sense.
- Section 7.5 also shows that combining single-FC instances into multi-FC instances can yield better results. Solving all FCs together proves to be too hard a task, so making smart partitions is advantageous.
- Section 7.5 ultimately shows that partition IV on the MRI instances yields the best results overall, outperforming MR + I by 2.68–5.06% on the five days we analysed.

8

Discussion

Having presented our algorithms and the design decisions behind it, and seen empirically its performance, we now take the opportunity to reflect upon the work. We address each algorithmic approach in turn, discuss its advantages and drawbacks, and assess where we might have done things differently with the knowledge at the time of writing. We also discuss our final empirical evaluation.

8.1. ALNS

We opted for the ALNS in the first place since it was often used for Pickup and Delivery Problems in literature. Especially problems that lie conceptually close to ours such as Grimault et al. [24] and Sarasola and Doerner [46] used such methods also. It was however halfway through our research that we came across these papers. By that time, we had already independently come up with our method for dealing with the resource constraint. This method seems to be very close to that of Sarasola and Doerner, but their paper does not present the technical details to an extent that we can be sure. We contacted Grimault, who was willing to help but no longer had his source code. We argued how our method compares to Grimault's, and decided that while there is scientific value in experimental comparison, we preferred to focus on more novel approaches instead.

Regarding the ALNS itself, Shaw [50] showed in his original paper how the framework lends itself well for incorporating additional constraints. Looking ahead at how Picnic may build on this algorithm in the future (see section 9.2) makes this approach a favourable one.

We did some experiments regarding parameter optimisation, where we found that the value for the destroy parameter strongly influenced performance. Results were in line with earlier research. Simulated annealing proved to be less effective for us. In fact, disabling it led to significantly better solutions early on, but did run the risk of getting stuck in local optima. A lower initial temperature compared to related research proved to be the better trade-off.

There are many more parameters associated to the ALNS, which we chose not to investigate specifically. Opinions on the use in doing so differ throughout the literature, with some showing considerable improvements by tuning these, where others show no significant influence whatsoever. We deemed such investigations less interesting than venturing in different research directions.

8.2. ALNS+LS

In an attempt to improve on the ALNS algorithm, we opted to include local search heuristics. These are less often used for PDPs, likely because many VRP LS operators would become less effective, or even ineffective on PDPs. Note how a crossover operation can only be performed when both the pickup node and the delivery node remain in the same route segment. In other words, a crossover can only be performed at a point where the truck is empty, which for many PDPs is hardly the case. Since it is the case for our problem, we can employ it. We also argued how other operators can become useful from a node-based perspective. This suggests that Savelsbergh's efficient local search evaluation is no longer possible, or at least requires reworking. Since this technique becomes even harder to use when routes are interdependent under the resource constraint, we did not attempt to incorporate these techniques. This may in fact be subject to research on its own. It does mean that the local search phase is costly: upon each LS iteration, the costs for all possible moves have to be reevaluated. However, we do observe a better performance under the same runtime limits, which demonstrates the power of local search heuristics.

As we had the ALNS algorithm as a basis, we incorporated local search in the existing framework. The ALNS provided a way to obtain an initial solution, and to escape local optima. We did not consider alternate techniques, such as Tabu Search or Guided Local Search, but these could very well be applicable too. Our hypothesis is that it would not significantly improve the results, but we have no evidence to back this up.

The ablation study we performed on the three local search operators showed that especially the 'cheap' relocate and crossover operators together are very close in terms of performance to using all three. This supports our decision to leave out other operators with higher complexity, but we cannot guarantee that our set of operators is optimal.

8.3. Matheuristic

The matheuristic was inspired by Grangier et al. [23], who relaxed the resource constraint on their transfer point, to later impose it using a CP model. We transferred this idea to our problem with resource constraints on all locations. We showed that doing this naively would not give satisfactory results, but we proposed to alterations to improve this. The rationale behind this was to guide the candidate solutions to be more like feasible solutions without actually imposing the constraint. We argued that it is important to keep the problem complexity low, or we would lose the speedup we got from the relaxation. Both route splitting and slack penalties serve this purpose, but there may be other innovations that can do so. Especially large instances remain difficult to solve.

We opted for a CP approach because the subproblem of resource scheduling was highly constrained by routing precedences, making CP suitable according to its strengths. Further, we were familiar with the technique as we tried solving the full problem with CP too. One could try other techniques such as MIP, but any exact technique is unlikely to meet the runtime constraints. We saw that, while the CP solver was generally quick (within tenths of a second), much time was lost in setting up the solver each time, and in other overhead. A scheduling heuristic could also be a faster alternative, but this suffers from potentially rejecting feasible solutions.

Finally, we note that as we relaxed the problem for the ALNS+LS step, Savelsbergh's constant time evaluation techniques could be implemented for further speed-up. While this is definitely worthwhile, we believe that improving solution quality for large instances is not achieved by such speed-up only.

8.4. Final evaluation

In our final evaluation, we first observe that ALNS+LS yields the best results on average and with least variation. It outperforms ALNS on all benchmark instances, which is not necessarily as expected, since ALNS shows to be very effective throughout literature. We do believe that greedily applying local search heuristics after an ALNS step can only improve the solutions, and doing this rather than doing many more ALNS-only steps is apparently beneficial. We cannot rule out the possibility that our ALNS heuristics are not tailored optimally to our problem, even though they are largely based on related research. It could even be that our non-linear objective function plays a role in this; this possibility could be investigated.

The matheuristic comes remarkably close to the ALNS+LS's performance on smaller instances of the benchmark set. It starts to lag behind on larger ones. For such instances, the increased number of routes and shipments inherently increase the number of precedence constraints in the CP model. With this growing number, a candidate solution is ought to be less likely to be feasible.

Regarding solving full Picnic instances, we tested several static decompositions of the problem, and found that groups of two FCs would give the best results. We did not exhaust the space of possible partitions, but rather chose a subset based on common sense. Combining shipments from FCs that lie close together should give the best combinatorial potential. We propose ways to improve on this in the next chapter.

9

Conclusions and recommendations

We end this thesis by presenting the most important conclusions of our work in section 9.1, and by making recommendations both to Picnic and for future research in sections 9.2 and 9.3.

9.1. Conclusions

Picnic's fast-growing supply chain and increasingly challenging truck planning problem served as a motivation to write this thesis. We defined our research goal as:

Develop an algorithm that finds a solution to Picnic's truck planning problem with the lowest possible costs within 30 minutes

We walk through the subquestions as posed in Chapter 1 in order:

- **Which vehicle routing/scheduling model suits Picnic's use case best, and how should such existing models be extended to account for specific constraints from Picnic's operations?**
  We found that the use case was best described by a Pickup and Delivery Problem formulation. This formulation prescribes that, in addition to regular VRP constraints, a shipment is described by two nodes, such that the pickup node is service before the delivery node by the same vehicle. The resource constraint we consider causes routes in a solution to become interdependent, which adds to the problem complexity. Other intra-route constraints could be added more easily.
- **What exact solving technique, if any, can be used to obtain a good solution to the problem in the specified time?**
  We defined an exact CP model in Chapter 2, which we implemented in MiniZinc and attempted to solve. In Chapter 7 we used this model to obtain solutions to small instances, which sometimes equalled our best heuristic results. Performance quickly degraded for larger instances, making this approach not useful in the end.
- **What inexact solving technique(s) can be used to obtain a good solution to the problem in the specified time?**
  We proposed three heuristic approaches to solve our problem. The ALNS was is closely related to that of Grimault et al. [24], whereas the ALNS+LS and matheuristic applied techniques used on less related problems. We found that the ALNS+LS algorithm performs consistently as good or better than the other two approaches, on a representative benchmark set of 14 real-life instances.

- **Can we solve full instances effectively, or does it pay off to decompose the problem into subproblems and solve those separately?**
  First of all, we show that, albeit under many assumptions, we outperform Picnic's current practice on a subset of the shipments. Ultimately, when looking at full instances, we found that solving a decomposition of the studied instances often yields better results than solving the full instances at once. We attribute this to the trade-off between increasing problem size and decreasing combinatorial potential when combining more FCs. We finally found that integrating the different transport flows (Inbounds and Outbounds) yield 2.6–5.0% better results on the instance set compared to keeping these separated.

Coming back to our main goal, we conclude that our ALNS+LS approach on a decomposition of the problem yields the best results, and outperforms Picnic's current approach. Local search methods are not often applied to PDPs, but we showed that under some conditions they can still be powerful.

9.2. Recommendations to Picnic

The reader is referred to the Thesis Confidential Supplement *for this section.*

9.3. Future research

All points mentioned in the previous section would open up new research areas to some extent, or tie in with existing ones. Besides these, we discuss future research areas more closely related to the algorithms we designed and implemented for this thesis.

- **Comparison to existing research**
  The ALNS with the resource constraint we employ is very closely related to that by Grimault et al. [24], but we did not have the opportunity to compare against it.

- **Further experiments on configurations**
  Due to limited time, we chose to focus on only a subset of the tunable parameters and configurations that the algorithms (can) have. A good example would be to assess the performance of the different LNS heuristics, as well as the influence of the adaptive aspect. The same goes for trying other local search heuristics in that part of the algorithm.
- **Improving the matheuristic**
  While the matheuristic we present shows promising results on smaller instances, it can be improved in several ways still. First of all, since the resource constraint is removed from the ALNS+LS step, we could incorporate Savelsbergh's [48] techniques for more efficient local search. Also, the CP solver still takes considerable time, mostly due to overhead. Different solvers, possibly integrated in the code base, could be considered, as well as heuristic scheduling methods.

  The slack penalties we introduce positively influence the results, but our analysis is limited to only a few values. More research can be put into these values, possibly making them a function of problem parameters, or adaptive based on performance.

  Finally, recall that the greatest drawback of the matheuristic is that it does not guarantee a feasible solution. Hybridising the matheuristic with resource-constrained ALNS could solve this, and might even yield better overall solutions.
- **Solving full instances**
  We ultimately found that our full problem instances were solved best by decomposing them into smaller chunks. This is effectively a manual way of clustering shipments, which leads to believe that this could be done automatically. Fixing clusters manually may not be robust to all

(future) instances, whereas an automated clustering procedure could account for differences in instance characteristics.

In a similar vein, an ALNS algorithm could for each iteration cluster its routes in k clusters, and apply its heuristics per cluster, rather than on the whole problem. This process could also be parallellised.

References

[1] Claudia Archetti and M Grazia Speranza. “A survey on matheuristics for routing problems”. In: *EURO Journal on Computational Optimization* 2.4 (2014), pp. 223–246.

[2] Russell Bent and Pascal Van Hentenryck. “A two-stage hybrid algorithm for pickup and delivery vehicle routing problems with time windows”. In: *Computers & Operations Research* 33.4 (2006), pp. 875–893.

[3] Marco A Boschetti et al. “Matheuristics: Optimization, simulation and control”. In: *International Workshop on Hybrid Metaheuristics*. Springer. 2009, pp. 171–177.

[4] Kris Braekers, Katrien Ramaekers, and Inneke Van Nieuwenhuyse. “The vehicle routing problem: State of the art classification and review”. In: *Computers & Industrial Engineering* 99 (2016), pp. 300–313.

[5] Olli Bräysy and Michel Gendreau. “Vehicle routing problem with time windows, Part I: Route construction and local search algorithms”. In: *Transportation science* 39.1 (2005), pp. 104–118.

[6] Jan Christiaens and Greet Vanden Berghe. “Slack induction by string removals for vehicle routing problems”. In: *Transportation Science* 54.2 (2020), pp. 417–433.

[7] Jean-Francois Cordeau and Québec) Groupe d’études et de recherche en analyse des décisions (Montréal. *The VRP with time windows*. Groupe d’études et de recherche en analyse des décisions Montréal, 2000.

[8] Jean-François Cordeau, Michel Gendreau, and Gilbert Laporte. “A tabu search heuristic for periodic and multi-depot vehicle routing problems”. In: *Networks: An International Journal* 30.2 (1997), pp. 105–119.

[9] Timothy Curtois et al. “Large neighbourhood search with adaptive guided ejection search for the pickup and delivery problem with time windows”. In: *EURO Journal on Transportation and Logistics* 7.2 (2018), pp. 151–192.

[10] George B Dantzig and John H Ramser. “The truck dispatching problem”. In: *Management science* 6.1 (1959), pp. 80–91.

[11] Emrah Demir, Tolga Bektaş, and Gilbert Laporte. “An adaptive large neighborhood search heuristic for the pollution-routing problem”. In: *European Journal of Operational Research* 223.2 (2012), pp. 346–359.

[12] Jacques Desrosiers et al. “Vehicle routing with full loads”. In: *Computers & Operations Research* 15.3 (1988), pp. 219–226.

[13] Michael Drexl. “Synchronization in vehicle routing—a survey of VRPs with multiple synchronization constraints”. In: *Transportation Science* 46.3 (2012), pp. 297–316.

[14] Yvan Dumas, Jacques Desrosiers, and Francois Soumis. “The pickup and delivery problem with time windows”. In: *European journal of operational research* 54.1 (1991), pp. 7–22.

[15] Nizar El Hachemi, Michel Gendreau, and Louis-Martin Rousseau. “A heuristic to solve the synchronized log-truck scheduling problem”. In: *Computers & Operations Research* 40.3 (2013). Transport Scheduling, pp. 666–673. issn: 0305-0548. doi: https://doi.org/10.1016/j.cor.2011.02.002. url: https://www.sciencedirect.com/science/article/pii/S0305054811000426.

[16] Raafat Elshaer and Hadeer Awad. “A taxonomic review of metaheuristic algorithms for solving the vehicle routing problem and its variants”. In: *Computers & Industrial Engineering* 140 (2020), p. 106242.

[17] Aurélien Froger et al. “A matheuristic for the electric vehicle routing problem with capacitated charging stations”. PhD thesis. Centre interuniversitaire de recherche sur les reseaux d’entreprise, la …, 2017.

[18] Birger Funke, Tore Grünert, and Stefan Irnich. “Local search for vehicle routing and scheduling problems: Review and conceptual integration”. In: *Journal of heuristics* 11.4 (2005), pp. 267–306.

[19] Asvin Goel. “Truck driver scheduling in the European Union”. In: *Transportation Science* 44.4 (2010), pp. 429–441.

[20] Bruce Golden et al. “The fleet size and mix vehicle routing problem”. In: *Computers & Operations Research* 11.1 (1984), pp. 49–66.

[21] *Google OR-Tools*. https://developers.google.com/optimization. Accessed: 2022-03-30.

[22] Philippe Grangier et al. “A matheuristic based on large neighborhood search for the vehicle routing problem with cross-docking”. In: *Computers & Operations Research* 84 (2017), pp. 116–126. issn: 0305-0548. doi: https://doi.org/10.1016/j.cor.2017.03.004. url: https://www.sciencedirect.com/science/article/pii/S0305054817300631.

[23] Philippe Grangier et al. “The vehicle routing problem with cross-docking and resource constraints”. In: *Journal of Heuristics* 27.1 (2021), pp. 31–61.

[24] Axel Grimault, Nathalie Bostel, and Fabien Lehuédé. “An adaptive large neighborhood search for the full truckload pickup and delivery problem with resource synchronization”. In: *Computers & Operations Research* 88 (2017), pp. 1–14.

[25] Christoph Hempsch and Stefan Irnich. “Vehicle routing problems with inter-tour resource constraints”. In: *The vehicle routing problem: latest advances and new challenges*. Springer, 2008, pp. 421–444.

[26] Hossein Hojabri et al. “Large neighborhood search with constraint programming for a vehicle routing problem with synchronization constraints”. In: *Computers & Operations Research* 92 (2018), pp. 87–97.

[27] *Integer programming*. https://en.wikipedia.org/wiki/Integer_programming. Accessed: 2022-03-30.

[28] Philip Kilby. “Tutorial: Constraint Programming for the Vehicle Routing Problem (slides)”. In: Sept. 2013.

[29] Scott Kirkpatrick, C Daniel Gelatt Jr, and Mario P Vecchi. “Optimization by simulated annealing”. In: *science* 220.4598 (1983), pp. 671–680.

[30] Adrianus Leendert Kok et al. “A dynamic programming heuristic for the vehicle routing problem with time windows and European Community social legislation”. In: *Transportation Science* 44.4 (2010), pp. 442–454.

[31] Yiannis A Koskosidis, Warren B Powell, and Marius M Solomon. “An optimization-based heuristic for vehicle routing and scheduling with soft time window constraints”. In: *Transportation science* 26.2 (1992), pp. 69–85.

[32] Raphael Kramer et al. “A matheuristic approach for the pollution-routing problem”. In: *European Journal of Operational Research* 243.2 (2015), pp. 523–539.

[33] Gilbert Laporte. “Optimal solutions to capacitated multidepot vehicle routing problems”. In: *Congressus Nemerantium* 4 (1984), pp. 283–292.

[34] E.L. Lawler. *The Travelling Salesman Problem: A Guided Tour of Combinatorial Optimization*. Wiley-Interscience series in discrete mathematics and optimization. John Wiley & Sons, 1985. url: https://books.google.nl/books?id=qbFlMwEACAAJ.

[35] Fabien Lehuédé et al. “A multi-criteria large neighbourhood search for the transportation of disabled people”. In: *Journal of the Operational Research Society* 65.7 (2014), pp. 983–1000.

[36] Hongping Lim, Andrew Lim, and Brian Rodrigues. “Solving the pickup and delivery problem with time windows using” Squeaky Wheel” optimization with local search”. In: AIS. 2002.

[37] Renaud Masson, Fabien Lehuédé, and Olivier Péton. “Efficient feasibility testing for request insertion in the pickup and delivery problem with transfers”. In: *Operations Research Letters* 41.3 (2013), pp. 211–215.
[38] *MiniZinc*. https://www.minizinc.org/. Accessed: 2022-03-30.
[39] *MiniZinc Challenge*. https://www.minizinc.org/challenge.html. Accessed: 2022-03-30.
[40] Nenad Mladenović and Pierre Hansen. “Variable neighborhood search”. In: *Computers & operations research* 24.11 (1997), pp. 1097–1100.
[41] Ilhan Or. “Traveling salesman-type combinatorial problems and their relation to the logistics of blood banking”. In: *PhD thesis (Department of Industrial Engineering and Management Science, Northwestern University)* (1976).
[42] Sophie N Parragh, Karl F Doerner, and Richard F Hartl. “A survey on pickup and delivery problems”. In: *Journal für Betriebswirtschaft* 58.1 (2008), pp. 21–51.
[43] Ted K Ralphs et al. “On the capacitated vehicle routing problem”. In: *Mathematical programming* 94.2 (2003), pp. 343–359.
[44] Stefan Ropke and David Pisinger. “An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows”. In: *Transportation science* 40.4 (2006), pp. 455–472.
[45] Francesca Rossi, Peter Van Beek, and Toby Walsh. *Handbook of constraint programming*. Elsevier, 2006.
[46] Briseida Sarasola and Karl F Doerner. “Adaptive large neighborhood search for the vehicle routing problem with synchronization constraints at the delivery location”. In: *Networks* 75.1 (2020), pp. 64–85.
[47] Dimitrios Sariklis and Susan Powell. “A heuristic method for the open vehicle routing problem”. In: *Journal of the Operational Research Society* 51.5 (2000), pp. 564–573.
[48] Martin WP Savelsbergh. “The vehicle routing problem with time windows: Minimizing route duration”. In: *ORSA journal on computing* 4.2 (1992), pp. 146–154.
[49] Martin WP Savelsbergh and Marc Sol. “The general pickup and delivery problem”. In: *Transportation science* 29.1 (1995), pp. 17–29.
[50] Paul Shaw. “A new local search algorithm providing high quality solutions to vehicle routing problems”. In: *APES Group, Dept of Computer Science, University of Strathclyde, Glasgow, Scotland, UK* 46 (1997).
[51] Paul Shaw. “Using constraint programming and local search methods to solve vehicle routing problems”. In: *International conference on principles and practice of constraint programming*. Springer. 1998, pp. 417–431.
[52] *Supermarkt & Ruimte*. https://www.supermarktenruimte.nl/wp-content/uploads/2021/05/Supermarktomzet-op-waarde-geschat.pdf. Accessed: 2022-04-05.
[53] Frank A Tillman. “The multiple terminal delivery problem with probabilistic demands”. In: *Transportation Science* 3.3 (1969), pp. 192–204.
[54] Christos Voudouris. “Guided local search for combinatorial optimisation problems.” PhD thesis. University of Essex, 1997.

A

Paper outline

A Multi-Stage Metaheuristic Approach for Multi-Depot Pickup and Delivery Problems with Resource Constraints

A.1. Introduction

Picnic is the fastest growing online(-only) supermarket in the Netherlands [52], and possibly beyond those borders. Picnic also is a quintessential example of a company with an intricate logistical operation, consisting of two stages: the last-mile delivery, i.e. the delivery to the customer, and the truck transport, which covers everything before that. This paper takes the latter as a use case for solving a specific multi-depot pickup and delivery problem with resource constraints.

Picnic's truck planning problem specifically entails creating a truck schedule such that all shipments between its warehouses on a given day are shipped in time. Hundreds of shipments are to be shipped on any given day, which is reasonably large compared to benchmark instances of PDPs in the literature. On top of that, Picnic has to deal with resource constraint at its warehouses. This leads to routes becoming dependent on one another, which adds considerably to the complexity of the problem. Grimault et al. [24] attempt to solve a similar problem, but show how even medium-sized instances are hard to solve.

What makes this use case more specific, is that the distribution network is partly decentralised. The number of pickup locations is limited, and delivery locations are generally close to their pickup locations. The problem can therefore be decomposed rather naturally, which can be taken advantage of. Furthermore, we face several intra-route constraints, including lower and upper bounds on working hours and switching drivers on a truck halfway through the day.

We propose an algorithm that alternates between Adaptive Large Neighbourhood Search and Local Search operators (ALNS+LS), while taking into account the resource constraint using ordered list structures. We compare this to an ALNS-only approach and show how the addition of local search improves solution quality by benchmarking both approaches on a subset of real-life-inspired instances. We also show how full instances are best solved when decomposed and solved separately using ALNS+LS.

The remainder of this paper is structured as follows. Section A.2 explores relevant research in this area, and section A.3 defines the problem formally. Section A.4 describes our solution method in detail, the results are presented in section A.5 and we draw our conclusions in section A.6

A.2. Related work

A condensed version of Chapter 3.

A.3. Problem formulation

Relevant parts of Chapter 2, including section 2.4

A.4. Solution method

- The general ALNS procedure
- The mechanism to impose the resource constraint
- Considerations regarding the additional constraints
- The local search extensions

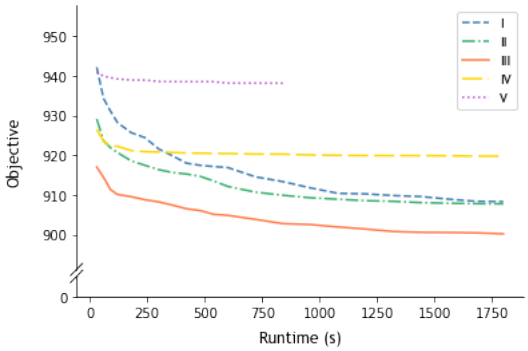
A.5. Computational results

A combination of sections 4.7 and 5.4, and Chapter 7.

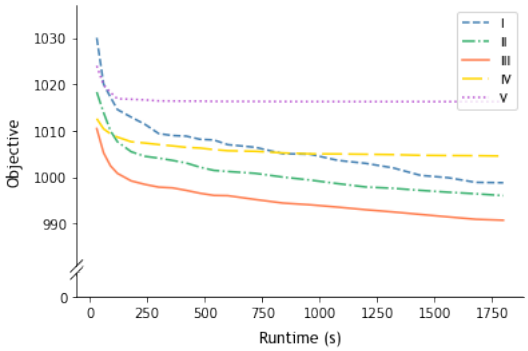
A.6. Conclusion

B

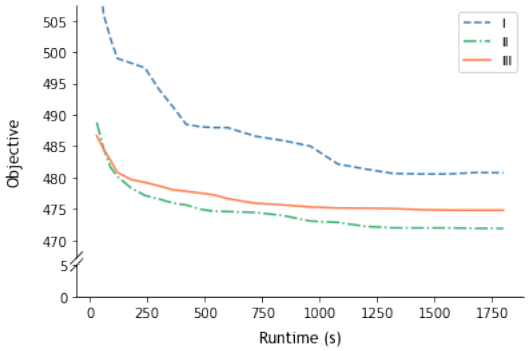
Tables and figures



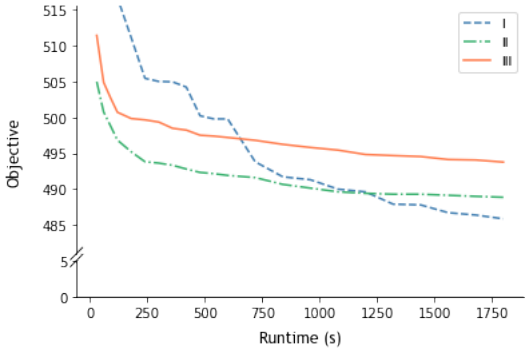
(a) Day 2 MR



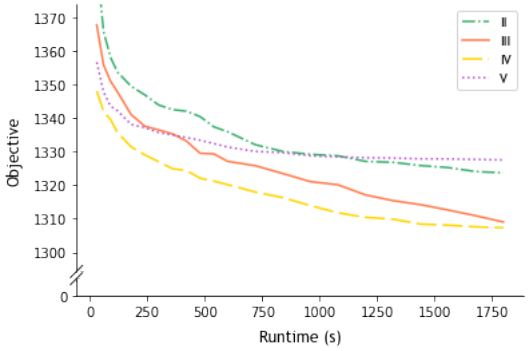
(b) Day 4 MR



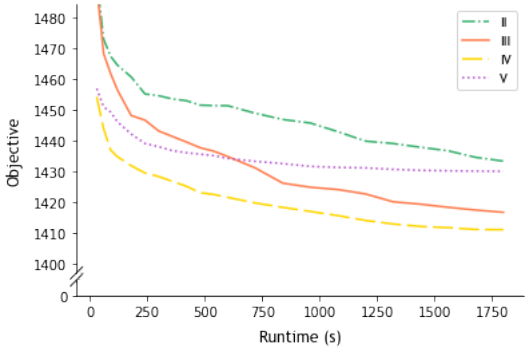
(c) Day 2 I



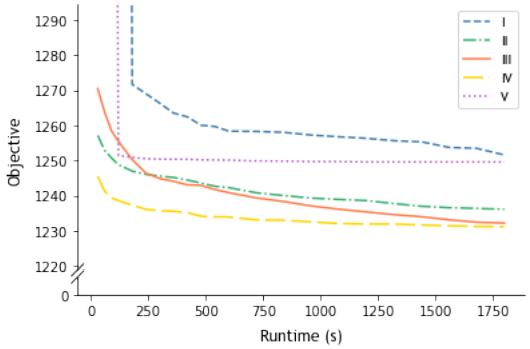
(d) Day 4 I



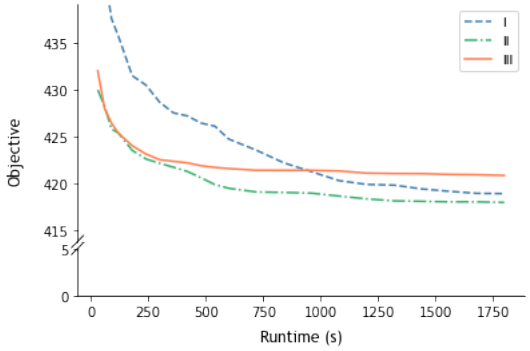
(e) Day 2 MRI



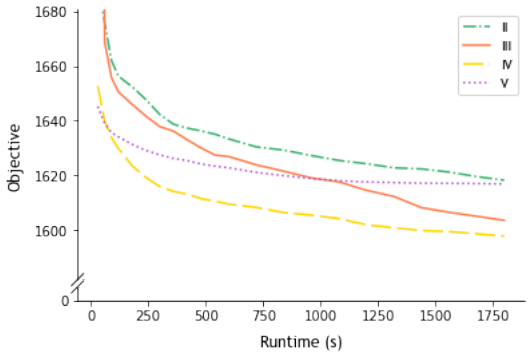
(f) Day 4 MRI



(g) Day 5 MR



(h) Day 5 I



(i) Day 5 MRI

Figure B.1: Average cumulative objective for partitions, Day 2, 4 and 5

Table B.1: Single FC, Regular Outbound, Days 2-5. Worst solutions across 4 runs. The last column states whether the solver found the same solution. This is marked **bold** when the solution is proven optimal.

Instance	FMS	ALNS+LS without MH	Gap (%)	ALNS+LS with MH	Gap (%)	Equal to BFS CP?
Day 2 FCA	92.19	88.84	-3.6	86.67	-6.0	**yes**
Day 2 FCB	157.69	146.76	-6.9	141.45	-10.3	yes
Day 2 FCC	133.10	127.06	-4.5	120.77	-9.3	yes
Day 2 FCD	101.94	99.47	-2.4	99.47	-2.4	**yes**
Day 2 FCE	70.27	69.05	-1.7	69.05	-1.7	**yes**
Day 2 FCF	137.41	135.06	-1.7	133.29	-3.0	yes
Day 2 FCG	104.41	101.62	-2.7	101.62	-2.7	**yes**
Total	797.02	767.86	-3.7	752.32	-5.6	-
Day 3 FCA	99.97	95.96	-4.0	84.95	-15.0	**yes**
Day 3 FCB	164.60	154.26	-6.3	145.83	-11.4	yes
Day 3 FCC	121.31	112.60	-7.2	106.06	-12.6	**yes**
Day 3 FCD	89.44	82.58	-7.7	76.18	-14.8	**yes**
Day 3 FCE	58.39	54.94	-5.9	54.94	-5.9	**yes**
Day 3 FCF	118.51	113.31	-4.4	110.34	-6.9	yes
Day 3 FCG	101.89	97.54	-4.3	97.54	-4.3	**yes**
Total	754.11	711.19	-5.7	675.83	-10.4	-
Day 4 FCA	97.23	94.66	-2.6	92.31	-5.1	**yes**
Day 4 FCC	121.31	112.88	-6.9	108.96	-10.2	yes
Day 4 FCD	117.37	114.14	-2.7	114.14	-2.7	yes
Day 4 FCE	71.77	70.64	-1.6	70.64	-1.6	**yes**
Day 4 FCF	136.16	135.16	-0.7	135.16	-0.7	yes
Day 4 FCG	101.39	93.93	-7.4	93.93	-7.4	**yes**
Total	645.22	621.41	-3.7	615.14	-4.7	-
Day 5 FCA	94.81	91.15	-3.9	87.71	-7.5	**yes**
Day 5 FCB	199.70	184.98	-7.4	184.98	-7.4	yes
Day 5 FCC	129.90	120.25	-7.4	117.56	-9.5	yes
Day 5 FCD	108.07	100.52	-7.0	100.52	-7.0	**yes**
Day 5 FCE	100.45	92.20	-8.2	92.20	-8.2	**yes**
Day 5 FCF	149.42	145.56	-2.6	145.56	-2.6	yes
Day 5 FCG	131.71	126.19	-4.2	126.19	-4.2	yes
Total	914.06	860.86	-5.8	854.73	-6.5	-