Master Thesis Applied Mathematics

Optimizing the transport scheduling of an online grocer

Franka van Dijken

Optimizing the transport scheduling of an online grocer

by

Franka van Dijken

to obtain the degree of Master of Science
at the Delft University of Technology
to be defended publicly on 10th of June 2021.

Committee

Prof.dr. K.I. Aardal
Dr. C. Kraaikamp
Ir. A. Braemer

An electronic version of this thesis is available at `http://repository.tudelft.nl/`.
Cover illustration ©Mila Mochèl

TU Delft Delft University of Technology

Abstract

In this thesis, we consider the Multi Depot Vehicle Scheduling Problem with Time Windows with driver day duration restrictions that addresses the task of assigning a given set of time window shipments to trucks with consideration of practical requirements. The goal of this thesis is to design an algorithm that finds a good solution for the problem within 15 minutes. In the literature, the problem is usually applied to public transport cases. We consider the freight transport application of an online grocer.

We introduce a Multi-Commodity Minimum Network Flow ILP implementation for MDVSPTW with short computation time. Also, based on the Concurrent Scheduler algorithm for MDVSP from the literature, we introduce the Greedy Scheduler algorithm that is able to find a feasible solution within short computation time for our problem. These two implementations form building blocks for the three main algorithms we introduce, that find good feasible solutions for our problem within 15 minutes: the ILP + Greedy algorithm, the Random Search algorithm and the Random Search and Fix algorithm. We incorporate the driver day duration restriction in the three algorithms by either allowing a driver change, that is two drivers executing one truck day, or not, and compare the results. We show that the ILP + Greedy algorithm performs the best. In the case an algorithm that does not involve an ILP is preferred, e.g., for robustness reasons, the Random Search and Fix algorithm performs better than the Random Search algorithm.

Acknowledgements

In September 2021, I embarked on the journey of writing a thesis to obtain the Master of Science in Applied Mathematics at the Delft University of Technology. Its conclusion also marks the end of the student period of my life. I look back on a time that taught me so much and of which most came together in this project: The discipline it takes to execute such a big project; The work-life balance in which you have to cut yourself some slack (work in progress); Taking ownership, initiative and responsibility for your own success; Checking your privileges; Showing resilience in the face of things that do not go down as planned and that you do not control; Being determined to make things exactly right; Finding balance between everything. But perhaps the most important lesson I learned in the past seven years: The work is awesome, but what really matters are the people. That is why I want to thank my committee for their involvement and enthusiasm in a project that we made be. In general, I am thankful that I have been guided by people that have consistently trusted me and have given me the freedom to add my own spices to the project. I thank you for your relentless enthusiasm regarding my work and your motivation to guide me to your best ability.

Karen, without your conscientious feedback, my thesis would have looked completely differently. You taught me that in Mathematics, I do not have to feel guilty for the excessive perfectionism that is often punished in other areas. In addition, your work ethic motivated me to really get everything out of the project that was there to be had.

Arjan, your involvement in the project has far exceeded my expectations. Our weekly meetings were my mind's anchors throughout this process. And your enthusiasm on the subject combined with the power to distil its essence enabled me to take big steps content-wise and come to better final result.

Cor, of course we have not spoken much during the project, but I would like to take the opportunity to dwell on the fact that this thesis would have never been here were it not for you. Your help in my first year of studies made my find my way in Mathematics, which guided my development through the past seven years and now results in me obtaining a Master of Science. Despite it being unsure whether I will continue to pursue Mathematics in an academic sense, it will always be of great importance in my life.

The help and guidance I received from Picnic employees and their willingness to answer my every question have also been a great support during this process.

Finally, I would like to thank my family and friends. Without the unconditional support of my family during my entire education, I would have never been able to put as much focus on Mathematics. I am also thankful for my friends who were always there for me to listen to my mathematical stories or support me in the process. And of course Joris, thank you for all the support, help and understanding. I hereby promise to never again drag you to the office on a Sunday to program together!

Franka van Dijken,
Amsterdam, June 3, 2021

Contents

Abbreviations and notations

Table 1: List of abbreviations in alphabetical order.

Abbreviation	**Meaning**
CSP	Crew Scheduling Problem
DC	Distribution Center
ePV	electric Picnic Vehicle
FC	Fulfillment Center
ILP	Integer Linear Program
IVCSP	Integrated Crew and Vehicle Scheduling Problem
LP	Linear Program
MDVSP	Multi Depot Vehicle Scheduling Problem
MDVSPTW	Multi Depot Vehicle Scheduling Problem with Time Windows
SDVSP	Single Depot Vehicle Scheduling Problem
TMS	Transport Management System
TSN	Time-Space Network
TU	Totally Unimodular
VSP	Vehicle Scheduling Problem
VSPLPR	Vehicle Scheduling Problem with Length of Path Restriction
VSPRTC	Vehicle Scheduling Problem with Route and Time Constraints
VSPTW	Vehicle Scheduling Problem with Time Windows

Table 2: List of mathematical notation in order of appearance.

Notation	Meaning
d_{max}	Maximal driver day duration
d_{min}	Minimal driver day duration
$S = \{s_1, \ldots, s_n\}$	Set of n discrete shipments
d	Depot d
s_i	Discrete shipment
ST_i	Start time of shipment s_i
ET_i	End time of shipment s_i
SL_i	Start location of shipment s_i
EL_i	End location of shipment s_i
$d(L_1, L_2)$	Driving time between location L_1 and L_2
$c : p \mapsto c(p)$	Cost function for truck planning p
A	Left hand side $m \times n$ coefficients matrix coefficients in standard form ILP
$\mathbf{b}$	Right hand side vector in standard form ILP
$\mathbf{x}$	Primal variable vector in standard form ILP
X	Solution space
$\mathbf{c}$	Objective vector in standard form ILP
$\mathbf{x}^*$	Optimal solution vector or optimum
z^*	Value of optimal solution
z_{LP}	Lower bound obtained by LP relaxation
$\underline{z}$	Lower bound of optimal value z^*
$\overline{z}$	Upper bound of optimal value z^*
τ_{max}	Time limit for ILP implementation
δ_{gap}	Bound on optimality gap
$\mathcal{P}(X)$	Powerset of the set X
$\mathcal{N} : X \to \mathcal{P}(X)$	Neighborhood function
$\mathcal{N}(\mathbf{x})$	Neighborhood of $\mathbf{x}$
$\mathbf{x}' \in \mathcal{N}(\mathbf{x})$	Neighbor of $\mathbf{x}$
z'	Objective value of neighbor
T	Temperature in Simulated Annealing
$G = (V, A)$	Graph with vertices V and arcs A
$c : A \to \mathbb{R}_{\geq 0}$	Cost function on arcs
c_{ij}	Cost of arc (i, j)
w_{dt}	Weight of the empty driving time
w_{wt}	Weight of the waiting time
w_{fc}	Fixed costs of adding a new truck to the planning
f_{ij}	Amount of flow through arc (i, j)
u_{ij}	Upper bound on the flow through arc (i, j)
I	Identity matrix
$D = \{d_1, \ldots, d_m\}$	Set of m depots
κ_d	Capacity (number of trucks available) of depot $d \in D$
$G^m = (V^m, A^m)$	Graph consisting of m layers of $G = (V, A)$

$\bar{S} = \{\bar{s_1}, \ldots, \bar{s_n}\}$	Set of n time window shipments
$\bar{s_i}$	Time window shipment
EST_i	Earliest start time of time window shipment $\bar{s_i}$
LST_i	Latest start time of time window shipment $\bar{s_i}$
EET_i	Earliest end time of time window shipment $\bar{s_i}$
LST_i	Latest end time of time window shipment $\bar{s_i}$
δ_s	Step size parameter in time window discretization
δ_m	Maximum number of discrete shipments generated by a single time window shipment parameter in time window discretization
$n_i^{\delta_s}$	Number of discrete shipments generated by $\bar{s_i}$ with step size δ_s
$\bar{\mathbf{S}}$	Set of all possible time window shipments
$\mathbf{S}$	Set of all possible discrete shipments
$p : \bar{\mathbf{S}} \rightarrow \mathcal{P}(\mathbf{S})$	Shipment discretization mapping
n_i	Number of discrete shipments generated by $\bar{s_i}$
$t \in \mathbf{T}$	Task, i.e., sequence of activities
$h \in \mathbf{H}$	Duty, i.e., sequence of tasks
y_h	Decision variable in CSP ILP formulation
A_c	Set of compatibility arcs
A_b	Set of backwards arcs
b_{ij}	Amount of flow through backwards arc (i, j)
$[S]$	Ordered list of shipments
t_{wt}^{max}	Maximal waiting time
s_i^j	Cheapest feasible shipment from $\bar{s_i}$ assigned after s_j
l_e	Last shipment indicator: end time
l_d	Last shipment indicator: duration of truck day with shipment
e_s	First shipment indicator: start time
e_d	First shipment indicator: duration of truck day with shipment
c_{max}	Low-cost truck indicator: maximal costs
δ_{min}	Low-cost truck indicator: minimal duration
I	Number of iterations for Random Search
ϕ_{min}	Minimal number of shipment not fixed after fixing phase
I_1	Number of iterations in Fixing Phase
I_2	Number of iterations in Searching Phase

1

Introduction

The online market for supermarkets is growing. In contrast to physical stores, each individual customer expects to receive their order on a chosen time and location. As a result, it is vital to optimize the complex logistics of the supply chain for the enormous number of deliveries per day. In this thesis, we report on research done on the optimization of transport planning for Picnic Technologies (referred to as *Picnic* from here on), an online grocer that was founded in 2015, and has grown exponentially ever since. Conducting research on this subject is relevant from two perspectives: an industry one and an academic one. For the industry, the impact of improving the efficiency of the supply chain is financially and environmentally interesting. Even though in this thesis the industry perspective comes from Picnic, we expect that similar distribution problems occur in related companies, and that our research is therefore relevant in a broader setting as well. From an academic perspective, the practical restrictions of the problem are of such high complexity that optimizing touches upon the academic literature on mathematical optimization problems and challenges state-of-the-art algorithms to find solutions for these problems. This chapter gives an introduction to the problem and presents the goals of this thesis.

1.1. Picnic's case

The problem we consider is based on Picnic's transport planning and called Picnic's case from here on. In order to completely understand this case, it is important to know the relevant aspects from Picnic's supply chain, the occurring problems and the goals to achieve from a business perspective. This section explains all three.

Figure 1.1: Current Picnic locations in the Netherlands.

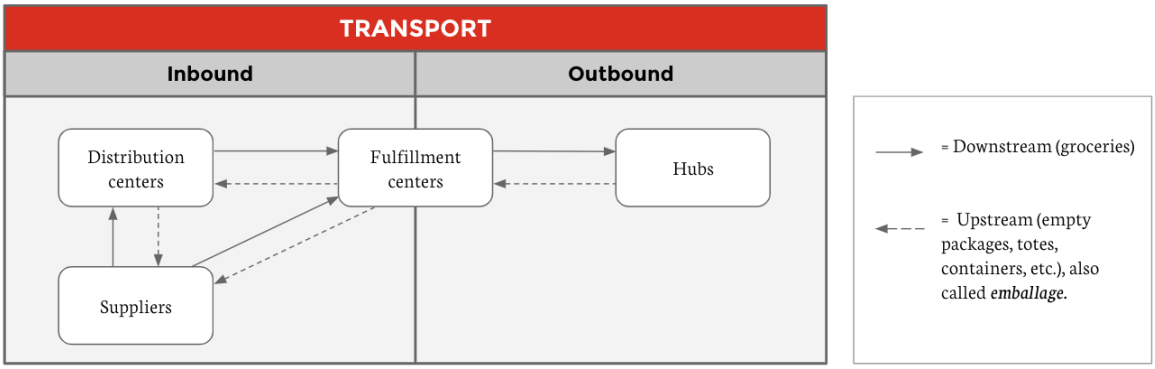


Figure 1.2: Transport in the supply chain of Picnic.

1.1.1. Picnic's supply chain

Picnic is an online grocer that delivers groceries to approximately 300.000 customers in the Netherlands on a daily basis. In order to make sure that the right groceries end up at the right customer in an efficient way, Picnic organizes a complex supply chain. First, the groceries are transported from suppliers to either a Distribution Center (DC) or directly to a Fulfillment Center (FC). In the FCs, the groceries are placed in crates, and sorted into frames. These frames are then shipped from the FC to the Hubs, smaller distribution centers, located near more densely populated places. From there, the frames are loaded into the electric Picnic Vehicles (ePVs), which execute the final part of the supply chain by delivering the groceries to the customers. Figure 1.1 shows an overview of all current DCs, FCs and Hubs that play a part in Picnic's supply chain.

This thesis considers the *transport* part of the supply chain, that is, all shipments be-

tween the DCs, FCs, Hubs and suppliers. Put differently, that is the whole supply chain, except for the part executed by ePVs between Hubs and customers. Figure 1.2 shows that the transport can be divided into two stages: the *inbound* (IB) refers to all shipments executed before the FCs, the *outbound* (OB) refers to all shipments executed after the FCs. All transport is currently executed by transport company *St vd Brink* (referred to as *vdBrink* from here on), that currently has around 60 trucks driving on the road for Picnic every day. A *truck planning* is a schedule that assigns shipments to trucks. The truck plannings are constructed partly by Picnic and partly by vdBrink. Currently, Picnic uses an algorithm to find a good solution for the scheduling of the OB shipments, and the IB shipments are manually processed into the final truck planning by vdBrink.

1.1.2. Motivation

Picnic has grown exponentially over the last 6 years (Rintoul, 2019). As a result, the complexity of the transport flows and the quantity of shipments that are to be planned is increasing, causing transport planning to be an increasingly difficult task. Another factor that complicates the transport planning is the introduction of morning shifts in addition to the three regular shifts in the afternoon. Before this introduction, customers were only able to place their order until 22:00 and receive it the next day during one of the regular shifts. With the introduction of morning shifts, customers are also able to receive their delivery in the morning, if they placed the order before 15:00 the day before. Figure 1.3 shows the conceptual timelines corresponding to one order and delivery cycle from the perspective of the customer, before and after the introduction of the morning shifts. The introduction

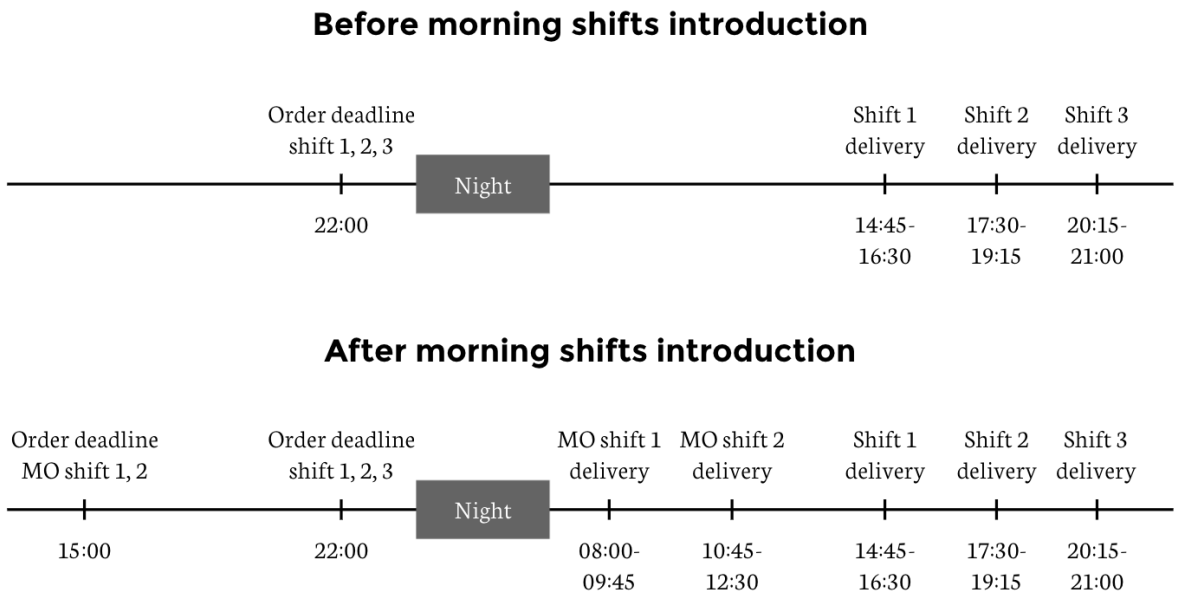


Figure 1.3: One order and delivery cycle before and after the introduction of morning shifts.

of morning shifts has two important consequences for the transport:

- Morning shifts cause the working days of the drivers in transport to become longer than legally permissible, hence a new constraint is required.
- Morning shifts create the need for more than one planning moment per day, as the deadline of the demand for the morning shifts differs from that of the regular shifts. Preferably, it is also possible to plan new shipments on an existing planning, instead of planning all shipments for one day at once.

The current way of transport planning is reaching its limits with respect to the points above. Therefore, Picnic would like to investigate how to design a better system to efficiently plan the transport.

1.1.3. The business goal

The business goal of this thesis is to contribute to a Transport Management System (TMS) for Picnic that is able to plan the transport in a cost-efficient manner. In particular, this thesis focuses on developing an algorithm that efficiently schedules one day of shipments using fewest possible trucks, resulting in a feasible truck planning for that day. The scope is to consider one day at a time. Picnic's shipments are defined by a start location, end location, start time window, end time window and shipment type. The last property is only considered when Picnic-specific details are mentioned. The question for every shipment is: at what time and by which truck should the shipment be executed, such that the total costs of the feasible truck planning are minimized?
Picnic's cost function is very specific because a part of the trucks, also called the *fixed trucks*, is situated at Picnic's depots and the rest is situated at vdBrink in Ermelo. The hourly tariff is decreases if the fixed trucks have a lower average utilization rate. Therefore, a new fixed truck is only placed at a depot if the high utilization rate of this truck can be guaranteed. As a result, the number of fixed trucks are lower than it would have been without this setup of the hourly tariff. In this thesis, we choose not to take this specific cost function into account, because we want the research to be applicable to other use cases as well. Therefore, we assume a more general cost function that is similar to the ones we encountered in the literature and takes into account the following factors:

- The number of trucks.
- The relative empty driving time, that is, the percentage of time in the truck planning that is spent on driving empty, either from or to the depot, or between two consecutive shipments.

- The relative waiting time, that is, the percentage of time in the truck planning that is spent on waiting.

The *inefficiency* of a truck planning, a performance indicator we often refer to, is the sum of the relative empty driving time and the relative waiting time. The total costs of a truck planning are defined by the weighted sum of the three components.

The following requirements for TMS should be taken into account when developing the algorithm:

- **Flexible**
  Picnic's exponential growth causes substantial and continuous changes in the transport planning problem definition on a yearly basis. The system is supposed to be flexible enough to incorporate these changes easily.

- **Robust**
  Picnic's supply chain operation can be chaotic, because of last minute changes. As a result, abnormalities in the transport planning input are plausible. The system should be able to deal with this by guaranteeing a feasible solution as output for each input instance.

- **Fast**
  The total computation time used by the algorithm to generate an output is bounded by 15 minutes on Picnic's server. Running the algorithm on the single laptop on which the research is executed is not representative. However, as the computation time only decreases when running on Picnic's server, resulting in an even better computation time, this difference is negligible.

The feasibility of a truck planning depends on several restrictions. In addition to the trivial restrictions, such as a truck not being able to be at two locations at the same time, the following restrictions follow from Picnic's supply chain in particular:

- **The multi-depot restriction.**
  The available trucks are distributed over various depots: DCs, FCs and Ermelo (where vdBrink is situated). Each truck has to start and end the day at the same location, to make sure that the drivers can park their car at that location and turn home at the end of the day.

- **The driver day duration restriction.**
  The drivers are legally allowed to work for 15 hours and drive for 9 hours on a single day. When starting their day before 5:00, the maximal driver day duration is 12 hours.

In order to make sure that these bounds are always respected in practice, Picnic currently plans driver days with a maximal duration of 11 hours if the driver starts the day before 5:15, and 13.5 hours otherwise. Depending on the amount of waiting time on a truck day, the maximal duration can be extended, as shown below. The 9-hour drive bound is never reached in practice, but should be kept in mind when developing the algorithm.
The driver day duration is also bounded from below by the contracts between vdBrink and each driver, who are payed for a minimal number of hours of work per week. Because the scope of this thesis is to take into account one day at a time, this restriction is incorporated by defining a minimal driver day duration of 7 hours.
For ease of notation, we introduce the following variables for the maximal and minimal duration of a driver day.

$$d_{max} = \begin{cases} 13.5, & \text{if driver day starts after 5:15 and has} < 1 \text{ hour of waiting time} \\ 14, & \text{if driver day starts after 5:15 and has} \geq 1 \text{ hour of waiting time} \\ 11, & \text{if driver day starts before or at 5:15 and has} < 1 \text{ hour of waiting time} \\ 11.25, & \text{if driver day starts before or at 5:15 and has} \geq 1 \text{ hour of waiting time} \end{cases}$$

$$d_{min} = 7$$

- **The time window restriction.**
  Every defined shipment includes start and end time windows. The start and end times assigned to a shipment should be in the start and end time windows, respectively.

- **The docking restriction.**
  Every location has a fixed number of trucks that is able to load and unload, i.e.,*docking*, at the same time. For Hubs, this number is often one, but for some of the FCs it can be up to twenty. The total number of trucks docking at the same time at a specific location should not exceed this bound. Since the drivers typically can resolve possible docking conflicts on site, we ignore this restriction in our model.

1.1.4. Driver change

One of the goals of this thesis is to quantify the impact of allowing a driver change. In order to respect the driver day duration restriction, but also make maximal use of trucks, the possibility of one driver executing the first part of a truck day and another driver executing the second part, i.e., *a driver change*, is allowed. Yet, each driver should start and end her or

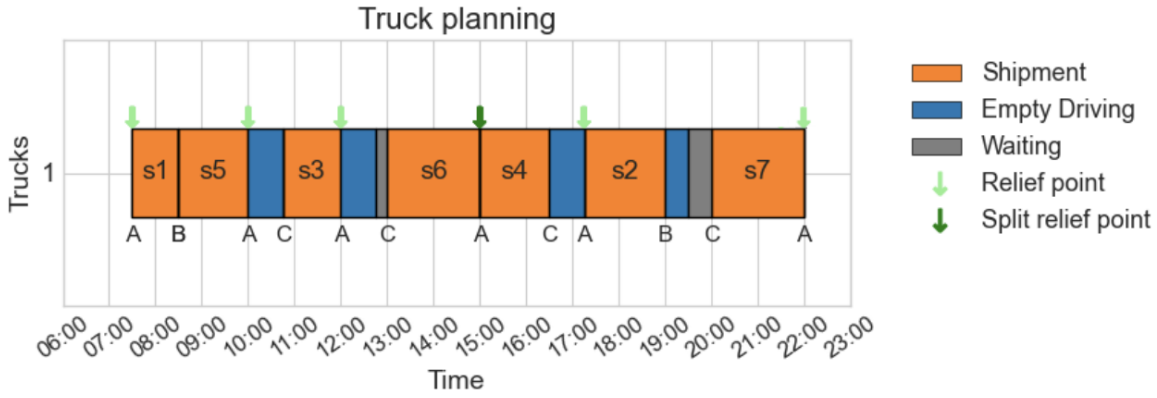


Figure 1.4: Example of a splittable truck day.

his day at the same location, and have a work day duration between d_{min} and d_{max} hours. Therefore, we define the moments in time when a truck returns to the depot where it also started its day to be *relief points*. A truck is able to facilitate a driver change if and only if there is a relief point that splits the truck day into two feasible driver days. A relief point of this type is called a *split relief point*. If a truck day contains at least one split relief point, it is called *splittable*. Figure 1.4 shows an example of a truck day that is splittable. The driver days have durations 7 and 7.5 hours and start and end at the same location, hence they are feasible.

A driver change may decrease the costs of a truck planning, by using fewer trucks in total. Even so, it may also be the case that introducing the use of a driver change decreases the efficiency of the planning, because in the process of finding splittable truck days efficiency is lost. Therefore, we investigate whether we should allow for a driver change or not, in order to find the best solutions.

The concept of a driver change could be generalized to more than two drivers executing the same truck day. Within the scope of this thesis, however, this does not occur, because all shipments are executed between 04:00 and 01:00 and the minimal driving time (d_{min}) is equal to seven, making it very unlikely that a truck day can be splitted into three driver days. Therefore, we assume, that a truck day is always executed by at most two drivers.

1.2. The Vehicle Scheduling Problem and its variants

A well-known mathematical optimization problem that resembles Picnic's case is the Vehicle Scheduling Problem (VSP) (Daduna et al., 1995). The input of this problem takes a list of $n \in \mathbb{N}$ shipments $S = \{s_1, \ldots, s_n\}$ that are to be executed by trucks. These shipments have a set of properties: a start time ST_i, an end time ET_i, a start location SL_i and an end

location EL_i, for all $i \in [n]$. Between every possible location, L_1 and L_2, the driving time is given by $d(L_1, L_2) \geq 0$. The goal is to find an assignment from shipments to trucks such that the following three conditions hold (Bunte et al., 2009):

- Each shipment is assigned to exactly one truck,
- Each truck performs a feasible sequence of shipments,
- The overall costs are minimized.

A solution of VSP defines a truck planning that is unique up to the order of waiting and empty driving in between executing shipments. The cost function that is minimized, assigns a cost $c(p) \in \mathbb{R}_{\geq 0}$ to any truck planning p. The cost function represents the objective and thus takes into account the total number of trucks, the relative empty driving time and the relative waiting time, by taking the weighted sum.

As with many optimization problems, there are multiple variants of VSP. The ones most relevant for this thesis are the following (Bodin, 1983):

1. Single Depot Vehicle Scheduling Problem (SDVSP): all trucks have to start and finish at the same single depot.
2. Multi Depot Vehicle Scheduling Problem (MDVSP): all trucks are assigned to one of the multiple depots available, from which they have to start and finish.
3. Vehicle Scheduling Problem with Time Windows (VSPTW): all shipments are allowed to start and end in a given time window instead of at a specific start and end time.
4. Integrated Vehicle and Crew Scheduling Problem (IVCSP): in addition to assigning every shipment to a truck, every task, i.e., executing a shipment or driving empty from one location to another, is assigned to a driver.
5. Vehicle Scheduling Problem with Length of Path Restrictions (VSPLPR): each truck day has a maximal day duration.

The problem most similar to Picnic's case depends on whether a driver change is allowed or not. When allowing a driver change, the Multi Depot Integrated Vehicle and Crew Scheduling Problem with Time windows (MDIVCSPTW) represents Picnic's case the best. Without allowing a driver change, the Multi Depot Vehicle Scheduling Problem with Length of Path Restrictions and Time Windows (MDVSPLPRTW) resembles Picnic's case the most. However, when modelling both problems, the problem sizes become too large to be able to find a solution within reasonable time. Therefore, the definition of the main problem we are

trying to find a solution to is the Multi Depot Vehicle Scheduling Problem with Time Windows (MDVSPTW) in combination with the driver day duration restriction either with or without allowing a driver change. When referring to *our problem* or *Picnic's case*, we mean finding a solution to this problem within limited computation time.

The VSP variants above are related to our problem as subproblems. In describing the existing literature and explaining the algorithms for our problem, we build up the theory by considering several of the subproblems from the list above. The simplest subproblem, SDVSP, comes with the advantage of being solvable in polynomial time. All other variants belong to the complexity class NP-hard, meaning these problems are unlikely to be solvable in polynomial time (Bunte et al., 2009).

1.3. Research question

The main goal of this thesis is to answer the following question:

Which algorithm is the most suitable for solving Picnic's case by finding a good solution for MDVSPTW and taking into account the driver day duration restriction within limited computation time?

The following subquestions are considered:

- Which ILP models exist to solve MDVSPTW with the driver day duration restriction?
- Which heuristics exist to find good solutions to MDVSPTW with the driver day duration restriction?
- How can we design algorithms based on the existing ILP models and heuristics that find good solutions to MDVSPTW with the driver day duration restriction and satisfies Picnic's requirements for TMS?
- What is the best algorithm when allowing a driver change?
- What is the best algorithm when not a driver change?
- What is the impact of allowing a driver change?

1.4. Contributions

As stated above, VSP is a well-studied problem. But in the literature, the problem is often applied to public transport bus scheduling problems, not to freight transport. The general

concepts are similar. Instead of assigning trips to buses available at depots, we assign shipments to trucks available at distribution centers (which we call depots). However, there are some specific differences between the two real-world applications.

- In transport, there is more flexibility time-wise. A truck is supposed to arrive at a certain distribution center before a certain deadline, since the freight has to be processed operationally within the distribution center. The shipment should not arrive too early, as the freight will occupy valuable ground space in the distribution center. But the deadlines are not as tight as in public transport. A couple of minutes early or late can always be corrected in other phases of the operational processes, while 5 minutes early or late in public transport is already a substantial delay.

- In transport, waiting at a location is less costly. At distribution centers, there is often a small parking space for trucks to wait. In public transport, on the other hand, waiting at a stop is not desirable, as there is seldom parking space. As a result, in public transport driving back to the depot to wait there, may be a cost efficient option, whereas in freight transport this never is the case.

- In transport, vehicles and drivers are more bound to each other. A truck driver generally drives the same truck and prefers not to switch trucks too often, whereas a bus driver is used to switching vehicles and driving several buses in one working day. Bus drivers can also join a trip as a passenger to turn back to the place where they started, while in transport this is unconventional.

- In transport, there is less certainty that all trucks will drive as planned. Each shipment is dependent on other processes in the supply chain, that are in their turn dependent on other processes. If one of the hundreds of employees that work in the distribution centers calls in sick, this might affect the speed at which the freight is prepared, which might cause a shipment delay. In public transport, however, less factors determine the accuracy of the actual start and end times of the trips.

Furthermore, there is also a specific property to Picnic's Case that distinguish it from a general transport shipment scheduling problem:

- Since customers are able to order their groceries until 22:00 and receive them the following day (or 15:00 and receive them the following morning), the truck schedules are constructed based on a forecast. Because the forecast is not always a perfect prediction of reality, shipments are cancelled or added last minute. This uncertainty decreases the need to optimize up to detail, as some parts of the solution will probably change anyway.

To the best of our knowledge, there is no algorithm that solves MDVSPTW with the driver day duration restriction and takes into account the specific properties mentioned above. The goal of this thesis is to investigate the existing solutions for MDVSPTW with the driver day duration restriction and leverage the properties that distinguish transport from public transport to write an algorithm that works specifically well for Picnic's Case.

1.5. Outline of the report

Figure 1.5 provides an overview of the chapters. The first three chapters provide theoretical background from the literature. Chapter 2 provides general theoretical background on mathematical optimization. Chapter 3 and Chapter 4 provide more specific background regarding our problem. These contain a short literature review and explanation of the particular existing algorithms, based on integer linear programming and heuristics respectively. The next two chapters regard the implementation and contain algorithms that are inspired on the theoretical background chapters, but a better fit for our problem specifically compared to the algorithms encountered in the literature. Chapter 5 presents two algorithms, that are directly derived from algorithms encountered in the literature. These function as important building blocks for the three main algorithms that find good feasible solutions to our problem, and are introduced in Chapter 6. Chapter 7 considers the algorithm configurations of the building blocks and the main algorithms, that is, the choice of specific performing values. Chapter 8 shows the experimental results of the three main algorithms. Finally, Chapter 9 draws conclusions and recommends topics for future research.

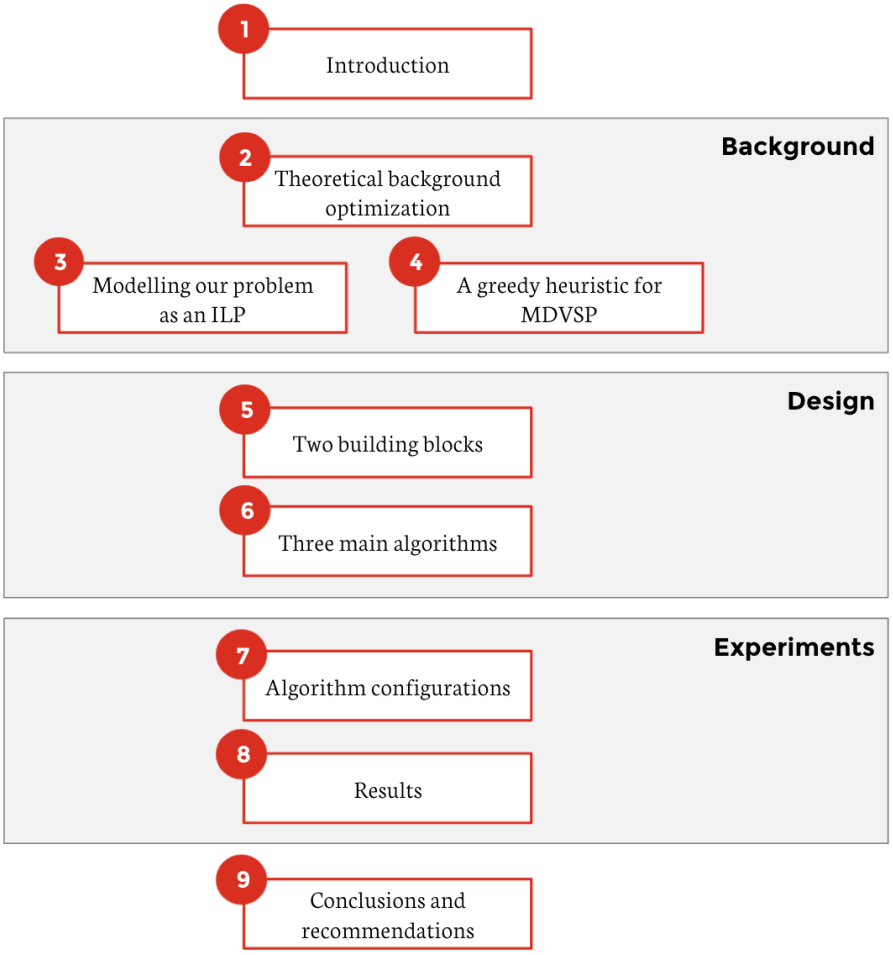


Figure 1.5: An overview of the chapters.

2

Theoretical background optimization

This chapter introduces the general mathematical optimization theory on which the work in this thesis in based. Section 2.1 and Section 2.2 explain the relevant theoretical topics regarding integer linear programming and heuristics respectively. Section 2.3 explains the theoretical background of assessing the performance of an optimization algorithm.

2.1. Integer Linear Programming

This section briefly introduces Integer Linear Programs (ILPs) and well-known methods to solve them. For a more extensive explanation of Integer Linear Programming and related subjects, the reader is advised to read Wolsey (1998).

2.1.1. The definition of an ILP

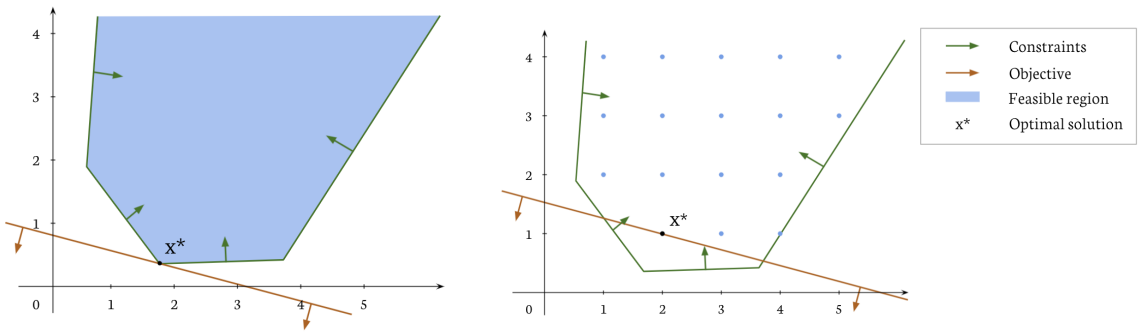
In optimization problems, the goal is to find an optimal solution with regard to an objective from a set of implicitly available alternatives. The set of possible solutions, often called the solution space or feasible region, is defined by a set of constraints. In linear programming, these constraints are represented by linear inequalities. Given $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$, the solution space $X \subseteq \mathbb{R}^n$ is defined by the set of constraints

$$A\mathbf{x} \leq \mathbf{b} \quad \text{with} \quad \mathbf{x} \geq 0,$$

The vector $\mathbf{x} \in \mathbb{R}^n$ is called the variable vector. Each vector $\mathbf{x}$ represents an element of the solution space and therefore a solution. The objective is represented by a vector $\mathbf{c} \in \mathbb{R}^n$ that yields the objective value by taking the inner product with the variable vector

$$\mathbf{c}^T\mathbf{x}.$$

The above standard form of a linear program shows that solving the optimization problem equals finding the element of a polyhedron X that optimizes the inner product with $\mathbf{c}$, also called the optimum, denoted by $\mathbf{x}^*$. The objective value of the optimum is called the optimal value and denoted by $z^*_{LP} = \mathbf{c}^T\mathbf{x}^*$. The geometric interpretation of linear programming is to 'push' a hyperplane given by $\mathbf{c}$ as far as possible in the right direction over the polyhedron X. If the problem is feasible and has a bounded optimum, the convexity of the polyhedron ensures that an optimal solution $\mathbf{x}^*$ is in an extreme point of the polyhedron X. Using the Simplex algorithm of Dantzig, linear programs can be solved efficiently in practice (Dantzig, 1951). Every linear program is polynomially solvable by the ellipsoid algorithm, however, this is not a fast algorithm in practice (Khachiyan, 1979). Integer Linear



(a): Linear Programming. **(b):** Integer Linear Programming.

Figure 2.1: Geometric interpretation two-dimensional Linear Programming and Integer Linear Programming.

Programming requires the solution to have integer values, i.e., each element of the vector $\mathbf{x}$ should be integral. From here on, the assumption is made that the optimality goal is to minimize. The standard form of an integer linear optimization problem is the following:

$$\begin{aligned} \text{minimize} \quad & \mathbf{c}^T\mathbf{x} \\ \text{subject to} \quad & A\mathbf{x} \leq \mathbf{b}, \\ & \mathbf{x} \geq 0, \\ & \mathbf{x} \in \mathbb{Z}^n, \end{aligned}$$

where $A \in \mathbb{Q}^{m \times n}$ and $b \in \mathbb{Q}^m$, and the optimal value is denoted by z^*_{IP}. The geometrical interpretation remains similar, but instead of the solution space being a polyhedron, the integer points intersected with the polyhedron defines the solution space. Integer linear programming is NP-hard, as in particular, the decision version of $\{0,1\}$-integer programming, where all elements of $\mathbf{x}$ are restricted to values from $\{0,1\}$, is one of Karp's 21 NP-complete problems (Karp, 1975).

2.1.2. Methods to solve ILPs

Even though solving ILPs is NP-hard in general, there are several approaches towards ILPs that enable us to extract useful information about the optimum and sometimes even find the optimum within a reasonable computation time. This subsection briefly explains some of these approaches.

A straightforward approach is to compute the LP relaxation, that ignores the integrality constraint and solves the associated Linear Program (LP). This approach is only able to find the optimum for the specific subclass of ILPs for which the extreme point of the polyhedron corresponding to the optimum is an integer point. In general, rounding the LP solution guarantees integrality, but not feasibility and is therefore also not an effective approach. Lemma 2.1 states a necessary and sufficient condition under which the solution of the LP relaxation is integral, but first we need a definition.

Definition 2.1. *A matrix A is* Totally Unimodular *(TU) if every square submatrix of A has determinant equal to* $1, -1$ *or* 0.

Lemma 2.1. *Let $A \in \mathbb{R}^{m\times n}$, $\mathbf{b} \in \mathbb{R}^n$ and $\mathbf{c} \in \mathbb{R}^n$ define the LP*

$$\min\left\{\mathbf{c}^T\mathbf{x} \colon A\mathbf{x} \leq \mathbf{b},\, \mathbf{x} \in \mathbb{R}^n_{\geq 0}\right\}.$$

This LP has an integral optimal solution for all integer vectors $\mathbf{b}$ *for which it has a finite optimal value if and only if A is TU.*

Even though the LP relaxation is generally not able to solve the ILP, it is able to provide a lower bound of the optimal value for every ILP, as $z^*_{LP} \leq z^*_{IP}$. This is very useful when assessing the performance of an optimization algorithm, which is further explained in Section 2.3.

One method to approach an ILP that guarantees to find the optimum is to apply a Cutting Plane algorithm. This class of algorithms adds linear constraints, i.e., *cutting planes*, to the program with the goal of cutting off part of the polyhedron, that does not contain integer solutions, such that the ILP solution eventually becomes an extreme point. Cutting Plane algorithms are not practical as a stand-alone, because, even though it can be proven that after a finite number of steps these algorithms find the optimum (Gomory, 1958) (see also Wolsey (1998)), the convergence is often slow.

2.1.3. The Branch and Bound algorithm

A method for solving ILPs that is often used in practice is the Branch and Bound algorithm, in particular when combined with cutting planes. The method was first proposed by Land

et al. (1960), but the name was coined by Little et al. (1963). Branch and Bound systematically searches the solution space, by *branching* the main problem into smaller subproblems. This can be intuitively visualized by a binary tree, from here on referred to as the search tree, with the root node representing the main problem and each other node representing a subproblem. The method aims to solve the main problem without having to visit the complete search tree. By keeping track of the bounds on the optimal value, generated by feasible solutions and LP relaxations, we are able to *prune* subtrees from the search tree without actually visiting every node in the particular subtrees. In practice, we hope that pruning yields a much smaller search tree than the complete tree.

Branching often takes place by detecting the 'most fractional' variable in the LP relaxation, i.e., the variable x_i closest to the decimal number $n.5$ with n any integer number, and splitting the problem into two subproblems. One subproblem assumes that $x_i \geq \lceil x_i \rceil$ and the other subproblem assumes that $x_i \leq \lfloor x_i \rfloor$.

At the start of the search, the upper bound is set to the value of a feasible solution, and if no fesaible solution is known we set it to $+\infty$. At each node, a lower bound is determined for the tree rooted in that node. The lower bound is calculated by solving the LP relaxation. If the LP solution is integral, the objective value of that solution gives an upper bound. If this upper bound is smaller than the currently best known feasible solution, we update the value of the currently best know upper bound.

The choice to not examine any further nodes in the tree rooted in a certain node, i.e., *pruning*, happens in the following cases:

- Pruning by optimality: the LP relaxation of the subproblem represented by the node has an integral solution and the subproblem is therefore solved to optimality. Branching is not necessary anymore and a feasible solution is found. The upper bound of the main problem is updated if the feasible solution has a smaller upper bound than the upper bound that was stored.
- Pruning by bound: The LP relaxation of the subproblem generates a lower bound that is greater or equal to the current upper bound of the solution of the main problem. Branching is not necessary, because it does not lead to a better solution than the solution already known.
- Pruning by infeasibility: the LP relaxation of the subproblem is infeasible. Trivially, no integer solution to this subproblem exists either.

Once all leaves of the search tree are pruned, the main problem has been solved.

An important advantage of Branch and Bound is that even if the algorithm is not able to finish completely within a pre-set time bound, it is typically possible to find good feasible solutions. That is because, during the searching process, we can store all feasible solutions found so far, from here on referred to as the *solution pool*. When terminating the algorithm before finding the optimal solution, the algorithm returns the best solution from the solution pool. There are different types of termination criteria for Branch and Bound. The algorithm could simply be terminated after a fixed number of seconds τ_{max} has passed, or we could set a fixed number of nodes in the search tree to visit, before returning the best feasible solution found so far. Another option is to terminate the algorithm if a feasible solution found is 'good enough'. This can be determined by setting a bound on the relative difference of the lower and upper bound, that we update during the search process. Recall that while searching the tree, we have a least upper bound $\overline{z}$, defined by the best feasible solution found so far, and a lower bound $\underline{z}$, defined by the LP relaxation. The relative difference of the lower and upper bound is defined to be the *optimality gap*. By bounding the optimality gap, we define a feasible solution to be good enough if the following holds:

$$\text{optimality gap} = \frac{|\overline{z} - \underline{z}|}{|\overline{z}|} \leq \delta_{gap}, \tag{2.1}$$

where $0 < \delta_{gap} < 1$.

This property of Branch and Bound is useful because in practice, good feasible solution may already be of great value. However, the algorithm does not guarantee to always return a feasible solution. This motivates the consideration of another type of algorithms that does guarantee the return of a feasible solution.

2.2. Heuristics

In this section, we consider a type of algorithms that focuses on finding feasible solutions: heuristics. We explain what the definition of a heuristic is, which different classes exists and how to assess the performance of a heuristic. For a more extensive explanation on various types of heuristics, the reader is advised to read Aarts et al. (2003).

2.2.1. The definition of a heuristic

Every optimization algorithm has a trade-off between the quality of the solution and the computation time of the algorithm. A heuristic prioritizes a shorter computation time over finding the provably optimal solution. There is no guarantee for the solution to be optimal

and there is no guarantee for the computation time to be polynomial, but the user can control the computation time by including various stopping criteria. The goal of a heuristic is to find (εὑρίσκω means *to find* in ancient Greek) a *good* feasible solution within *reasonable* computation time. What the precise definitions of *good* and *reasonable* are depend on the user-defined requirements.

Heuristics often have the general strategy to efficiently search the feasible solution space. Based on this strategy, we usually distinguish between two classes of heuristics: construction heuristics and improvement heuristics (Blum et al., 2003).

2.2.2. Construction heuristics

Construction heuristics generate a feasible solution from scratch. The first priority is to find a feasible solution within a short computation time, the second priority is to guarantee quality of this feasible solution. This class of heuristics often defines the first step in a search algorithm. There are many possible methods that define construction heuristics. We introduce one method that is relevant in particular for our implementation.

Apply an intuitive algorithm with short computation time.
Depending on the specifics of the problem, one can define intuitive, often greedy, rules that together construct an initial solution. This approach can be difficult to use if the problem is highly capacitated. That is, for example, if we only have a small number of available trucks to assign the shipments to. If there is always a new truck available that can be added to the truck planning, however, designing an intuitive algorithm to construct an initial solution is easier. The performance of an intuitive algorithm as a construction heuristic solely depends on the effectiveness of the rules. The intuitiveness of this method has two main advantages: transparency, robustness. The transparency of the algorithm creates understanding of the characteristics of the solution and therefore allows us to improve the solution further and communicate the workings of the algorithm easier to all people concerned with the algorithm in practice. The robustness result from the fact that construction heuristics generally run in polynomial time and and therefore able to guarantee an output. This is an important property when dealing with algorithms running in operational systems, such as in Picnic's case, where no output is not acceptable.

Another method to construct a good feasible solution is to combine the two methods above. This can be done by defining an easier ILP corresponding to a subproblem of the complex ILP one is trying to solve. After solving the easier ILP (multiple times, if necessary), one constructs a solution to the complex ILP by applying intuitive rules. In finding an initial

solution for MDVSP, Pepin et al. (2009) uses this method by leveraging the fact that SDVSP is solvable in polynomial time. The performance of this method depends on the existence of an easier ILP for a subproblem and the possibility to define good intuitive rules.

2.2.3. Improvement heuristics

Improvement heuristics start from an initial solution and iteratively improve the solution by searching the solution space close to the current solution. This method is also called Local Search. In explaining Local Search in more detail, we return to the mathematical notation regarding optimization problems as introduced in Section 2.1.

The solution space is denoted by X, and a particular solution by $\mathbf{x} \in X$. Local Search is a search strategy, that iteratively explores the neighborhood of solutions 'close' to a given solution. Therefore, a Local Search algorithm is induced by the definition of a neighborhood for every feasible solution $\mathbf{x}$ and a neighborhood exploration method. Formally, the neighborhood of a solution is defined by a neighborhood function, as follows:

Definition 2.2. *A* neighborhood function *is a mapping* $\mathcal{N} : X \rightarrow \mathcal{P}(X)$ *that defines for every solution* $\mathbf{x} \in X$ *a set* $\mathcal{N}(\mathbf{x}) \subseteq X$ *of solutions that are in some sense close to* $\mathbf{x}$*. The set* $\mathcal{N}(\mathbf{x})$ *is called the* neighborhood *of* $\mathbf{x}$*, and each element* $\mathbf{x}' \in \mathcal{N}(\mathbf{x})$ *is a neighbor of* $\mathbf{x}$*.*

Given a truck planning, an example of a neighborhood could be all truck plannings that are generated by assigning one shipment to a different truck. This would generate a neighborhood that consists of $n \times t$ neighbor truck plannings, where n is the number of shipments and t is the number of trucks. In order to effectively explore this neighborhood, we need to define a *neighborhood exploration method*. A neighborhood exploration method considers $\mathcal{N}(\mathbf{x})$ and replaces $\mathbf{x}$ by neighbor $\mathbf{x}'$ with the goal of improving the final solution. An example of a neighborhood exploration method is *first improvement*, that replaces the current solution by an improved solution, as soon as one if found. Another method, called *best improvement*, is to first explore (a part of) the neighborhood, before replacing the current solution with the best solution found so far (Aarts et al., 2003). A straightforward approach would be to stop, when no improvement seems probable anymore. Therefore a possible termination criterion is to quit when no improving solution is found in the neighborhood of the current solution. The disadvantage of a simple improvement heuristic is that, depending on the properties of the objective function, the risk of getting stuck in a poor-quality local optimum is plausible.

There are several methods to escape local optima. Again, it depends on the specific characteristics of the optimization problem which method is preferable. We introduce two of the most common used methods in practice.

- **Tabu Search**
  Tabu Search was introduced by Glover (1986) and formalized three years later (Glover, 1989). The method is based on the idea that we might be able to escape a local optimum with an Local Search heuristic, if we are also allowed to accept worse solutions and move away from the local optimum, without being drawn back to its relative good objective value. When moving away, at some point improving solutions to accept appear again, that lead towards another, hopefully better, (local) optimum. In order to achieve this, a list of recently visited solutions that are not allowed to be visited again, i.e., a *tabu list*, is kept. When a better solution is found, it is accepted if it is not on the tabu list, and, when no better solution is found, we are allowed to visit worse solutions.

- **Simulated Annealing**
  Simulated Annealing was first coined by Kirkpatrick et al. (1983). The method decides whether to accept a neighbor as the new solution based on a probabilistic process. The probability of accepting a solution depends on the difference between the objective value of the current solution $\mathbf{x}$ and the candidate solution $\mathbf{x}'$ and the *temperature* T. The temperature is time-dependent: starting at a positive value, the more time passes, the closer the temperature gets to zero. When an improving solution is found, it is always accepted. However, when a worse solution is found, the solution is accepted with a probability that is higher when the solution is better and the temperature is higher. The probability of accepting a solution was introduced by Kirkpatrick et al. (1983) to be

$$P(z, z', T) = \begin{cases} 1, & \text{if } z' < z \\ e^{\frac{-(z'-z)}{T}}, & \text{otherwise} \end{cases} \tag{2.2}$$

  where z is the objective value of the current solution, z' is the objective value of the candidate solution and T is the temperature.

There are various other methods to escape local optima, such as Variable Depth Search, Genetic Algorithms and Ant Colony optimization. However, these are out of the scope of this thesis. The reader is again advised to read Aarts et al. (2003) for more theory regarding methods to escape local optima.

2.3. Assess the performance of an optimization algorithm

If all optimization methods would be able to find the optimum, a performance measure would solely be the computation time. However, as mentioned above, it is often not possible to solve an optimization algorithm to optimality in practice, because of practical time restrictions. Therefore, we construct algorithms to find a suboptimal solution within a fixed

amount of time. The most important performance measure of an optimization algorithm is therefore the quality of the suboptimal solution. So, when comparing various heuristics to each other, the heuristic that generates the solution with the minimal objective value within the fixed amount of time, naturally, performs the best.

Assessing the absolute performance of a heuristic is more difficult, as the optimal value is typically not known. We are, however, able to find a lower bound for the optimal value by solving a relaxation of the problem. With this lower bound, we are able to bound the difference between the best objective value found and the optimum. The quality of this approximation is determined by the quality of the lower bound, that depends on the problem characteristics and type of relaxation. This could be an LP relaxation, but we could also choose to relax another constraint.

3

Modelling our problem as an ILP

The Vehicle Scheduling Problem (VSP) belongs to the class of optimization problems that can be formulated as an ILP. Therefore, well known Integer Linear Programming techniques, such as the Branch and Bound algorithm, are suitable for finding a solution. This chapter considers Integer Linear Programming models and techniques applicable to MDVSPTW with the driver day duration restriction. Section 3.1 contains a literature review on Integer Linear Programming techniques applied to VSP. The other sections explain the modelling of our problem by building it up from relevant subproblems. Section 3.2 considers SDVSP, Section 3.3 adds the multi-depot property by considering MDVSP, and Section 3.4 adds the time window property by considering MDVSPTW. Finally, Section 3.5 considers the incorporation of the driver day duration restriction in the ILP model for MDVSPTW.

3.1. Literature review

We build up this section the same way up we build up the chapter: by considering SDVSP, MDVSP, MDVSPTW and finally MDVSPTW with the driver day duration restriction. There are various ways to model the variants of VSP. We shortly mention models of the relevant subproblems, but for a more extensive explanation of the models, the reader is referred to the survey papers on ILP modelling of VSP of Bunte et al. (2009) and Daduna et al. (1995).

3.1.1. SDVSP literature review

As mentioned in Chapter 1, SDVSP is solvable in polynomial time. Saha (1970) introduced the first ILP modelling method for SDVSP: to formulate the problem as a Minimal Decomposition model and reformulate it as a Network Flow Problem. Drawbacks are that this model only solves the minimum fleet size, without taking operational costs into account,

and no upper bound for the fleet size can be set. To be able to incorporate the operational costs as well, Orloff (1976) formulated SDVSP as an Assignment Problem. However, the maximal number of vehicles could still not be taken into account. Gavish et al. (1979) finally was able to construct a model that could also take into account the maximal number of vehicles, by formulating the problem as a Transportation Problem. Later, J. P. Paixão et al. (1987) reformulated this model to the closely related Quasi-Assignment model, on which the currently best performing algorithm to solve SDVSP is based (Freling et al., 2001). Finally, Bodin (1983) modelled SDVSP as a Minimum-Cost Network Flow model, motivated by the work of Dantzig and Fulkerson (1954). We explain this model in more detail in Section 3.2, because it naturally extends to MDVSP.

3.1.2. MDVSP literature review

MDVSP was proven to be NP-hard by Bertossi et al. (1987). The main three modelling approaches are: Single-Commodity models, Multi-Commodity models and Set Partitioning models. The optimization problem was first modelled as an ILP by Carpaneto et al. (1989) in a Single-Commodity model with Subtour Breaking Constraints. However, the problem size of this model is extremely large, therefore Mesquita and J. Paixão (1992) introduced a similar model, but with a smaller problem size, by modelling the problem as a Single-Commodity model with Assignment Variables. A more intuitive way to model MDVSP, is by extending the Minimum-Cost Network Flow model of SDVSP. This results in the Multi-Commodity Connection-Based Networks model. Because this model, again, naturally extends to MDVSPTW, we choose to explain it in more detail in Section 3.3. The model is the basis for several heuristic approaches that find a solution for MDVSP. We do not name them explicitly, but refer to Bunte et al. (2009) for an overview. Another Multi-Commodity model is the Time-Space Networks model, introduced by Kliewer, Mellouli, et al. (2006). The advantage of this model is the decreased problem size compared to the Connection-Based model. The final modelling approach of MDVSP is Set Partitioning, which was first presented by Ribeiro et al. (1994). Compared to the Multi-Commodity model, this model has only few constraints, but a huge number of variables, as it enumerates all possible feasible sequences of trips. Again, several different heuristic approaches are based on this way of modelling, and often entail Column Generation, because of the large number of non-basic variables.

Each modelling approach has the same optimal integer solution, but the lower bounds generated by the LP relaxations differ per approach. The quality of these bounds is relevant for reasons introduced in Section 2.3. Mesquita and J. Paixão (1999) proved that the lower bound provided by the Single-Commodity model with Subtour Breaking Constraints

is smaller than or equal to the lower bound provided by the Single-Commodity with Assignments Variables, which is weaker than the Multi-Commodity model lower bound. Ribeiro et al. (1994) presented the proof that the LP bound of the Multi-Commodity model and the Set Partitioning model have the same value. So, the quality of the LP bounds can be summarized as follows:

$$z_{LP}^{\text{Subtour Breaking}} \leq z_{LP}^{\text{Assignment Variables}} \leq z_{LP}^{\text{Connection Based}} = z_{LP}^{\text{Set Partition}}. \tag{3.1}$$

3.1.3. MDVSPTW

In general, two approaches are introduced for modelling the time window extension: discrete and continuous processing of the time windows. Discrete processing adds a number of extra trips to the model for different possible start times within the time window. This method was first applied for aircraft fleet routing by Levin (1971) and later adapted to the Multi-Commodity Connection-Based Networks model by Ferland et al. (1988). Section 3.4 explains this method in more detail. The second approach for modelling the time windows, is continuous processing, a natural extension of the set partition model and therefore often combined with Column Generation, see Desaulniers et al. (1998).

3.1.4. MDVSPTW with driver day duration restrictions

When incorporating the driver day duration, we make a distinction in allowing a driver change and not allowing a driver change. When no driver change is allowed, the optimization problem is an extension of MDVSPTW with a maximal day duration. In the literature, this extention of VSP is referred to as the Vehicle Scheduling Problem with Length of Path Restrictions (VSPLPR), the Vehicle Scheduling Problem with Route and Time Constraints (VSPRTC) or VSPRTC with only Route or Time Constraints. In this thesis, we refer to this algorithm by VSPLPR.

Bodin (1983) first described an extention of the Minimum-Cost Network Flow model of SDVSP to VSPLPR. Because the natural extension from the Minimum-Cost Network Flow model is in line with the modelling extentions of the other subproblems, we choose to explain this model in more detail in Section 3.5. Later, also extensions from MDVSP models were made that incorporated a maximal day duration. Mingozzi et al. (1995) introduced a set partitioning formulation for MDVSP, extended with the maximal day duration constraint. Finally, Haghani et al. (2002) introduced an exact method to solve the same problem that naturally extends from the Minimum-Cost Network Flow model. Even though this model may be relevant to our problem, we did not incorporate it into this chapter because it is not a single ILP formulation but a dynamic method. Instead of adding constraints for all possible too long days to the ILP, which would result in an enormous number of con-

straints, the method iteratively solves the MDVSP ILP and adds the constraint corresponding to a too long day once it occurs. There is no guarantee that the method terminates in time, even though it seemed to work well in the experiments by Haghani et al. (2002). Investigating the application of this method to our problem may be interesting for future research.

When allowing a driver change, the Integrated Vehicle and Crew Scheduling Problem (IVCSP), also referred to as the Vehicle and Crew Scheduling Problem, is relevant to our problem. The optimization problem contains the traditional Vehicle Scheduling Problem and the Crew Scheduling Problem (CSP) as subproblems, and aims to find minimum cost sets of vehicle blocks and crew duties such that both vehicle and crew schedules are feasible and mutually compatible. Section 3.5 explains CSP and the relation to IVCSP in more detail. As shown by Fischetti et al. (1989), CSP with working time constraints is NP-hard. The problem is usually modelled as a Set Paritioning or Set Covering Problem. Because the number of feasible duties (and thus columns) is extensive in real-world problem instances, a Column Generation is often used, for example by Desrochers et al. (1989). Freling (1997) introduced the first integrated approach for solving VSP and CSP together. Based on this approach, Huisman (2004) and Huisman et al. (2005) presented the first general mathematical formulation for MDICVSP. Mesquita and Paias (2008) introduced two similar formulations, but with less constraints. All solution schemes involve techniques such as Column Generation and Lagrangian Relaxation in order to deal with the large problem sizes. We were only able to find one reference that covers all constraints of our problem and allows for a driver change. Kliewer, Amberg, et al. (2012) suggested an algorithm that solves MDICVSPTW.

3.2. An ILP to solve SDVSP

In a first step to formulate MDVSPTW as an ILP, we consider the subproblem SDVSP. As described below, SDVSP can be formulated as a Minimum-Cost Network Flow Problem. This section formally defines SDVSP and then considers the modelling as an ILP.

3.2.1. Defining SDVSP

Bodin (1983) formulated the problem as a Minimum-Cost Network Flow model. In order to formalize the model, we use the following notation for the various inputs of SDVSP:

- Shipments: $S = \{s_1, \ldots, s_n\}$
- Depot: single depot d
- Properties of shipment: start location SL_s, end location EL_s, start time ST_s and end time ET_s, $\forall s \in S$

- Driving time between locations: $d(L_1, L_2) \geq 0$, $\forall$ locations L_1, L_2
- A cost function c that assigns costs to a truck planning in a deterministic manner.

All locations relevant to the problem are the start and end locations of the shipment and the depot. Also, all shipments are required to start and end on time.

Example: SDVSP instance

Table 2.1 and Table 2.2 provide an example instance of SDVSP with depot D, locations A, B, C, and six shipments to be scheduled.

Table 3.1: Driving Time Matrix (minutes).

	D	A	B	C
D	0	45	30	60
A	45	0	45	45
B	30	45	0	30
C	60	45	30	0

Table 3.2: Input list of shipments.

s_i	ST_i	ET_i	SL_i	EL_i
s_1	7:30	9:00	A	B
s_2	14:00	16:30	A	B
s_3	10:30	12:00	C	A
s_4	13:30	15:30	B	C
s_5	8:30	10:00	B	A
s_6	12:00	13:00	C	B

Now, the goal is to construct a feasible truck planning that covers every shipment exactly once and minimizes the costs. For this instance, one can intuitively verify that the truck planning in Figure 3.1 is a good solution, because there is relatively little waiting time and empty driving time, and few trucks are used. What the optimal solution is depends on the exact definition of the cost function, which we define in the next section.

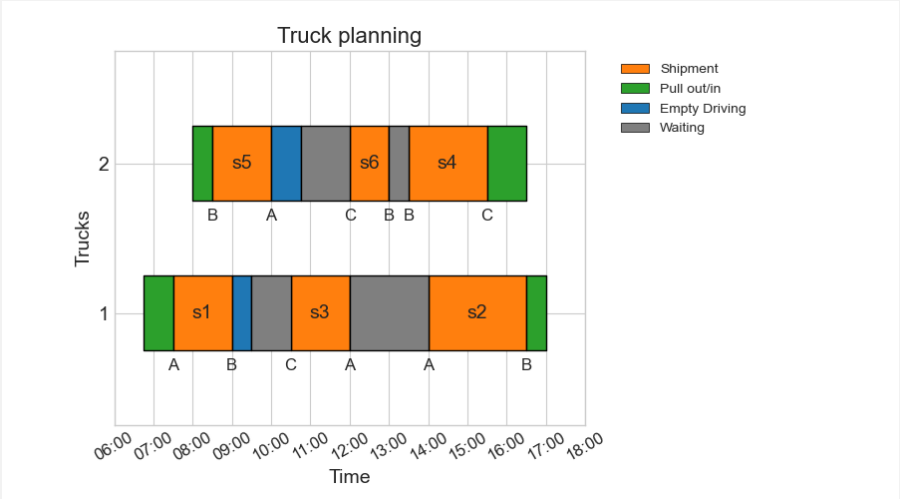


Figure 3.1: Solution for example instance.

3.2.2. The Minimum-Cost Network Flow model for SDVSP

Bodin (1983) constructs the Minimum-Cost Network Flow model by defining a directed graph $G = (V, A)$ and assigning costs and flow capacities to the arcs. Note that we used A before as matrix in the standard notation of ILP. The A used here is non-related and represents the arcs of the graph. The set of nodes consists of the source node, the sink node and the shipments, i.e., $V = \{s, S, t\}$. The source and sink nodes both represent the depot. One flow unit represents a single truck, and each path from s to t represents a potential schedule for a single truck. The arcs of the graph are assigned costs and capacities based on four types:

- *Pull out arcs*: from the source node to every shipment node. The costs are equal to the operational costs of driving from the depot to the start location of the shipment plus fixed costs for adding a new truck to the planning. The capacity is equal to one.
- *Pull in arcs*: from every shipment node to the sink node. The costs are equal to the operational costs of driving from the end location of the shipment to the depot. The capacity is equal to one.
- *Compatibility arcs*: from one shipment node s_j to another shipment node s_k, if it is feasible for a truck to execute shipment s_k after shipment s_j. Formally, there is a compatibility arc connecting s_j to s_k if and only if the following holds:

  $$ET_j + d(EL_j, SL_k) \leq ST_k. \tag{3.2}$$

  The costs are equal to the operational costs of executing s_k after s_j and the capacity is equal to one.
- *Circulation arc*: from the sink node to the source node. The costs are equal to zero and the capacity is equal to n.

Figure 3.2 shows a conceptual example of the graph. Three parameters play a part in assigning the costs to the arcs. In general, the operational costs are equal to the weighted sum of the driving time and the waiting time. Therefore, the parameters w_{dt} (weight of the driving time) and w_{wt} (weight of the waiting time) are introduced. Also, the fixed costs w_{fc} of adding a new truck to the planning is introduced. The cost function is defined as the function $c : A \rightarrow \mathbb{R}_{\geq 0}$. For ease of notation, the cost of an arc $c((i, j))$ is denoted by c_{ij}.

$$c_{ij} = \begin{cases} \text{operational costs}(d, s_j) + \text{fixed costs} & \text{if } (i,j) \text{ in a pull out arc} \\ \text{operational costs}(s_i, d) & \text{if } (i,j) \text{ in a pull in arc} \\ \text{operational costs}(s_i, s_j) & \text{if } (i,j) \text{ in a compatibility arc} \\ 0 & \text{if } (i,j) \text{ in a circulation arc} \end{cases}$$

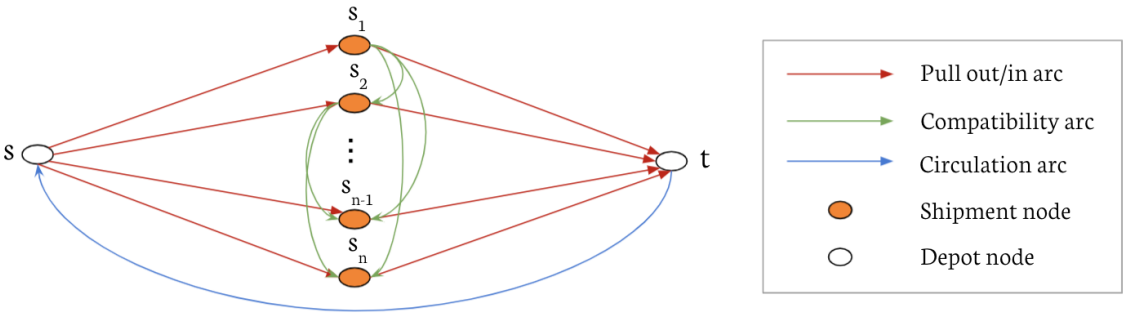


Figure 3.2: The directed graph representing SDVSP.

where

$$\text{fixed costs} = w_{fc} \tag{3.3}$$

$$\begin{aligned}\text{operational costs}(d, s_j) &= w_{dt} \times \text{empty driving time}(d, s_j) \\ &= w_{dt} \times d(d, SL_j)\end{aligned} \tag{3.4}$$

$$\begin{aligned}\text{operational costs}(s_i, d) &= w_{dt} \times \text{empty driving time}(s_i, s_j) \\ &= w_{dt} \times d(EL_i, d)\end{aligned} \tag{3.5}$$

$$\begin{aligned}\text{operational costs}(s_i, s_j) &= w_{wt} \times \text{waiting time}(s_i, s_j) + w_{dt} \times \text{empty driving time}(s_i, s_j) \\ &= w_{wt} \times (ST_j - ET_i - d(EL_i, SL_j)) + w_{dt} \times d(EL_i, SL_j)\end{aligned} \tag{3.6}$$

The ILP of the minimum flow cost model is defined as follows.

$$\min \quad \sum_{(i,j)\in A} c_{ij} f_{ij} \tag{3.7}$$

$$\text{s.t.} \quad \sum_{i\in V} f_{ij} - \sum_{k\in V} f_{jk} = 0 \qquad \forall j \in V \tag{3.8}$$

$$\sum_{i\in V} f_{ij} = 1 \qquad \forall j \in S \tag{3.9}$$

$$0 \le f_{ij} \le u_{ij} \qquad \forall (i,j) \in A \tag{3.10}$$

$$f_{ij} \in \mathbb{Z} \qquad \forall (i,j) \in A \tag{3.11}$$

The decision variables are $f_{ij} \ge 0$ corresponding to the amount of flow through arc $(i,j) \in A$, with each an assigned cost c_{ij} and capacity u_{ij}. The objective function (3.7) minimizes the sum of the total costs of all arcs in the solution. Constraints (3.8) guarantee flow conservation in every node. Constraints (3.9) require an in-flow (and thus, by (3.8), also out-flow) of exactly one, for all nodes in S. Constraints (3.10) guarantee that the flow does not exceed the capacity and constraints (3.11) ensure that it is integral.
This ILP is solvable in polynomial time, because by Lemma 3.1 the extreme points of the

polyhedron are integer points, making the integrality constraints redundant. Hence, linear programming methods are sufficient in finding the optimal solution of the ILP.

Lemma 3.1. *The extreme points of the polyhedron defined by* (3.8)-(3.10) *above are integer points.*

Proof. By Lemma 2.1, it suffices to show that the constraint matrix A and the vector $\mathbf{b}$ of constraints (3.8)-(3.10) are respectively TU and integral. The three types of constraints divide A and $\mathbf{b}$ into three parts:

$$A = \begin{bmatrix} A_1 \\ A_2 \\ A_3 \end{bmatrix} \qquad \mathbf{b} = \begin{bmatrix} \mathbf{b_1} \\ \mathbf{b_2} \\ \mathbf{b_3} \end{bmatrix}$$

Here, A_1, $\mathbf{b}_1$ correspond to the flow conservation constraints (3.8), A_2, $\mathbf{b}_2$ to the flow requirement constraints (3.9) and A_3, $\mathbf{b}_3$ to the flow capacity constraints (3.10). First, we check the integrality of $\mathbf{b}$. We see that

$$b_i = \begin{cases} 0, & \text{for } b_i \in \mathbf{b_1} \\ 1, & \text{for } b_i \in \mathbf{b_2} \\ u_{ij}, & \text{for } b_i \in \mathbf{b_3} \end{cases}$$

As the flow capacity u_{ij} is integer by construction, we can conclude that $\mathbf{b}$ is also integer. Now, we need to prove that A is TU. In general, it holds that if matrix B is TU, then matrix $\binom{B}{I}$ is TU, where I is the identity matrix (Wolsey, 1998). So, as $A_3 = I$, it is sufficient to show that $\binom{A_1}{A_2}$ is TU.

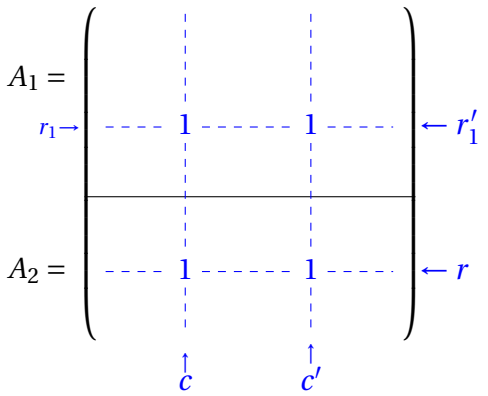
It also holds in general that a matrix B is TU if all entries are in $\{-1, 0, +1\}$ and any collection of rows of B can be split into two parts so that the sum of the rows in one part minus the sum of the rows in the other part is a vector with all entries in $\{-1, 0, +1\}$, see p.269 Theorem 19.3 from Schrijver (1998). We prove that this is true for $\binom{A_1}{A_2}$. Note that A_1 is the incidence matrix corresponding to the graph, with a row for each node and a column, containing exactly one $+1$ and one -1, for each arc. Also note that A_2 is a matrix with a row for each shipments node, and a column containing exactly one $+1$, for each arc. If we only consider the 1-entries, the rows of A_2 are copies of the rows of A_1 corresponding to the same shipment node.

Let a_{ij} denote the elements of $\binom{A_1}{A_2}$. Indeed $a_{ij} \in \{+1, -1, 0\}$ for all elements a_{ij} of A_1 and $a_{ij} \in \{+1, 0\}$ for all elements a_{ij} of A_2.

Now, let R be an arbitrary collection of rows of $\begin{pmatrix} A_1 \\ A_2 \end{pmatrix}$. The columns corresponding to all arcs, except for the pull in arcs (from all shipment nodes to t) contain exactly two $+1$'s and one -1. The columns corresponding to the pull in arcs, i.e., *pull in columns* contain one $+1$ and one -1. So for the pull in columns, we know that difference of the sum of the two parts of R is a vector with entries in $\{-1,0,+1\}$. Now, we only have to prove that for the matrix $\begin{pmatrix} A_1 \\ A_2 \end{pmatrix}$ minus the pull in columns, R can be split into two parts $R = R_1 \sqcup R_2$ for which the sum of the rows in R_1, denoted by σ_1, minus the sum of the rows in R_2, denoted by σ_2, is a vector with all entries in $\{-1,0,+1\}$.

We suggest a method to construct the partition and prove that all entries of $\sigma_1 - \sigma_2$ are in $\{-1,0,+1\}$. Because each column now contains exactly two $+1$'s and one -1, we know that the sum of the rows in R is a vector ψ with entries in $\{-1,0,+1,+2\}$. Let all rows in R that belong to A_1 be in R_1. For the rows in R that belong to A_2, we assign them to R_2 if they contribute to the 2-entries in ψ and assign the remaining rows to R_1. In the following paragraph we prove by contradiction that for this partition all entries of $\sigma_1 - \sigma_2$ are in $\{-1,0,+1\}$.

Assume there is a column c such that $\sigma_1(c) - \sigma_2(c)$ is not in $\{-1,0,+1\}$. Because, $\sigma_1(c) - \sigma_2(c)$ is not in $\{-1,0,+1\}$, we know that $\sigma_1(c) - \sigma_2(c) = -2$, where $\sigma_1(c) = -1$ and $\sigma_2(c) = +1$. Because $\sigma_1(c) = -1$, we know that the row corresponding to the 1-entry of column c, say r_1, in A_1 is not in R. Let $r \in R_2$ be the row that corresponds to the 1-entry at column c. By construction of the partition, r contributes to a 2-entry in ψ, that is, there is a column c' such that $r(c') = 1$ and $\sigma_1(c') = \sigma_2(c') = 1$. Because $\sigma_1(c') = 1$, we know that the row corresponding to the 1-entry of column c', say r'_1, in A_1 is in R. However, because the rows in A_2 are copies of the rows in A_1 if we only consider the 1-entries, we know that r_1, that was not in R, is the same row as r'_1, that is in R. Therefore, we found a contradiction and thus, $\begin{pmatrix} A_1 \\ A_2 \end{pmatrix}$ is TU. So, we can conclude that A is also TU, and that the extreme points of the polyhedron are integer points.



□

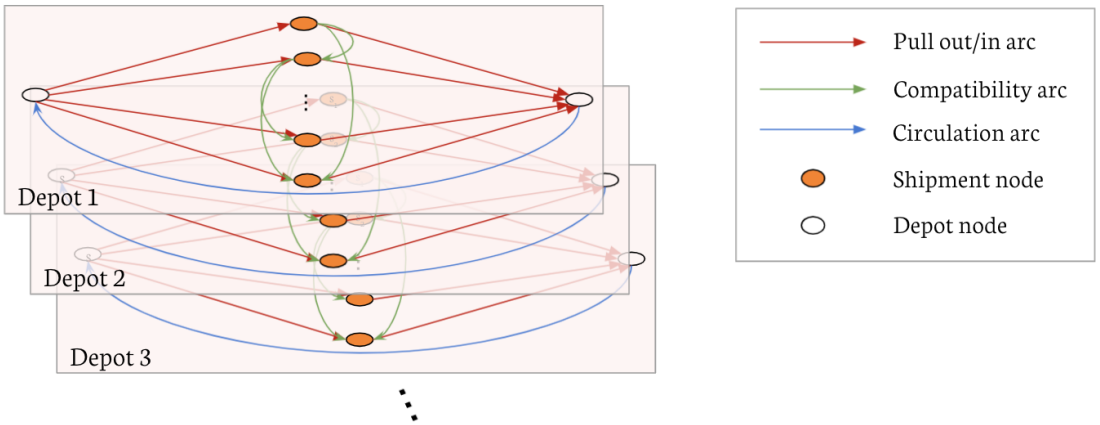


Figure 3.3: The directed graph representing MDVSP.

Since all extreme points of (3.8)-(3.10) are integral, we can ignore (3.11). Once problem (3.7)-(3.10) is solved, the optimal solution provides a subgraph of G, consisting of all arcs (i, j) such that $f_{ij} > 0$. The solution consists of a collection of cycles, each of which represents a truck.

3.3. An ILP to solve MDVSP

In this section the possibility that multiple depots are available is added to the problem described in the previous section. First, Subsection 3.3.1 formally defines MDVSP, then Subsection 3.3.2 describes the formulation as a minimum flow cost model.

3.3.1. Defining MDVSP

The formal definition of MDVSP is similar to SDVSP. The difference is that the input consists of multiple depots, with each a number of trucks available at that depot. The requirement that each truck has to start and end at the same location remains. Additional to the notation introduced in the SDVSP, the following notations are introduced:

- Depots: $D = \{d_1, \ldots, d_m\}$
- Number of trucks available at each depot: $\kappa_d \in \mathbb{N}$ for $d \in D$.

3.3.2. The Minimum-Cost Network Flow model for MDVSP

The optimization problem is again formulated as an ILP by modelling it as a Minimum-Cost Network Flow model. The difference compared to SDVSP is that the graph consists of m layers, each representing a depot and consisting of the SDVSP graph. The notation of the graph is therefore $G^m = (V^m, A^m)$. Figure 3.3 shows the multi-layered graph. Each copy $G_l = (V_l, A_l)$, with $l \in [m]$, gives the flow of trucks starting and ending at d_l as output. The

set of nodes per layer consists of $V_l = \{s_l, T_l, t_l\}$, hence s_l and t_l represent d_l. The costs and capacities assigned to every arc are determined in the same way as in the SDVSP model. The only difference is that the circulation arc has a flow capacity of κ_{d_l} instead of n.
The ILP corresponding to this model is the following:

$$\min \quad \sum_{l=1}^{m} \sum_{(i,j)\in A_l} c_{ij}^l f_{ij}^l \tag{3.12}$$

$$\text{s.t.} \quad \sum_{i\in V_l} f_{ij}^l - \sum_{k\in V_l} f_{jk}^l = 0 \qquad \forall j \in V_l,\ \forall l \in [m] \tag{3.13}$$

$$\sum_{l=1}^{m} \sum_{(i,j)\in A_l} f_{ij}^l = 1 \qquad \forall j \in V_l \tag{3.14}$$

$$0 \le f_{ij}^l \le u_{ij} \qquad \forall (i,j) \in A_l \tag{3.15}$$

$$f_{ij}^l \in \mathbb{Z} \qquad \forall (i,j) \in A_l \tag{3.16}$$

The decision variables are $f_{ij}^l \ge 0$, corresponding to the amount of flow through every arc $(i,j) \in A_l$. The objective function (3.12) and constraints (3.13), (3.15), (3.16) are intuitively extended from the SDVSP model. Constraints (3.14) require a total in-flow of exactly one for every node corresponding to the same task. Note that these constraints differ from SDVSP in the sense that not every node representing a shipment in the graph has to have in-flow one. Intuitively this makes sense, because every shipment needs to be executed by one truck in total, not one truck per depot. If the latter were the case, the problem would have been a cumulative set of m SDVSP problems, which would have made the program solvable in polynomial time. Constraints (3.15) are similar as before.

3.4. An ILP to solve MDVSPTW

In this section the time window property is added; shipments do not have fixed start and end times, but are allowed to start and end in given time windows. First, Subsection 3.4.1 defines MDVSPTW, then Subsection 3.4.2 describes the formulation as a Minimum-Cost Network Flow model.

3.4.1. Defining MDVSPTW

The formal definition of MDVSPTW is similar to MDVSP. The difference is that each shipment is equipped with a start window and an end window. A new type of constraints requires that the chosen start (and end) time of a shipment should be in the start (and end) window. The notation introduced for MDVSP remains the same, only the input shipments are of a different type. Instead of being defined by fixed start and end times, from here on referred to as *discrete shipments*, we now consider shipments defined by start and end

time windows, called *time window shipments*. The input set of n time window shipments is denoted by $\bar{S} = \{\bar{s}_1, \ldots, \bar{s}_n\}$ and each time window shipment $\bar{s}_i$ has the following properties:

- Start window: earliest start time EST_i and latest start time LST_i
- End window: earliest end time EET_i and latest end time LET_i
- Start and SL_i end location EL_i

Without loss of generality, we assume that the length of the start window is the same as the length of the end window.

3.4.2. The Minimum-Cost Network Flow model for MDVSPTW

The problem is again formulated as an ILP by modelling it as a Minimum-Cost Network Flow model. The difference compared to MDVSP is that the time windows are to be taken into account. In order to achieve this, a time window discretization method is applied. The method maps each time window shipment to a set of discrete shipments. Instead of one shipment node per input shipment per layer, we now have (possibly) multiple shipment nodes per input shipment per layer that represent the same shipment, but with different start times.

The time window discretization is constructed by setting a minimum step size δ_s (in minutes) and a maximum number of discrete shipments that is to be generated by a single time window shipment $\delta_m \geq 2$. Because the length of a shipment is fixed, we only consider start times in defining the discrete shipments. For every time window shipment, a discrete shipment is defined for the earliest start time, every δ_s minutes later, and the latest start time. The step size δ_s results in the generation of n^{δ_s} discrete shipments. If this number exceeds δ_m, we redefine the step size by the minimum step size such that the total number of discrete shipments is equal to δ_m. In order to achieve this, the step size is adjusted to

$$\text{step size} = \left\lceil \frac{LST_i - EST_i}{\delta_m - 1} \right\rceil$$

Algorithm 1 provides the pseudocode of the time window discretization procedure for a single time window shipment and Figure 3.4 shows two conceptual examples of time window discretization.

To formalize notation, we denote the set of all possible time window shipments and discrete shipments to be $\bar{\mathbf{S}}$ and $\mathbf{S}$. The discretization mapping that applies the discretization

Algorithm 1: Time window discretization procedure.

INPUT
Time window shipment $\bar{s}_i$
Parameters: δ_s and δ_m

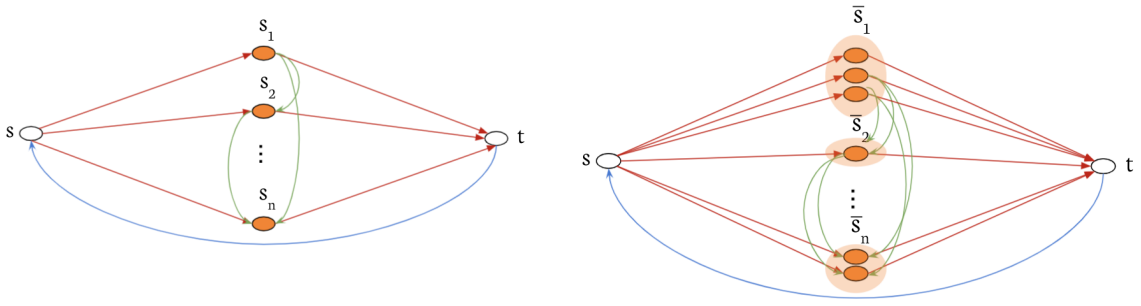
RULES
Define the of shipments generated with step size δ_s:
$n_i^{\delta_s} = \left\lceil \frac{LST_i - EST_i}{\delta_m} \right\rceil + 1$
if $n_i^{\delta_s} \leq \delta_m$ **then**
    Define a discrete shipment for EST_i, every δ_s minutes in the time window, and for LST_i

else
    Define a discrete shipment for EST_i, every $\left\lceil \frac{LST_i - EST_i}{\delta_m - 1} \right\rceil$ minutes in the time window, and for LST_i

method as described above, is denoted as follows:

$$p : \bar{\mathbf{S}} \longrightarrow \mathcal{P}(\mathbf{S})$$
$$s_i \longmapsto p(s) = \{\bar{s}_1^i, \ldots, \bar{s}_{n_i}^i\}$$

where $n_i \leq \delta_m$ is the number of discrete shipments generated by time window shipment $\bar{s}_i$. By defining the discretization mapping this way, we are able to adjust the trade-off between defining few discretized shipments, and therefore not increasing the problem size or defining many discretized shipments and therefore allowing for more flexibility which leads to a better quality solution.

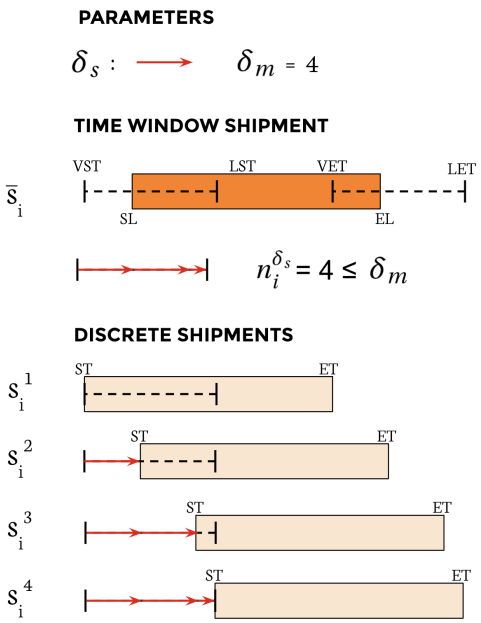


(a): MDVSP graph.

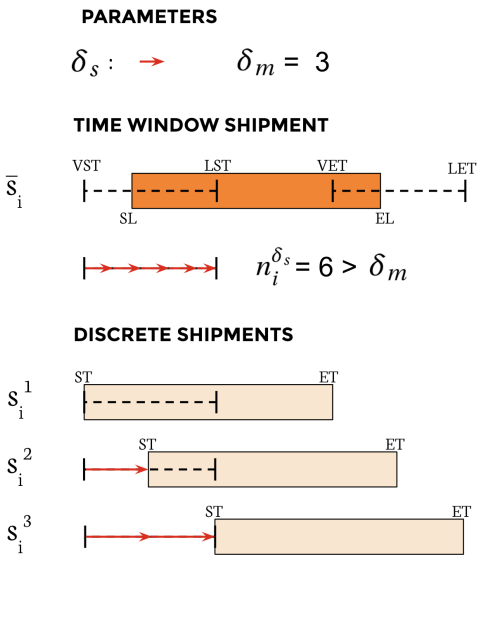
(b): MDVSPTW graph.

Figure 3.5: Schematic view of a single layer of a graph corresponding to the MDVSP and MDVSPTW model.

Figure 3.5 shows the difference between a single layer of the MDVSP graph, where each shipment is represented by a single shipment node, and the MDVSPTW graph, where time



(a): δ_m is greater or equal to the amount of shipments generated by the time window with given step size.



(b): δ_m is smaller than the amount of shipments generated by the time window with given step size.

Figure 3.4: Two conceptual examples of time window discretization.

window shipments are often represented by multiple discrete shipment nodes. The nodes of the MDVSPTW graph are described by $V^m = \{s, p(S), t\}^m$ and the arcs are similar as before. The complexity of the graph increases by applying the time window discretization. Figure 3.6 shows the graph of the model. The ILP corresponding to the MDVSPTW model is as follows

$$\min \quad \sum_{l=1}^{m} \sum_{(i,j)\in A_l} c_{ij}^l f_{ij}^l \tag{3.17}$$

$$\text{s.t.} \quad \sum_{i\in V_l} f_{ij}^l - \sum_{k\in V_l} f_{jk}^l = 0 \qquad \forall j \in V_l,\ \forall l \in [m] \tag{3.18}$$

$$\sum_{l=1}^{m} \sum_{j\in p(j)} \sum_{(i,j)\in A_l} f_{ij}^l = 1 \qquad \forall j \in V_l \tag{3.19}$$

$$0 \leq f_{ij}^l \leq u_{ij} \qquad \forall (i,j) \in A_l \tag{3.20}$$

$$f_{ij} \in \mathbb{Z} \qquad \forall (i,j) \in A_l \tag{3.21}$$

Note that the only difference compared to the MDVSP ILP is the flow requirement constraints (3.19). The inflow of all shipment nodes generated by a single shipment should be exactly one. In the MDVSPTW model this means that not only the shipment nodes in different layers, but also the shipment nodes with different start times corresponding to the same original shipment should be summed over, explaining the additional sum compared

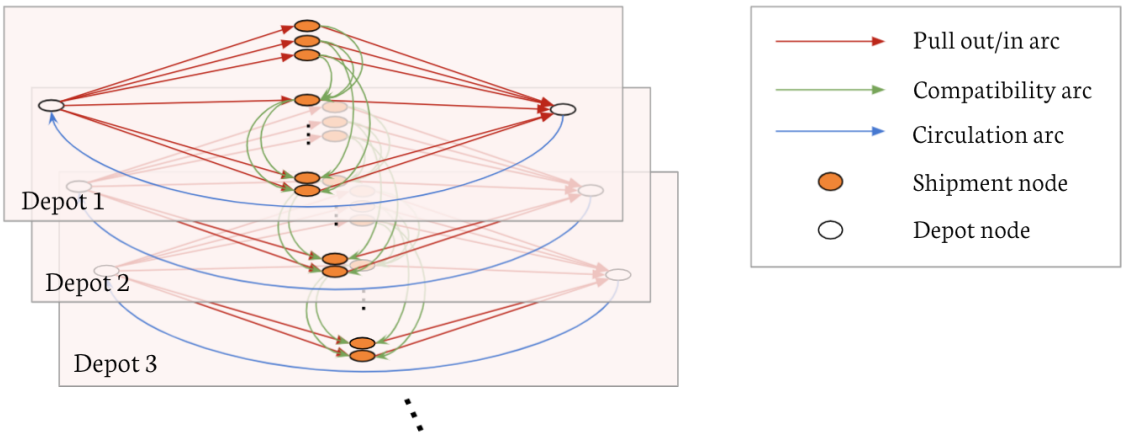


Figure 3.6: The directed graph representing MDVSPTW.

to before.

3.5. The driver day duration restriction

Solving MDVSPTW formulated as an ILP finds an efficient truck planning that uses a minimum number of trucks, while taking into account the depots with their capacities and the time windows in which the shipments are allowed to start and end. However, the model does not take into account the driver day duration restriction. As mentioned in Chapter 1, the driver day duration should be between d_{min} and d_{max} and start and end at the same location in order to be feasible. When modelling the ILP of MDVSPTW as a Minimum-Cost Network Flow Problem, no constraint considers the 'length' of the path. Hence, it is likely that too long truck day durations occur, as maximal truck use benefits the objective. And thus, we cannot guarantee that the truck days are to be executed by one or two drivers while respecting the driver day duration restriction.

Within of Integer Linear Programming, there are two solutions to this problem. One is applicable when allowing a driver change and one when not allowing a driver change.

1. When allowing a driver change: combine the Integrated Vehicle and Crew Scheduling Problem (IVCSP) with MDVSPTW.

2. When not allowing a driver change: combine the Vehicle Scheduling Problem with Length of Path Restrictions (VSPLPR) with MDVSPTW.

The following subsections elaborate on both VSP variants and explain why they are not implemented in this thesis.

3.5.1. Combine IVCSP with MDVSPTW

Vehicle scheduling and crew scheduling are usually approached in a sequential manner, by first scheduling the vehicles and then scheduling the crew. However, when integrating the problems, we are able to find better solutions. The goal is to find minimal-cost feasible truck days and driver days such that both truck and driver plannings are feasible and mutually compatible. First we explain CSP from a public bus transport perspective and then relate the problem to our freight transport case. The Crew Scheduling Problem aims to cover all trips and driving activities with feasible working days. In order to model this, we need the following concepts:

- A *relief point* is, in the context of public bus transport, defined by a location and time where a driver may change its vehicle.
- A *task* $t \in \mathbf{T}$ is a sequence of activities between two consecutive relief points and represents an elementary portion of work that can be assigned to a driver.
- A sequence of tasks that are assigned to a single driver form a *duty* $h \in \mathbf{H}$.

The goal of CSP is assigning tasks to duties, such that

- Each task is executed.
- Each duty contains a feasible sequence of tasks.
- The total cost of the duties is minimizes.

A duty is feasible if it respects certain constraints, such as a minimal and maximal working duration. Notice here the similarity to VSP. However, where VSP could be modelled as a Minimum-Cost Network Flow model, as shown in the previous sections, CSP is usually modelled as a Set Partitioning or Set Covering Problem, as mentioned in Section 3.1. In order to define the ILP model of CSP, we need a new type of decision variable. The decision variable $y_h \in \{0,1\}$ is defined for every duty $h \in \mathbf{H}$ and equal to 1 if and only if the duty belongs to the optimal solution. The set of all duties containing task t is denoted by $\mathbf{H}_t$. The cost function $c : h \mapsto c(h) = c_h \in \mathbb{R}_{\geq 0}$ assigns costs to every duty.

The ILP formulation of the Set Partitioning model of CSP is

$$\min \quad \sum_{h \in \mathbf{H}} c_h y_h \tag{3.22}$$

$$\text{s.t.} \quad \sum_{h \in \mathbf{H}_t} y_h = 1 \qquad \forall t \in \mathbf{T} \tag{3.23}$$

$$y_h \in \{0,1\} \qquad \forall h \in \mathbf{H} \tag{3.24}$$

Objective (3.22) minimizes the total cost of the crew schedule and constraints (3.23) ensure that every task is covered by exactly one duty. In order to model constraints (3.23), all possible feasible duties have to be generated. This results in a huge number of decision variables, making Column Generation a often used method in solving CSP.

When extending this problem to IVCSP, the number of decision variables increases even further, as we have decision variables f_{ij}, in addition to decision variables y_h. This problem can finally be extended to the ILP corresponding to our problem MDVCSPTW. For a detailed explanation on the ILP modelling and solving of MDIVCSPTW, we refer to Kliewer, Amberg, et al. (2012).

Even though MDIVCSPTW is applicable to our problem, we choose not to implement it, because of the following reasons:

- Our crew scheduling restrictions differ from the crew scheduling restrictions that we found in the literature and explained above. In our problem, drivers never change trucks. So, a *relief point* is defined by a location and time where a driver may start or end his or her day. And only at a *split relief point*, that divides a truck day into two feasible driver days, a driver day is allowed to end and begin. This difference compared to the bus public transport application, causes a heuristic sequential method, to be better applicable than in the public transport case.
- The problem size of the ILP is enormous. As a result, it is not likely that we are able to find a feasible solution within reasonable time. However, for bench-marking purposes, the ILP can still be useful. Therefore, implementing the ILP might be interesting for future research.
- The solution scheme presented by Kliewer, Amberg, et al. (2012) involves, among other methods, Lagrangian Relaxation, Column Generation, Subgradient Method and Decomposed Pricing Strategy. Because of the time restrictions of this thesis, we had to choose between either developing a more advanced ILP algorithm or developing heuristics for our problem. The ILP algorithms are not able to guarantee an output

for every input instance, because of the NP-hardness. As a result, we chose to develop heuristics. Also, heuristics are preferred by Picnic, because they are easier to involve on in the ever-changing landscape of the scale-up.

3.5.2. Combine VSPLPR with MDVSPTW

The Vehicle Scheduling Problem with Length of Path Restrictions bounds the length of truck day durations by adjusting the graph of the SDVSP model, see Figure 3.2. The shortcoming of the SDVSP model is that we are not able to consider, and thus bound, the length of a flow cycle. In order to change this, we remove the two depot nodes and adjacent arcs, as representatives of the start and end of the day, and add *backward arcs*. A backward arc connects shipment s_k to shipment s_j if and only if starting the day executing s_j and ending the day executing s_k results in a feasible day duration and is thus feasible. Formally, this is the case if the following holds:

$$d_{min} \leq d(d, SL_j) + (ET_k - ST_j) + d(EL_k, d) \leq d_{max}. \tag{3.25}$$

The new graph is defined by the shipment nodes $V = \{S\}$ and the arcs $A = \{A_c, A_b\}$, where A_c is the set of compatibility arcs and A_b the set of backward arcs. Instead of all flow cycles in the solution flowing back to the start depot node through the circulation arc, flow cycles are constructed by a path from one shipment node to another and a single backward arc, that represents the finishing of a working day. Figure 3.7 shows a graph corresponding to VSPLPR and an example of a solution consisting of two truck days. The costs of the

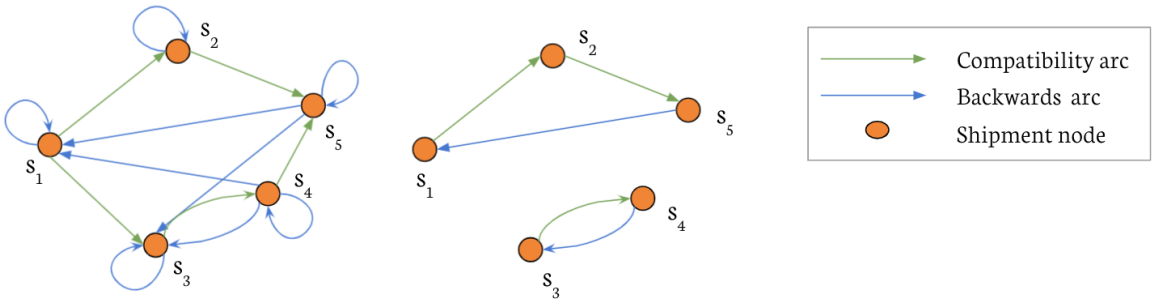


Figure 3.7: The directed graph representing VSPLPR and an example of a solution.

compatibility arcs remain equal to the operational costs. The costs of the backward arcs are equal to the operational costs of the pull out and pull in trips and the fixed costs of adding a new truck to the planning. Note that this is equal to the costs that were assigned to the pull out/in arcs before. The ILP corresponding to this model introduces in addition to variables f_{ij}, that now only correspond to compatibility arc (i, j), another type of decision variable

q_{ij} corresponding to backward arc (i, j).

$$\min \quad \sum_{(i,j)\in A_c} c_{ij} f_{ij} + \sum_{(i,j)\in A_b} c_{ij} q_{ij} \tag{3.26}$$

$$\text{s.t.} \quad \sum_{i:(i,j)\in A_c} f_{ij} + \sum_{i:(i,j)\in A_b} q_{ij} - \sum_{i:(j,i)\in A_c} f_{ji} - \sum_{i:(j,i)\in A_b} q_{ji} = 0 \qquad \forall j \in V \tag{3.27}$$

$$\sum_{i:(i,j)\in A_c} f_{ij} + \sum_{i:(i,j)\in A_b} q_{ij} = 1 \qquad \forall j \in V \tag{3.28}$$

$$\sum_{(i,j)\in A_c\cap C} f_{ij} + \sum_{(i,j)\in A_b\cap C} q_{ij} \leq |C| - 1 \qquad \forall \text{ cycles } C \text{ with } |A_b \cap C| \geq 2 \tag{3.29}$$

$$0 \leq f_{ij} \leq u_{ij} \qquad \forall (i,j) \in A_c \tag{3.30}$$

$$0 \leq f_{ij} \in \mathbb{Z} \qquad \forall (i,j) \in A_c \tag{3.31}$$

$$0 \leq q_{ij} \leq 1 \qquad \forall (i,j) \in A_b \tag{3.32}$$

$$q_{ij} \in \mathbb{Z} \qquad \forall (i,j) \in A_b \tag{3.33}$$

Apart from the separated arcs in the notation, the ILP has similar constraints as in the SDVSP case: the costs are minimized by constraints (3.26), the flow is conserved by constraints (3.27), the flow requirements through all shipment nodes is guaranteed by constraints (3.28) and the amount of flow is restricted by a capacity and integer by constraints (3.30) and constraints (3.31). The only new type of constraints is (3.29), that guarantees that each flow cycle in the solution contains exactly one backward arc. As the meaning of a backward arc is 'finishing the working day', visiting two backward arcs within one truck day is not possible.

Experiments indicate that the problem size of the ILP above is too large to be solved for instances of the size that occur at Picnic. That is the case, because the number of backward arcs far exceeds the number of stem and circulation arcs that they replaced and the addition of constraints (3.29). Even when adding assumptions that decrease the number of backward arcs, such as defining a minimum start time for shipments that are allowed to be the last shipment of a truck day, the problem size remains too large. And if we would be able to reduce the problem size of the ILP above, we did not incorporate the multi-depot and time window restrictions yet, that increase the problem size substantially. Thus, even though modelling a constraint that bounds the duration of truck days would define an ILP that represents our problem when allowing no driver change, we do not implement VSPLPR in our algorithms because it increases the problem size too much.

Because we are not able to implement IVCSP and VSPLPR into our ILP, it remains a challenge to incorporate the driver day duration restriction. Within Integer Linear Programming, it is difficult to find a solution for this problem, because of the time restriction. And even if we would be able to construct an algorithm that is able to find a feasible solution for the ILP of our problem within reasonable time for our test data, the algorithm violates the robustness requirement of the algorithm. Because of the NP-hardness, we are not able to guarantee an output within reasonable time for every input instance.

However, heuristics are able to guarantee the return of a feasible solution within reasonable time, at the cost of not being able to find an optimal solution. In Chapter 6, an algorithm is introduced that takes a solution to the ILP of MDVSPTW as initial solution and heuristically applies adjustments in order to create a solution that respects the driver day duration restriction. But first, we introduce a Greedy algorithm from the literature that is able to generate a solution for MDVSP in very short time.

4

A Greedy heuristic for MDVSP

In the previous chapter, we noted that Integer Linear Programming can be used to model MDVSPTW and find an optimal solution. However, as the problem is NP-hard, we are not able to guarantee a feasible solution within limited computation time. Also, the solution for MDVSPTW is not a feasible solution for our problem, because the driver day duration restriction is not incorporated. While in practice, a good feasible solution might already be of great value. For these reasons, this chapter reports on existing heuristic approaches from the literature for our problem and relevant subproblems. Section 4.1 considers a short literature review on heuristics for our problem. Section 4.2 considers a Greedy scheduling heuristic for MDVSP, the Concurrent Scheduler algorithm. This heuristic lays the foundation for the Greedy Scheduler algorithm for MDVSPTW that is introduced in the next chapter and plays an important role in the main algorithms.

4.1. Literature review: heuristics to solve our problem

Because every subproblem of our problem is NP-hard, except for SDVSP, often heuristical approaches are proposed in the literature. The majority of these approaches involve solving ILPs heuristically. Specifically, Langrangian heuristic in combination with Column Generation often occurs. We choose to focus on finding heuristical approaches that do not involve ILP formulations, because these are easier to involve on in the ever-changing landscape of Picnic as a scale-up. Therefore, we consider the heuristical approaches from the literature that do not involve an ILP below. For more details on the Column Generation and Langrangian heuristic approaches for out problem, we refer the reader to (Kliewer, Amberg, et al., 2012).

Only few heuristics for our problem or relevant subproblems that do not involve an ILP, are introduced in the literature. And all algorithms we found, aim to find a solution for the subproblem MDVSP. The first Greedy approach in the literature is the Concurrent Scheduler algorithm, introduced by Bodin et al. (1978). We explain this algorithm detail in Section 4.2, because it lays the foundation for the Greedy Scheduler algorithm we introduce in Chapter 5. Pepin et al. (2009) compared five heuristics for MDVSP, of which the final two, Large Neighborhood Search in combination with Column Generation and Tabu Search, were not introduced before. From all five methods, only the Tabu search algorithm does not involve an ILP formulation. The Tabu search algorithm defines two types of neighborhoods: based on shifting and swapping. Shifting means that one trip assigned to a truck is *shifted* to another truck, i.e., assigned to another truck. Swapping means that two trips assigned to trucks, *swap* trucks, i.e., each are assigned to the other truck. The algorithm does not provide the quality of solution that the other four methods are able to provide. In other study, Laurent et al. (2009) introduced an Iterated Local Search algorithm for MDVSP, that defines a neighborhood by allowing *block-moves*, that is, apply the following scheme to a truck planning:

1. Randomly select a truck and randomly select a *block* of consecutive trips assigned to that truck. Remove the block from the truck.
2. Randomly select a different truck to assign the block to this new truck.
3. If no conflicts occur, we are done. Otherwise, continue.
4. Remove all trips from the truck that are in conflict with the block.
5. Assign all removed trips to the cheapest truck possible.
6. If there is no other truck to assign a trip to, assign the trip to a new truck added to the planning.

The neighborhood structure based on block moves outperforms the shift and swap neighborhood structures. Finally, Wen et al. (2016) presents an Adaptive Large Neighborhood Search algorithm for the Electric Vehicle Scheduling Problem, but this problem is too different from our problem to be applicable directly.

4.2. The Concurrent Scheduler algorithm for MDVSP

As mentioned in Chapter 2, one method for finding a feasible solution in short time is to apply an intuitive, often Greedy, algorithm with short computation time. This section considers the Concurrent Scheduler algorithm, a Greedy algorithm to find a solution for MDVSP. The Concurrent Scheduler algorithm was coined by Bodin et al. (1978) and constructs

a feasible solution for MDVSP by iterating through an ordered list of shipments and greedily assigning the shipments to the trucks. Algorithm 2 shows the pseudocode of the Concurrent Scheduler.

First, Bodin et al. (1978) order the shipments according to increasing start times. Every

Algorithm 2: Concurrent Scheduler for MDVSP.

```
1  INPUT
2  Ordered list of discrete shipments [S] based on increasing start times
3  Set of depots D with capacities κ_d for all d ∈ D
4
5  RULES
6  while [S] ≠ ∅ do
7      for s_i ∈ [S] do
8          if feasible active trucks ≠ ∅ then
9              Assign s_i to the feasible active truck with last shipment s_j that minimizes
                the operational costs(i, j)
10
11         else
12             Assign s_i to non-active truck that minimizes the operational costs(d, s_i)
                without exceeding κ_d
13
14         Remove s from [S]
15
```

shipment s_i is assigned to a truck by first considering the trucks that already have shipments assigned to them, i.e., *active trucks*. An active truck is *feasible* for s_i if for the last shipment s_j assigned to the truck, the following holds:

$$ET_j + d(EL_j, SL_i) \leq ST_i. \tag{4.1}$$

If the set of feasible active trucks is non-empty, the costs are determined for executing the shipment after the currently last shipment on the truck. These costs are equal to the operational costs of executing potential new shipment s_i after current last shipment s_j, similarly defined as in (3.6).

$$\begin{aligned} \text{operational costs}(s_i, s_j) &= w_{wt} \times \text{waiting time}(s_i, s_j) + w_{dt} \times \text{empty driving time}(s_i, s_j) \\ &= w_{wt} \times (ST_j - ET_i - d(EL_i, SL_j)) + w_{dt} \times d(EL_i, SL_j) \end{aligned} \tag{4.2}$$

Shipment s_i is assigned to the cheapest feasible active truck. If there is no active truck that can be at the start location at the start time of the shipment, the shipment is assigned to a

truck that has no shipments assigned to it yet, i.e., *non-active truck*. From the non-active trucks, the shipment is assigned to a truck from depot d that minimizes the operational costs defined as follows:

$$\begin{aligned} \text{operational costs}(d, s_j) &= w_{dt} \times \text{empty driving time}(d, j) \\ &= w_{dt} \times d(d, SL_j) \end{aligned}$$

without exceeding the capacity κ_d. After assigning all shipments to the trucks, the algorithm returns the obtained truck planning.

5

Two building blocks

This chapter introduces two algorithms that build upon the existing approaches for our problem and relevant subproblems from the literature, presented in the previous two chapters. Section 5.1 introduces assumptions to decrease the problem size of the implementation of the ILP from Section 3.4. As a result, we are able to find a good solution for the ILP within reasonable computation time. Section 5.2 introduces the Greedy Scheduler algorithm, inspired by the Concurrent Scheduler algorithm from Section 4.2, but specifically applied to our problem. Both algorithms are important building blocks in defining the main algorithms for our problem in the next chapter.

5.1. Finding a solution for the ILP of MDVSPTW

The commercial optimization solver Gurobi Optimization (2020) enables us to implement an ILP and applies built-in optimization techniques, such as Branch and Bound and Cutting Plane algorithms, to find a solution. Section 3.4 introduced the ILP that represents MDVSPTW, a relaxation of our problem without the driver day duration restriction incorporated. We aim to find a good solution for this relaxation of our problem and then adjust the solution in order to be feasible for our problem. These adjustments are applied heuristically in the algorithm introduced in Section 6.1. Experiments indicate that the computation time required to solve the MDVSPTW ILP for instances of the size existing at Picnic far exceed the acceptable computation time. Therefore we consider methods that decrease the problem size of MDVSPTW in order to be able to define an ILP implementation that finds a feasible solution within reasonable time.

5.1.1. Methods to decrease the problem size of the MDVSPTW ILP

The number of nodes and arcs of the MDVSPTW graph determines the problem size of the ILP, as the decision variables correspond to the arcs and the constraints are based on the in- and outflow of the nodes. Recall that the input consists of m depots and n time window shipments, where realistically $m \ll n$. Each time window shipment generates a maximal number of d_m discrete shipments. So, the total number of nodes is of the order $\mathcal{O}(m * d_m * n)$, dominantly determined by the number of shipment nodes. The total number of arcs is of the order $\mathcal{O}((d_m * n)!)$, dominantly determined by the number of compatibility arcs. As a result, we aim to decrease the number of shipment nodes and the number of compatibility arcs. The following method decrease the number of compatibility arcs:

- **Add a maximum waiting time t_{wt}^{max}.**
  As given in Section 3.2, the compatibility arc (s_j, s_k) is added if the following equation holds:

  $$ET_j + d(EL_j, SL_k) \leq ST_k. \tag{5.1}$$

  Compatibility arcs connect every pair of shipments that can be executed after each other. But, for a shipment s_j that has to be executed in the morning and a shipment s_k that has to be executed in the evening, the constraint above is likely to not be restrictive. Therefore, we do not add compatibility arcs between shipments that have a long waiting time between them, without losing optimality. To formalize this idea, the equation determining if shipments s_j and s_k are connected by a compatibility arc is updated to the following:

  $$ET_j + d(EL_j, SL_k) \leq ST_k \leq ET_j + d(EL_j, SL_k) + t_{wt}^{max}. \tag{5.2}$$

  In determining the value of t_{wt}^{max}, we want to find a balance between the effectiveness in decreasing the number of compatibility arcs and not losing optimality. In order to decide on what assumptions to make to decrease the problem size, but not lose quality of solution, we set up an experiment in Chapter 7.

The following methods decrease the number of nodes:

- **Minimize the time window discretization parameter d_m and maximize the time window discretization parameter d_s.**
  As explained in Section 3.4, choosing the maximal number of discrete shipments d_m that are allowed to be generated by a single time window shipment and the minimal step size d_s, is a trade-off between problem size and quality of solution. When aiming to decrease the problem size, we can choose d_m to be as small as possible, and d_s to be as large as possible, without decreasing the efficiency of the truck planning too

much.

- **Make assumptions on the depot from which a certain type of the shipments are executed, also called *depot assumptions*.**
  The depot from which a shipment is executed in an efficient planning, is not completely random. In an efficient truck planning, a shipment is likely to be executed by a truck that starts and finishes at a depot close to the start and end location(s) of the shipment. This insight gives us the possibility to decrease the problem size, by making assumptions on the depot from which a truck executes a shipment, with a small probability of cutting off good solutions. Given a shipment, each depot that we exclude from the set of depots from which this shipment might be executed, decreases the number of nodes by at most d_m. That is the case, because excluding the depot means removing all shipment nodes corresponding to the shipment from the depot-layer of the graph.

  In deciding on what assumptions to make, we distinguish between IB and OB shipments. Because OB shipments contain the trip from the FC to the Hub and back, they have an FC as the same start and end location. Therefore, it is easier to predict from what depot the OB shipments are to be executed in an efficient planning: the FC they start and end, or a location close to that FC. For IB shipments, this is more difficult, because the driving between the start and end location might be 2 hours. In order to decide on what depot assumptions to make to decrease the problem size, but not lose quality of solution, we set up an experiment in Chapter 7.

The methods above enable us to bring the computation time down to our set time limit. Whenever referring to *our ILP implementation*, we mean the implementation of the ILP of MDVSPTW with the problem size decreasing methods above.

There are two methods in the literature to decrease the problem size that we did not implement in our algorithms. The first method is to rewrite the problem to a Time-Space Network, as introduced by Kliewer, Mellouli, et al. (2006). This method exploits the transitivity property of partially ordered sets by aggregating possible connections between groups of compatible shipments. The second method is to remove part of the compatibility arcs that are most costly, as they are not likely to appear in the optimal solution (Haghani et al., 2002). This increases the risk of losing optimality, but the proportion of arcs to remove can be tweaked precisely when bench-marking with the complete ILP. Both methods could be implemented if considering larger instances in future work.

5.1.2. Defining a termination criterion for solving the ILP

In addition to the methods to reduce the problem size of MDVSPTW, we can also control the computation time by not solving the ILP to optimality, but to return the best solution from the solution pool when reaching a termination criterion. Of the three termination criteria mentioned in 2.1, two are relevant for our implementation:

- **Time limit τ_{max}.**
  The time limit is a fixed number of seconds after which the algorithm terminated and returns the best solution from the solution pool. If the time limit is set too small, however, the solution pool may be empty and no solution is returned.

- **Optimality gap bound δ_{gap}.**
  The algorithm is terminated if a feasible solution is found for which the optimality gap smaller is than the set bound.

These termination criteria allow the ILP implementation to return a feasible solution in time, instead of continuing until the optimum is found.

5.2. The Greedy Scheduler algorithm

Apart from finding good feasible solutions for our problem with Integer Linear Programming techniques, for robustness reasons we also want to find a heuristic approach to solve our problem, that does not involve an ILP. This section explains the extention from the Concurrent Scheduler algorithm for MDVSP, introduced in Section 4.2, to the Greedy Scheduler algorithm, specifically designed for our problem.

5.2.1. Finding a feasible solution for our problem

Recall that for our problem the input shipments $\bar{S} = \{\bar{s}_1, \ldots, \bar{s}_n\}$ are time window shipments, and therefore defined by earliest start time EST_i, latest start time LST_i, earliest end time EET_i, latest end time LET_i, start location SL_i and end location EL_i, for all $i \in [n]$. Similar to the Concurrent Scheduler algorithm, we iterate through an ordered list of shipments and greedily assign them to an active truck if possible and otherwise to a non-active truck. However, the following specifics are different than before:

1. **The order of the shipments.**
   As the shipments have different properties compared to MDVSP, and no fixed start time, we have to define a new property to order them by. We choose to order the shipments by latest start time LST_i, as this indicates which shipments have their 'deadlines' first, and therefore have a higher priority to be scheduled first. Because it might occur that many shipments have equal latest start time, a second property is

chosen to break ties. This is the length of the time window, scheduling the shipments that have small time windows, and are therefore less flexible, first.

2. **The evaluation of the feasible active trucks.**
   A truck with shipment discrete s_j assigned to it last is now feasible for a time window shipment $\bar{s}_i$ if the following holds:

$$LST_i \geq ET_j + d(EL_j, SL_i). \tag{5.3}$$

   The costs of assigning $\bar{s}_i$ to a feasible active truck are equal to the cost of the cheapest feasible discrete shipment, generated by $\bar{s}_i$, assigned to the truck after s_j, this discrete shipment is denoted by s_i^j. We only need to define the start time of s_i^j, as all other properties are already determined by $\bar{s}_i$. The start time of the cheapest feasible shipment s_i^j is the earliest start time possible, after s_j, that is:

$$ST_i = \min\{EST_i,\ ET_j + d(EL_j, SL_i)\}. \tag{5.4}$$

   Now, the costs of assigning $\bar{s}_i$ are equal to the operational costs, defined by equation (4.2), of assigning s_i^j to the truck with last shipment s_j.

3. **The evaluation of the non-active trucks.**
   This procedure is very similar to before, the only difference is that the fixed start time of $\bar{s}_i$ is set to be equal to the earliest start time: $ST_i = EST_i$, as this leads to more possibilities for another shipment to be assigned after s_i later on.

The algorithm is now able to find a feasible solution for MDVSPTW. But in order to make it applicable to our problem, we have to include the driver day duration restriction. Therefore, we first present a method to guarantee that each driver day duration is more than d_{min} and then a method to guarantee the duration is less than d_{max}.

In order to guarantee that each driver day duration is more than d_{min}, we make an adjustment to the truck planning obtained after applying the rules above. If all shipments have been assigned to the trucks, we detect the truck days that have a duration less than d_{min}. We add *additional waiting time* to the truck day, until the duration is equal to d_{min}. Of course, in practice the driver does not have to wait during these hours. Figure 5.1 shows an example of a truck day with additional waiting time for $d_{min} = 7$. The truck day duration was 4.5 hours, and therefore too short. In order to make the truck day feasible, 2.5 hours of additional waiting time is added.

In order to guarantee that the driver day duration is less than d_{max}, we add a criterion

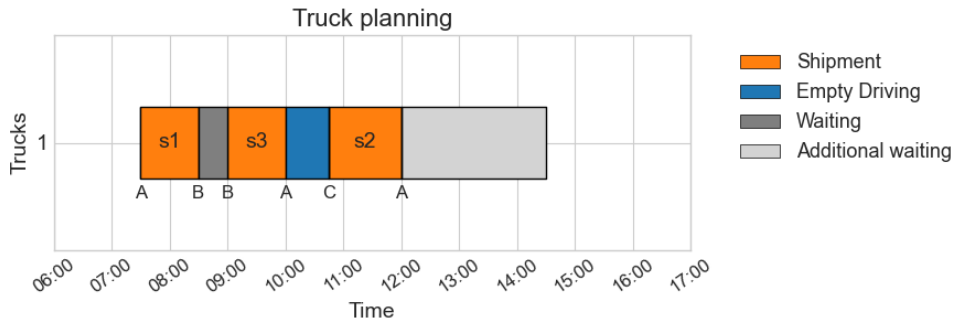


Figure 5.1: Example of a short truck day with additional waiting time with $d_{min} = 7$.

to equation (5.3) that decides whether an active truck is feasible for a shipment. Instead of allowing the shipment to be assigned to every truck that is able to execute it after its currently last shipment, we take the total duration of the truck day into account. An active truck with depot d_t, first shipment s_k and last shipment s_j is feasible for shipment $\bar{s}_i$, with cheapest feasible shipment s_i^j, if in addition to (5.3), the following holds:

$$\text{truck day duration with shipment } = d(d_t, SL_k) + (ET_i - ST_k) + d(EL_i, d_t) \leq d_{max}. \quad (5.5)$$

When executing the Greedy Scheduler algorithm in the main algorithms, we might want to relax this constraint, in particular if we want to generate splittable truck days. We refer to the Greedy Scheduler algorithm without the truck day duration constraint as the *Greedy Scheduler algorithm without maximal day duration*. We refer to the version of the algorithm with the constraint above incorporated into it as the *Greedy Scheduler algorithm with maximal day duration* or simply the *Greedy Scheduler*.

5.2.2. Improving the algorithm

Taking the adaptations to the Concurrent Scheduler algorithm into account, we can construct a feasible truck planning for our problem from a set of time window shipments. However, we can improve the quality of the truck planning by adding the following improvements to the algorithm:

- **The introduction of last shipment indicators.**
  While greedily assigning the shipments to the trucks, the algorithm does not take into account that every truck should return to the depot where it started by the end of the day. Therefore, it is likely that many trucks drive back empty to their start depot at the end of their day, which increases the inefficiency of the planning. In order to prevent this from happening, we would like to predict what shipments will be the last shipments of the truck day. Because then we can give a preference to the shipments that return at the same depot where the truck started. We do this by first defining last

shipment indicators, to decide which shipment should be taken into account when giving this preference. A shipment s_i, assigned to a truck, is classified as a potential last shipment if the following holds:

$$ET_i > l_e \quad \text{or} \quad \text{duration of truck day with the shipment} > l_d, \tag{5.6}$$

where l_e is a fixed time and l_d is a fixed duration. Now, for any shipment s_i for which the above is true, the operational costs include the costs of driving back to the depot d:

$$\text{operational costs}(s_j, s_i) = \text{operational costs}(s_j, s_i) + \text{operational costs}(s_i, d). \tag{5.7}$$

Again, the operational costs are equal to the operational costs in the ILP algorithm and defined by equations (3.6) and (3.5). Here, operational costs (s_i, d) functions as a penalty when the distance between d and EL_i is large, making it preferable to select a last shipment on a truck day that ends close to d.

- **The introduction of a tie-breaking rule.**
  Because shipments have no fixed start times, but start time windows, the probability that a shipment is able to start precisely when the truck arrives at the start location of the shipment, increases significantly compared to Concurrent Scheduler for MD-VSP. Therefore, when considering a shipment to assign to a truck, many trucks have no waiting time and operational costs equal to only the empty driving costs. This is an advantage to the efficiency of the planning. However, it also causes more ties to appear: the operational cost of every truck that is able to execute the shipment without waiting time is equal if the truck is currently placed at the same location. When choosing the truck with minimal costs, our experiments indicate that for 24% of all shipments multiple trucks have minimal costs, see Appendix B.1.

  We choose to leverage the occurrence of ties to prioritize trucks that remain close to their depot. Considering the set of trucks with minimal costs, we prefer to assign a shipment to a truck that has a depot not to far away from the end location of the shipment. This results in less empty driving hours, as the truck stays close the the depot it returns to at the end of the day. It also leads to more relief points, as the truck is be more likely to return to execute a shipment at its depot if it stays relatively close to it, which increases the probability that splittable truck days occur. Moreover, there are operational advantages to keeping trucks local, as a disturbance in the planning also stay locally. We formally implement this idea by considering the truck of minimal costs, and assigning the shipment to a truck that minimizes the distance between

the depot of the truck and the end location of the shipment. Our experiments indicate that for 18% of the shipments, multiple trucks have minimal costs and minimal distance. We assign the shipment to a randomly chosen truck from this set, see Appendix B.1.

- **Not only scheduling shipment after a truck day, but also before.**
  Until now, when assigning a shipment to a truck, we only considered the possibility to schedule the shipment after the current truck day. However, it may as well be an efficient option to schedule the shipment before the current truck day. This is in particular relevant, when we give active trucks to the algorithm as input. In order to formalize this, we notice that most *after* arguments and equations used above can be symmetrically translated to *before* arguments and equations. For example, a truck with first shipment s_k and last shipment s_j is now feasible for shipment $\bar{s}_i$ if the following holds:

$$LST_i \geq ET_j + d(EL_j, SL_i) \qquad \text{or} \qquad LET_i - d(EL_i, SL_k) \leq ST_k, \tag{5.8}$$

  and the truck day duration with shipment is smaller than the maximal day duration d_{max}. For the evaluation of non-active trucks, we predict whether a shipment is likely to be a first or last shipment based on the latest end time of the shipment. If the latest end time is earlier than 16:00, we assign the shipment to a truck close to its start location, and define the start time to be its earliest start time. For a latest end time later than or equal to 16:00, naturally, the truck is close to its end location and the start time is the latest start time. Formally, we define the start time of shipment $\bar{s}_i$ as follows:

$$ST_i = \begin{cases} EST_i, & \text{if } LET_i \text{ is earlier than } 16:00. \\ LST_i, & \text{otherwise.} \end{cases} \tag{5.9}$$

  The last shipment indicators are slightly adjusted. A shipment is classified as a last shipment if the following holds:

$$ET_i > l_e \quad \text{or} \quad (\text{duration of truck day with the shipment} > l_d \quad \text{and} \quad ST_i > 16:00). \tag{5.10}$$

  In general, last shipments start after 16:00. The requirement that the start time is later than 16:00 is added to distinguish last shipments from first shipments, that start before 16:00 in general. In addition to the last shipment indicators, we have the first shipment indicators e_s and e_d, that classify a shipment s_i as a first shipment if the

following holds:

$$ST_i < e_s \quad \text{or} \quad (\text{duration of truck day with the shipment} > e_d \quad \text{and} \quad ST_i \leq 16:00). \tag{5.11}$$

By symmetric argumentation, we assume that $e_d = l_d$. The great time window length of a shipment might allow a shipment to be feasibly executable before and after a truck day. In order to prevent ties and errors in the argumentation, we choose to give preference to scheduling the shipment after the truck day, if both is possible.

Algorithm 3: Greedy Scheduler for MDVSPTW.

```
1  INPUT
2  Ordered list of shipments [S̄] = {s̄_1, ..., s̄_n} with start and end time windows
3  Set of depots D = {d_1, ..., d_m} with κ_d trucks available
4
5  RULES
6  while [S̄] ≠ ∅ do
7      for s̄_i ∈ [S] do
8          if feasible active trucks ≠ ∅ then
9              Assign s̄_i to the feasible active truck with last shipment s_j that minimizes
                the operational costs(s_j, s_i^j), where s_i^j is the cheapest feasible shipment.
                Define the start time of s_i to be ST_i = min{EST_i, ET_j + d(EL_j, SL_i)}.
10
11         else
12             Assign s̄_i to a non-active truck that minimizes the operational costs(d, s_i)
                without exceeding κ_d. Define the start time of s_i to be ST_i = EST_i.
13
14         Remove s from S_l
15
```

Algorithm 3 provides the pseudocode of the Greedy Scheduler algorithm for MDVSPTW.

The Greedy Scheduler has various advantages:

- The algorithm is able to generate a feasible solution within short computation time, because of its simplicity.
- The algorithm is able to take an incomplete truck planning as input, by initialising not all trucks to be empty, but already with shipments assigned to them. Then, we continue with assigning the set of shipments that were not yet assigned to trucks, and complete the planning. This is a useful for combining the algorithm with other algorithms, which we do in main algorithms.

- The algorithm is flexible. Adding a new constraint, such as a maximal day duration, is relatively easy compared to the ILP implementation.

The Greedy Scheduler algorithm is applicable to MDVSPTW. In addition, depending on the precise definition of the feasible active trucks, the algorithm is also applicable to MDVSPTW with a maximal truck day duration, as is further explained in Chapter 6.

5.3. Additional post-processing steps

The two implementations introduced in the previous sections are important building blocks for the main algorithms that the following chapter presents. In addition, we introduce two post-processing steps that are not integrated parts of the two algorithms above, but can be seen as small improvement building blocks. These post-processing steps take a truck planning as input and are only able to improve its objective value.

1. **Depot improvement**
   The goal of depot improvement is to decrease the total pull out and pull in time, i.e., the sum of the driving time between the depot and the start location of the first shipment, and the driving time between the end location of the last shipment and the depot. Each truck day is essentially defined by a depot and a feasible sequence of shipments. The algorithms above do not guarantee that each shipment sequence is assigned to a depot that minimizes this sum. As a result, rearranging the combinations of shipment sequences and depots might decrease the total costs of the truck planning. The depot improvement procedure is executed by checking for every shipment sequence whether there is a depot other than the current depot, that has not reached its capacity yet, and that strictly decreases the total pull out and pull in time. If such a depot exists, the sequences of shipments are removed from the current truck and assigned to a truck at the improving depot. Figure 5.2 shows a conceptual example of depot improvement, where the total pull out/in time is reduced by 0.5 hour, because we changed the depot from A to B.

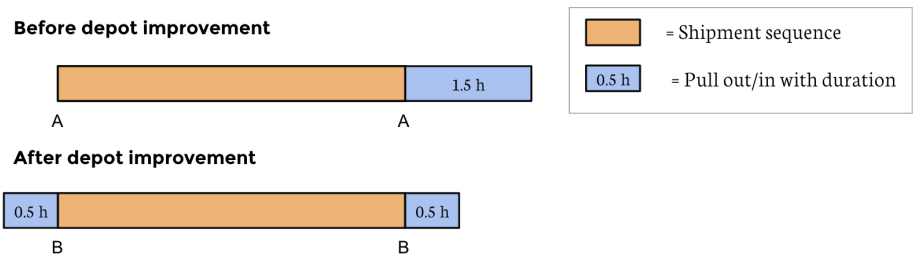


Figure 5.2: Conceptual example of depot improvement.

2. **Shipment sliding**

   The time windows enable us to *slide* shipments, that is, assign new start times to the shipments without changing the order of the sequence. Because, neither algorithms above guarantee minimal waiting time, *shipment sliding* might decrease the waiting time of a truck. This can be seen as pressing the 'air' out of a truck day, by pushing the shipments closer to each other. The shipment sliding procedure is executed by considering two consecutive shipments at a time, and iterating through the sequence. We can either choose to start at the front of the sequence, and execute *forward sliding*, or start at the end of the sequence and execute *backward sliding*. Whenever we refer to *shipment sliding*, we mean executing both variants in sequence. Figure 5.3 and Figure 5.4 show conceptual examples of effective forward and backward sliding respectively.

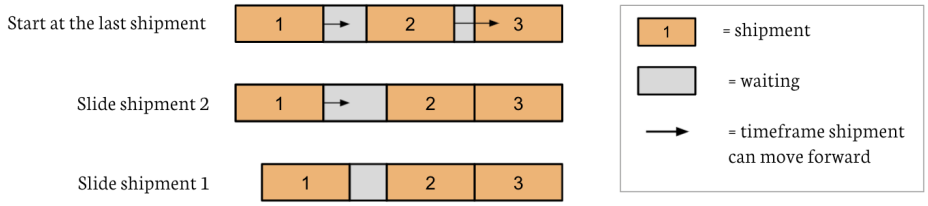


Figure 5.3: Conceptual example of forward shipment sliding.

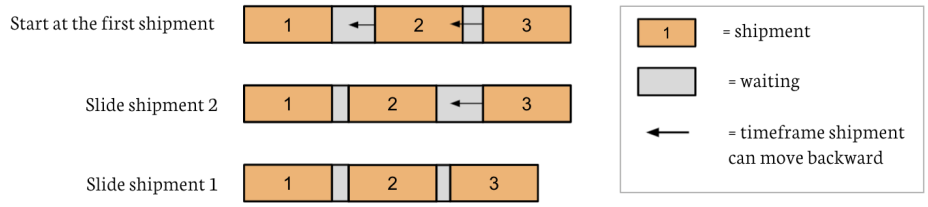


Figure 5.4: Conceptual example of backward shipment sliding.

6

Three main algorithms

The previous chapter provided us with two important building blocks: the ILP implementation and the Greedy Scheduler. Based on the building blocks, we define three main algorithm that find good feasible solutions for our problem. This chapter names and explains these three main algorithms. Section 6.1 introduces the ILP + Greedy algorithm that combines the ILP implementation and the Greedy Scheduling algorithm. Sections 6.2 and 6.3 introduce two heuristics, Random Search and Random Search and Fix, that only use the Greedy Scheduler algorithm. For each of the three main algorithms we introduce one version that allows a driver change and one version that does not allow a driver change.

6.1. ILP + Greedy algorithm

With our ILP implementation from Section 5.1, we are able to find a good or even optimal solution for MDVSPTW within reasonable computation time. This truck planning is likely to be efficient, compared to the truck plannings from the greedy approaches. However it is not a feasible solution for our problem, as incorporating the driver day duration restriction in the ILP formulation causes the computation time to exceed our limits. The more flexible character of Greedy algorithms causes the driver day duration restriction to be easily incorporated in the Greedy Scheduler algorithm. However, the algorithm lacks guarantee of solution quality. The complementary properties of the two methods suggests a combination of both.

This section introduces the ILP + Greedy algorithm, that takes the ILP solution as and adjusts it to a feasible solution, by applying the Greedy Scheduler algorithm. First, we consider a version of the algorithm in which the driver day duration restriction is incorporated

with allowing a driver change. Then we consider a version in which the driver day duration restriction is incorporated when not allowing a driver change.

6.1.1. ILP + Greedy without driver change

The algorithm consists of four steps:

1. Find a feasible solution for MDVSPTW by using our ILP implementation.
2. Remove the infeasible trucks from the planning.
3. Split the splittable truck days into two separate truck days.
4. Reassign all removed shipments to the truck planning using the Greedy Scheduler algorithm.

Figure 6.1 provides an overview of the algorithm. When a driver change is not allowed, all truck days have to be feasible driver days. The ILP solution contains three types of truck days: truck days that are feasible driver days, truck days that exceed the maximal driver day duration, but are splittable and truck days that exceed the maximal driver day duration and are not splittable. The first type is feasible for our problem, so we keep them. The second type is infeasible, so we remove them from the truck planning. The third type is also infeasible, but we can make the truck days into feasible driver days by splitting them into two truck days. After splitting the splittable truck days, the final step is to add the removed shipments, that were assigned to the removed trucks, to the planning again. We do this by taking the feasible trucks as input trucks for the Greedy Scheduler and assign the removed shipments in a Greedy manner.

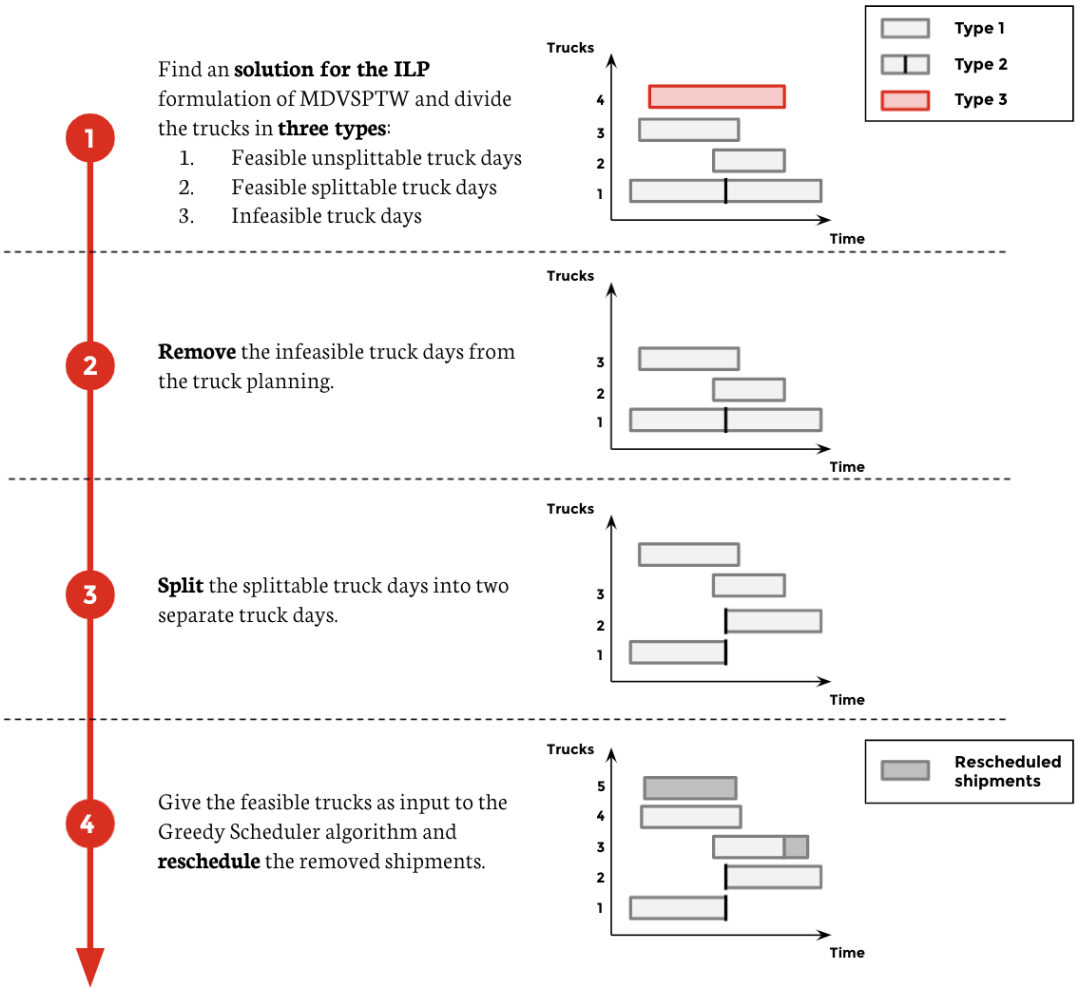


Figure 6.1: An overview of ILP + Greedy without algorithm driver change.

6.1.2. ILP + Greedy with driver change

This algorithm is very similar to the algorithm when no allowing a driver change. The only difference is that we skip the third step. So, the algorithm consists of three steps:

1. Find a feasible solution for MDVSPTW by using our ILP implementation.
2. Remove the infeasible trucks from the planning.
3. Reassign all removed shipments to the truck planning using the Greedy Scheduler algorithm.

Figure 6.2 provides an overview of the algorithm. The ILP solution contains of the same three types of truck. But, because we allow a driver change, not only the first type, but also the second type is already feasible for our problem. That is, not only the truck days that are feasible driver days, but also the truck days that exceed the maximal driver day duration and are splittable. Therefore, we only have to remove the truck days that exceed the maximal driver day duration and are splittable, and add the removed shipments again to the planning, using the Greedy Scheduler algorithm.

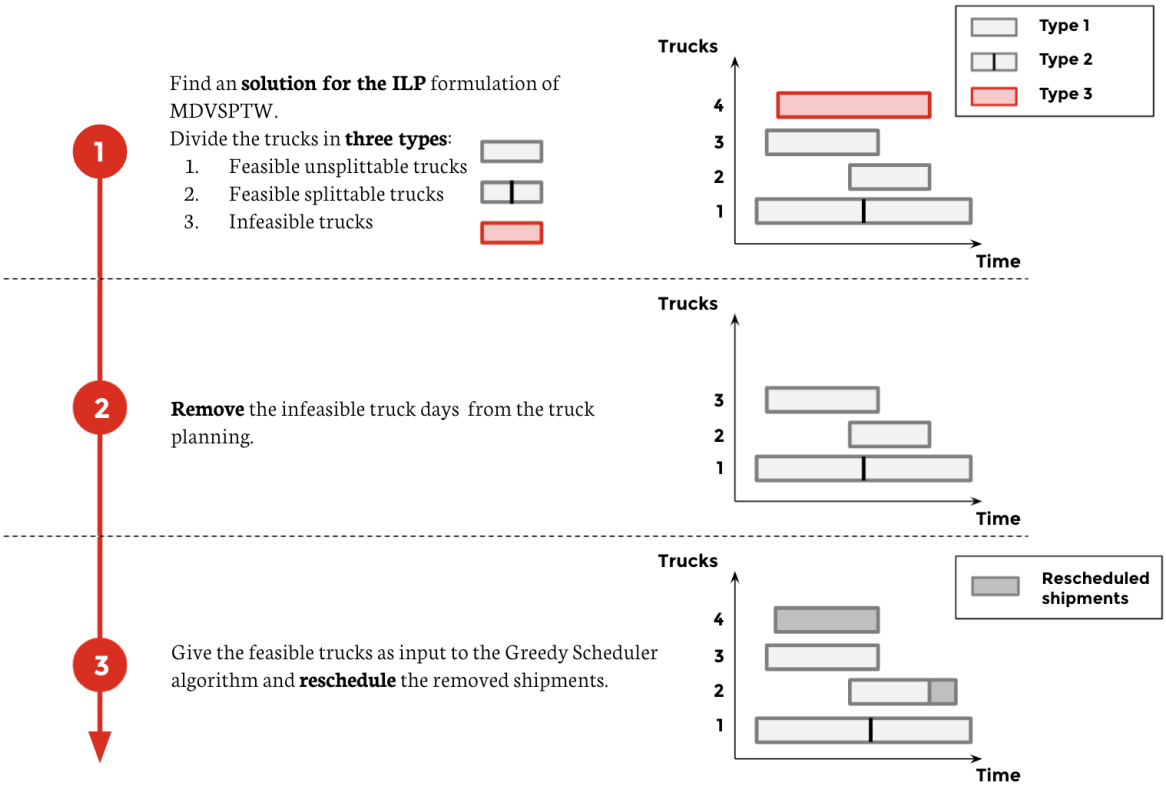


Figure 6.2: An overview of ILP + Greedy algorithm with driver change.

6.2. Random Search algorithm

The Greedy Scheduler algorithm is not deterministic, because ties are broken by choosing a random truck for 18% of the shipments, as mentioned in 5.2. Therefore, executing the algorithm multiple times and selecting the best solution found is a method to hopefully improve the solution compared to executing the algorithm once. We call this method Random Search. Again, we first consider a version of the algorithm in which the driver day duration restriction is incorporated with allowing a driver change. Then we consider a version in which the driver day duration restriction is incorporated when not allowing a driver change.

6.2.1. Random Search without driver change

The Greedy Scheduler algorithm takes into account minimal and maximal driver day duration for truck days. Therefore, every solution generated by the algorithm is already feasible for our problem. We aim to optimize the quality of this solution by applying Random Search. The Random Search algorithm without driver change, therefore simply executes the Greedy Scheduler algorithm I times and returns the best truck planning found. Figure 6.3 shows an overview and Algorithm 4 shows the pseudocode of the Random Search algorithm without driver change.

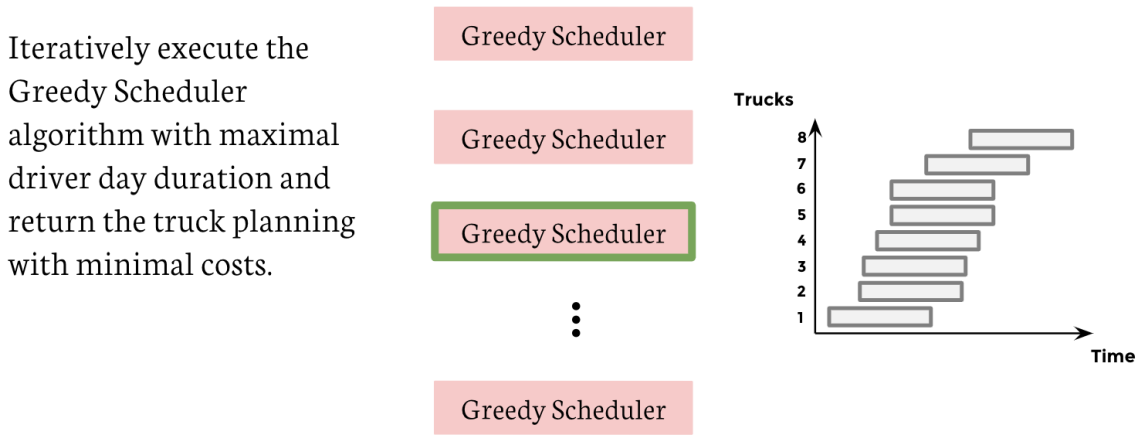


Figure 6.3: An overview of the Random Search algorithm without driver change.

Algorithm 4: Random Search without driver change.

```
1  INPUT
2  Set of time window shipments S
3  Set of depots D with truck capacity κ_d for every d ∈ D
4  Number of iterations I ∈ ℕ
5
6  RULES
7  best solution = GreedyScheduling(S, D)
8  best costs = cost(best solution)
9  i = 0
10 while i < I do
11     solution = GreedyScheduling(S, D)
12     costs = cost(solution)
13     if costs < best costs then
14         best solution = solution
15         best costs = cost(best solution)
16     i = i + 1
```

6.2.2. Random Search with driver change

In order to find trucks that enable a driver change, a straightforward approach would be to apply the Greedy Scheduler algorithm without the maximal truck day duration restriction and reschedule the shipments assigned to infeasible trucks, similarly as in the ILP + Greedy algorithm with driver change. However, experiments indicate that on average 4 splittable truck days occur when applying this approach, see Appendix B.2 for the exact results of this experiment. Almost all trucks turn out to be infeasible, because they are too long for one driver to execute the truck day, but not splittable. This results in a truck planning with very little driver changes. In order to leverage the possible advantage of a driver split, we want to define a method that results in more splittable truck days and less infeasible truck days in the solution of the Greedy Scheduler algorithm.

A method to find more splittable truck days, is to apply the Greedy Scheduler algorithm without maximal truck duration multiple times, and fix all splittable truck days that we encounter. When applying this method, our experiments indicate that the average number of splittable truck days in the solution increases from 4 to 13, see Appendix B.2. Figure 6.4 shows an overview of the Random Search algorithm with driver change. The Random Search with driver change algorithm is divided into two phases:

1. **The Fixing Phase**
   In the Fixing Phase, we iteratively generate truck plannings by executing the Greedy Scheduler without maximal truck duration and fix all splittable truck days that we encounter. The Fixing Phase terminates if a either the minimal number of shipments ϕ_{min} that are not fixed is reached, or a fixed number of iterations I_1 have been executed. The parameter ϕ_{min} is introduced to ensure that the shipments that still need to be assigned to trucks in the second phase have enough scheduling possibilities to generate efficient trucks. The Fixing Phase provides a number of splittable truck days that is presumably higher than before.

2. **The Searching Phase**
   In the Searching Phase, we aim to assign all shipments, that were not assigned to a splittable truck day in the first phase, to feasible driver days. We do this by executing Random Search with no driver change. The Searching Phase terminates after executing a fixed number I_2 of iterations. Finally, we merge the splittable truck days with the feasible driver days into one truck planning, that is our final solution.

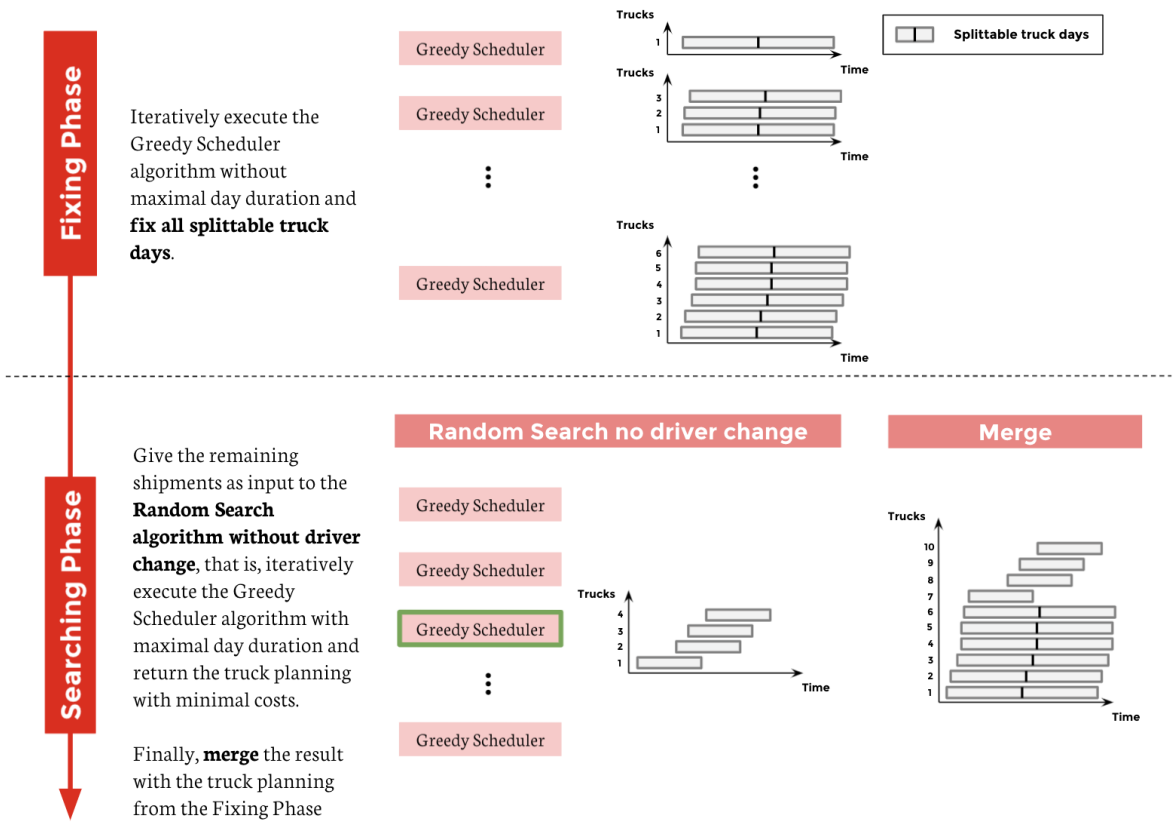


Figure 6.4: An overview of the Random Search change algorithm with driver change.

6.3. Random Search and Fix algorithm

When executing Random Search with driver change, we fix certain parts of solutions that we encounter in the Fixing Phase. Instead of fixing splittable truck days, we could also fix other types of truck that we prefer to keep in our truck planning. This section introduces Random Search and Fix, an algorithm that works similarly to Random Search with driver change, but fixes low-cost trucks in the Fixing Phase. We formally define low-cost truck days to be truck days for which the following holds:

$$\text{costs of the truck} < c_{max} \quad \text{and} \quad \delta_{min} < \text{truck day duration} < d_{max}. \tag{6.1}$$

The first criterion ensures that we fix truck days with relative short waiting and empty driving time. The second criterion ensures that we fix trucks with a relative long duration, in order to create an optimal use of trucks. Based on the truck plannings we have available, we choose c_{max} to be equal to the costs of a truck day that has one hour of empty driving time or two hours of waiting time, and δ_{min} to be equal to ten hours.

6.3.1. Random Search and Fix without driver change

The Random Search and Fix algorithm without driver change consists of two phases:

1. **The Fixing Phase**
   There are two differences compared to the Random Search algorithm with driver change. First, because we do not want the low-cost truck days to exceed d_{max}, we apply the Greedy Scheduler algorithm with maximal truck day duration instead of without. Second, as mentioned before, we fix the low-cost truck days instead of the splittable truck days. Again, the Fixing Phase terminates if either the minimal number ϕ_{min} of shipments that are not fixed is reached, or a fixed number I_1 of iterations has been executed.

2. **The Searching Phase**
   The Searching Phase is exactly the same as in the Random Search algorithm with driver change. The Random Search algorithm is executed to schedule the shipments that were not fixed, by executing the Greedy Scheduler algorithm I_2 times. Finally, we merge the splittable truck days with the feasible driver days into one truck planning, that is our final solution.

Figure 6.5 shows an overview of the algorithm.

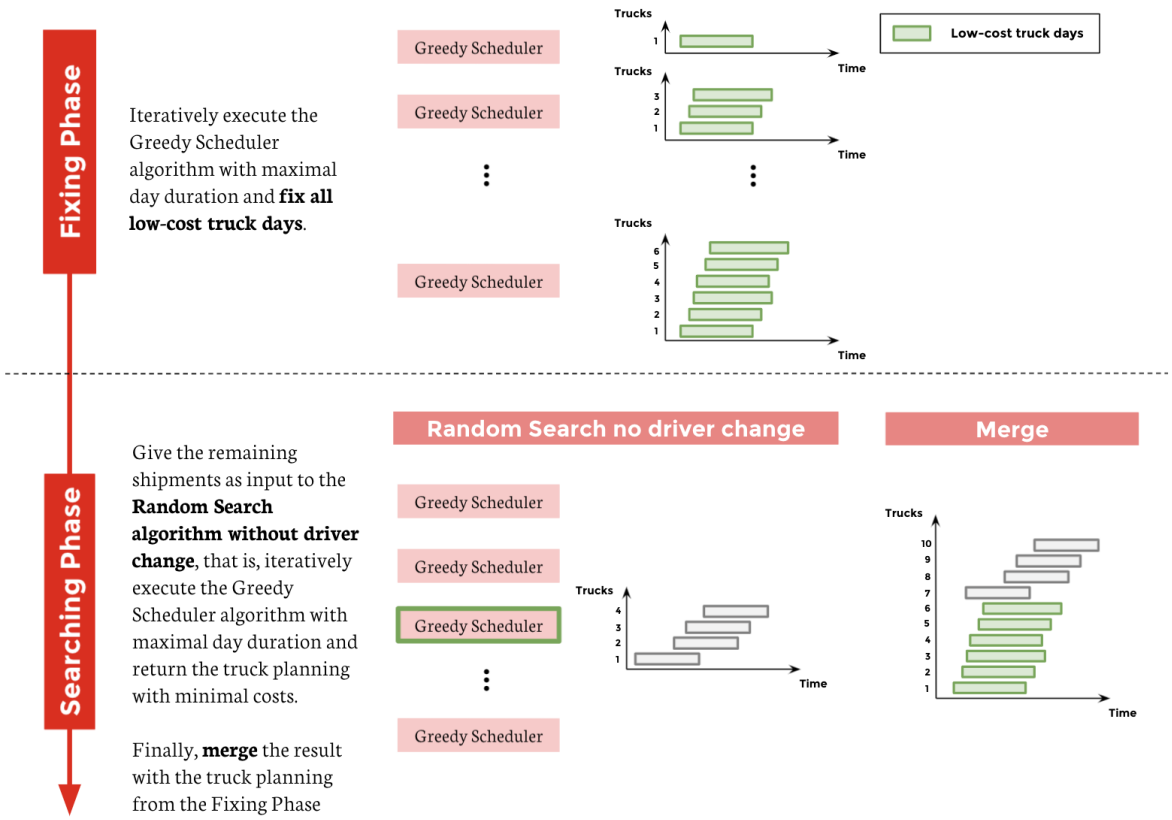


Figure 6.5: An overview of the Random Search and Fix algorithm without driver change.

6.3.2. Random Search and Fix with driver change

When a driver change is allowed, we essentially combine the Random Search with driver change algorithm and the Random Search and Fix algorithm.

1. **The Fixing Phase**
   In the Fixing Phase, all splittable truck days and low-cost truck days are fixed. We apply the Greedy Scheduler algorithm without maximal truck day duration. Again, the Fixing Phase terminates if either the minimal number ϕ_{min} of shipments that are not fixed is reached, or a fixed number I_1 of iterations has been executed.

2. **The Searching Phase**
   The Searching Phase is exactly the same as before. The Random Search algorithm is executed to schedule the shipments that were not fixed, by executing the Greedy Scheduler algorithm I_2 times. Finally, we merge the splittable truck days with the feasible driver days into one truck planning, that is our final solution.

Figure 6.6 shows an overview of the algorithm.

Figure 6.6: An overview of the Random Search and Fix algorithm with driver change.

7

Algorithm configurations

This chapter considers the algorithm configurations of the main algorithms introduced in the previous chapter. If the reader is mainly interested in the results, we suggest to skip this chapter, and continue reading at page 93. Appendix A.2 provides an overview of all choices of parameter values, as explained in this chapter. Section 7.1 introduces the test instances on which the configurations are based. Section 7.2 introduces our choices for the objective function parameters. Section 7.3 considers the configuration of the first building block: the ILP algorithm. Section 7.4 considers the configuration of the second building block: the Greedy Scheduler algorithm. The final two sections regard the configuration of the main algorithms. We divided the main algorithms into two types that have similar parameters:

1. Main algorithms that do not contain a Fixing Phase. The algorithms of this type are: ILP + Greedy (with and without driver change) and Random Search without driver change.

2. Main algorithms that contain a Fixing Phase. The algorithms of this type are: Random Search with driver change and Random Search and Fix (with and without driver change).

Section 7.5 considers the configuration of the first type of main algorithms and Section 7.6 considers the configuration of the second type of main algorithms.

Finding values for parameters that result in the best performance of an algorithm is often a trade-off between more than two elements. Because of the complex dependencies between all elements, there is not always one final perfect answer. However, we aim to be

transparent in our decision process, not only to justify our final configuration, but also to create more understanding in the working of the algorithms.

7.1. Test days

As test instances, we take the actual shipments executed by vdBrink on

- Wednesday March 31, 2021,
- Thursday April 15, 2021,
- Friday April 16, 2021 and
- Saturday April 17, 2021.

We assume every FC and DC to be a depot with a truck capacity of 15 trucks. From now on, the instances are denoted by their day and month, e.g., 31_03, and from here on referred to as *test days*. Table 7.1 shows the number of shipments of every instance.

Instance (*day_month*)	Total number of shipments
31_03	190
15_04	195
16_04	188
17_04	194

Table 7.1: The four test instances with their size.

7.2. Objective function parameters

As mentioned before the objective function is the weighted sum of the number of trucks, the empty driving time and the waiting time. These elements have the weights w_{fc}, w_{dt} and w_{wt}. Because our first priority is to minimize the number of trucks, we choose to set $w_{fc} = 100000$. Our second priority is to minimize the waiting time and the empty driving time. Empty driving is more costly compared to waiting because we pay not only for the working hours of the driver, but also for the gasoline of the truck. We choose to set $w_{dt} = 60$ and $w_{wt} = 30$. The large difference with the fixed costs of the truck, enables us to extract the number of trucks and the approximate inefficiency from the costs of a truck planning, which is useful for implementation purposes.

In choosing the parameter values for all algorithms, we often measure the impact of small changes in parameter values on the objective value. Therefore, it is important to understand that the first two digits of the objective value in general represents the number of

trucks. For some parameters, we want to focus on the inefficiency instead of the number of trucks. In that case, we do not consider the objective value, but the inefficiency, i.e., the percentage of time in a truck planning spend on waiting or empty driving.

7.3. ILP algorithm configuration

In this section, we explain our choice for the parameters and assumptions in the ILP implementation. As the main challenge in finding a good solution for the ILP is the limited computation time, we aim to find parameter values and assumptions that decrease the computation time, without decreasing the quality of the solution too much. Recall that the goal of the final algorithm is to generate a good feasible solution within 15 minutes and that the ILP is only able to provide an initial solution that is in general not feasible for our problem. We therefore assume that the ILP should terminate within 7.5 minutes, leaving 7.5 minutes for other phases of the main algorithms. We consider the time window discretization parameters, the maximal waiting time between consecutive shipments, the depot assumptions, the termination criterion and the post-processing steps. We refer to Section 5.1 and Section 5.3 for the exact definitions and explanations of the parameters.

7.3.1. The time window discretization parameters

In order to decide on the the values of the maximal number of discrete shipments that is allowed to be generated by a single time window shipment d_m and the minimal step size d_s, we take into account the operational meaning of the parameters. In consultation with the operational managers at Picnic, we came to the conclusion that $d_s \geq 0.25$ hours, as smaller intervals are too detailed considering the uncertainty of truck plannings. We also decide that in general the intervals should not exceed 0.5 hours too far, as more than that makes a significant difference in the truck planning. Figure 7.1 shows the distribution of the time window lengths of the input shipments from our test data set. The majority of the shipments (74%) has a time window length smaller of equal to 3 hours. Therefore assuming $d_s = 0.25$, we choose $d_m = 7$, to guarantee that for the majority of the time window shipments are discretized with an interval between 0.25 hours and 0.5 hours.

7.3.2. The maximal waiting time

When setting no maximal waiting time between consecutive shipments, the ILP has an average number of 1.75 million variables and constraints. Experiments indicate that the ILP algorithm takes far more than 15 minutes before finding a feasible solution. Therefore, we want to set a maximal waiting time t_{wt}^{max} that decreases the problem size. We decide what values to test for t_{wt}^{max} based on the following considerations. From an intuitive point

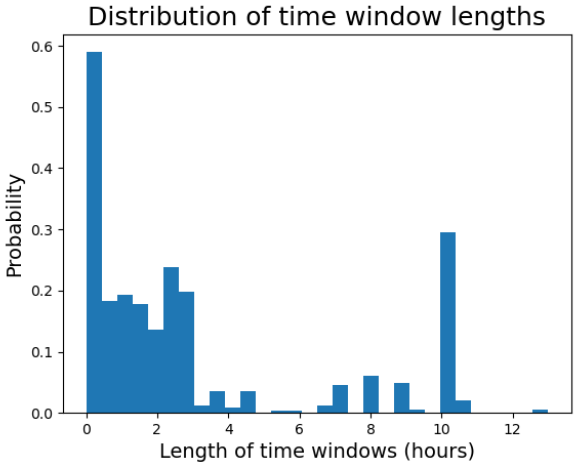


Figure 7.1: The distribution of the time window lengths of the input shipments from our test data set.

of view, a long waiting time in between executing shipments is undesirable, because it is inefficient. As derived from the many truck plannings we have available, we assume that waiting 3 hours in between shipments classifies as a very long waiting time. Also, if the maximal waiting time is restricted to be less than one hour, we know that many good solutions are excluded from our feasible region, because one hour of waiting time often occurs in efficient truck plannings. Therefore, we test the values $t_{wt}^{max} \in \{1, 1.5, 2, 2.5, 3\}$. We assume that, the solver should at least be able to find a feasible solution within 7.5 minutes. Table 7.2 shows the average number of second that the solver requires to find a feasible solution for the ILP, for different values of t_{wt}^{max}. Figure 7.2 shows the average objective value of the four test days for the different values of t_{wt}^{max}. In order to decide on the best value for the

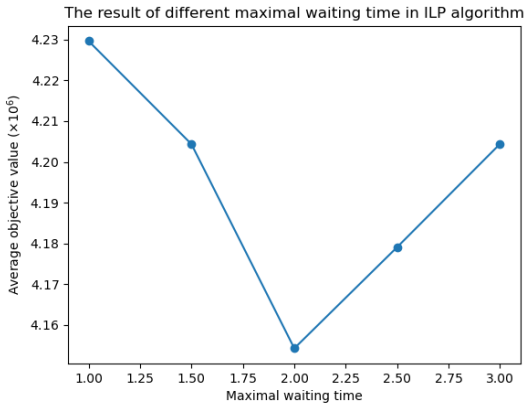


Figure 7.2: Average objective value over the four test days for different maximal waiting times.

	Time to find feasible solution (in seconds)	
t_{wt}^{max}	Average	Maximum
1	253	446
1.5	290	410
2	422	713
2.5	576	849
3	505	781

Table 7.2: Time to find feasible solution for different maximal waiting times.

maximal waiting time, we first consider the computation times to find a feasible solution. If the ILP is terminated before finding a feasible solution, we have no output. As this scenario is highly unfavorable, we choose to exclude values of t_{wt}^{max} that do not find a feasible

solution within 7.5 minutes. From the table we see that this is the case for the maximal waiting times of 2 hours or more. Therefore, we choose $t_{wt}^{max} < 2$. Considering the average objective values for $t_{wt}^{max} \in \{1, 1.5\}$ in Figure 7.2, we decide $t_{wt}^{max} = 1.5$.

7.3.3. The depot assumptions

In deciding what assumptions to make on the depot from which a truck that executes a certain shipment is likely to start, we state five potential assumptions and test the impact on the computation time and the solution quality. As explained in Section 5.1, the depot of the truck that executes an OB shipment is easier to predict than the depot that executes a IB shipment. Therefore, we distinguish between IB an OB shipments in the potential assumptions. Assumption 0 is equal to implementing no assumption.

0. All IB and OB shipments can be executed by trucks that start from any depot.
1. An OB shipment can only be executed by a truck that starts from a depot that is at most 1 hour driving from the end location of the shipment.
2. In addition to the assumption above, an IB shipment can only be executed by a truck that starts from a depot that is at most 1.5 hour driving away from the end location of the shipment.
3. In addition to the assumption above, an OB shipment can only be executed by a truck that starts from a depot that is at most 0.5 hour driving away from the end location of the shipment.
4. In addition to the assumption above, an IB shipment can only be executed by a truck that starts from a depot that is at most 1 hour driving away from the end location of the shipment.

We define the assumptions such that the average percentage of IB and OB shipments in the optimal ILP solution that respects the assumptions, varies between 100% and approximately 60%. Here, we assume that if the percentage would be lower, the assumption would rule out too many truck-shipment possibilities that might lead to a good solution. Table 7.3 shows the percentage of shipments that respects the assumptions in the final solution when running the algorithm without any assumptions taken into account. In order to decide on which assumption finds the right balance between decreasing the computation time, and maintaining quality of solution, we run the ILP algorithm with the different assumptions for 10 minutes for the four test days and conclude which assumption minimizes the objective function. Figure 7.3 shows the average objective values corresponding to the

	Average percentage of shipments that respects assumptions	
Assumption	IB shipments	OB shipments
0	100	100
1	100	76
2	92	76
3	92	69
4	59	69

Table 7.3: The average percentage of IB and OB shipments that respect the assumptions, as defined above, in an optimal ILP solution.

different assumptions. Table 7.4 shows the average and maximum number of minutes before finding a feasible solution corresponding to the ILP algorithms with the different assumptions. From the figure we can conclude that for assumption 2, 3 and 4, the average

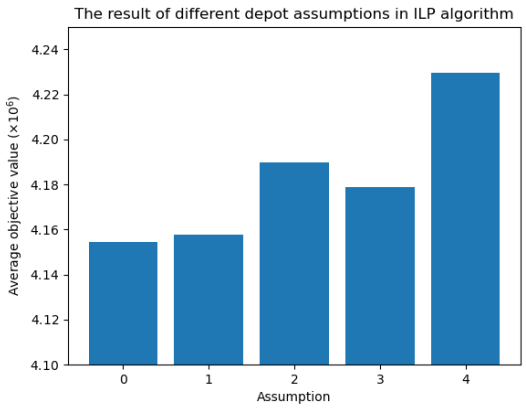


Figure 7.3: Average objective value over the four test days for different depot assumptions.

	Time to find feasible solution (in seconds)	
Assumption	Average	Maximum
0	290	410
1	75	92
2	74	85
3	17	22
4	5	7

Table 7.4: Time to find feasible solution for different assumptions.

objective value increases (by costs approximated by adding 0.5 truck to the planning) compared to the average objective value of assumptions 0 and 1. Considering the computation time to find a feasible solution, we see that all computation times are less than 7.5 minutes. However, the computation time with assumption 1 has a substantially decreased computation time compared to assumption 0, while the average objective value remained more or less the same. Therefore, we choose to incorporate assumption 1 in the ILP algorithm.

7.3.4. The termination criterion

As mentioned in Section 5.1, we set two parameters that define the termination of the ILP algorithm: the optimality gap δ_{gap} and the time limit τ_{max}. The algorithm stops when the solution is 'good enough', because of the optimality gap, or when the maximal time has passed, by the time limit. The value for τ_{max} is equal to 450 seconds, because we assumed that after 7.5 minutes, we want to have a feasible solution for our ILP.

For the value of δ_{gap}, we recall that the optimality gap is the relative difference of the lower bound and upper bound. Assume that we have a truck planning with 50 trucks. We want the algorithm to continue searching within the time limit, until the optimality gap is less than 1 hour of waiting in costs. However, one hour of waiting, increases the costs by 30 in our cost function. Therefore, we want the gap to be of the order $\frac{30}{50\times100000} = 6\cdot10^{-6}$, so we set $\delta_{gap} = 1\cdot10^{-5}$. We understand that this is a relative small gap bound, that is not likely to be restrictive within the set time limit. For our experiments, the implementation of δ_{gap} is therefore not required. However, when running the algorithm with other instances or other cost function parameters, the parameter might be useful, so we do not remove it from the algorithm.

7.3.5. Post-processing steps

The post-processing steps introduced in Section 5.3 are depot improvement and shipment sliding. When applying depot improvement step, the average inefficiency of the ILP solution decreases from 12.42% to 12.34%. Even though this effect is not substantial, we choose to implement the post-processing step, because the additional computation time is negligible (< 0.01 seconds).

Shipment sliding has more effect on the inefficiency of the ILP solution. Because the ILP algorithm works with time window discretization, it is likely that in between many consecutive shipments, waiting time occurs. By applying shipment sliding after finding a solution for the ILP, we expect to be able to remove some of the waiting time from the truck planning. Our experiments indicate that, without applying shipment sliding, the average inefficiency of the ILP solution is 12.42%. After adding shipment sliding to the algorithm, the average inefficiency is 9.37%. Because we only apply the rule once, the additional computation time is negligible (< 0.01 seconds). Thus, we choose to implement shipment sliding as a post-processing step after applying the ILP algorithm.

When decreasing the waiting time or the empty driving time, splittable truck days might not be splittable anymore, because the truck day duration is decreased. Therefore, we choose to only apply the post-processing steps to truck days that are not splittable.

7.4. Greedy Scheduler configuration

In this section, we explain our choice for parameter values in the Greedy Scheduler algorithm. In contrast to the ILP implementation, the Greedy Scheduler has a short computation time. Therefore, when setting the parameter values, we mainly aim to decrease the

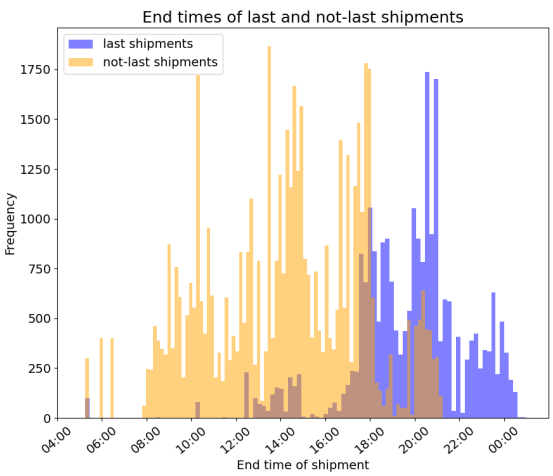
objective value of the final solution instead of decreasing the computation time. We consider the last and first shipment indicators and the implementation of the post-processing steps. We refer to Section 5.2 and Section 5.3 for the exact definitions and explanations of the parameters. Because the Greedy Scheduler algorithm in not deterministic, we have to run the algorithm multiple times and take the average result, when measuring the effect of different parameter values. For the experiments below, we run the Greedy Scheduler algorithm 100 times for the four test days in order to monitor the impact of changing parameter values. After 100 times for the four test days, we assume that, based on our experience with the algorithm, we have a representative average output of the algorithm.

7.4.1. The last and first shipment indicators

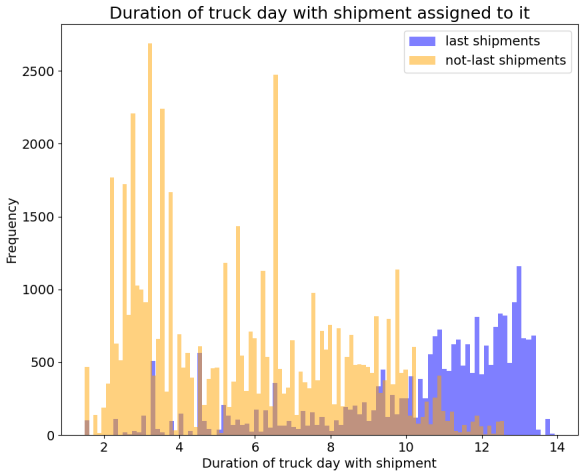
In order to decide on the last shipment indicators, we analyse the solutions without the indicators. The goal is to predict if a shipment is going to be the last shipment of a truck day. We choose two properties, that we expect to be good indicators, based on implicit reasoning:

- The end time of the shipment.
  If the end time is later than l_e, the shipment is expected to be a last shipment.
- The duration of the truck day with the shipment.
  If the duration is more than l_d, the shipment is expected to be a last shipment.

We run the Greedy Scheduler without the last shipment indicators 100 times for the four test days in order to generate a test set of truck plannings that we can analyze. Figure 7.4 shows the distribution of the last shipments and not-last shipments over the two indicator properties. Based on these distributions, we choose the values of l_e and l_d. Identifying a not-last shipment as a last shipment results in overestimating the costs of a shipment, while the other way around results in underestimating the costs of a shipment. We choose to prevent underestimation of the costs, because that might lead to unexpected high empty driving costs, while we expect the effect of overestimating the costs to be less bad. Therefore, we choose the values of l_e and l_d such that a relatively high percentage of the last shipments is identified, even though this also causes a relatively high percentage of the not-last shipments to be identified as last shipments. Based on the results given in Figure 7.4, we set l_e to be 18 : 00 and $l_d = 11$, and identify a last shipment if either the end time is at least 18 : 00, or the truck day duration with shipment is at least 11 hours and the start time is later than 16 : 00. The start time being later than 16 : 00 is required, because we want to distinguish last shipments from first shipments. Figure 7.5 shows a plot of the indicator properties of all shipments. The red dotted lines represent the values of l_e and l_d. With the given values, 90% of the last shipments is correctly identified as last shipment.



(a): The end time of the shipment.



(b): The duration of the truck day with the shipment.

Figure 7.4: Distribution of last shipment indicator properties.

On the other hand, 13% of the not-last shipments is falsely identified as last shipments. These percentages are relatively high compared to the percentages corresponding to other parameter values. Adding the last shipment indicator parameters to the algorithm mainly has an impact on the inefficiency of the truck planning, as the goal is to prevent long pull in trips. When running the algorithm without the parameters 100 times for the four test days, the average inefficiency of the truck planning is 22.47 percent, while with the parameters, the average inefficiency is 20.48 percent.

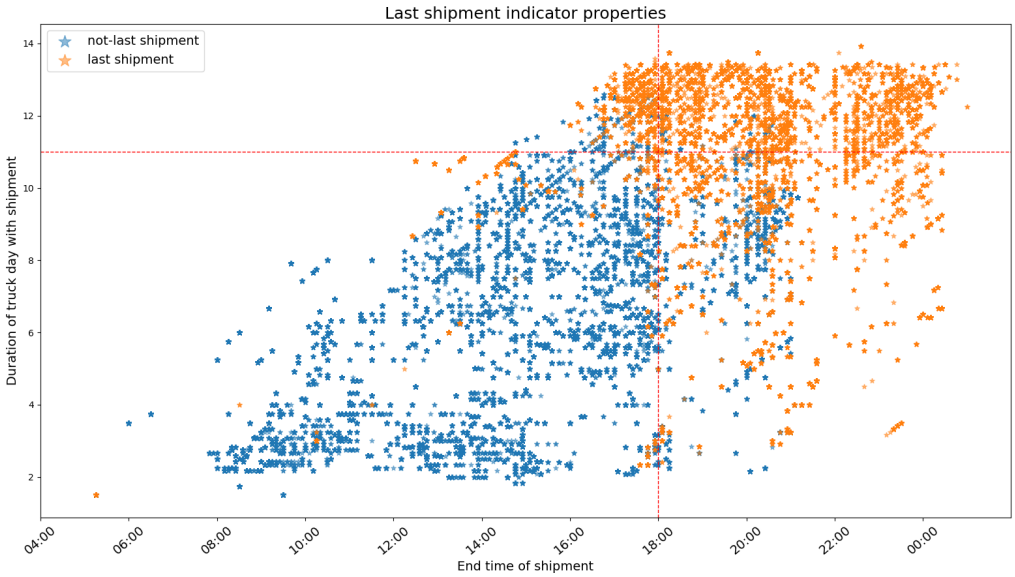


Figure 7.5: Last shipment indicator properties.

By symmetric argumentation, we already assumed that $e_d = l_d$. In order to decide on the value of e_s, we again consider the distribution of the start times of the first and not-first shipments after running the Greedy Scheduler algorithm 100 times for the four test days, see Figure 7.6. Based on the distribution, we choose e_s to be 12:00. This results in 75% of the first shipments to be correctly identified and 26% of the not-first shipments to be falsely identified as first shipments. We do not measure the impact on the objective function, as the first shipment indicators are only useful when active trucks are given as input to the Greedy Scheduler and the impact highly depends on the specific characteristics of the active trucks.

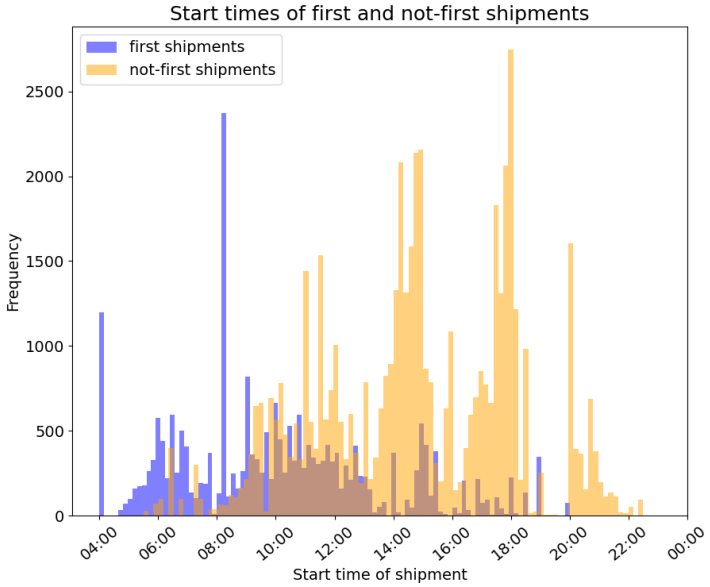


Figure 7.6: Distribution of start times of first shipments and not-first shipments.

7.4.2. Post-processing steps

In order to decide on the implementation of the post-processing steps, as introduced in Section 5.3, we compare the results of the algorithm with and without the implementation of the post-processing steps, depot improvement and shipment sliding. As both steps do not impact the number of trucks of a truck planning, but decrease the amount of empty driving and waiting time, we only consider the inefficiency of the truck plannings. Again, we run the Greedy Scheduler algorithm 100 times for the four test days and compare the average inefficiency percentages. Table 7.5 shows the average inefficiency percentage when implementing Greedy Scheduler with and without the post-processing steps. The impact of depot improvement is less compared to the impact of shipment sliding. However, both post-processing steps decrease the average inefficiency of the truck planning, without increasing the time per iteration too much. Therefore, we choose to add both post-processing steps to the Greedy Scheduler algorithm.

Post-processing steps	Average inefficiency (%)	Average time per iteration (s)
No post-processing step	20.48	0.23809
Depot improvement	20.43	0.24129
Shipment sliding	19.49	0.24333
Depot improvement and shipment sliding	19.37	0.24651

Table 7.5: The impact of implementing post-processing steps in the Greedy Scheduler algorithm.

7.5. Configuration of main algorithms without a Fixing Phase

This section considers choosing the parameter values for the main algorithms that do not contain a Fixing Phase, that is, algorithms that run the Random Search algorithm once. The algorithms that do not contain a Fixing Phase are the ILP + Greedy algorithms and the Random Search algorithm without driver change. These algorithms have only the number of iterations I for the Random Search algorithm, which has to be determined. In future research, it also might be useful to add a termination criterion that stops the algorithm after running for a fixed amount of time, similar as the ILP termination criterion τ_{max}. Because our computation times are far below the time limit, this is not necessary. In order to decide which value of I allows for enough opportunity to find better solutions, but does not take unnecessary long, we run the algorithms with $I = 1000$ for the four test days. We make a distinction between *significant* and *non-significant* improvements. An improving solution is called *significant* if the decrease in objective function value in comparison to the previous best solution value is more than the costs that represent one hour of waiting.

The average time per iteration per test day ranges between 0.09 and 0.15 seconds for the ILP + Greedy algorithms and between 0.22 and 0.25 seconds for the Random Search algorithm without driver change. As a result of this short computation time, we can overestimate the number of iterations required, without substantial consequences for the total computation time. For example, an overestimation of 100 iterations results in only 9 to 25 seconds redundant running time.

7.5.1. ILP + Greedy without driver change

Figure 7.9 shows the development of the objective value after running the Random Search algorithm with $I = 1000$ for the four test days. Table 7.6 contains the final improvement iterations for a significant improvement and an improvement in general. Because all significant improvements are made within the first 100 iterations, and overestimation of 100 iterations results in 25 extra seconds at most, we choose $I = 200$ for the Random Search algorithm without driver change.

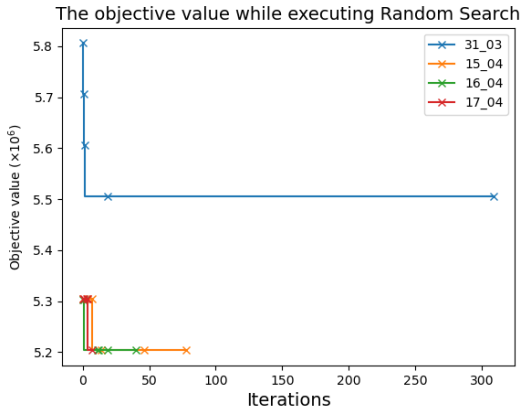


Figure 7.7: The objective value while executing the Random Search algorithm on the ILP solution without driver change for $I = 1000$.

	Final improvement iterations	
Date	Significant	General
31_03	2	309
15_04	78	78
16_04	40	40
17_04	7	7

Table 7.6: The final improvement iterations of the test days.

7.5.2. ILP + Greedy with driver change

Figure 7.9 shows the development of the objective value after running the Random Search algorithm with $I = 1000$ for the four test days. Table 7.7 contains the final improvement iterations for a significant improvement and an improvement in general. Based on these results, we choose the number of iterations required for the Random Search algorithm without driver change to be $I = 200$.

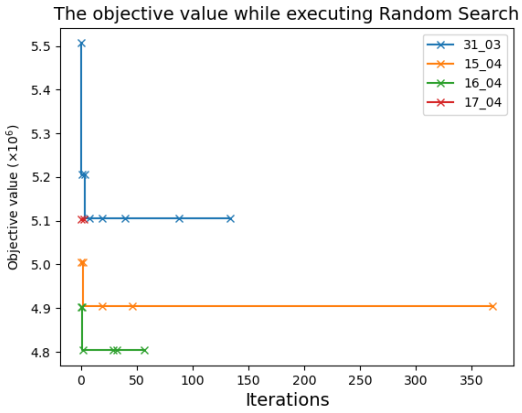


Figure 7.8: The objective value while executing the Random Search algorithm on the ILP solution with driver change for $I = 1000$.

	Final improvement iteration	
Date	Significant	General
31_03	40	134
15_04	46	369
16_04	29	57
17_04	3	3

Table 7.7: The final improvement iterations of the test days.

7.5.3. Random Search without driver change

Figure 7.9 shows the development of the objective value after running the Random Search algorithm with $I = 1000$ for the four test days. Table 7.8 contains the final improvement iterations for a significant improvement and an improvement in general. Based on 189 being the overall final iteration that generates a significant improvement, we choose the number of iterations required for the Random Search algorithm without driver change to be $I = 250$.

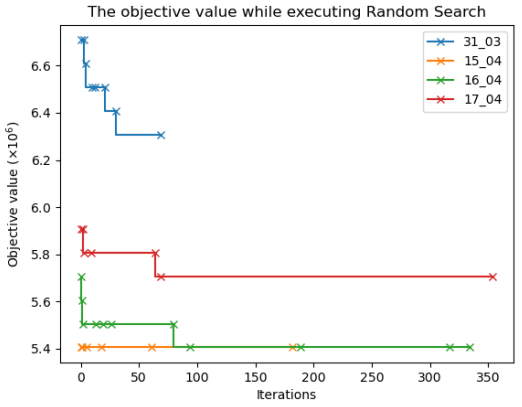


Figure 7.9: The objective value while executing the Random Search algorithm for $I = 1000$.

	Final improvement iteration	
Date	Significant	General
31_03	69	69
15_04	61	182
16_04	189	334
17_04	69	354

Table 7.8: The final improvement iterations of the test days.

7.6. Configuration of main algorithms with a Fixing Phase

The main algorithms that contain a Fixing Phase are Random Search with driver change and the Random Search and Fix algorithms. For these algorithms, we do not only have to decide on the termination criterion for the Searching Phase, but also for the Fixing Phase.

As introduced in Chapter 6, the Fixing Phase is terminated if a fixed number I_1 of iterations is executed, or less than a minimal number ϕ_{min} of shipments is left to be scheduled. The Searching Phase is terminated after executing I_2 iterations. We first analyse the termination criterion of the Fixing Phase, by assuming $I_2 = 1000$. We run the algorithm for $I_1 \in \{1, 20, 50, 100, 200\}$ and monitor the number of fixed truck days found per iteration, the number of shipments to be scheduled and objective value of the solution, after executing the Searching Phase with $I_2 = 1000$. Based on the results we choose values for I_1 and ϕ_{min}. Finally, we decide on the value for I_2, by detecting the improvement iterations, similar as in the previous section.

7.6.1. Random Search with driver change

Figure 7.10 shows the linear regression of the number of fixed splittable truck days after the first iteration and the costs of the final solution. This indicates a negative correlation, implying that the higher the number of splittable truck days fixed in the Fixing Phase, the better the final solution. This is expected, as splittable truck days decrease the total number of trucks in use. As a result of this relation, our goal is to maximize the number of splittable truck days in the Fixing Phase. When running the Fixing Phase, we notice that all splittable truck days are detected early in the iteration process. Table 7.9 shows the final iteration in which a new splittable truck day was detected, when running the Fixing Phase for $I_2 = 1000$, for the four test days. From these results, we can conclude that all splittable truck days are likely to be found within 150 iterations. Therefore, we choose to set $I_1 = 150$.

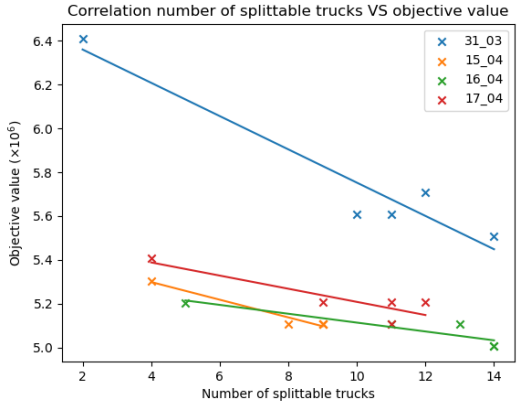


Figure 7.10: The objective value of Random Search with driver change for a different number of iterations in the Fixing Phase.

Date	Final iteration new splittable truck day
31_03	10
15_04	13
16_04	40
17_04	76

Table 7.9: The final iteration that finds new splittable truck day.

The average number of shipments left to schedule decreases from 173 to 135, whenever I_1 increases. We assume that this results in enough freedom for scheduling the remaining shipments in an efficient way. We set $\phi_{min} = 80$, but based on our experiments, we expect

not to reach this bound.

In order to decide on the value of I_2, we run the algorithm with the determined values for I_1 and ϕ_{min} and $I_2 = 1000$. Figure 7.11 shows the objective value improvements while executing the algorithm, and Table 7.10 shows the final improvement iterations. Based on these results, we choose the number of iterations in the Searching Phase to be $I_2 = 500$.

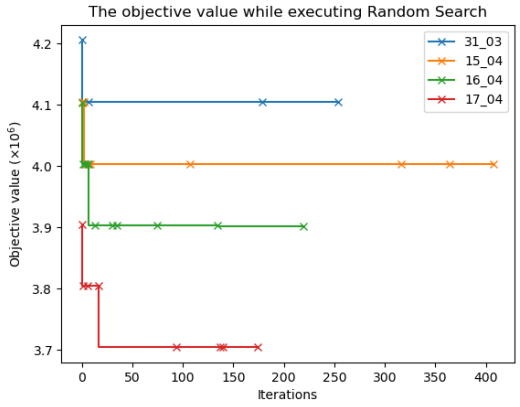


Figure 7.11: The objective value while executing the ILP + Greedy algorithm without driver change and $I_1 = 150$, $\phi_{min} = 50$ and $I_2 = 1000$.

	Final improvement iteration	
Date	Significant	General
31_03	179	254
15_04	316	408
16_04	219	219
17_04	174	174

Table 7.10: The final improvement iterations of the test days.

7.6.2. Random Search and Fix without driver change

In Random Search and Fix, we fix the low-cost truck days that we find in each iteration. Our experiments indicate that this method does not have a significant effect on the total number of trucks. However, there is a negative correlation between the number of low-cost truck days fixed in the Fixing Phase, and the inefficiency score of the final solution. Figure 7.12 shows this correlation, by showing the trend line from the linear regression. Therefore, we aim to maximize the number of low-cost truck days fixed in the Fixing Phase. Again, Table 7.11 shows the final iterations in which new low-cost truck days are detected. From these results, we can conclude that all low-cost truck days are likely to be found within 150 iterations. Therefore, we choose to set $I_1 = 150$.

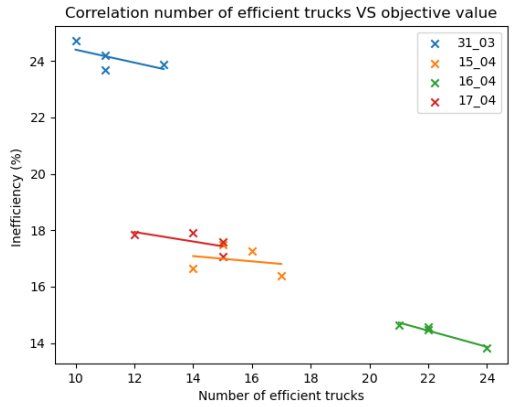


Figure 7.12: The inefficiency percentage of Random Search and Fix without driver change solution for a different number of iterations in the Fixing Phase.

Date	Final iteration new low-cost truck day
31_03	24
15_04	13
16_04	7
17_04	65

Table 7.11: The final iteration that finds new low-cost truck day.

The average number of shipments left to schedule decreases from 148 to 105, whenever I_1 increases. We assume that this ensures enough freedom in scheduling the remaining shipments. We set $\phi_{min} = 80$, but based on our experiments, we expect not to reach this bound. In order to decide on the value of I_2, we run the algorithm with the determined values for I_1 and ϕ_{min} and $I_2 = 1000$. Figure 7.13 shows the objective value improvements while executing the algorithm, and Table 7.12 shows the final improvement iterations. Based on these results, we choose the number of iterations in the Searching Phase to be $I_2 = 600$.

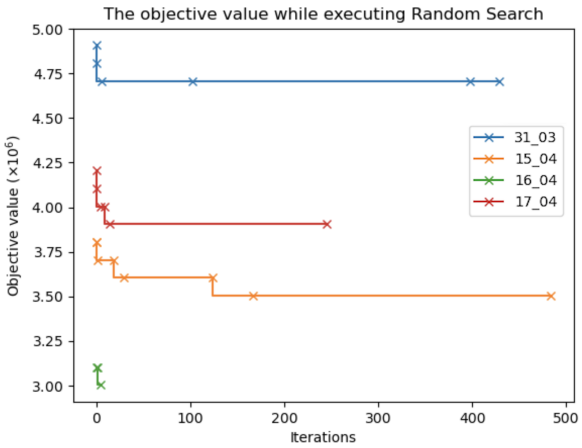


Figure 7.13: The objective value while executing the Random Search algorithm with driver change and $I_1 = 150$, $\phi_{min} = 50$ and $I_2 = 1000$.

	Final improvement iteration	
Date	Significant	General
31_03	398	429
15_04	484	484
16_04	5	5
17_04	245	245

Table 7.12: The final improvement iterations of the test days.

7.6.3. Random Search and Fix with driver change

In the Random Search and Fix algorithm with driver change, both splittable and low-cost truck days are fixed in the Fixing Phase. As we expect, splittable truck days decrease the total number of trucks and low-cost trucks decrease the inefficiency. Figure 7.14 shows the

linear regression of the number of fixed truck days in the Fixing Phase and the objective value of the final solution. However, we can also see that the relation is less linear, compared to Figure 7.10 and Figure 7.12. Unlike before, we are able to see an increase in costs when running the Fixing Phase for too long. We expected that this could be explained by the little freedom in the Searching Phase, because too many shipments are fixed. When we examine the number of shipments left to be scheduled, however, the average over the test days, varies between 103 and 138 shipments. This does not seem to be too little to create an efficient truck planning.
We set $I_1 = 50$, based on the minimum in Figure 7.15. Also we set $\phi_{min} = 80$, but we again do not expect this bound to be reached based on our experiments.

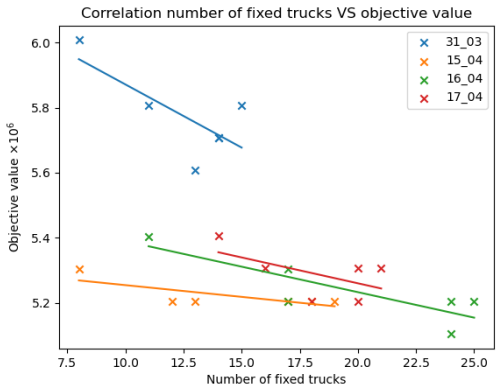


Figure 7.14: Correlation between the number of fixed truck days after the first phase and the objective value of the final solution.

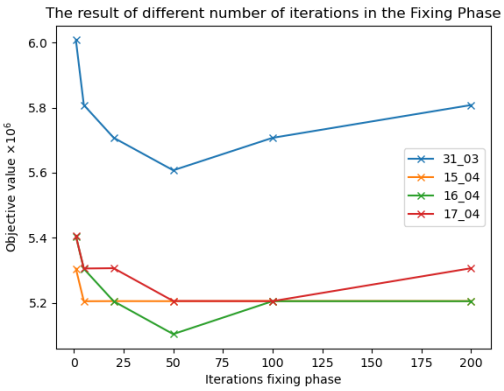


Figure 7.15: The relation between the number of iterations in the Fixing Phase and the objective value of the final solution.

In order to decide on the value of I_2, we run the algorithm with the determined values for $I_1 = 50$ and $\phi_{min} = 80$ and $I_2 = 1000$. Figure 7.16 shows the objective value improvements while executing the algorithm, and Table 7.13 shows the final improvement iterations. Based on these results, we choose the number of iterations in the Searching Phase to be $I_2 = 400$.

Notice, that the parameter ϕ_{min} is not likely to be restrictive in any of our implementations. However, we choose to not remove the parameter, because for other instances, it might be restrictive.

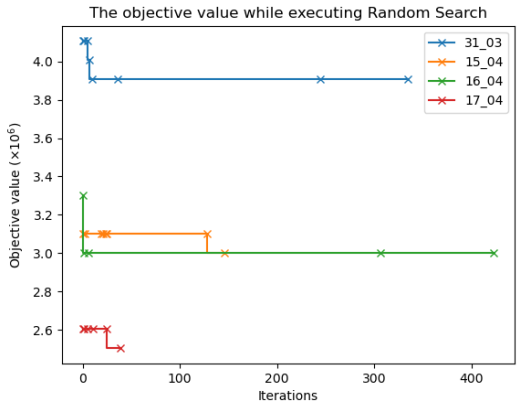


Figure 7.16: The objective value while executing the ILP + Greedy algorithm without driver change and $I_1 = 150$, $\phi_{min} = 50$ and $I_2 = 1000$.

	Final improvement iteration	
Date	Significant	General
31_03	334	334
15_04	146	146
16_04	306	423
17_04	39	39

Table 7.13: The final improvement iterations of the test days.

8

Results

This chapter reports on the results of the computational experiments with the main algorithms, with parameter values chosen as in Chapter 7, and are summarized in Appendix A.2. Section 8.1 shows the results of the main algorithms when not allowing a driver change and Section 8.2 shows the results of the algorithms when allowing a driver change. Section 8.3 considers the impact of the driver change given the results.

8.1. Comparison of the algorithms without driver change

In order to provide a complete assessment of the quality of the truck plannings that are generated by the algorithms, we monitored various performance indicators of the truck plannings, in addition to the objective value. Recall that the inefficiency of a truck planning is the sum of the relative waiting time and the relative empty driving time. The *short driver days* are defined as the the driver days that have a duration less than 7 hours, and therefore were supplemented by additional waiting time in order to guarantee the feasibility. Table 8.1 shows the results, with the best performing metrics in bold.

The ILP + Greedy algorithm performs best on most of the metrics, while the Random Search algorithm does not outperform the other algorithms on any of the metrics. The inefficiency of the ILP + Greedy solutions is with 13.54% on average particularly low compared to the 16.98% and 16.63% of the Random Search and Random Search and Fix truck plannings respectively. This is in line with the total planned hours being 580 on average for the ILP + Greedy truck plannings compared to approximately 600 for the two heuristics. The computation time of the ILP + Greedy algorithm is also lower than the computation time of other two algorithms, but as all computation times of all test days are far below our set time limit

15 minutes, we conclude that the computation times of all algorithms are feasible. The waiting time percentage is slightly lower for the Random Search and Fix algorithm with 3.46%, than for the ILP + Greedy algorithm with 3.58%, this is, however, compensated by the empty driver percentage on which the ILP + Greedy algorithm scores 9.95%, while the Random Search and Fix algorithm scored 13.18%. The driver days of the Random Search and Fix algorithm are slightly more efficient with only 4 short days and 11.2 hours duration on average, compared to 6 short days and 10.8 hours duration on average. And the number of trucks is minimal for both the ILP + Greedy algorithm and the Random Search and Fix algorithm with both 54.0 trucks on average.

Metric	ILP + Greedy	Random Search	Random Search and Fix
Objective value	**5,404,656**	5,606,340	5,406,183
Number of trucks	**54.0**	56.0	**54.0**
Number of drivers	**54.0**	56.0	**54.0**
Waiting (%)	3.58	3.89	**3.46**
Empty driving (%)	**9.95**	13.09	13.18
Inefficiency (%)	**13.54**	16.98	16.63
Total planned hours	**580**	602	603
Short days (<7 hours)	6	5	**4**
Average driver day duration (hours)	10.8	10.8	**11.2**
Computation time (seconds)	**104**	132	332

Table 8.1: Average results over the four test days of the main algorithms without driver change.

8.2. Comparison of the algorithms with driver change

We now consider the results of the main algorithms when driver change is allowed. We measured the same performance indicator metrics as in the previous section. Table 8.2 shows the results, with the best performing metrics in bold.

Again, the ILP + Greedy algorithm performs the best on most of the metrics, while the Random Search algorithm does not outperform the other two algorithms on any of the metrics. The inefficiency of the ILP + Greedy solutions is with 15.03% on average lower than the 17.36% and 16.99% of Random Search and Random Search and Fix. Also, the total planned hours is minimal for the ILP + Greedy algorithm with 587 hours in total compared to 609 for both other algorithms. Again, the computation time of the ILP + Greedy algorithm is minimal, however, all computation times of all test days are far below our set time limit of 15 minutes, are therefore feasible. The waiting time of the Random Search and Fix algorithm is minimal with 2.97% compared to 5.03% for the ILP + Greedy algorithm. But the empty driving time compensates with 10.00% for the ILP + Greedy algorithm compared to

14.02% for the Random Search and Fix algorithm. The average number of trucks of the ILP + Greedy algorithm is slightly less with 49.8 compared to the 50.5 of Random Search and Fix. However, the average number of drivers is slightly more for the ILP + Greedy algorithm with 59.8 compared to the average number of drivers 58.5 of the Random Search and Fix solutions. The driver days of the Random Search and Fix algorithm are more efficient on average than the ILP + Greedy driver days: 2 short days compared to 6 short days and a duration of 10.4 on average compared to 9.8.

Metric	ILP + Greedy	Random Search	Random Search and Fix
Objective value	**4,980,343**	5,156,485	5,056,303
Number of trucks	**49.8**	51.5	50.5
Number of drivers	59.8	63.8	**58.5**
Waiting (%)	5.03	3.05	**2.97**
Empty driving (%)	**10.00**	14.31	14.02
Inefficiency (%)	**15.03**	17.36	16.99
Total planned hours	**587**	609	609
Short days (<7 hours)	6	4	**2**
Average driver day duration (hours)	9.8	9.6	**10.4**
Computation time (seconds)	**109**	203	165

Table 8.2: Average results over the four test days of the main algorithms with driver change.

8.3. Impact of driver change

Table 8.3 shows the average difference of the metrics when allowing a driver change, compared to not allowing a driver change. We see that the average number of trucks decreases by 4, 5 and 3.5 trucks, that is a decrease of approximately 8%. This results in an increase in drivers of 6, 8 and 4.5, that is approximately 10%. In general, the impact of allowing a driver change on the efficiency of the truck plannings is relatively small. Except for the number of trucks and drivers, average driver day duration decreases with approximately 1 hour on average and the total planned hours increases with 6 hours on average, that is approximately 1%. The other changes in performance metrics are negligible. The actual cost impact in euros of allowing a driver change are now a small calculation involving the number of trucks, number of drivers, total planned hours and average driver day duration and the specific cost function. A detailed cost savings calculation for the specific case of Picnic has not been included, since we set up this thesis not taking into account Picnic's specific cost function, but a more general cost function for other use cases as well.

Metric	ILP + Greedy	Random Search	Random Search and Fix
Number of trucks	-4	-5	-3.5
Number of drivers	6	8	4.5
Waiting (%)	1	-1	-0.5
Empty driving (%)	0	1	0.8
Inefficiency (%)	1	0	0.4
Total planned hours	7	7	5.4
Short days (<7 hours)	0	-1	-1.8
Average driver day duration (hours)	-1	-1	-0.8

Table 8.3: Average difference of performance metrics when allowing a driver change compared to not allowing a driver change.

9

Conclusions and recommendations

This chapter presents the most important conclusions of the research in Section 9.1 and the recommendations for future research in Section 9.2.

9.1. Conclusions

The growing market for online supermarkets gives rise to challenges when planning the increasingly complex supply chain for online grocers, such as Picnic. This motivated us to research the optimization problems that model to the planning of the transport. To tackle this problem, we took a rigorous mathematical approach to design and evaluate suitable algorithms. The goal of this research was to answer our main question.

Which algorithm is the most suitable for solving Picnic's case by finding a good solution for MDVSPTW and taking into account the driver day duration restriction within limited computation time?

We investigated existing ILP approaches for our problem and relevant subproblems. The problem size of the ILP formulations of our problem is far too large to find a feasible solution within reasonable computation time. However, we did find an ILP formulation that finds a solution to MDVSPTW, and for which the ILP implementation can be adjusted to find a feasible solution within our set time limit. By incorporating additional assumptions on the final truck planning, we were able to decrease the problem size, and find a feasible solution for the new ILP formulation of MDVSPTW within our time constraints.

However, this ILP does not provide a feasible solution for our problem, as the driver day

duration restriction is not incorporated. Also, ILP approaches are not able to guarantee an output, because of the NP-hardness. Therefore, we investigated existing heuristic approaches for our problem and relevant subproblems, in addition to the ILP approaches. Based on the Concurrent Scheduler heuristic for MDVSP from prior work, we introduced the Greedy Scheduler algorithm, that finds a feasible solution for our problem within relatively short time. The Greedy Scheduler has the advantage of being able to take active trucks, i.e., trucks that already have shipments assigned to them, as input and plan remaining shipments on the existing truck planning. This property enabled us to combine the ILP implementation with the Greedy Scheduler into the main algorithm ILP + Greedy to find a good feasible solution to our problem. For robustness reasons, we also defined two other main algorithms that solely depend on heuristics: Random Search and Random Search and Fix. In answering our main question, we compare the performance of the three main algorithms when allowing a driver change and when not allowing a driver change.

Our results show that ILP + Greedy is the best algorithm both when allowing and not allowing a driver change. This is specifically due to the algorithm generating truck plannings with a lower inefficiency compared to the other two algorithms. If, however, we would prefer to choose an algorithm that does not involve an ILP, e.g., for robustness reasons, the Random Search and Fix algorithm performs better compared to the Random Search algorithm.

The impact of allowing a driver change, when executing our main algorithms, is that the number of trucks decreases by approximately 8%, while the number of drivers increases by approximately 10%. The total planned hours increases slightly (1%) and the average diver day duration decreases by approximately one hour. Whether this results in an increase or decrease of monetary costs depends on the company-specific cost function.

9.2. Recommendations for future research

During this research, we noted multiple fruitful ideas and approaches that could be interesting for future research. In this section, we review these ideas and approaches. We divide our recommendations into four categories.

The recommendations regarding the general research in general:

- **Constraint Programming approach**
  We solved our problem using ILP and heuristic techniques. However, there is another approach often used on scheduling problems: Constraint Programming. This approach is based on logic rules that determine whether a solution is feasible and

whether a change in solution is an improvement. Follow-up studies might compare the results of this approach to our ILP and heuristic results.

- **Dataset size**
  We chose to test our algorithms with data derived from Picnic's actual planning for four days. Increasing the size of the dataset would allow an improved empirical evaluation of our algorithms. This may shed light on opportunities for further improvements.

The recommendations regarding the ILP models:

- **Set Partitioning modelling**
  We chose to implement the Connection-Based Minimum Cost Network models for our relevant subproblems. In the literature, however, Set Partitioning models that might be better extendable when implementing the driver day duration restriction. Comparing the results of these models to the results of the Connection-Based Minimum Cost Network models may be insightful. Also, this could result in an ILP formulation of our problem that is solvable to optimality, and therefore defines a benchmark.

- **Implementation of large ILPs**
  In constructing the ILP models of our problems, we did not implement the final ILP formulations that represent our problem, because the computation time exceeded our reasonable limit even for smaller subproblems. However, it may be interesting to implement these formulations and try to solve them, perhaps with Column Generation. These ILPs might be able to provide an optimal solution to our problem, or otherwise allow us to define a lower bound.

The recommendations regarding the heuristics:

- **Other types of heuristics**
  We investigated heuristics that are Greedy or based to an intuitive Local Search algorithm. In the literature, however, we also found algorithms based on less intuitive heuristics, such as Genetic Algorithms. Implementing these heuristics and comparing the performance to our main algorithms may be insightful.

- **Local optima escaping methods**
  In our main algorithms, we find a good feasible solution for our problem, by making decisions in a greedy manner. This has the disadvantage that the algorithm might get stuck in a local optimum. Researchers are encouraged to apply strategies introduced in Chapter 2, that aim to escape local optima.

A specific idea in this direction, is to not only fix trucks, but also *unfix* trucks in the Random Search and Fix algorithm, see Appendix C.2. A Simulated Annealing or Tabu Search approach might be able to improve the current results of the algorithm.

- **Change the order of the shipments**
  When executing the Greedy Scheduler algorithm, the order in which the shipments are scheduled is fixed. We ordered the shipments lexicographically on latest start time and time window length. However, other changing the order might result in more efficient truck plannings. We encourage future researchers to explore this degree of freedom when examining the Greedy Scheduler algorithm. See Appendix C.1 for a more detailed explanation on this idea.

The recommendations regarding the operational application of the algorithms:

- **Time scope**
  In order to construct a truck planning that is even more realistic for Picnic, we may want to take into account restrictions that consider a larger time horizon than one day. For example, every driver should have a fixed number of hours rest at night. When considering the truck plannings on a week-basis instead of day-basis, we are able to take this into account.

- **Docking restriction**
  It might be relevant to incorporate the docking restriction. By measuring the number of times the restriction is violated in the current truck plannings, we can decide whether it is relevant to include this constraint into our algorithms. The constraint could be implemented by adding a *Location* class to the code structure, also see Appendix A.1.

References

[1] E. Aarts, E. H. Aarts, and J. K. Lenstra. *Local search in combinatorial optimization*. Princeton University Press, 2003.

[2] A. A. Bertossi, P. Carraresi, and G. Gallo. On some matching problems arising in vehicle scheduling models. *Networks* 17.3 (1987), pp. 271–281.

[3] C. Blum and A. Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM computing surveys (CSUR)* 35.3 (2003), pp. 268–308.

[4] L. Bodin. Routing and scheduling of vehicles and crews, the state of the art. *Comput. Oper. Res.* 10.2 (1983), pp. 63–211.

[5] L. Bodin, D. Rosenfield, and A. Kydes. UCOST: a micro approach to a transportation planning problem. *Journal of Urban Analysis* 5.1 (1978).

[6] S. Bunte and N. Kliewer. An overview on vehicle scheduling models. *Public Transport* 1.4 (2009), pp. 299–317.

[7] G. Carpaneto, M. Dell'Amico, M. Fischetti, and P. Toth. A branch and bound algorithm for the multiple depot vehicle scheduling problem. *Networks* 19.5 (1989), pp. 531–548.

[8] J. R. Daduna and J. M. P. Paixão. Vehicle scheduling for public mass transit—an overview. *Computer-aided transit scheduling*. Springer, 1995, pp. 76–90.

[9] G. B. Dantzig. Application of the simplex method to a transportation problem. *Activity analysis and production and allocation* (1951).

[10] G. B. Dantzig and D. Fulkerson. *Minimizing the number of carriers to meet a fixed schedule*. Tech. rep. RAND CORP SANTA MONICA CA, 1954.

[11] G. Desaulniers, J. Lavigne, and F. Soumis. Multi-depot vehicle scheduling problems with time windows and waiting costs. *European Journal of Operational Research* 111.3 (1998), pp. 479–494.

[12] M. Desrochers and F. Soumis. A column generation approach to the urban transit crew scheduling problem. *Transportation science* 23.1 (1989), pp. 1–13.

[13] J. A. Ferland and P. Michelon. The vehicle scheduling problem with multiple vehicle types. *Journal of the Operational Research Society* 39.6 (1988), pp. 577–583.

[14] M. Fischetti, S. Martello, and P. Toth. The fixed job schedule problem with working-time constraints. *Operations Research* 37.3 (1989), pp. 395–403.

[15] R. Freling. Models and techniques for integrating vehicle and crew scheduling. PhD thesis. Thesis Publishers Amsterdam, 1997.

[16] R. Freling, A. P. Wagelmans, and J. M. P. Paixão. Models and algorithms for single-depot vehicle scheduling. *Transportation Science* 35.2 (2001), pp. 165–180.

[17] B. Gavish and E. Shlifer. An approach for solving a class of transportation scheduling problems. *European Journal of Operational Research* 3.2 (1979), pp. 122–134.

[18] F. Glover. Future paths for integer programming and links to artificial intelligence. *Computers & operations research* 13.5 (1986), pp. 533–549.

[19] F. Glover. Tabu search—part I. *ORSA Journal on computing* 1.3 (1989), pp. 190–206.

[20] R. Gomory. Essentials of an algorithm for integer solutions to linear programs. *Recent Advances in Mathematical Programming* (1958), pp. 269–302.

[21] I. Gurobi Optimization. *Gurobi Optimizer Reference Manual*. 2020. URL: `https://www.gurobi.com/documentation/9.1/refman/index.html`.

[22] A. Haghani and M. Banihashemi. Heuristic approaches for solving large-scale bus transit vehicle scheduling problem with route time constraints. *Transportation Research Part A: Policy and Practice* 36.4 (2002), pp. 309–333.

[23] D. Huisman. Integrated and Dynamic Vehicle and Crew Scheduling (2004).

[24] D. Huisman, R. Freling, and A. P. Wagelmans. Multiple-depot integrated vehicle and crew scheduling. *Transportation Science* 39.4 (2005), pp. 491–502.

[25] R. M. Karp. On the computational complexity of combinatorial problems. *Networks* 5.1 (1975), pp. 45–68.

[26] L. G. Khachiyan. A polynomial algorithm in linear programming. *Doklady Akademii Nauk*. Vol. 244. 5. Russian Academy of Sciences. 1979, pp. 1093–1096.

[27] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *science* 220.4598 (1983), pp. 671–680.

[28] N. Kliewer, B. Amberg, and B. Amberg. Multiple depot vehicle and crew scheduling with time windows for scheduled trips. *Public Transport* 3.3 (2012), pp. 213–244.

[29] N. Kliewer, T. Mellouli, and L. Suhl. A time–space network based exact optimization model for multi-depot bus scheduling. *European journal of operational research* 175.3 (2006), pp. 1616–1627.

[30] A. H. Land and A. G. Doig. An Automatic Method of Solving Discrete Programming Problems. *Econometrica* 28.3 (1960), pp. 497–520.

[31] B. Laurent and J.-K. Hao. Iterated local search for the multiple depot vehicle scheduling problem. *Computers & Industrial Engineering* 57.1 (2009), pp. 277–286.

[32] A. Levin. Scheduling and fleet routing models for transportation systems. *Transportation Science* 5.3 (1971), pp. 232–255.

[33] J. D. Little, K. G. Murty, D. W. Sweeney, and C. Karel. An algorithm for the traveling salesman problem. *Operations research* 11.6 (1963), pp. 972–989.

[34] M. Mesquita and A. Paias. Set partitioning/covering-based approaches for the integrated vehicle and crew scheduling problem. *Computers & Operations Research* 35.5 (2008), pp. 1562–1575.

[35] M. Mesquita and J. Paixão. Multiple depot vehicle scheduling problem: A new heuristic based on quasi-assignment algorithms. *Computer-aided transit scheduling*. Springer, 1992, pp. 167–180.

[36] M. Mesquita and J. Paixão. Exact algorithms for the multi-depot vehicle scheduling problem based on multicommodity network flow type formulations. *Computer-aided transit scheduling*. Springer, 1999, pp. 221–243.

[37] A. Mingozzi, L. Bianco, and S. Ricciardelli. An exact algorithm for combining vehicle trips. *Computer-aided transit scheduling*. Springer, 1995, pp. 145–172.

[38] C. S. Orloff. Route constrained fleet scheduling. *Transportation Science* 10.2 (1976), pp. 149–168.

[39] J. P. Paixão and I. Branco. A quasi-assignment algorithm for bus scheduling. *Networks* 17.3 (1987), pp. 249–269.

[40] A.-S. Pepin, G. Desaulniers, A. Hertz, and D. Huisman. A comparison of five heuristics for the multiple depot vehicle scheduling problem. *Journal of scheduling* 12.1 (2009), pp. 17–30.

[41] C. C. Ribeiro and F. Soumis. A column generation approach to the multiple-depot vehicle scheduling problem. *Operations research* 42.1 (1994), pp. 41–52.

[42] J. Rintoul. *Online Supermarket "Picnic" is the Fastest Growing Company in the Netherlands*. Mar. 2019. URL: https://dutchreview.com/news/economy/online-supermarket-picnic-is-the-fastest-growing-company-in-the-netherlands/.

[43] J. Saha. An algorithm for bus scheduling problems. *Journal of the Operational Research Society* 21.4 (1970), pp. 463–474.

[44] A. Schrijver. *Theory of linear and integer programming*. John Wiley & Sons, 1998.

[45] M. Wen, E. Linde, S. Ropke, P. Mirchandani, and A. Larsen. An adaptive large neighborhood search heuristic for the electric vehicle scheduling problem. *Computers & Operations Research* 76 (2016), pp. 73–83.

[46] L. A. Wolsey. *Integer programming*. Wiley Online Library, 1998.

A

Implementation

A.1. Code setup

We introduce the way the code is set up by introducing the most important classes. Figure A.1 provides an overview of the dependencies of the classes. The figure depicts two types of classes. The purple classes, *Input*, *Config*, *ShipmentTW*, *Shipment*, *Truck* and *Schedule*, are fixed classes for every algorithm. The pink classes, *BuildingBlock* and *MainAlgorithm*, differ depending on what algorithm is executed. For example, if executing the Random Search algorithm without driver change, *MainAlgorithm* is replaced by the *RandomSearch* class and *BuildingBlock* is replaced by the *GreedyScheduler* class. The attributes and public methods listed for every class are examples. In the actual class more attributes and public methods appear, that could not all be displayed. If one would like to implement the docking constraint, or other constraints that are location-based, adding a *Location* class might be useful. This class could initiate an object for every depot, and monitor the number of trucks currently at the location.

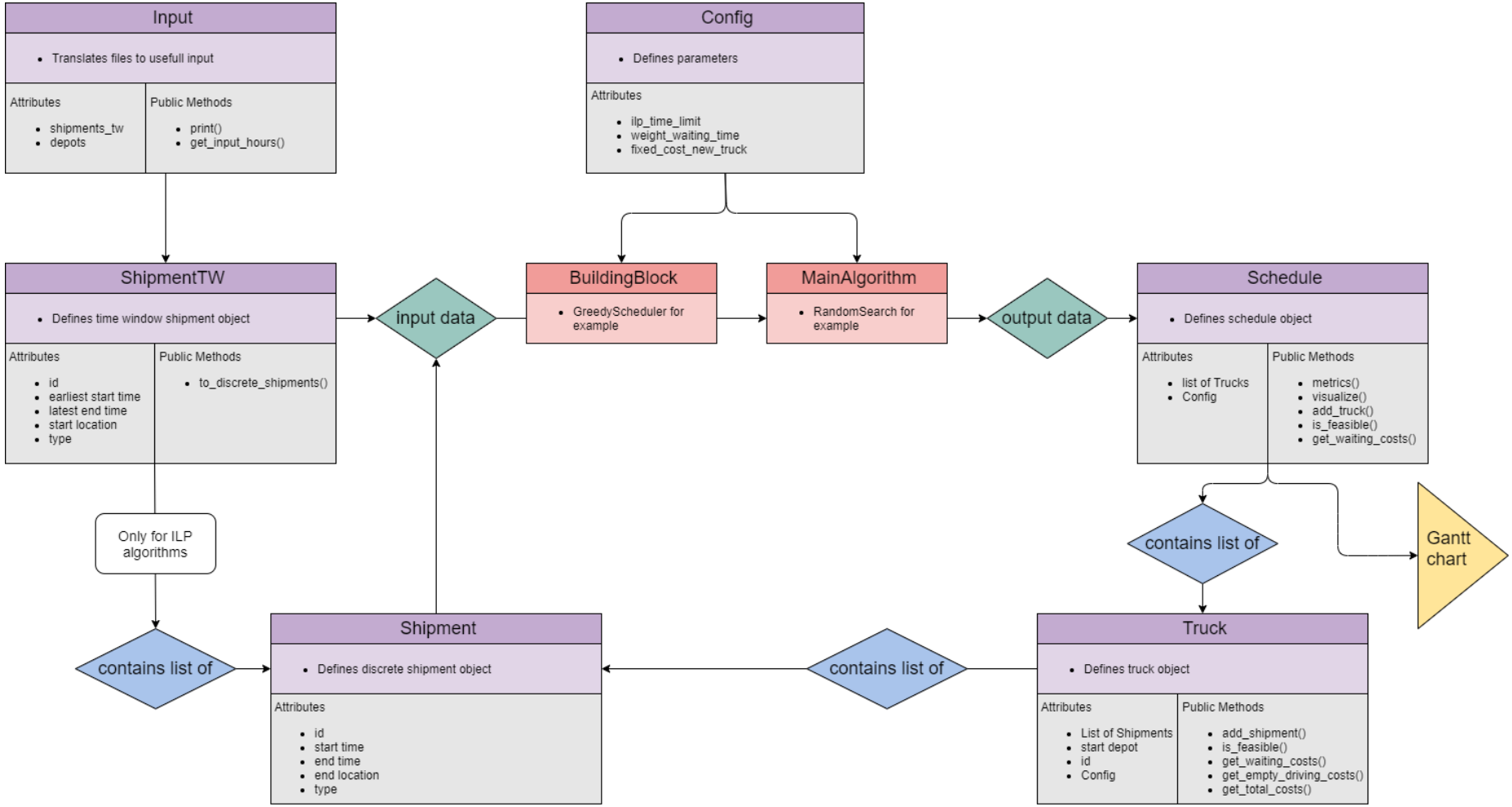


Figure A.1: UML diagram of the main classes in our code.

A.2. Our choices for parameter values

This section contains a short summary of our configurations for the algorithms. Table A.1 shows a list of all parameters for the objective function, ILP implementation and Greedy Scheduler, and two parameters from the Random Search and Fix algorithm. Table A.2 shows the number of iterations for the main algorithms that do not contain a Fixing Phase and Table A.3 show the chosen parameter values for the main algorithms that contain a Fixing Phase. For the exact definitions and explanations of all parameters, we refer to Chapter 5 and 6.

For the ILP implementation we choose to assume that an OB shipment can only be executed by a truck that starts from a depot that is at most 1 hour driving from the end location of the shipment. Also, we implement depot improvement and shipment sliding as post-processing steps, but only for truck days that are not splittable. For the Greedy Scheduler algorithm we also implement both post-processing steps only for truck days that are not splittable.

Parameter	Value
w_{fc}	100000
w_{dt}	60
w_{wt}	30
c_{max}	$w_{fc} + w_{dt}$
δ_{min}	10
δ_s	15
δ_m	7
t_{wt}^{max}	1.5
τ_{max}	450
δ_{gap}	$1 \cdot 10^{-5}$
l_e	18:00
l_d	11
e_s	12:00
e_d	11

Table A.1: An overview of the parameter values we chose for our algorithms.

Algorithm	I
ILP + Greedy no driver change	200
ILP + Greedy with driver change	200
Random Search no driver change	250

Table A.2: The parameter values for the main algorithms that do not contain a Fixing Phase.

Algorithm	ϕ_{min}	I_1	I_2
Random Search with driver change	80	150	500
Random Search and Fix no driver change	80	150	600
Random Search and Fix with driver change	80	50	400

Table A.3: The parameter values for the main algorithms that contain a Fixing Phase.

B

Additional experiments

B.1. Tie analysis for the Greedy Scheduler algorithm

When executing the Greedy Scheduler algorithm, multiple trucks might have minimal costs for a given shipment. Providing insight in the number of ties is import for understanding the degree of randomness of the algorithm, and for making the choice in how to break ties. In order to analyse the occurrence of ties, we executed the Greedy Scheduler algorithm 1000 times for all dates and counted the average number of ties per execution of the algorithm.

Table B.1 shows the average number of ties in one execution of the Greedy Scheduler algorithm, when only minimizing the costs. We can conclude that for 24% of all shipments, more than one truck has minimal costs. Table B.2 shows the average number of ties when minimizing costs and the distance of the depot and the end location of the shipment. From these results we can conclude that for 18% of the shipments ties occur.

	31_03	15_04	16_04	17_04	average
absolute number of ties	17	34	39	37	32
percentage of ties (%)	14	24	30	28	24

Table B.1: Average number of ties within one execution of the Greedy Scheduler algorithm, after executing the algorithm 1000 times, when only taking the costs into account.

	31_03	15_04	16_04	17_04	average
absolute number of ties per iteration	13	22	34	27	24
percentage of ties per iteration	11	16	26	20	18

Table B.2: Average number of ties within one execution of the Greedy Scheduler algorithm, after executing the algorithm 1000 times, when taking the costs and distance into account.

B.2. Splittable trucks analysis for the Random Search with driver change algorithm

When defining an algorithm that allows a driver change, we want to ensure that truck days appear that enable driver changes. This section analyzes the number of splittable truck days that appear when executing two strategies:

1. A straightforward approach is to execute Random Search without a maximal day duration restriction.

2. In order to ensure more splittable truck days to appear, we execute Random Search without a maximal day duration multiple times, and fix all splittable truck days that occur.

In order to examine the performance of the two strategies, we execute them with $I = 200$ for all test days five times and compare the average results. Table B.3 shows the number of splittable truck days that appear, when executing the first strategy. On average, the number of splittable truck days is 4. Table B.4 shows the number of splittable truck days, when executing the second strategy. On average, the number of splittable truck days is 13. Hence, fixing all splittable truck days that occur while executing Random Search without maximal day duration is an effective strategy, when aiming to increase the number of splittable truck days.

i	31_03	15_04	16_04	17_04
1	6	5	2	3
2	5	7	2	3
3	4	6	3	1
4	4	5	4	4
5	3	3	4	3
average	4	5	3	3

Table B.3: Number of splittable truck days when applying the first strategy described above.

i	31_03	15_04	16_04	17_04
1	11	13	12	12
2	13	14	13	13
3	13	13	13	14
4	13	14	12	10
5	14	13	13	15
average	13	13	13	13

Table B.4: Number of splittable truck days when applying the second strategy described above.

C

Additional algorithms

C.1. List Search algorithm

In the Greedy Scheduler algorithm the order in which the shipments are considered has a great influence on the final truck planning. Choosing the order to be based on increasing latest start time and length of time window does not lead to the optimal planning per se. Therefore, changing the order may result in a better planning. We implemented the List Search algorithm that exploits this fact by switching a random pair of shipments in the order and comparing the new solution with the best solution found so far. If the new solution is better than the best solution so far, the switched shipments remain switched. If not, the two shipments switch back. Table C.1 shows the results when executing this algorithm for $I = 500$ for the four for test days and taking the average, compared to the Random Search algorithm. We conclude that the performance of the two algorithms is comparable. It might be interesting to investigate if more sophisticated order changes result in better performance of the algorithm.

C.2. Random Search and Fix dynamic algorithm

The Random Search and Fix algorithm defines low-cost trucks in a static way: a truck is either low-cost or not, and the requirement remains the same throughout all iterations. The disadvantage of this approach is that we might reject trucks in the first phase that do not meet the requirement up to a small difference, but we would have liked to keep in hindsight, as we end up with worse when applying Random Search in the second phase. Inspired by Simulated Annealing, we choose to fix a certain truck from a solution constructed in a certain iteration with a probability that depends on how good the truck is and how far we are in the search process. The probability that a truck with cost t_c is accepted, which

Metric	Random Search	List Search
Objective value	5,606,340	5,606,268
Number of trucks	56	56
Number of drivers	56	56
Waiting (%)	3.89	3.05
Empty driving (%)	13.09	13.9
Inefficiency (%)	16.98	16.9
Total planned hours	602	607
Short driver days (<7 hours)	5	3
Average driver day duration (hours)	10.8	10.9
Running time (seconds)	132	132

Table C.1: Average results over the four test days of the Random Search algorithm and List Search algorithm.

means that it is fixed, in iteration $i \in \mathbb{N}$, is:

$$\text{probability of acceptance}(t_c, i) = e^{\frac{t_c - w_{fc}}{\gamma \cdot i}} \tag{C.1}$$

The probability is 1 if the truck costs are equal to the fixed costs, w_{fc}, and the higher the truck costs are, the smaller the acceptance probability is. But, the further we are in the search process, the higher the probability that a truck with certain costs is accepted gets. We introduce parameter $\gamma > 0$ that determines the pace at which the acceptance probability increases as we get further into the search process.

Using Simulated Annealing or Tabu Search techniques, we could also *unfix* trucks after have fixed them. An example is to unfix the fixed truck with highest costs when the unfixed shipments have to generate trucks above a certain costs.