Erasmus University Rotterdam

Erasmus School of Economics

Master Thesis

Econometrics & Management Science

Operations Research & Quantitative Logistics

The Resource Planning Problem

Abstract

The Resource Planning Problem (RPP) combines multiple well-studied Vehicle Routing Problems (VRP) into one rich problem. The problem consists of constructing a least-cost set of routes and finding the optimal resource assignment given an extensive set of constraints. To our best knowledge, this comprehensive VRP variant has not been considered before. We propose a Two-Stage Adaptive Memory Procedure (TSAMP), decomposing the problem into a route construction and resource assignment phase. The first phase revolves around an Adaptive Memory Procedure, which constructs routing plans by iteratively selecting routes from a memory. The second phase consists of assigning drivers, trucks and trailers to the selected routes. Simulated Annealing is incorporated in order to intensify the search. The algorithm has a flexible design and can be easily modified to other VRP variants. In order to assess the performance of the TSAMP, a diverse set of benchmark instances is generated. For most of these instances, special purpose solver CPLEX is not able to find feasible solutions, whereas the TSAMP finds competent solution within several minutes. Furthermore, the TSAMP finds significantly improved solutions compared to a Greedy Randomized Adaptive Search Procedure (GRASP), especially for larger instances.

Author: Jesper Nijboer (458799)

Supervisor: Dr. Oguzhan Vicil

Second assessor: Dr. Wilco van den Heuvel

Contents

1 Introduction

The aim of this research is to develop an efficient algorithm for the Resource Planning Problem (RPP). The RPP amounts to finding a routing plan and resource assignment such that each transport order is covered and the costs are minimized. This research is conducted in cooperation with Adaption, an IT company offering software solutions to logistic businesses.

In the RPP setting, the set of resources comprises of trucks, trailers and drivers, whereas a transport order consists of one or multiple activities such as pickup and delivery of certain goods. The considered RPP is a variant of the Vehicle Routing Problem (VRP) with additional constraints such as trailer capacity, heterogeneous vehicles, pickup and delivery requirements, time windows, multiple routes per vehicle and drivers' regulations. By considering all these facets, the resulting algorithm is suitable for assisting transport planners in decision making and resource management.

Several practical aspects touch upon the considered RPP features. First, deadhead kilometers (vehicles driving without goods) are estimated to cover approximately 20% of the total kilometers in freight transport (Terrazas, 2019). For intermodal transport with heavy load, the share of deadhead kilometers can even reach to 35%. From both an environmental and an economic perspective, these percentages urge the need for an efficient planning, where minimizing (deadhead) kilometers is one of the primary objectives. Second, in a distressing report by the ETSC (European Transport Safety Council, 2001), it is stated that driver fatigue plays a significant role in every fifth commercial road transport crash. Regulations for truck drivers is thus, not surprisingly, still an ongoing topic in the European Union. Neglecting these regulations in the planning stage might lead to excessive driving schedules, violation of the regulations and potential accidents. Last, transport companies face various types of transport orders with different requirements. Fresh food needs to be transported in a trailer with a cooling engine, airplane propellors have to be transported on an open low-loader and some other orders might require a tailgate for fast unloading. These features and additional restrictions as weight capacity and size of the load have to be considered in the planning stage in order to obtain a feasible planning.

Despite the wide range of developed solving techniques in Operations Research for similar problems, most transport planners still rely on experience and manually operate the planning. When the number of orders and available resources are substantial, this manual planning process can be complex and time consuming. An efficient planning tool could therefore benefit a transportation company in both planning efficiency and decreased labor cost of human planners, which stresses the practical relevance of this research.

Several similar problems include subsets of the constraints considered in the RPP. However, to our best knowledge, the RPP is not yet considered in the existing VRP literature. Hence, this work extends the literature by presenting methodology on a new problem, that combines many VRP variants into one extensive problem.

We propose a Two-Stage Adaptive Memory Procedure (TSAMP) inspired by the work of Rochat and Taillard (1995), Gendreau et al. (1999) and Olivera and Viera (2007). The algorithm is flexible and can be easily extended or simplified in order to include more or less restrictions. The TSAMP first constructs a

routing plan by selecting solution parts (routes) from different solutions and then assigns drivers, trucks and trailers to the routes. These two steps are iterated throughout the algorithm and enforced with Simulated Annealing.

New benchmark instances are constructed using generation principles inspired by the research of Uchoa et al. (2017). The instances vary from ten to 100 nodes and are diversified with respect to depot positioning, customer positioning, demand distribution and fleet size. In total, thirty instances are generated in order to assess the performance of the TSAMP.

From the computational results, we find that the performance of the TSAMP for small instances is mediocre. Special purpose solver CPLEX finds better solutions for all small instances. Furthermore, for half of the small instances, the TSAMP is not able to improve the starting solution, which is obtained by a randomized greedy construction procedure. The TSAMP however surpasses the construction procedure for larger instances up to 100 nodes. In contrary to the small instances, the TSAMP substantially improves the starting solution for each of the large instances. The CPLEX solver is not able to find feasible solutions or lower bounds for any of the large instances within a predefined time limit. Hence, the TSAMP outperforms the CPLEX solver in terms of feasibility for larger instances. However, no inferences can be made regarding optimality gaps.

The remainder of this research is structured as follows. Section 2 is dedicated to literature on problems related to the RPP. In Section 3, we explain the problem dynamics and introduce relevant mathematical notation. Section 4 starts with a thorough explanation of the model formulation and then continues with an elaboration on the TSAMP. Data generation, parameter tuning and computational results are presented in Section 5. In the last section, the conclusion of this research is presented.

2 Literature Review

The VRP is introduced by Dantzig and Ramser (1959) and initially defined as a truck dispatching problem. Their work ushered years of research into a plethora of related problem settings. In this section we highlight literature on several VRP variants. Section 2.1 is devoted to literature on the classical Vehicle Routing Problem and Sections 2.1.1 - 2.1.3 to relevant extensions. Subsection 2.2 is devoted to literature on drivers' regulations in routing problems. In Section 2.3 we aim to determine whether this extensive VRP variant has been considered before and which solution techniques have been proven successful.

2.1 Vehicle Routing Problem

The classical VRP with vehicle capacities is the most studied variant. According to Braekers et al. (2016), 90.5% of all published VRP papers in the period of 2009 to 2015 consider capacitated vehicles. In short, the VRP consists of designing a set of least-cost routes for a homogeneous fleet such that all customers are visited exactly once, each route starts and ends at a central depot and the accumulated demand of all customers on any route does not exceed the vehicle capacity. Finding this optimal routing plan is NP-hard (Toth and Vigo, 2002). The same applies for the considered variants in the next subsections, since they are generalizations of the VRP.

There are three main types of Mixed Integer Programming (MIP) formulations that are commonly used to model the problem. The Vehicle Flow formulation (Laporte and Nobert, 1983), the Commodity Flow formulation (Baldacci et al., 2004) and the Set Partitioning formulation (Balinski and Quandt, 1964). The first two formulations use binary variables to indicate arc traversing, which is particularly useful when the total cost can be expressed as the aggregated cost of all traversed arcs. However, many practical applications cannot be modelled using these formulations. The Set Partitioning formulation uses binary variables to indicate which route is selected and allows for a general cost function, alongside many practical constraints. However, the major drawback of this formulation is the exponential number of decision variables.

Exact branching algorithms have been widely used to tackle this problem. Toth and Vigo (2002) provide a comprehensive overview of several Branch-and-Bound algorithms, whereas Laporte (2007) emphasizes Branch-and-Cut algorithms based on all three MIP formulations. State-of-the-art branching algorithms are able to solve instances up to approximately 100 customers (Laporte, 2009). Exact Dynamic Programming (DP) algorithms are less successful and cannot compete with state-of-the-art branching algorithms. Christofides et al. (1981) propose a DP algorithm with a state-space relaxation, which is able to solve instances up to 25 customers. The area of DP algorithms seems extinguished and occurs more frequently in stochastic environments.

Due to the complexity of the problem, extensive research has been conducted in designing efficient meta-heuristics. Evolutionary algorithms proposed by Prins (2004), Baker and Ayechew (2003) and Bell and McMullen (2004) produce near-optimal solutions for large instances within a reasonable time. These algorithms mimic organic processes and operate on a population of solutions. Moreover, a plethora of heuristics is based on the concept of Local Search, which searches the solution space by considering small (local) changes. Gendreau et al. (1994) construct a Tabu Search heuristic that repeatedly removes and reinserts vertices. The algorithm proposed in Arnold and Sorensen (2019) completely revolves around Local Search, which is guided by pruning and perturbation techniques.

2.1.1 Time Windows

The Vehicle Routing Problem with Time Windows (VRPTW) assumes that each customer requires a certain time interval, in which visitation should take place. Usually, these time window restrictions are considered as hard constraints, however some papers include time windows as soft constraints and use a cost-service tradeoff as objective (Figliozzi, 2010). The problem is first introduced in Solomon (1987), in which a relatively easy heuristic based on insertions is presented. Braysy and Gendreau (2005-a) and Braysy and Gendreau (2005-b) give an excellent overview of algorithms considering this variant. In the latter research, it is concluded that algorithms based on Tabu Search or evolutionary techniques perform best. Cordeau et al. (2001) provide a robust Tabu Search heuristic, of which the strength lies in its simplicity and flexibility. Their heuristic is able to deal with multiple depots and periodic planning. The two evolution strategies in Homberger and Gehring (1999) produce new best solutions for several instances indicating the effectiveness of evolutionary-based algorithms.

2.1.2 Pickup and Delivery

In the literature on the Vehicle Routing Problem with Pickups and Deliveries (VRPPD), two different model settings are considered. Firstly, allowance of mixing of pickup and delivery within the same route and, secondly, the possibility of both pickup and delivery requirements simultaneously at the same location. In this research, we use the most general settings and allow for mixing and simultaneous pickup and delivery.

Nagy and Salhi (2005) propose a Local Search based heuristic. Ten different Local Search operators are used to intensify and diversify the search. The authors allow for intermediate infeasibility, in which case the search is guided towards a feasible solution. Chen and Wu (2006) follow a similar approach and iteratively search different neighborhoods. To avoid solution cycling, a Tabu list is used to store previously obtained solutions. In Ropke and Pisinger (2006), the pickup and delivery constraints are considered alongside time window constraints. The authors propose a Large Neighborhood Search, which is based on the principle of iteratively ruining and recreating the solution.

2.1.3 Heterogeneous Fleet

According to Gendreau et al. (1999), no exact algorithms exist for the Vehicle Routing Problem with Heterogeneous Fleet (HVRP). The authors propose a Tabu Search algorithm embedded in an Adaptive Memory Procedure (AMP), which stores partial solutions and combines superior solution components into new solutions. This principle can be seen as a generalization of Genetic algorithms. The authors in Choi and Tcha (2007) use a Column Generation approach and outperform many of the existing heuristics in terms of solution quality and computation time. Prins (2009) presents two different Memetic algorithms. These evolutionary algorithms enforced with Local Search compete with state-of-the-art heuristics and even produce several new best known solutions.

2.1.4 Multiple Trips

The concept of multiple trips is formally introduced in the work of Fleischmann (1990) and allows vehicles to perform more than one trip within the planning horizon. This variant frequently occurs with additional time window constraints. Exact algorithms are scant in the literature. However, branching algorithms seem to be the most pursued direction. One of the first exact methods for the Multi-Trip Vehicle Routing Problem (MVRP) appears in Koc and Karaoglan (2011). The authors construct a Branch-and-Cut algorithm and solve instances up to 50 customers. Mingozzi et al. (2013) propose an advanced Branch-and-Price algorithm based on two Set Partitioning formulations and find optimal solutions to 42 benchmark instances.

Tabu Search and population-based algorithms produce the best solutions for larger instances. Olivera and Viera (2007) use an AMP within a Tabu Search framework. The authors generate solutions for the classical VRP and use a Bin Packing approach to combine trips. The Memetic algorithm in Cattaruzza et al. (2014) also initially considers the classical VRP. The chromosome construction and splitting is similar to Prins (2004). A Dynamic Programming approach is used to transform the VRP solution into the best MVRP solution. For a more detailed overview of exact and heuristical approaches, we refer to Cattaruzza et al. (2016).

2.2 Drivers' Regulations

The majority of literature on routing problems does not consider regulations regarding driving and working hours. Considering these regulations is crucial in obtaining feasible driver schedules. This observation on deficiency in most VRP literature is also stated in Archetti and Savelsbergh (2009). The authors propose a polynomial time algorithm for verifying whether a feasible driver schedule exists for a given sequence of pickup and delivery requests with time windows. Goel (2010) presents a Breath First Search algorithm that is guaranteed to find a feasible driver schedule. The principle of such a Breath First Search is to find certain nodes in tree-based data structures. The author has produced several papers on drivers' regulations. In Goel (2012), the problem of finding a feasible driver schedule with minimum duration is considered. The author presents a flexible DP approach able to deal with legislation in both the EU and the US. In another research, the author considers the integrated problem of simultaneously vehicle routing and driver scheduling (Goel, 2009). The author shows that methods for handling drivers' regulations can be incorporated within a vehicle routing algorithm and proposes a Large Neighborhood Search. In contrary to Goel (2009), the Dynamic Programming heuristic in Kok et al. (2010) includes all legal rules that apply to a weekly planning period. The authors show that incorporating modifications and exceptions on the set of basic rules, which are usually neglected, allows for more flexible schedules and thus better solutions.

2.3 Discussion

The VRP variants discussed above are commonly combined into more general variants. For example, the time window constraints are frequently considered in combination with pickup and delivery constraints (Lin, 2008) or with allowance of multiple trips (Wang et al., 2014). However, we conclude, to our best knowledge, that there exists no VRP literature on the extensive variant considered in this research. This finding underlines the scientific relevance of this paper.

The two most successful heuristical approaches for VRP problems in general are evolutionary population-based algorithms and Local Search frameworks such as Tabu Search or Iterated Local Search. Algorithms following the evolutionary approach usually represent a single solution as a giant tour and use a splitting procedure to convert the tour into a set of feasible VRP routes. Efficient splitting procedures are established for both the MTVRP and the HVRP. We believe for this extensive variant that an evolutionary approach could succeed, but would result in notoriously difficult splitting procedures. Local Search frameworks usually do not need any intermediate procedure and are more straightforward in their design. Furthermore, efficient metaheuristics revolving around Local Search show an excellent time-quality tradeoff. Hence, a framework based on Local Search seems a promising direction for the highly constrained RPP.

3 Problem Description

The main goal of the RPP is to construct a routing plan and to determine which resources should be assigned to which trips such that the total costs are minimized. This section gives a comprehensive description of

the problem and is outlined as follows. In Section 3.1, we elaborate on the resources. Section 3.2 contains a description on transport orders. The objective of the RPP is considered in Section 3.3 and in the last section (3.4) we discuss all RPP restrictions.

3.1 Resources

The vehicle resources consist of a heterogeneous set of trucks $\mathcal{L}$ and a heterogeneous set of trailers $\mathcal{T}$. A truck is a motor vehicle used to pull a trailer, while a trailer is an unpowered vehicle used to carry materials and goods. It is assumed that the resources are stored at a central depot, from which they can start serving transport order. Each trailer t (truck l) has a trailer (truck) type with specific characteristics such as steerable axes or refrigeration mechanisms. Some of the truck-trailer combinations are excluded due to compatibility issues. We define $\mathcal{L}_t$ as the set of trucks compatible with trailer t.

Trucks and trailers are not continuously available for planning purposes. Preventive maintenance, changing tires or cleaning requirements are typical reasons for planning unavailability. Let $\mathcal{E}_l$ and $\mathcal{E}_t$ be the set of external events for truck l and trailer t respectively. Each external event has a fixed start and end time. Throughout this paper we assume that all time variables and parameters are measured in hours after the start of the planning horizon and that $t = 0$ is the start of the planning horizon.

All trucks l and trailers t have respective weight capacities Q_l and Q_t. The total weight capacity of a combination of truck l and trailer t can be obtained by adding the individual weights capacities, $Q_l + Q_t$. Another used capacity measure is the amount of load meters in a trailer (LDM_t), a commonly used criteria in freight transport. One LDM is equal to one meter of loading space of the trailer's length and can be interpreted as a simplified one-dimensional way to measure volume capacity.

The set of drivers is denoted by $\mathcal{D}$. Each driver d has a certain skill level and possesses one or multiple licences or certificates. For example, an ADR-certificate is required for transporting dangerous goods. Similar to the vehicle resources, drivers can be unavailable for planning. The set of external events for drivers is similar defined as the set for vehicle resources and is denoted by $\mathcal{E}_d$ for driver d with fixed start and end times. However, the unavailability reasons for drivers are different then for vehicle resources and can, for example, be holiday or maternity leave. In Table 1, an illustrative example of some fictional data is given.

Trailers					
Name	Type	Q	LDM	External Events	Features
01-AB-CD	Tautliner	100,000	10.5	Maintenance [55, 127]	
02-EF-GH	Flatbed	85,000	8	Cleaning [40,42]	Tailgate
03-IJ-KL	Standard	55,000	7		Refrigerated
Trucks					
Name	Type	Q		External Events	Features
04-MN-OP	Cab Over	50,000		Cleaning [40,42]	
05-QR-ST	Box Truck	25,000			Steerable axes
Drivers					
Name				External Events	Features
John Doe				Doctor [48,51]	ADR License
Jane Doe				Long Weekend [144,192]	

Table 1: Example resource data for a small instance.

3.2 Transport Orders

A transport order, which we refer to as an *order*, is a request from a customer to have some goods transported. We define the set of *orders* as $\mathcal{O}$. Each *order* consists of an ordered set of *activities* $\mathcal{I}_o$, where each *activity* represents a pickup or delivery subtask. Each *activity* i requires service time S_i, for (un)loading, and specifies a time window $[E_i, L_i]$, in which the service should be started. E_i represents the earliest start time (release) and L_i the latest start time (deadline). The time windows are agreed upon with the customers in a preplanning stage. For early arrivals, we assume that the resources have to wait inoperative. The goods collected or delivered at each *activity* i are quantified by measures weight Q_i and load meters LDM_i. We assume that the distances $D_{i,j}$ and travel times $T_{i,j}$ between each two *activity* nodes i and j are preprocessed and available in a constant time lookup table.

We distinguish between two *order* types:

- $|I_o| = 1$: *order* o consists of only one pickup (delivery) *activity*. The goods are collected (delivered) at the *activity* node and stored at (distributed from) a central depot. The required service time at the pickup (delivery) node is also incurred at the central depot for unloading (loading) on arrival (departure). It is assumed that the (un)loading at the depot is performed by a working crew.

- $|I_o| = 2$: *order* o consists of one pickup *activity* and one delivery *activity*, where the pickup node has to be visited before the delivery node. No service time is needed at the central depot in this case.

For an *order* to be served, it needs a truck, trailer and driver assigned to it. Not all resources are allowed to serve all *orders*. Some *orders* might require a specific trailer type, such as a 'Tautliner', or some additional features (i.e. cooling engine). This requirements can be specified by the planner based on experience or requested by the customer. Furthermore, special driver licences might be necessary for certain goods and the

planner can decide to set a required skill level for an *order*. The set of allowed trailers and drivers for *order* o are denoted by $\mathcal{T}_o$ and $\mathcal{D}_o$, respectively.

In Figure 1 an example of an *order* is presented. Some machine part needs to be collected at node A between 10:00 - 10:15 and delivered at node B between 12:00 - 12:30. Both collecting and delivering takes 15 minutes, which results in a total service time of half an hour. The distance between node A and B is 80 kilometer, which corresponds to one hour of driving. Suppose that the resources arrive at node A at 10:00 sharp, then node B can be visited 11:15 at the earliest. For early arrivals, we assume that the resources have to wait inoperative. The trailer type has to be a 'Tautliner' with a tailgate as additional feature. No specifications on drivers are given, implying that every driver is allowed to serve this *order*.

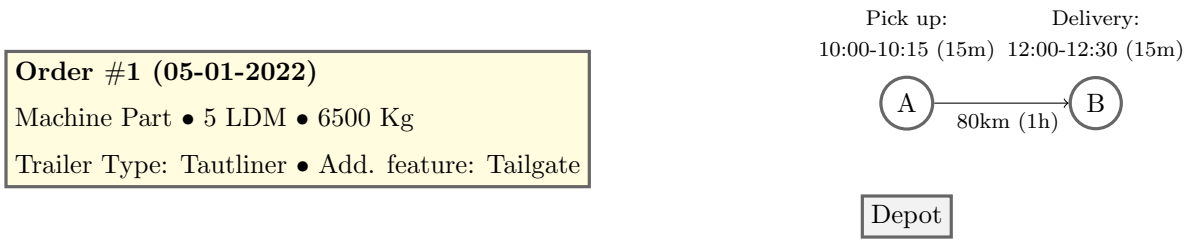

Figure 1: Example of an *order*. The yellow rectangle shows some details of the *order*, whereas the right figure shows a simplified geographical situation.

3.3 Objective

The objective of the RPP is to minimize the total cost, which consists of four parts similar to the fourfold cost structure in Xu et al. (2003)

The first part amounts to the total mileage cost. In the heterogeneous fleet setting, we have that the cost per kilometer is dependent on the used truck-trailer combination. Secondly, there is a fixed setup cost of using a particular resource, which also varies across the different resources. We do not include costs associated with assigning drivers to *orders*. We assume that drivers work on a contractual base and hence finding a feasible driver schedule is sufficient. Thirdly, for intensive driver schedules, an overnight stay in a hotel might be required. The costs resulting from arranging such sleeping accommodations are referred to as layover costs. Lastly, waiting cost per time unit is incurred in case of early arrival. The waiting costs are the only non-tangible costs and can be interpreted as penalty costs.

3.4 Restrictions

In this section, we discuss the considered RPP restrictions. Section 3.4.1 discusses *order* coverage constraints. In the next section, we describe constraints regarding trips. Section 3.4.3 explains resource assignment restrictions and the last section discusses drivers' regulations.

3.4.1 Order Coverage

The most obvious constraint is that each *order* should be covered by exactly the required resources. This amounts to exactly one truck, one trailer and one driver. It is possible that there exists no feasible solution with complete *order* coverage. In this case, the planning tool should still provide a solution, where some *orders* are put on hold. External resources can then be used to cope with this infeasibility.

3.4.2 Trip Restrictions

We define a *trip* as an ordered set of *activities* describing a route starting and ending at the central depot. We say that a *trip* is *locally feasible*, when some necessary constraints are satisfied and the *trip* has the potential to be part of RPP solution. To illustrate this, suppose that during the execution of *trip* v, a maximum of 12 LDM is attained and that the available resources are identical to the resources listed in Table 1. The maximum LDM capacity is present at trailer 01-AB-CD and amounts to 10.5 LDM. Hence, *trip* v can never be part of an RPP solution, since no trailer exists that potentially can serve this *trip*. In the remainder of this subsection, the constraints for a *trip* to be *locally feasible* are described.

The time windows of each *activity* within the *trip* should be respected. Arriving too early is allowed, however arriving after the specified time window of an *activity* yields an *locally infeasible trip*.

When an *order* consists of two *activities*, all *activities* should reside in the same *trip* and, moreover, the visitation order within the *trip* should comply with precedence constraints (i.e. pickup before delivery). These constraints are superfluous for *orders* with only one *activity*.

Following the example at the beginning of the section, the maximum attained weight and LDM during a *trip*, should be supported by at least one truck-trailer combination. Otherwise, no resource combination exists that is able to serve this *trip*.

The intersection of sets $\mathcal{T}_o$ and $\mathcal{D}_o$ for each order o in a *trip* should be non-empty. For example, if a *trip* contains both an *order* requiring refrigeration and an *order* requiring a tailgate, then no trailer in Table 1 is able to serve this *trip*. Both features are present, however not at the same trailer. This implies that these two *orders* cannot be together in one *trip*.

3.4.3 Resource Assignment Restrictions

Compatibility between trucks and trailers should be respected. Trucks and trailers that are incompatible can thus not be assigned to the same *trip*.

When a truck-trailer combination is assigned to a *trip*, the capacity (in terms of weight and LDM) should be sufficient. Suppose that a *trip* attains a maximum LDM of 10 and a maximum weight of 130,000, then the combination of trailer 01-AB-CD and truck 04-MN-OP is the only combination from Table 1 that has sufficient capacity. A combination of trailer 01-AB-CD and truck 05-QR-ST has sufficient LDM capacity, but not enough weight capacity, since $100.000 + 25.000 < 130.000$.

Next to the capacity constraints, the assigned trailer should possess all features incurred by the *orders* in a *trips*. This concretely means that a trailer should be in the intersection of all sets $\mathcal{T}_o$ for all *orders* o in a

certain *trip*. The same applies for drivers.

When assigning resources to *trips*, the availability of the resources have to be taken into account. We distinguish between two types of availability. The external availability of resources depends on external events that are unrelated to the *trip* assignments. To illustrate, driver Jane Doe from Table 1 is not available for planning during $[144, 192]$, since a long free weekend is planned during this time interval. The second type of availability follows from the resource planning itself. In our multi-*trip* setting, resources may be assigned to more than one *trip*. However, resources are not allowed to perform two distinct time-overlapping *trips* simultaneously.

3.4.4 Drivers' Regulations

In this research, we consider a subset of the regulations established by the European Union (European Union, 2006), since all customers of Adaption operate in this area. We assume that rules regarding small breaks and driving uninterruptedly are incorporated by the planner in the travel times. The considered rules are listed below.

- At most twice in a calendar week, it is allowed to drive for ten hours in one day. For the other days in the same calendar week the maximum is nine hours.
- The maximum driving time per calendar week is 56 hours.
- The maximum driving time per any two consecutive calendar weeks is 90 hours.
- The maximum working time per calendar week is 60 hours. In the RPP, we assume that working time includes driving time, service time at each *activity* and inoperative waiting at each *activity* node. The driver is excluded from working time arising from service at the depot.
- The daily rest period should be at least eleven consecutive hours. At most 24 hours after the end of the previous daily rest period, a new daily rest period should should be finished. We assume that a daily rest period can be either taken during the execution of a *trip* or at home in between performing *trips*.
- At most 144 hours after the previous weekly rest period, a new weekly rest period of at least 45 consecutive hours has to be started. We assume that a weekly rest period cannot be taken during the execution of a *trip*, which implies that a *trip* cannot span more than 144 hours.

For the calculation of driving time per day, we assume that the total driving time between two *activity* nodes is fully included in the calculation of driving hours for the day of departure. For example, if a driver leaves *activity* node i at 22:00 on day k and arrives at the next *activity* node $i+1$ at 01:00 on day $k+1$, then the three hours of driving time are fully included in the total driving hours for day k, whereas zero driving hours are added to day $k+1$. Furthermore, we assume that the working time emerging from inoperative waiting and providing service is incurred at the day, where the service at the corresponding *activity* is started.

In Table 2, an overview of all regulation parameters is given. Let $\mathcal{K}$ be the set of days included in the planning horizon and let $\mathcal{W}$ be the set of calendar weeks that (partly) coincide with the considered planning

horizon. Subscripts $w \in \mathcal{W}$ and $d \in \mathcal{D}$ are added to the last four parameters to allow for tuning for individual drivers and to cope with weeks that only partially fall in the planning horizon.

Symbol	Value	Description
S_{day}^{rest}	11 hours	Minimum duration of a daily rest period
S_{week}^{rest}	45 hours	Minimum duration of a weekly rest period
H_{consec}^{work}	144 hours	Maximum duration between the end and start of two consecutive weekly rest periods
H_{day}^{drive}	9 hours	Regular maximum driving hours per calendar day
H_{day}^{ext}	1 hour	Maximum extension of H_{day}^{drive}
$N_{d,w}$	2	Maximum number of times that H_{day}^{drive} can be extended per calendar week
$H_{d,w}^{work}$	60 hours	Maximum working hours per calendar week
$H_{d,w}^{drive}$	56 hours	Maximum driving hours per calendar week
$H_{d,w,w+1}^{drive}$	90 hours	Maximum driving hours per two consecutive calendar weeks

Table 2: Legislation parameters imposed by European Union (2006).

4 Methodology

The methodology used to solve the RPP is presented in this section. In Section A in the appendix, we give a MIP formulation for the RPP and extensively elaborate on some modelling choices, model assumptions and used notation. We strongly recommend to first read Section A before continuing to Section 4.1, since it gives a deep understanding of the RPP. In Subsection 4.1, we propose an algorithm inspired by the AMP presented in Olivera and Viera (2007). This technique is successfully implemented for several related problems and we believe highly suitable for the tightly constrained RPP.

4.1 Two-Stage Adaptive Memory Procedure

In this section, we present a two-stage heuristic revolving around an Adaptive Memory Procedure (AMP): a Two-Stage Adaptive Memory Procedure (TSAMP). The first stage consists of selecting *trips* from a memory and the second stage consists of iteratively applying local search and assigning resources to the selected *trips*. The main idea behind an AMP is to construct adequate solutions by combining parts from several good solutions, which are stored in a dynamic memory. The first appearance of an AMP in the VRP setting is in the research of Rochat and Taillard (1995). The authors conclude that the AMP technique can be used for a wide range of problem settings. This is confirmed by successful implementations for the heterogeneous (Gendreau et al., 1999) and multi-trip (Olivera and Viera, 2007) setting.

4.1.1 Outline Algorithm

The TSAMP framework yields similarities to the algorithm presented in Olivera and Viera (2007). However, several aspects of the algorithm, such as the memory initialization, Local Search procedure and resource

assignment are substantially different. The mathematical notation in this section is adopted from the used notation in Olivera and Viera (2007) and extended, if necessary.

Memory M consists of a list of *trips* with corresponding labels indicating the quality of the *trip*. The list is sorted according to the labels of the *trips*, where the best labels are put in the first positions. An example of M for a small instance with four *activity* nodes is presented in Table 3. As Table 3 shows, memory M contains parts (i.e. *trips*) of many different RPP solutions.

Trip	Label
$1 \rightarrow 3 \rightarrow 2$	400
$3 \rightarrow 4$	450
$2 \rightarrow 4$	600
$1 \rightarrow 3 \rightarrow 2 \rightarrow 4$	640
$3 \rightarrow 2 \rightarrow 4$	690
1	690
$4 \rightarrow 3$	800
4	900

Table 3: Small example of memory M for four *activity* nodes.

We do not allow M to contain *locally infeasible trips* according to the constraints described in Section 3.4.2, which implies that the *trips* in M form a subset of $\mathcal{V}$. In each TSAMP iteration, a routing plan s^{VRP} is constructed by selecting *trips* from memory M. The routing plan is subsequently transformed into an RPP solution by assigning resources to the *trips*. In the second stage of the iteration, local search is applied to the current solution in order to intensify the search. At the end of the iteration, each *trip* from the resulting RPP solution is reinserted in memory M. General pseudocode for the complete TSAMP algorithm is presented in Algorithm 1.

Algorithm 1 High-level Outline of the Two-Stage Adaptive Memory Procedure

1: initialize memory M using a Greedy Randomized Adaptive Search Procedure (Algorithm 2)
2: **for** $i = 1$ to $TSAMP^{iter}$ **do**
3: construct routing plan s^{VRP} by probabilistically selecting *trips* from M (Algorithm 3)
4: apply local search & resource assignment and obtain s^{RPP} (Algorithm 4)
5: update M using the *trips* in s^{RPP}
6: **end for**

The expectation is that in the early stages, memory M contains a divers and potentially poor set of *trips*. The routing plans in Step 3 are then expected to vary considerably among subsequent iterations. In this way, the local search in Step 4 is diversified, yielding a search in various areas of the solution space. As the TSAMP progresses, memory M is expected to contain more and more *trips* corresponding to good solutions. The local search is then intensified to richer areas of the solution space.

4.1.2 Memory Initialization

Memory M is initialized by a Greedy Randomized Adaptive Search Procedure (GRASP), which is an iterated randomized version of a greedy construction heuristic. The used construction procedure is based on the famous Savings Heuristic developed by Clarke and Wright (1964). Instead of a savings measure, we use a relatedness measure $R(o_i, o_j)$ between two *orders* $o_i, o_j \in \mathcal{O}$, which includes relatedness regarding distance, time, weight, LDM and the number of resources that are allowed to serve both *orders*.

Let $v = \{v_1, ...v_n\}$ be the sub-*trip* resulting from first serving all *activities* in $\mathcal{I}_{o_i}$ and then serving all *activities* in $\mathcal{I}_{o_j}$, without starting and ending at the depot. Then we define total distance D_{o_i,o_j} as $D_{v_1,v_2} + ...+D_{v_{n-1},v_n}$. Let $Span_{o_i,o_j}$, be the time between arriving at the first *activity* and leaving the last *activity* given the tightest time schedule. In Sections B.3 and B.4 in the appendix, it is explained how the tightest schedule for a given *trip* can be constructed. Let $Arrive_{v_1}$ be the time of arrival at first *activity* v_1 and let $Leave_{v_n}$ be the departure time from the last *activity* v_n, then the span can be calculated as $Span_{o_i,o_j} \leftarrow Leavev_n - Arrive_{v_1}$.

The parameters controlling the weights of the several relatedness measures are displayed as the first six parameters in Table 8. The total distance and time span are generally smaller for two *orders* with only one *activity*, than for two *orders* consisting of both two *activities*. Hence, subscript n is added to the first two parameters in order to account for a varying number of *activities* per *order*.

Relatedness measure $R(o_i, o_j)$ is defined as

$$R(o_i, o_j) = \begin{cases} \alpha_n D_{o_i,o_j} + \gamma_n Span_{o_i,o_j} + \theta^{weight}|Q_{o_i} - Q_{o_j}|+ & \\ \theta^{LDM}|LDM_{o_i} - LDM_{o_j}| + \lambda^D\left(1 - \frac{|\mathcal{D}_{o_i} \cap \mathcal{D}_{o_j}|}{\min\{|\mathcal{D}_{o_i}|,|\mathcal{D}_{o_j}|\}}\right)+ & \text{if } v \text{ is } \textit{locally feasible} \\ \lambda^V\left(1 - \frac{|\mathcal{T}_{o_i} \cap \mathcal{T}_{o_j}|}{\min\{|\mathcal{T}_{o_i}|,|\mathcal{T}_{o_j}|\}}\right) & \\ \infty & \text{otherwise.} \end{cases} \tag{1}$$

The weight and LDM relatedness is calculated by taking the absolute differences. The resource relatedness for drivers and trailers is calculated by considering the intersection of allowed resources for both *orders*. The denominator in the expression for trailers is added to compare the intersection size relative to the sizes of $\mathcal{T}_{o_i}$ and $\mathcal{T}_{o_j}$ and not to the size of $\mathcal{T}$. A similar reasoning applies to the relatedness for drivers. The lower the relatedness measure, the more related the two *orders*. Note that in general it holds that $R(o_i, o_j) \neq R(o_j, o_i)$. In the GRASP algorithm, we use a standardized version of $R(o_i, o_j)$ denoted by $\hat{R}(o_i, o_j)$, where the distance, span, weight and LDM measures are standardized such that they only take values in $[0, 1]$. In this way we have $\hat{R}(o_i, o_j) \in [0, \alpha_n + \gamma_n + \theta^{weight} + \theta^{LDM} + \lambda^D + \lambda^T]$.

Pseudocode for the GRASP algorithm is presented in Algorithm 2. Similar to the original Savings Heuristic, the solution is initialized with a *trip* for each *order* in Step 3. Then the algorithm attempts to iteratively merge *orders* that are highly related, where the selection of a relatedness label is the randomized greedy step. The probability of selecting the i^{th} label in $\tilde{RL}$ is equal to $2\frac{|\tilde{RL}|+1-i}{|\tilde{RL}|(|\tilde{RL}|+1)}$, which gives a higher probability to *orders* that are more related. This probability distribution is obtained as follows. Suppose that list $\tilde{RL}$ contains n elements. Then we assign weight $n+1-i$ to the i^{th} element such that the first element has a weight of n and the last element a weight of 1. The sum of all weights equals $\sum_{i=1}^{n} n+1-i = \sum_{i=1}^{n} i = \frac{n(n+1)}{2}$. Scaling all

weights by this expression gives the used probability distribution. Parameter $GRASP^{iter}$ controls the number of iterations and thus the number of solutions used to initialize memory M.

Algorithm 2 GRASP algorithm

Input: list of *order*-pairs RL sorted according to $\hat{R}(o_i, o_j)$ (ascending)

1: **for** $i = 1$ to $GRASP^{iter}$ **do**
2: $\tilde{RL} \leftarrow RL$
3: initialize s_i^{VRP} by constructing a *trip* for each *order* $o \in \mathcal{O}$
4: **while** $\tilde{RL} \neq \emptyset$ **do**
5: probabilistically select a relatedness label $\hat{R}(o_i, o_j)$ from $\tilde{RL}$
6: validate whether the merging the *trips* belonging to o_i and o_j results in a *locally feasible trip* (Steps 4 - 15 in Algorithm 11)
7: **if** possible to merge **then**
8: merge the two *trips* and update s_i^{VRP}
9: remove labels $\hat{R}(o_i, :)$, $\hat{R}(:, o_j)$ and $\hat{R}(o_j, o_i)$ from $\tilde{RL}$
10: **else**
11: remove label $\hat{R}(o_i, o_j)$ from $\tilde{RL}$
12: **end if**
13: **end while**
14: **end for**

Output: s_i^{VRP} for $i = 1, ..., GRASP^{iter}$

4.1.3 Memory Management

The management of memory M is almost similar to the memory management in Olivera and Viera (2007). Each *trip* in M is labelled according to the objective value of the corresponding RPP solution. Suppose *trip* v belongs to solution s^{RPP}, then *trip* v is labelled with $f(s^{RPP})$, where $f(\cdot)$ is defined as in (54). The *trips* in M are ordered in ascending order based on their respective labels, which implies that *trips* corresponding to good solutions reside in the first positions of the memory.

When the memory is updated at the end of a TSAMP iteration, the *trips* of the new RPP solution are labelled accordingly and added to the memory, while maintaining the ascending sorting. In case of duplicates, only the *trip* with the best label is retained. Parameter M^{size} limits the size of the memory. When the memory size $|M|$ exceeds this parameter, the surpassing *trips* with the highest labels are removed from the memory.

Algorithm 3 Procedure for selecting *trips* from the memory

Input: memory M, iteration number i

1: $\tilde{\mathcal{O}} \leftarrow \mathcal{O}$
2: $\tilde{M} \leftarrow M$
3: **if** mod $\{i, \kappa\} = 0$ **then**
4: draw integer $L^{uncover}$ probabilistically from interval $[0, L^{uncover}_{max}]$
5: randomly remove $L^{uncover}$ *orders* from $\tilde{\mathcal{O}}$
6: remove all *trips* from $\tilde{M}$ that contain *orders* in set $\mathcal{O} \setminus \tilde{\mathcal{O}}$
7: **end if**
8: $s^{VRP} \leftarrow \emptyset$
9: **while** $\tilde{M} \neq \emptyset$ **and** $\tilde{\mathcal{O}} \neq \emptyset$ **do**
10: probabilistically select *trip* v from $\tilde{M}$ and add to s^{VRP}
11: remove all *trips* from $\tilde{M}$ that have *orders* in common with v
12: remove all *orders* from $\tilde{\mathcal{O}}$ that are contained in v
13: **end while**
14: solve one iteration of the GRASP algorithm with the unrouted *orders* in $\tilde{\mathcal{O}}$

Output: routing plan s^{VRP}

Pseudocode for the memory selection process is given in Algorithm 3. Routing plan s^{VRP} is constructed by probabilistically selecting *trips* from the memory (Step 10). In order to favor *trips* corresponding to good solutions, the probability of selecting a certain *trip* is dependent on the relative position in the memory similar to the selection of the relatedness labels in Algorithm 2.

Throughout the algorithm, it can happen that we are not able to find an RPP solution, where all *orders* are served. It would then be myopic to always attempt to construct a routing plan from M containing all *orders*. Therefore, we sometimes omit some randomly selected *orders*. This is done as follows: every κ^{th} TSAMP iteration, a routing plan is constructed using a randomized subset of *orders*. This omitting of *orders* is displayed in Steps 3 - 7.

When memory $\tilde{M}$ is empty and there are still unrouted *orders* in $\tilde{\mathcal{O}}$, one iteration of the GRASP algorithm is executed in order to create *trips* for the unrouted *orders* (Step 14).

4.1.4 Local Search Procedure

In each TSAMP iteration, we apply a Simulated Annealing (SA) procedure to the selected *trips*. The procedure improves the constructed routing plan and avoids reinserting the same *trips* in the memory over and over again. The main principle behind SA is to allow the incumbent solution to deteriorate in order to prevent getting stuck in local optima. The probability of accepting worse solutions is gradually decreased such that the search is diversified in the early stages and intensified in the later stages of the SA procedure.

We define neighborhood $N(s^{VRP})$ as the feasible routing plans, containing *locally feasible trips* only, that can be obtained from incumbent solution s^{VRP} by performing one of the local search moves listed below.

- *Relocation*: This move attempts to relocate one *order* to another position. The new *trip* does not necessarily have to be a different *trip*, however all *activities* of the relocated *order* have to be inserted in the same *trip*.

- *Exchange*: This move tries to exchange two *orders* of two distinct *trips*. The new positions do not necessarily have to be the respective former positions, hence this move also considers exchanges between *orders* with an unequal number of *activities*

- *Reinsertion*: This move tries to reinsert an *order* that has been put on hold. All *activities* of the considered *order* have to be reinserted in the same *trip*. Only *orders* that are put on hold during the resource assignment procedures can be reinserted. *Orders* that are removed in Step 5 from Algorithm 3 are neglected throughout the entire TSAMP iteration.

All local search moves operate on an incumbent routing plan s^{VRP}, which means that the assignment of resources to *trips*, the exact amount of waiting time and the number of layovers are not yet determined. Therefore, the real change in objective, as defined in (54), of a certain local search move cannot be calculated. To overcome this problem, we adopt an alternative measure using the change in mileage, time span and uncovered *orders*. Suppose that $s^{VRP'} \in N(s^{VRP})$ and that this local search move modifies *trips* v^i and v^j into *trips* $v^{i'}$ and $v^{j'}$ respectively. We define $\Delta(s^{VRP}, s^{VRP'})$ as the cost of this local search move. For the *Relocation* and *Exchange* move, this alternative measure is calculated as

$$\begin{aligned}\Delta(s^{VRP}, s^{VRP'}) = &- (\tau Span_{v_i} + \min_{l,t:feasible}\{C_{v_i,l} + C_{v_i,t}\}) \\ &+ (\tau Span_{v_{i'}} + \min_{l,t:feasible}\{C_{v_{i'},l} + C_{v_{i'},t}\}) \\ &- (\tau Span_{v_j} + \min_{l,t:feasible}\{C_{v_j,l} + C_{v_j,t}\}) \\ &+ (\tau Span_{v_{j'}} + \min_{l,t:feasible}\{C_{v_{j'},l} + C_{v_{j'},t}\}).\end{aligned} \tag{2}$$

Each subterm represents the alternative cost of a single *trip* consisting of the sum of the total time span, as defined in (1), scaled by parameter τ and the assignment cost of the cheapest feasible truck-trailer combination. For the *Reinsertion* move, the measure is calculated slightly different. Let v_j be the *trip* that is used for the reinsertion, then the alternative measure can be derived by substituting the first two terms in (2) by $-C_{omit}$

Instead of determining a resource assignment at the end or beginning of the SA procedure, we compute a resource assignment in each SA iteration directly after performing a single Local Search move. This comes at the cost of additional computation time, but has two major advantages. First, all intermediate routing plans are transformed into an RPP solution and thus evaluated in terms of the actual objective. Hence, no potential good solutions are discarded. Second, without these intermediate resource assignments, the routing plan may be optimized perfectly with respect to the alternative measure (2), but might require extensive ad-hoc modifications in order to obtain a feasible RPP solution.

In Algorithm 4 below, the SA procedure is presented. From the initial routing plan, a RPP solution is constructed in Steps 1 and 2.

Algorithm 4 Simulated Annealing procedure for iteratively improving routing plan s^{VRP} and assigning resources

Input: routing plan s^{VRP}

1: assign drivers to the *trips* in s^{VRP} (Algorithm 5)
2: assign trucks and trailers to the *trips* in s^{VRP} and obtain s^{RPP} (Algorithm 9)
3: $s^{VRP} \leftarrow$ current routing plan in s^{RPP}
4: $T \leftarrow T_0$ & $s^{RPP}_{best} \leftarrow s^{RPP}$
5: **for** $i = 1$ to SA^{iter} **do**
6: randomly select $s^{VRP'} \in N(s^{VRP})$
7: **if** $\Delta(s^{VRP}, s^{VRP'}) < 0$ **then**
8: accept $s^{VRP'}$ with probability 1
9: **else**
10: accept $s^{VRP'}$ with probability $\exp(-\Delta(s^{VRP}, s^{VRP\prime})/T)$
11: **end if**
12: **if** accepted **then**
13: $s^{VRP} \leftarrow s^{VRP'}$
14: assign drivers to the *trips* in s^{VRP} (Algorithm 5)
15: assign trucks and trailers to the *trips* in s^{VRP} and obtain s^{RPP} (Algorithm 9)
16: $s^{VRP} \leftarrow$ current routing plan in s^{RPP}
17: **if** $f(s^{RPP}) < f(s^{RPP}_{best})$ **then**
18: $s^{RPP}_{best} \leftarrow s^{RPP}$
19: **end if**
20: **end if**
21: $T \leftarrow \psi T$
22: **end for**

Output: RPP solution s^{RPP}_{best}

A routing plan before resource assignment may be different than the routing plan after resource assignment, since it is not known in advance whether there exists a feasible resource assignment plan for a given routing plan. Hence, some *trips* may have to be split or some *orders* have to be put on hold in order to obtain a feasible RPP solution. Thus, after each resource assignment, routing plan s^{VRP} is subtracted from the incumbent RPP solution, which is done in Steps 3 and 16.

The for-loop in Steps 5 - 22 is similar to most SA frameworks. The number of iterations is controlled by parameter SA^{iter}. A new routing plan $s^{VRP'}$ is randomly selected from the neighborhood of s^{VRP} in Step 6. When $s^{VRP'}$ has a better objective in terms of the alternative measure, $s^{VRP'}$ is accepted as new incumbent routing plan (Step 8). Otherwise, $s^{VRP'}$ is accepted with the probability shown in Step 10.

When the new routing plan is accepted, resource assignment is performed in order to obtain a RPP solution. The best found RPP solution during the SA algorithm is stored in s^{RPP}_{best} and updated in Step 18.

Parameter T controlling the probability distribution in Step 10 is initialized with starting value T_0 in Step 4 and updated in Step 21 using parameter $\psi \in [0, 1]$.

4.1.5 Driver Assignment

The assignment of resources to *trips* in a given routing plan is split into a heuristic for drivers and a heuristic for vehicle resources. The constraints regarding drivers' regulations are quite extensive. Hence, the drivers are assigned in a separate algorithm, where weekly driving and working limits are considered and rest periods are iteratively incorporated. After the driver assignment phase, vehicle resources are assigned in a greedy way. The used methodology for assigning drivers is less straightforward than, for example, the SA algorithm. For that reason, we elaborate on the driver assignment heuristic (Algorithm 5) step by step using examples.

In Step 1 the tightest schedule for each *trip* in s^{VRP} is calculated. The *trips* are then stored in list SL and sorted in ascending order based on departure times from the depot (early departures first).

All variables that keep track of working and driving are initialized in Step 3. These variables are updated when a drivers is assigned to a *trip* and are used to check constraints regarding drivers' regulations. For the sake of consistency, we adopt the notation used in the MIP formulation.

In the main while-loop in Steps 4 - 27, we iteratively attempt to assign drivers to *trips*. In each iteration the *trip* in SL with the earliest departure time is selected (Step 5). If a *trip* cannot be served by any driver, the *trip* is split and reinserted in SL preserving the sorting.

By considering the *trips* in ascending order based on departure times, we always attempt to assign *trips* time-wise after *trips* assigned in previous iterations. In this way, arrival times at all nodes and start/end times of rest periods can be fixed after each assignment, which means that only the time window feasibility of the current *trip* has to be considered. Suppose, on the contrary, that we do allow to assign *trips* before or in between previously assigned *trips*, then we might have to shift time schedules of the previously assigned *trips* or incorporate some additional rest periods, which could result in cumbersome puzzle of considering multiple schedules for several *trips* simultaneously.

The set of drivers that are allowed to serve *trip* v is obtained by the intersection of $\mathcal{D}_o$ with $o \in \mathcal{O}_v$ and stored in temporary set $\tilde{\mathcal{D}}_v$ (Step 6). The drivers, that that are unavailable during the time span of the *trip*, are discarded in Steps 8 and 9. Suppose that current *trip* v spans $[25, 32]$, given the tightest schedule, and that driver $d \in \tilde{\mathcal{D}}_v$ has a private appointment during $[30, 31]$, then driver d cannot perform this *trip* and is discarded. Similarly, if a previously assigned *trip* for driver $d \in \tilde{\mathcal{D}}_v$ ends at the depot at $t = 27$, this driver is discarded.

The weekly service and weekly driving for each week w for *trip* v given the tightest schedule are calculated in Step 10. Suppose we are given the tightest schedule in Table 4 and weeks w_1 and w_2 with respective intervals $[0, 168)$ and $[168, 336)$. Despite the fact that the travel time from the depot to node 1 $[166, 169]$ overlaps with both weeks, this three hours of driving are fully incorporated in δ_{v,w_1} following the assumptions made in Section 3.4.4. Hence we have that $\delta_{v,w_1} = 3$ and $\delta_{v,w_2} = 2 + 2 = 4$. The start times of service for both nodes are in w_2, which implies that $\sigma_{v,w_1} = 0$ and $\sigma_{v,w_2} = 2.5$. These calculated variables can subsequently be used to check constraints in Steps 12 - 14 (bullet points 2 - 4 in Section 3.4.4). For each driver d, we add

the driving/work hours for the current *trip* v to the driving/working hours incurred by previous assignments. If for a certain driver the accumulated driving/working hours exceed one of the three legal bounds, the driver is discarded. Suppose that we have $\delta_{v,w} = 30$ for current *trip* v and that driver d is already assigned to some *trips* in week w resulting in $\delta_{d,w} = 40$. Assigning *trip* v to driver d would result in $30 + 40 = 70$ driving hours in week w, which is not allowed according to the legal maximum of 56 hours.

	Depot	Node 1	Node 2	Depot
S_i	0	1	1.5	0
$T_{i,j}$	3	2	2	-
$Arrive_i$	-	169	172	187.5
$Wait_i$	-	0	12	-
$Start_i$	-	169	184	-
$Leave_i$	166	170	185.5	-

Table 4: Tightest schedule of a *trip* visiting nodes 1 and 2. The time windows are not presented, since they are not relevant in this example.

We do not yet incorporate $\omega_{v,w}$, the waiting hours for the current *trip*, since daily rest periods are not yet incorporated in the *trip*, whereas daily rest periods might consume a large part of the waiting. For example, before node 2 in Table 4, there is enough waiting time to incorporate a daily rest period of eleven hours. Hence, counting this twelve hours of waiting as working hours gives a biased image of the actual working hours and leads to unnecessary discarding of drivers.

Constraints regarding daily driving hours are also not considered in this stage of the algorithm. Incorporation of daily rest periods is most likely to have a large influence on the driving hours per day. These constraints are thus considered in a later stage, when all rest periods are incorporated.

The drivers in set $\tilde{\mathcal{D}}_v$ are sorted based on the cardinality of SL_d, the set of *trips* in SL that driver d is allowed to perform (Steps 15 and 16). We want to favor drivers, where set SL_d is small. To illustrate this, when a driver d_1 is allowed to only perform the current *trip* and another driver d_2 is allowed to perform ten other *trips* that are still in SL, then we prefer driver d_1 for the current *trip*, since he or she cannot be assigned to any other *trip*.

The for-loop in 17 - 23 attempts to assign drivers from $\tilde{\mathcal{D}}_v$ to the current *trip* by validating whether daily and weekly rest periods can be feasibly planned when performing the current *trip*. The algorithm for planning all these rest periods is presented in Algorithm 7 and explained later. The number of iterations of this for-loop is controlled by parameter N^{driver}. The probabilistic selection of a driver in Step 18 is done using the same probability distribution as selecting *trips* from memory M and selecting relatedness labels in the GRASP algorithm. When Algorithm 7 indicates that a driver can perform *trip* v, the driver is assigned to the *trip* in Step 21 and the variables from Step 3 are updated using the driving and working hours incurred by the current *trip*.

When no driver can be assigned to the current *trip*, the *trip* is split into two new *trips*, which are reinserted

in SL. The procedure for this splitting process is described in Algorithm 6.

Algorithm 5 Driver assignment

Input: routing plan s^{VRP}

1: create tightest schedule for each *trip* in s^{VRP}
2: sort the *trips* based on departure times from the depot (ascending) and store in sorted list SL
3: initialize variables for all drivers d: $\beta_{d,k} \leftarrow 0$ (daily driving), $z_{d,k} \leftarrow 0$ (extension), $\delta_{d,w} \leftarrow 0$ (weekly driving), $\omega_{d,w} \leftarrow 0$ (weekly waiting), $\sigma_{d,w} \leftarrow 0$ (weekly service)
4: **while** $|SL| \neq 0$ **do**
5: $v^* \leftarrow$ *trip* in SL with earliest start time & $SL \leftarrow SL \setminus \{v^*\}$
6: $\tilde{\mathcal{D}}_{v^*} \leftarrow \mathcal{D}_{v^*}$, drivers that are allowed to perform this *trip*
7: let $[t_{v^*}^{startD}, t_{v^*}^{endD}]$ be the full time span of *trip* v^* given the tightest schedule
8: delete from $\tilde{\mathcal{D}}_{v^*}$ drivers d that have external events overlapping with $[t_{v^*}^{startD}, t_{v}^{*endD}]$
9: delete from $\tilde{\mathcal{D}}_{v^*}$ drivers d that have previously assigned *trips* overlapping with $[t_{v^*}^{startD}, t_{v^*}^{endD}]$
10: calculate $\delta_{v^*,w}$ (driving hours *trip* v^* in week w) and $\sigma_{v^*,w}$ (service hours *trip* v^* in week w) given the tightest time schedule and set $\omega_{v^*,w} \leftarrow 0$ (waiting hours *trip* v^* in week w)
11: delete drivers d that satisfy one of the following three violations:
12: 1) $\delta_{d,w} + \delta_{v^*,w} > H_{d,w}^{drive}$
13: 2) $\delta_{d,w} + \delta_{v^*,w} + \delta_{d,w+1} + \delta_{v^*,w+1} > H_{d,w,w+1}^{drive}$
14: 3) $\delta_{d,w} + \sigma_{d,w} + \omega_{d,w} + \delta_{v^*,w} + \sigma_{v^*,w} + \omega_{v^*,w} > H_{d,w}^{work}$
15: let $SL_d = \{v \in SL : d \in \mathcal{D}_v\}$ be the *trips* in SL that driver d is allowed to perform
16: sort $\tilde{D}_{v^*}$ based on $|SL_d|$ (ascending)
17: **for** $i = 1, .., N^{driver}$ **do**
18: probabilistically select driver d from $\tilde{D}_v$
19: run Algorithm 7 to verify whether driver d can perform *trip* v^* in a feasible way
20: **if** $feasible$ **then**
21: assign driver d to *trip* v^* and update all variables from Step 3
22: **end if**
23: **end for**
24: **if** no drivers can be assigned to v^* **then**
25: split *trip* v^* according to the procedure in Algorithm 6
26: **end if**
27: **end while**

Output: routing plan s^{VRP} equipped with drivers

The *trip* splitting process in Algorithm 6 can be described as follows. If more than one *order* is present in the *trip*, an attempt is made to split the *trip* at the most middle position possible in Step 2. Suppose we have two *orders* o_1 and o_2 in *trip* v with both a pickup and a delivery *activity*: $I_{o_1} = \{p_1, d_1\}$ and $I_{o_2} = \{p_2, d_2\}$. For *trip* $v = \{p_1, p_2, d_1, d_2\}$ no feasible splitting is possible, since all splits result in one of the two *orders*

divided over two *trips*, which is not allowed. In this case, only *trip* $v = \{p_1, d_1, p_2, d_2\}$ could potentially be split. If no split is possible, then the first appearing *order* is removed and put in a newly created *trip*. The new *trip* and reduced *trip* are reinserted in SL in Step 6.

When a *trip* cannot be assigned to any driver and the number of *orders* in the *trip* is one, no further splitting is possible and the *order* is put on hold in Step 9.

Algorithm 6 Procedure for splitting a *trip* into two new *trips*

Input: *trip* v

1: **if** $|\mathcal{O}_v| > 1$ **then**
2: let $v = \{v_1, ..., v_n\}$, then $i^* \leftarrow \max\{\min\{i, n-i\} : \textit{trip } v \text{ can be feasibly split at position } i\}$
3: **if** v can be split in a feasible way **then**
4: split *trip* v at position i^* and reinsert new *trips* in SL preserving the sorting
5: **else**
6: remove *order* o_1 corresponding to first *activity* and add the reduced *trip* and the *trip* only containing o_1 back to SL preserving the sorting
7: **end if**
8: **else**
9: remove *trip* v from s^{VRP} and put only *order* $o \in \mathcal{O}_v$ on hold
10: **end if**

Output: updated list SL

4.1.6 Insertion of Daily and Weekly Rest Periods

In Algorithm 7, it is validated whether a given driver can perform a given *trip* considering the daily and weekly rest requirements. To summarize, a daily rest period should last at least eleven hours and should be finished within 24 hours of the previous daily rest period (fifth bullet point in Section 3.4.4). A weekly rest period should last at least 45 hours and the time between two consecutive rest period can be at most 144 hours (last bullet point in Section 3.4.4). The presented algorithm is difficult to understand and read, hence the reader is guided through the algorithm using an example.

Suppose that *trip* v and driver d in Table 5 are input to Algorithm 7. The parameters in Steps 1 - 3 are presented in the most right column in Table 5.

In Steps 4 - 15 it determined when the next weekly rest period should planned. We have that the next weekly rest period should be started before $-35 + 144 = 109$. The if-statement in Step 4 is true, since the *trip*, given the schedule in Table 5, is finished at $t = 79$. Hence the new weekly rest period $\rho_{d,g}$ can be safely scheduled after this *trip*. The weekly rest period does not need to be planned yet, we only need to ensure that the *trip* does end later than $t' = 109$ (Step 5) such that the weekly rest period could theoretically be started at this latest deadline.

Parameter $t^{available}$ denotes that start time at which the driver is available for the current *trip* and $t^{deadline}$ the respective end time. These parameters are used to determine how much the current schedule can be delayed

or advanced without violating the driver's availability. In Step 6 these parameters are set to $t^{available} \leftarrow \max\{10, -70\} = 10$ and $t^{deadline} \leftarrow \min\{109, 100\} = 100$, which means that driver d is available during $[10, 100]$ for the current *trip*.

Trip v	Depot	Node 1	Node 2	Depot	Driver d	
S_i	0	1	1	0	t_d^{prev}	10
$[E_i, L_i]$	-	[50,52]	[75,78]	-	e_l	[-80,-70]
$T_{i,j}$	5	9	3	-	e_{l+1}	[100,150]
$Arrive_i$	-	52	62	79	$\rho_{d,f-1}$	[-5,6]
$Wait_i$	-	0	13	-	$\rho_{d,g-1}$	[-80,-35]
$Start_i$	-	52	75	-		
$Leave_i$	47	53	76	-		

Table 5: Tightest schedule of a *trip* visiting nodes 1 and 2 & needed information for driver d.

To illustrate Steps 7 - 14, suppose that $\rho_{d,g-1}$ would be planned during $[-150, -105]$, then we have that $-105 + 144 = 39 < 79$. In this case, the next weekly rest period should be planned before this *trip*. However, the time between the previous *trip* and current *trip* is $t_v^{startD} - t_d^{prev} = 47 - 10 = 37 < 45$ indicating that there is not enough time-space for a weekly rest period. Hence, the assignment of driver d to the current *trip* would be infeasible.

Since taking daily rest periods at home yields no layover costs, we attempt to schedule the next daily rest period directly before starting the current *trip*. In Step 16 it is checked whether there is enough time-space for a daily rest period. Since $47 - 10 = 37 > 11$, it is possible to schedule daily rest period $\rho_{d,f}$ directly before the *trip*. The start time and end times of $\rho_{d,f}$ are set to $t = 47 - 11 = 36$ and $t = 47$ respectively (Step 17). Technically, the constraint regarding consecutive daily rest periods is now violated. $\rho_{d,f-1}$ ends at $t = 6$ and $\rho_{d,f}$ ends at $t = 47$, which is more than 24 hours. However, driver d is not assigned to any *trip* in between those two daily rest periods. Therefore, this two daily rest periods can be interpreted as one long rest period.

Parameter $t^{available}$ is updated to $t^{available} \leftarrow \max\{10, 10+11\} = 21$ in Step 18. It might seem more logical to set $t^{available}$ to 47 in Step 18, since we scheduled a daily rest period during $[36, 47]$. However, the start and end times of $\rho_{d,f}$ are decided to be flexible such that they can be shifted together with the *trip* schedule. This is clarified with the following simplified example. Suppose that we have a *trip* that starts at $t_v^{startD} = 50$ and that the previous assigned *trip* for driver d ends at $t_d^{prev} = 35$. Using only this information, $t^{available}$ will be set to 35 and the remaining slack, in which the *trip* can be advanced, is initially 15 hours. However, since $50 - 35 = 15 > 11$ (Step 16), a daily rest period is incorporated directly before the *trip* during $[39, 50]$. It remains, that the *trip* and thus the daily rest period can be advanced at most fours hours, since the start time of the daily rest period ($t = 39$) will otherwise overlap with the previous *trip* (ends at $t = 35$). This change in availability can be captured by updating $t^{available}$ as in Step 18: $t^{available} = \max\{35, 35 + 11\} = 46$, which gives the desired four hours of advancing slack compared to start time $t = 50$.

The incorporation of daily rest periods within the *trip* is described in the while-loop in Steps 21 - 57. Step

22 calculates the deadline of the start time of the next daily rest period, which is $47 + 24 - 11 = 60$. If this deadline is after the end time of the current *trip* schedule, it is not necessary to incorporate any more daily rest periods. The driver can simply take his or her rest period after finishing the *trip*. In the example, the deadline $t^{deadline}_{\rho_{d,f}} = 60$ is before the end of the *trip* $t^{endD}_{v} = 79$, which means that at least one daily rest period needs to be incorporated in order for driver d to feasibly perform this *trip*.

When presenting the MIP formulation in Section A in the appendix, we assumed that daily rest periods during a *trip* can only be taken directly after leaving a node and thus not continuously on the interval between two *activity* nodes. This assumption is highly limiting and likely to cause modelling issues. When, for example, the travel time between two *activity* nodes is fifteen hours and it would not be possible to schedule a daily rest period somewhere during this fifteen hours, this arc can most likely not be traversed without violating the constraint of consecutive daily rest periods. To that extent, artificial nodes are included between each two *activity* nodes in order to be more flexible in scheduling daily rest periods during a *trip*. A more detailed explanation on the concept and generation of artificial nodes is given in Section B.2 in the appendix.

The set $\mathcal{P}$ in Step 26 contains all *activity* and artificial nodes in $\tilde{\mathcal{I}}_v$, after which the next daily rest period could potentially be taken.

The first constraint in the definition of $\mathcal{P}$ is straightforward, a rest period can only be scheduled after nodes, which are departed from before the rest deadline. The second constraint avoids that the new daily rest period can be inserted after nodes that are visited before the previous daily rest period. Instead of looking at the arrival time of the current node, the constraint considers the arrival time at the succeeding node. Otherwise it would not be possible to schedule two or more daily rest periods directly after the same node.

Figure 2 visualizes the *trip* including artificial nodes, where $AN_{i,j}$ denotes the j^{th} artificial node after *activity* i. All arrival and departure times for each node are displayed in Table 6 and obtained using the current schedule in Table 5 and the travel times in Figure 2. Set $\mathcal{P}$ can subsequently easily be constructed using Table 6: $\mathcal{P} \leftarrow \{AN_{0,1}, AN_{0,2}, \text{Node 1}, AN_{1,1}, AN_{1,2}\}$.

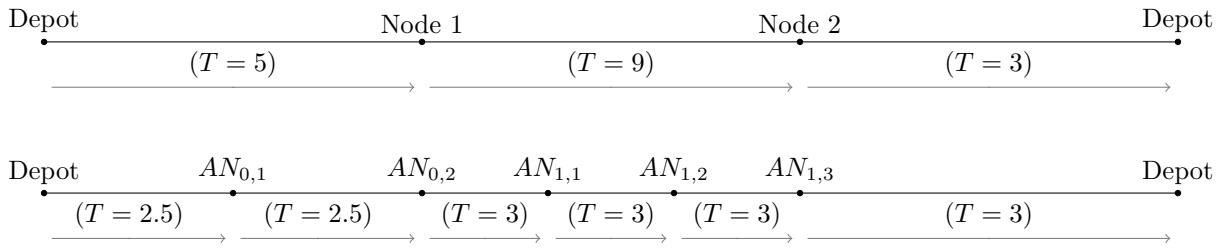

Figure 2: Simplified representation of the *trip* in the example. The first black line shows only the *activity* nodes and corresponding driving times. The second black line exhibits all artificial nodes. Nodes $AN_{0,2}$ & Node 1 and $AN_{1,3}$ & Node 2 are located at the same geographical location.

	Depot	$AN_{0,1}$	$AN_{0,2}$	Node 1	$AN_{1,1}$	$AN_{1,2}$	$AN_{1,3}$	Node 2	Depot
$Arrive_i$	-	49,5	52	52	56	59	62	62	79
$Leave_i$	47	49,5	52	53	56	59	62	76	-

Table 6: Arrival and departure times at each *activity* and artificial node given the current schedule.

Algorithm 7 Algorithm that validates whether rest periods can feasibly be taken

Input: driver d, *trip* v and corresponding $schedule_v$

1: let t_d^{prev} be the end time of the previously assigned *trip* (from either this instance or from the previous planning period)
2: let e_l, $e_{l+1} \in \mathcal{E}_d$ be the external events enclosing $schedule_v$
3: let $\rho_{d,f-1}$ and $\rho_{d,g-1}$ be the previous daily and weekly rest period respectively
4: **if** $t^{endD}_{\rho_{d,g-1}} + H^{work}_{consec} \geq t_v^{endD}$ **then**
5: $\quad \rho_{d,g}$ can be planned after v, but should start no later than $t' = t^{endD}_{\rho_{d,g-1}} + H^{work}_{consec}$
6: $\quad t^{available} \leftarrow \max\{t_d^{prev}, t_{e_l}^{endD}\}$ & $t^{deadline} \leftarrow \min\{t', t^{startD}_{e_{l+1}}\}$
7: **else**
8: $\quad$ **if** $t_v^{startD} - t_d^{prev} \geq S_{week}^{rest}$ **then**
9: $\quad\quad$ schedule $\rho_{d,g}$ before *trip* v with $t^{endD}_{\rho_{d,g}} = t_d^{prev} + S_{week}^{rest}$
10: $\quad\quad t^{available} \leftarrow \max\{t_d^{prev}, t_{e_l}^{endD}, t^{endD}_{\rho_{d,g}}\}$ & $t^{deadline} \leftarrow t_{e_{l+1}}^{endD}$
11: $\quad$ **else**
12: $\quad\quad feasible \leftarrow$ **false**
13: $\quad\quad$ **break** algorithm
14: $\quad$ **end if**
15: **end if**
16: **if** $t_v^{startD} - t_d^{prev} \geq S_{day}^{rest}$ **then**
17: $\quad t^{startD}_{\rho_{d,f}} \leftarrow t_v^{startD} - S_{day}^{rest}$, $t^{endD}_{\rho_{d,f}} \leftarrow t_v^{startD}$
18: $\quad t^{available} \leftarrow \max\{t^{available}, t_d^{prev} + S_{day}^{rest}\}$
19: $\quad f \leftarrow f + 1$
20: **end if**
21: **while** $feasible =$ **true do**
22: $\quad \rho_{d,f}$ should start before $t^{deadline}_{\rho_{d,f}} \leftarrow t^{endD}_{\rho_{d,f-1}} + H_{day} - S_{day}^{rest}$
23: $\quad$ **if** $t^{deadline}_{\rho_{d,f}} \geq t_v^{endD}$ **then**
24: $\quad\quad$ **break** while-loop
25: $\quad$ **end if**
26: $\quad \mathcal{P} \leftarrow \{i \in \tilde{\mathcal{I}}_v : Leave_i \leq t^{deadline}_{\rho_{d,f}}$ & $Arrive_{i+1} \geq t^{endD}_{\rho_{d,f-1}}\}$
27: $\quad foundPosition \leftarrow$ **false**

The for-loop in Steps 28 - 52 loops over all nodes in $\mathcal{P}$. Recall that a daily rest period during a *trip* yields

layover costs, hence the for-loop iterates in reversed order over the elements in $\mathcal{P}$ such that each new daily rest period is scheduled as late as possible. Node $AN_{1,2}$ is used in the first iteration, since this is the last element in $\mathcal{P}$.

Variable $flow_{\Rightarrow}^{max}$ in Step 29 denotes the amount of time the schedule can be delayed after node p, which is $AN_{1,2}$ in the first iteration. Node 2 is the only time window restricted node after $AN_{1,2}$. The arrival time at node 2 is $Arrive_2 = 62$ and the deadline is $L_2 = 78$, which means that the time schedule after node $AN_{1,2}$ can be delayed at most $78 - 62 = 16$ hours. Recall that $t^{deadline} = 100$ (Step 6) and that the *trip* currently ends at $t = 79$. Since $t^{deadline} - t_v^{endD} = 100 - 79 = 21 > 16$, the time window constraint at node 2 is dominant, which means that $flow_{\Rightarrow}^{max} \leftarrow 16$.

When, for example, $t^{deadline}$ would have been 80, the end time of the *trip* can be delayed at most one hour, since the current end time is $t = 79$. This implies that the driver can leave node 2 at $t = 77$ at the latest and thus start the service at $t = 76$ at the latest. Given the current arrival time $Arrive_2 = 62$, $flow_{\Rightarrow}^{max}$ would be set to $76 - 62 = 14$.

$flow_{\Leftarrow}^{max}$ in Step 30 is the counterpart of $flow_{\Rightarrow}^{max}$ and holds the value of the maximum amount of time the schedule can be advanced before and including node p. Only node 1 contains restricted time windows before node $AN_{1,2}$. We have that $Start_1 = 52$ and that $E_1 = 50$, which means that the schedule can be advanced at most $52 - 50 = 2$ hours. Since $t_v^{startD} - t^{available} = 47 - 21 = 26$, $flow_{\Leftarrow}^{max}$ is set to 2.

The if-statement in Step 31 checks whether the total available flow forward and backward is more than the required eleven hours. If this is the case, the daily rest period is inserted after the current node p, which is done in Steps 32 - 50.

Variable $flow_{\Leftarrow}$ represents the actual advancement of the schedule. Since it is desirable to schedule rest periods as late as possible, this variable is only positive, if the available flow forward only is not enough to fully incorporate the daily rest period. As $flow_{\Rightarrow}^{max}$ is 16 in the example, $flow_{\Leftarrow}$ is set to 0. If $flow_{\Leftarrow}$ is positive, the arrival times at all previous nodes and the current node are advanced with $flow_{\Leftarrow}$ hours. Furthermore, all previously incorporated daily rest periods during this call to Algorithm 7 are similarly advanced with $flow_{\Leftarrow}$ hours.

$flow_{\Rightarrow}$ in Step 34 represents the actual flow forward and is initialized as the maximum flow forward incremented with with $flow_{\Leftarrow}$. In the example $flow_{\Leftarrow} = 0$, which implies that $flow_{\Rightarrow} = 16 + 0 = 16$.

$cumulativeWait$ in the next step describes the cumulative amount of waiting at all nodes after node p. From Table 5 it can be obtained that at node 2 the driver has to wait for 13 hours, which is also immediately the value of $cumulativeWait$. This variable is used to determine the length of the daily rest period, since it might be beneficial to extend the rest period in order to consume some inoperative waiting. When there is no (or few) cumulative waiting, the additional time arising from the incorporated daily rest period is not consumed by inoperative waiting and leads to an delayed arrival at the depot. Therefore, we only want to extend a daily rest period more than necessary, if it consumes waiting time.

If the cumulative waiting is less then the flow forward in Step 36, the value of $flow_{\Rightarrow}$ is truncated to $cumulativeWait$ in Step 37. However, $flow_{\Rightarrow}$ cannot be less than 11, since this is the minimum duration of the rest period to be inserted. In the example $flow_{\Rightarrow} = 16$ is less then $cumulativeWait = 13$, hence

$flow_{\Rightarrow} \leftarrow \max\{13, 11\} = 13$.

The start and end times of the new daily rest period are determined in Steps 39 - 41. The start time is set to the departure time of the current node p, provided that there is no daily rest period already directly after p. If this is the case, the start time of $\rho_{d,f}$ is set to the end time of the previous daily rest period. In Step 39, $t^{startD}_{\rho_{d,f}}$ is set to $\max\{59, 47\} = 59$. The duration of the rest period is set in Step 40. All available flow forward is utilized with a maximum of λS^{rest}_{day}. Recall from constraint set (26) in the MIP, that λ is a parameter to control the length of daily rest periods. Suppose that $\lambda = 1.3$, then *duration* is set to $\min\{13, 1.3 * 11\} = \min\{13, 14.3\} = 13$. The end time in Step 40 is then set to $59 + 13 = 72$

The block in 42 - 47 is executed when the current daily rest period is planned directly after the previous daily rest period after the same node, which is not the case in the example. The remaining flow in calculated in Step 43. The current daily rest period is subsequently shifted with $shift$ hours. The current daily rest period should still start before the deadline $t^{deadline}_{\rho_{d,f}}$ from Step 22. Step 44 ensures that this deadline is not violated given $flow_{\Rightarrow}$. The reasoning behind this shift is that otherwise multiple daily rest periods could be planned straight after each other, which yields more layover costs than necessary. For example, in 50 hours time span usually two nights take place. When rest periods are planned without shifting, four rest periods could be planned to cover this 50 hours. This concretely means that four hotel nights are booked for this 50 hours, whereas two nights would be enough.

The schedule of the *trip* is updated in Step 49. The arrival time at the next node is determined using the end time of the inserted daily rest period and the still remaining travel time. The arrival time at the next node $A_{1,3}$ is calculated as $Arrive_{AN_{1,3}} = 72 + 3 = 75$, after which the schedule from Table 6 can be updated to the new schedule in Table 7. Since $flow_{\Leftarrow} = 0$, the schedule is not modified up to and including node $AN_{1,2}$. Suppose that we would not have truncated $flow_{\Rightarrow}$ in Step 37 from 16 to 13, then we would have that $Arrive_{AN_{1,3}} = Arrive_2 = 75 + 3 = 78$ and $Arrive_{\text{Depot}} = 79 + 3 = 82$, which is an unnecessary delay of the time schedule.

	$\rho_{d,f}$	Depot	$AN_{0,1}$	$AN_{0,2}$	Node 1	$AN_{1,1}$	$AN_{1,2}$	$\rho_{d,f}$	$AN_{1,3}$	Node 2	Depot
$Arrive_i$	36	-	49,5	52	52	56	59	59	75	75	79
$Leave_i$	47	47	49,5	52	53	56	59	72	75	76	-

Table 7: Arrival and departure times at each *activity* and artificial node given the modified schedule after the daily rest incorporation.

When it not possible to schedule a daily rest period after any node $p \in \mathcal{P}$, the assignment of driver d to *trip* is set to infeasible in Step 54 and the algorithm is terminated.

In the next iteration of the while-loop in Steps 21 - 57 we find that the deadline $t^{deadline}_{\rho_{d,f}}$ for the next daily rest period for driver d is $72 + 24 - 11 = 85$, which is after the arrival time at the depot $t^{endD}_{v} = 79$ in Step 23. The while-loop is stopped and the assignment of driver d to the *trip* is feasible.

In Step 58 the daily driving ($\beta_{v,k}$), weekly driving ($\delta_{v,w}$), weekly service ($\sigma_{v,w}$) and weekly waiting ($\omega_{v,w}$) are calculated given the modified schedule. This is done similarly to Step 10 in Algorithm 5.

In the last step these calculated variables are used to check the constraints listed in Steps 12 - 14 in Algorithm 5 and to check constraints (43) and (44) from the MIP regarding daily driving hours. These MIP constraints obviously only need to be checked for the current driver d, since a new *trip* is assigned to driver d only. This last step concludes the algorithm of inserting daily and weekly rest periods.

Algorithm 8 Algorithm that validates whether rest periods can feasibly be taken (remainder of Algorithm 7)

for all *activities* $p \in \mathcal{P}$ in reversed visitation order **do**
  $flow_{\Rightarrow}^{max} \leftarrow$ the maximum flow forward after p given time windows and $t^{deadline}$
  $flow_{\Leftarrow}^{max} \leftarrow \min\{\min_{i \leq p}\{Start_i - E_i\}, t_v^{startD} - t^{available}\}$ (maximum flow backwards)
  if $flow_{\Rightarrow}^{max} + flow_{\Leftarrow}^{max} \geq S_{day}^{rest}$ **then**
    $foundPosition \leftarrow$ **true**
    $flow_{\Leftarrow} \leftarrow \max\{S_{day}^{rest} - flow_{\Rightarrow}^{max}, 0\}$, $t_{\rho_{d,f}}^{deadline} \leftarrow t_{\rho_{d,f}}^{deadline} - flow_{\Leftarrow}$
    $flow_{\Rightarrow} \leftarrow flow_{\Leftarrow} + flow_{\Rightarrow}^{max}$
    $cumulativeWait \leftarrow \sum_{i>p} \max\{Wait_i 0\}$
    if $cumulativeWait < flow_{\Rightarrow}$ **then**
      $flow_{\Rightarrow} \leftarrow \max\{cumulativeWait, S_{day}^{rest}\}$
    end if
    $t_{\rho_{d,f}}^{startD} \leftarrow \max\{Leave_p, t_{\rho_{d,f-1}}^{endD}\}$
    $duration \leftarrow \min\{flow_{\Rightarrow}, \lambda S_{day}^{rest}\}$
    $t_{\rho_{d,f}}^{endD} \leftarrow t_{\rho_{d,f}}^{startD} + duration$
    if $\rho_{d,f}$ is directly after $\rho_{d,f-1}$ **then**
      $flow_{\Rightarrow} \leftarrow flow_{\Rightarrow} - duration$
      $shift \leftarrow \min\{t_{\rho_{d,f}}^{deadline} - t_{\rho_{d,f}}^{startD}, flow_{\Rightarrow}\}$
      $t_{\rho_{d,f}}^{startD} \leftarrow t_{\rho_{d,f}}^{startD} + shift$
      $t_{\rho_{d,f}}^{endD} \leftarrow t_{\rho_{d,f}}^{endD} + shift$
    end if
    $f \leftarrow f + 1$
    update schedule after p with $Arrive_{p+1} \leftarrow t_{\rho_{d,f}}^{endD} + T_{p,p+1}$
    break loop
  end if
end for
if $foundPosition =$ **false then**
  $feasible \leftarrow$ **false**
  break algorithm
end if
end while
calculate $\beta_{v,k}$, $\delta_{v,w}$, $\sigma_{v,w}$ and $\omega_{v,w}$ given the updated schedule with incorporated rest periods
check constraints 12 - 14 from Algorithm 5 & check constraints (43) and (44) (daily driving hours) from the MIP formulation $\Rightarrow$ store result in $feasible$

Output: boolean $feasible$ and a modified $schedule_v$ with incorporated rest periods, if feasible.

4.1.7 Truck-Trailer Assignment

The heuristic presented in this section is the final step in transforming a routing plan into an actual RPP solution. The output of the driver assignment algorithm in Section 4.1.5 is input to this final algorithm. In contrary to the driver assignment phase, the schedules of the *trips* are set to be fixed and can thus not be advanced or delayed. Otherwise, all carefully constructed schedules from Algorithms 5 and 7 are disrupted, which presumably leads to infeasible routing plans in terms of drivers' regulations. Hence, this section is solely concerned with assigning resources to fixed time slots. The pseudocode is presented in Algorithm 9.

In each iteration, the heuristic selects an uncovered *trip* based on two selection criteria, which are explained later. The assignment of resources to the selected *trip* is then done is a greedy way. Namely, the cheapest available truck-trailer combination is assigned to the current *trip*. If no truck-trailer combination is available for the current *trip*, previous assignments are iteratively cancelled en restored in order to still find a feasible assignment. When cancelling procedure fails to assign resources to the current *trip*, the *trip* is discarded from the routing plan and all *orders* are put on hold.

In Steps 1 and 2 all *trips* in routing plan s^{VRP} are stored in temporary set $\mathcal{V}'$ representing all uncovered *trips*. The main while-loop in Steps 3 - 57 iteratively attempts to cover all *trips* until set $\mathcal{V}'$ is empty and all *trips* are covered or discarded.

The selection criteria for selecting *trips* from $\mathcal{V}'$ are formulated in Steps 4 - 9. First, the *trips* covering the most *orders* are selected, since not covering these *trips* could lead to excessive omitting costs. When multiple *trips* contain the maximum amount of *orders*, ties are broken by considering the *trip* covering the most distance (Step 6).

From the set of allowed trailers for *trip* v^*, we discard all trailers that are not available during the time span of the *trip* (Step 11) . This unavailability can be caused by external events such as maintenance or by previously assigned *trips* that overlap with the current *trip*.

The for-loop in Steps 13 - 20 loops over all trailers and searches for the cheapest truck-trailer combination given the current trailer. All feasible combinations are stored in two-tuple set *combi*. The compatibility and availability constraints are captured in Steps 14 and 15 respectively. The weight capacity of the truck-trailer combination should be sufficient for the maximum attained weight during *trip* v^*, which is enforced in Step 16.

When set *combi* is not empty in Step 21, the cheapest combination from this set is assigned to the current *trip*. If the set is empty, we attempt to assign the *trip* by iteratively cancelling and restoring previous assignments in Steps 26 - 50.

We first explain the idea behind this cancelling and restoring procedure and then continue with elaborating on the algorithm. Suppose that routing plan s^{VRP} consists of three *trips* and that we have two trailers to our disposal. After two iterations we obtain the situation displayed in Figure 3, where the gray bars depict the time spans of the three *trips*. In the third iteration we still only have to assign the *trip* in the middle. Given the current assignments, both trailer 1 and trailer 2 cannot serve the middle *trip*, since they both overlap with previously assigned *trips*. When we would cancel the assignment of trailer 1 to the first *trip*, trailer 1 could

be assigned to the middle *trip*. After executing this step, it remains to restore the assignment of the first *trip*. Since the first and last *trip* do not overlap, trailer 2 can be feasibly assigned to the first *trip*, which concludes the cancellation and restoring procedure.

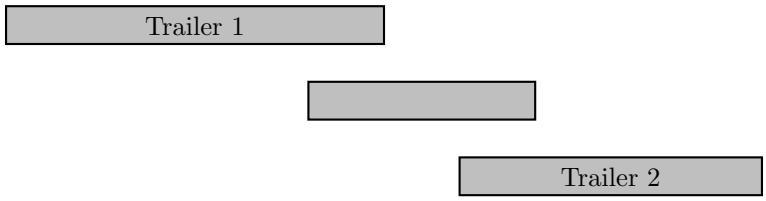

Figure 3: Result of the truck-trailer assignment heuristic after two iterations.

Let $\tilde{v}$ be the *trip* that needs to be assigned in the current iteration of the cancellation procedure. This *trip* is initialized with *trip* v^* in Step 25, since v^* is the reason for starting this cancellation procedure in the first place. In each iteration, we first attempt to restore the assignment of *trip* $\tilde{v}$ (Step 28) without any cancellations. This step is skipped in the first iteration, since $\tilde{v}$ is initialized with v^*, which currently cannot be assigned to any resources.

If it is possible to serve v^* without cancellations, $feasible$ is set to true in Step 31, which terminates the while-loop. When $\tilde{v}$ cannot be served, we iteratively cancel assignments in the for-loop in Steps 34 - 45 and anticipate that $\tilde{v}$ can be served after one of the assignment cancellations. If Step 36 reveals a feasible assignment for *trip* $\tilde{v}$ after a cancellation, this assignment is executed and the cancelled *trip* becomes $\tilde{v}$ in the next iteration of the cancellation procedure. When Step 36 finds no available truck-trailer combination, the cancellation is reverted and the for-loop proceeds to the next potential cancellation. When none of the cancellations results in an assignment for *trip* $\tilde{v}$, the while-loop is terminated by setting *counter* to N^{cancel} in Step 47. Parameter N^{cancel} is incorporated to control the number of cancellations that are actually executed and to avoid that the algorithm runs indefinitely. When the cancellation procedure indicates that it is not possible to serve current trip v^* given the already assigned *trips*, the assignments are reverted to the situation before the cancellation procedure in Step 52 and the *orders* in *trip* v^* are put on hold (Step 53).

Algorithm 9 Truck-trailer assignment

Input: routing plan s^{VRP}, where each *trip* has an assigned driver and a fixed start and end time

let $\mathcal{V}_{s^{VRP}}$ be the incumbent *trips* in s^{VRP}
initialize $\mathcal{V}' \leftarrow \mathcal{V}_{s^{VRP}}$
while $\mathcal{V}' \neq \emptyset$ **do**
  let $\mathcal{V}^*$ be the *trip(s)* covering the most *orders*: $\mathcal{V}^* \leftarrow \text{argmax}_{v \in \mathcal{V}'}\{|\mathcal{O}_v|\}$
  if $|\mathcal{V}^*| > 1$ **then**
    let v^* be the *trip* covering the most distance: $v' \leftarrow \text{argmax}_{v \in \mathcal{V}^*}\{D_v\}$
  else
    let v^* be the only *trip* present in $\mathcal{V}^*$
  end if
  let $\mathcal{T}_{v^*}$ be the set of trailers allowed to perform *trip* v^*
  discard from $\mathcal{T}_{v^*}$ trailers that are not available during $[t_{v^*}^{startV}, t_{v^*}^{endV}]$
  initialize two-tuple set (l, t) *combi* $\leftarrow \emptyset$
  for all $t \in \mathcal{T}_{v^*}$ **do**
    let $\mathcal{L}_t$ be the set of trucks compatible with trailer t
    discard from $\mathcal{L}_t$ trucks that are not available during $[t_{v^*}^{startV}, t_{v^*}^{endV}]$
    find cheapest truck: $l' \leftarrow \text{argmin}_{l \in \mathcal{L}_t}\{C_{v^*,l} : Q_t + Q_l \geq Q_{v^*}^{max}\}$
    if $l' \neq \emptyset$ **then**
      add (l', t) to *combi*
    end if
  end for
  if *combi* $\neq \emptyset$ **then**
    let (l', t') be the cheapest truck-trailer combination: $(l', t') \leftarrow \text{argmin}_{(l,t) \in combi}\{C_{v^*,l} + C_{v^*,t}\}$
    assign truck l' and trailer t' to *trip* v^*
  else

Algorithm 10 Truck-trailer assignment (remainder of Algorithm 9)

initialize $feasible \leftarrow$ **false**, $counter \leftarrow 0$ & $\tilde{v} \leftarrow v^*$
while $counter < N^{cancel}$ **and** $feasible =$ **false do**
  if $counter > 0$ **then**
    verify whether *trip* $\tilde{v}$ can be served by running Steps 10 - 23
  end if
  if possible to serve **then**
    $feasible \leftarrow$ **true**
  else
    initialize $foundRemoval \leftarrow$ **false**
    for all *trips* v in $\mathcal{V}_{sVRP} \setminus \mathcal{V}'$ **and while** $foundRemoval =$ **false do**
      cancel current assignment of (l, t) to v
      verify whether *trip* $\tilde{v}$ can be served by running Steps 10 - 23
      if possible to serve **then**
        let (l', t') be the cheapest available truck-trailer combination
        $foundRemoval \leftarrow$ **true** & assign (l', t') to *trip* $\tilde{v}$
        $\mathcal{V}' \leftarrow \mathcal{V}' \cup \{v\} \setminus \{\tilde{v}\}$ & $\tilde{v} \leftarrow v$
        $counter \leftarrow counter + 1$
      else
        reassign (l, t) to *trip* v
      end if
    end for
    if $foundRemoval =$ **false then**
      $counter \leftarrow N^{cancel}$
    end if
  end if
end while
if $feasible =$ **false then**
  restore the truck-trailer assignments and set $\mathcal{V}'$ as before Step 26
  put *orders* in *trip* v^* on hold
end if
end if
$\mathcal{V}' \leftarrow \mathcal{V}' \setminus \{v^*\}$
end while

Output: RPP solution

5 Results

In this section, we perform a computational experiment in order to assess the performance of the TSAMP. The MIP formulation from Section A is solved using the commercial solver IBM ILOG CPLEX Optimization Studio 22.1.0 and implemented in Java 12 using Eclipse IDE version 2022-3. The MIP computations are performed locally on a laptop equipped with an Intel Core i7-8550U processor working at 1.99 GHz using Windows 11 Home.

The TSAMP is fully integrated in the Adaption Logistics Cloud Suite, which operates on Oracle Database 18c. Section D in the appendix contains an introduction to this application including some examples. Procedural language PL/SQL is used to implement the TSAMP on the back-end. This language is designed by Oracle and can be used to directly access and manipulate data from the database using functions and procedures. The TSAMP calculations are performed on an external server equipped with an Intel Xeon E5-2660 v3 processor working at 2.59 GHz using Windows Server 2016 Standard.

Generally, the same programming language is used for all computations. However, no commercial MIP solver exists for language PL/SQL. On the other hand, connecting Java to an Oracle database is rather technical and thus decided to be out of scope for this research. The difference in both programming language and processor unfortunately complicate the comparison of both models. In the remainder of this section, the computation times obtained by CPLEX and the TSAMP are nonetheless compared. The reader should however keep the comparison complications in mind.

The outline of this section is as follows. In Section 5.1, we elaborate on the data generation process. Section 5.2 discusses the parameter setting and tuning. The computational results are presented and discussed in Section 5.3.

5.1 Data Instances

Since the RPP is a new problem, benchmark instances are generated in order to test the performance of the TSAMP algorithm. In Uchoa et al. (2017), extensive research is conducted into generating a diverse set of instances. Several generation mechanisms used in Uchoa et al. (2017) are therefore adopted and implemented in this research. In Sections 5.1.1 - 5.1.7, all generation techniques are discussed. Table 16 in Section C in the appendix exhibits a summary of all features per instance.

5.1.1 Instance Size

The considered instances consist of $X \in \{10, 50, 100\}$ *activity* nodes. We iteratively select one of the three following *orders* at random and add it to the instance.

- *Order* consisting of only one pickup *activity* (P)
- *Order* consisting of only one delivery *activity* (D)
- *Order* consisting of one pickup *activity* and one delivery *activity* (PD)

This is repeated, until X *activity* nodes are included. Note that the number of considered *orders* in an instance is generally less than the number of *activity* nodes, since some *orders* consist of two *activities*. For each considered instance size $X \in \{10, 50, 100\}$, we generate ten different instances. The set of instances with size X is denoted as nX and we refer to the i^{th} instance of size X as nX-i with $i \in \{1, ..., 10\}$.

5.1.2 Node Positioning

We assume that the depot and all *activity* nodes are located in a two-dimensional grid defined by $[-300, 300] \times [-300, 300]$. The distances between each two *activity* locations can be obtained using the Euclidean distance rounded to three decimals. The travel time between two *activity* is calculated using an average speed of 80 km/h. To illustrate, suppose $D_{i,j} = 200$, then $T_{i,j} = 200/80 = 2.5$.

Three different depot positions are examined:

- Random (R): Depot is positioned at a random location.
- Central (CE): Depot is positioned at $[0, 0]$.
- Corner (CO): Depot is positioned at $[-300, -300]$.

For the *activity* positioning we also consider three different positioning mechanisms:

- Random (R): Each *activity* is located at a random positioning.
- Cluster (C): Randomly select S seed *activities* with $S \sim Unif(1, 0.2X)$, that will function as cluster middle points. Next, each of the remaining $X - S$ *actitivies* is randomly assigned to a seed *activity*. The coordinates of an assigned *activity* are then obtained as follows. Suppose *activity* i is assigned to seed *activity* s with coordinates (x_s, y_s). The coordinates of *activity* i are then calculated as $(x_s + N_x, y_s + N_y)$ with $N_x \sim N_y \sim N(0, 40)$. Coordinates that fall outside range $[-300, 300]$ are truncated, what could result in some tight clusters in the corners. When a newly generated coordinate overlaps with another coordinate, the coordinates are regenerated. This could, for example, happen when coordinates $[350, 350]$ are truncated to $[300, 300]$ and coordinates $[325, 330]$ are also truncated to $[300, 300]$.
- Cluster-Random (CR): Half of the customers are positioned randomly. The other half is positioned according to the above described procedure.

For each instance, we randomly select one of the three depot and *activity* positioning mechanisms.

5.1.3 *Activity* Features

The demand in terms of weight and LDM is generated according to one of the following three mechanisms:

- Unitary (U): All weight demands have value 25,000 and all LDM demands have value 2.5.
- Random (R): All weight demands are drawn from $Unif(10,000; 50,000)$ and all LDM demands are drawn from $Unif(1, 5)$.

- Random Small-Large (SL): 80% of *activities* gets weight from $Unif(10,000; 30,000)$ and LDM from $Unif(1, 3)$. 20% of *activities* gets weight from $Unif(50,000; 100,000)$ and LDM from $Unif(5, 10)$.

Note that for *orders* consisting of two *activities* the generated weight and LDM are identical for both *activities*.

The planning horizon for each instance is assumed to span two weeks starting one a Monday such that two full calendar weeks are considered. When an *order* consists of only one *activity* i, the release date E_i is drawn from $Unif(0, 336)$, since exactly 336 hours reside in two calendar weeks. Given E_i, L_i is generated as $E_i + T$, where T is drawn from $Unif(0.25, 8)$. When an *order* consists of two *activities*, the time windows for the first *activity* i_1 are generated identically to the case of only *activity*. A necessary condition for the time windows of the second *activity* i_2 is that $E_{i_1} + T_{i_1,i_2} \leq L_{i_2}$. Otherwise, no feasible time schedule would exist for a *trip* containing such an *order*. Hence, deadline L_{i_2} is drawn from $Unif(E_{i_1} + T_{i_1,i_2}, E_{i_1} + T_{i_1,i_2} + 5)$ and release date E_{i_2} is drawn from $Unif(L_{i_2} - 8, L_{i_2} - 0.25)$ given L_{i_2}.

5.1.4 Resources Features

For all instances, we use an universal set of truck and trailer types with fixed weight and LDM capacities, which is presented in Table 15 in Section C in the appendix. We distinguish between five trailer master types: Tiny, Small, Normal, Large and Heavy. The weight capacities are identical for all trailer master types, while the LDM capacity is shifted two units for each trailer master type. In Table 15 we have $25 * 5 = 125$ different truck-trailer combinations. From these 125 combinations, we randomly make $0.2 * 125 \approx 37$ combinations incompatible.

For each instance we draw a proportion p^{truck} ($p^{trailer}$) in the interval $[0.2, 0.5]$, after which $\lceil p^{truck} * X \rceil$ trucks ($\lceil p^{trailer} * X \rceil$ trailers) are selected from Table 15 and included in the instance. Suppose we have to select three trucks, then for each truck we randomly select a truck type with repetition. This implies that we could have multiple trucks of the same truck type in one instance. The same holds for trailers and trailer types.

Similarly to trucks and trailer, proportion p^{driver} is drawn in interval $[0.2, 0.5]$ and used to create a set of drivers for each instance.

By using these proportions, the resource fleet size per instance is intentionally restrained. This is done for the following reasons. First, several constraints in the MIP formulation appear per truck, trailer or driver. With a a large set of resources, an exact approach is most likely not able to find a feasible solution for even small instances sizes. Since we want to compare the performance of the TSAMP to an exact approach, the set of available resources is restricted. Second, we are interested in whether the TSAMP is able to serve most or all *orders* given a limited fleet size. Considering a large or unlimited fleet size would disregard the multi-*trip* feature of the RPP, since it is then not necessary to reuse resources. Last, all customers of Adaption have a fixed an relatively small set of resources. Hence those customers are represented best when using these proportions.

For each driver, trailer and truck in each instance we use the following three scenarios for generating external events:

- Zero external events. This scenario has a probability of 0.5
- One external event. This scenario has a probability of 0.3
- Two external events. This scenario has a probability of 0.2

Similar to time windows, the start time of each external events is drawn uniformly from the considered planning horizon of two weeks $[0, 336]$. The duration of each external event is uniformly drawn in interval $[6, 72]$.

5.1.5 Allowed Resources

Sets $\mathcal{T}_o$ and $\mathcal{D}_o$ containing all allowed resources for *order* o are generated using a relatively simple mechanism. Each trailer (driver) in a given instance has a probability of 0.7 to be in set $\mathcal{T}_o$ ($\mathcal{D}_o$). When, unintentionally, set $\mathcal{T}_o = \emptyset$ ($\mathcal{D}_o = \emptyset$), one trailer (driver) is randomly selected from $\mathcal{T}$ ($\mathcal{D}$) and added to $\mathcal{T}_o$ ($\mathcal{D}_o$).

5.1.6 Cost Parameters

For each instance we fix the waiting cost per hour C_{wait} to 10, the cost of not covering an *order* C_{omit} to 100,000 and the layover costs $C_{layover}$ to 100.

The kilometer tariff for trailer type Heavy Type 5 is decided to be 8. All kilometers tariffs for the remaining trailer types are calculated according to the relative capacities. For example, the kilometer tariff for Normal Type 1 is calculated as $8 * \frac{1}{2} * (\frac{20{,}500}{201{,}000} + \frac{9}{21}) = 2.12$. Similarly, the kilometer tariff for Truck Type 5 is set to 2, while the kilometer tariffs for the remaining truck types are calculated using the weight capacity relative to Truck Type 5.

5.1.7 Remaining Characteristics

In this last subsection, we discuss two settings that are neither tangible instance attributes nor actual model parameters. First, parameter λ controlling the length of a daily rest period in constraint (26) in the MIP is set to 1.3, which concretely means that a daily rest period can last at most $1.3 * 11 = 14.3$ hours. Second, splitting interval K for the creation of artificial nodes is decided to be 3. This implies that each arc between any two *activities* is split into parts with travel times of at most 3 hours.

5.2 Model Parameters

The proposed TSAMP contains a plethora of parameters, which can be configured in order to improve the performance of the algorithm. An overview of all parameters is given in Table 8. Tuning all parameters in Table 8 is a time-consuming and cumbersome process. Therefore, several parameters that we believe to have less effect on the performance than other parameters, are fixed using educated guesses or support from the literature. Section 5.2.1 discusses the setting of those parameters. In Section 5.2.2, we discuss the tuning procedure for the parameters we believe are most important to the performance of the TSAMP.

Parameter	Description
α_n	Controls the distance relatedness
γ_n	Controls the time relatedness
θ^{weight}	Controls the weight relatedness
θ^{LDM}	Controls the LDM relatedness
λ^D	Controls the relatedness of allowed drivers
λ^T	Controls the relatedness of allowed trailers
$TSAMP^{iter}$	Number of iterations in the TSAMP (Algorithm 1)
$GRASP^{iter}$	The number of iterations in the GRASP heuristic (Algorithm 2)
M^{size}	Maximum size of memory M
κ	Every κ^{th} TSAMP iteration, some randomly selected *orders* are put on hold on purpose (Algorithm 3)
$L_{max}^{uncover}$	Upper bound on the number of *orders* that are left out of consideration (Algorithm 3)
τ	Controls the weight of the time span of a *trip* in the alternative cost measure (Equation 2)
SA^{iter}	Number of iterations in the SA heuristic (Algorithm 4)
ψ	Cooling parameter in the SA heuristic (Algorithm 4)
T_0	Initial temperature in the SA heuristic (Algorithm 4)
N^{driver}	Number of attempts to assign a driver to a certain *trip* (Algorithm 5)
N^{cancel}	Number of cancellations in the truck-trailer assignment heuristic (Algorithm 9)

Table 8: Overview of all parameters used in the Two-Stage Adaptive Memory Procedure.

5.2.1 Parameter Settings

All values of parameters corresponding to the relatedness function are adopted from Ropke and Pisinger (2006), which results in $\{\alpha_n, \gamma_n, \theta^{weight}, \theta^{LDM}, \lambda^D, \lambda^T\} \leftarrow \{\frac{9}{n-1}, \frac{3}{n-1}, 2, 2, 5, 5\}$. As explained in Section 4.1.2, subscript n is added to account for the different types of *orders*. When two *orders* consist of three *activities* in total, we have that $\alpha_n = \alpha_3 = 9/2 = 4.5$ and $\gamma_n = \gamma_3 = 3/2 = 1.5$.

The value of parameter $GRASP^{iter}$, the number of iterations in the GRASP algorithm, can be interpreted as the number of initial solutions used to initialize memory M. The value of this parameter is also adopted from Olivera and Viera (2007) and thus set to 20.

Parameters κ and $L_{max}^{uncover}$ are respectively set to 10 and $0.2 * |\mathcal{O}|$, which means that every tenth TSAMP iteration at most 20% of the *orders* are discarded.

For a given instance, τ is obtained by adding the average truck kilometer tariff and the average trailer kilometer tariff and multiplying this number by six. This concretely means that one time unit increase in time span is as heavily penalized as the average increase in assignment cost culminating from six additional

kilometers. This multiplication with six is added such that one additional kilometer is as heavily penalized a $60/6 = 10$ minute increase in time span.

In most SA frameworks in the literature, cooling parameter ψ takes on values in range $[0.9, 0.99]$. In this research, the value of ψ is set to 0.95. Initial temperature T_0 is set such that in the first iteration of the SA algorithm a 5% worse solution is accepted with a probability of 5%. To illustrate, suppose that after Step 2 in the SA framework in Algorithm 4, the objective in terms of the alternative measure amounts to 10,000. Then we want to find T_0 such that $0.05 = \exp(\frac{10.000-10.500}{T_0}) = \exp(\frac{-500}{T_0})$, which can easily be obtained by taking logarithms on both sides.

Parameters N^{driver} and N^{cancel} respectively controlling the driver assignment attempts and the number of cancellations are both set to 20.

5.2.2 Parameter Tuning

$TSAMP^{iter}$, M^{size} and SA^{iter} are believed to be the three most important parameters. Hence, a grid search is performed over three pre-selected lists of values: $TSAMP^{iter}$ & $SA^{iter} \in \{100, 200, 500\}$ and $M^{size} = n|\mathcal{O}|$ with $n \in \{2, 4, 10\}$. The first two parameters are selected from a static list, whereas M^{size} depends on the instance size. Larger instances generally contain more *locally feasible trips*, hence it seems rational to let the maximum memory size depend on the instance size. Each parameter setting is evaluated using six benchmark instances from Section 5.1. The selected instances are n10-1, n10-2, n50-1, n50-2, n100-1 and n100-2.

For each setting of $TSAMP^{iter}$ and SA^{iter}, we found that increasing the maximum memory size leads to an increase in the average objective value. Hence, M^{size} is fixed to $2 * |\mathcal{O}|$. A possible explanation for this result is that larger memories can potentially contain many *trips* corresponding to bad solutions and only a few promising *trips*. Hence the probability of selecting bad *trips* in each TSAMP becomes considerably large, when the maximum memory size is too large.

The tuning results for the remaining parameter settings are displayed in Table 9. The first column exhibits the considered parameter settings. The next three columns show the average objective, the average TSAMP iteration of the best found solution and the average running time in seconds for the six tuning instances. In the last three columns only instances n50-1, n50-2, n100-1 and n100-2 are considered, since instance n10-1 heavily influences the results. For this instance, the first parameter setting $[100, 100]$ finds an excellent solution with only *order* on hold, whereas most other parameter settings do not find such a solution. This leads to excessive penalty costs for the other parameter settings and thus to the suggestion that parameter setting $[100, 100]$ outperforms the other settings, while paradoxically considering less iterations. The randomness of the TSAMP, or even the chosen seed, could be the main culprit for this inconsistent result. Especially for small instances, we believe that the impact of randomness can be large. To this extent, the tuning results excluding the small instances are considered as the main tuning results.

The reduction in objective across the different parameter settings is marginal compared to the increase in computation time. In set 2, the objective reduction between the first and last parameter setting is 8.90%, whereas the increase in computation time is 163.58%. Depending on the available time and the preferences of the planner, it could be argued whether this objective reduction is worth the additional computation time.

The average iteration, at which the best objective is found, is 56.15% of $TSAMP^{iter}$ for set 1 and 78.82% for set 2, from which we can conclude that the best objective for small instances is found early in the algorithm, while for the larger instances it is beneficial to extend the number of iterations.

In this research, we assume that the objective is of primary interest and that the computation time, however still important, is of secondary interest. To that extent, $[TSAMP^{iter}, SA^{iter}] = [200, 500]$ is decided to be the best parameter setting. The reported average objective is the third best obtained objective in set 2 and approximately 4% worse than the best average objective in set 2 (last parameter setting). The decisive factor is however the computation time. Namely, the 4% objective improvement is obtained at the cost of 68.93% additional running time indicating that parameter setting $[200, 500]$ yields the best objective-running time tradeoff.

	Set 1: Including n10-1 and n10-2			Set 2: Excluding n10-1 and n10-2		
$[TSAMP^{iter}, SA^{iter}]$	Objective	Found	Time (s)	Objective	Found	Time (s)
$[100, 100]$	117,375.18	52.67	102	114,767.27	75.75	151
$[100, 200]$	147,355.88	54.00	107	114,664.78	70.00	158
$[100, 500]$	179,422.61	57.67	115	111,580.50	85.00	169
$[200, 100]$	130,525.17	137.00	135	113,597.70	159.00	199
$[200, 200]$	146,495.69	123.83	139	112,263.94	149.00	200
$[200, 500]$	127,632.92	142.00	162	108,623.10	184.75	237
$[500, 100]$	111,966.14	290.67	224	108,995.53	433.75	324
$[500, 200]$	144,016.92	248.50	236	107,740.66	372.75	343
$[500, 500]$	125,527.67	241.33	277	104,548.05	361.75	398

Table 9: Parameter tuning results. The middle block of columns considers all tuning instances, whereas the left block of columns neglect the smallest two tuning instances.

5.3 Computational Results

The computational results obtained by the CPLEX solver, GRASP algorithm and TSAMP algorithm are presented in this section. Section 5.3.1 elaborates on the performance of the CPLEX solver. In Sections 5.3.2 and 5.3.3, we respectively discuss the computational results of the GRASP algorithm and the TSAMP algorithm.

5.3.1 Mixed Integer Program

The MIP defined by constraints (3) - (53) and objective (54) is used as a benchmark for the TSAMP. To limit the computation time utilized by CPLEX, we allow 3,600 seconds of computation time per instance. The results are shown in Table 10. Column 'UB' contains the best found solution by CPLEX, whereas column 'LB' contains the best found lower bound on the optimal solution. By default, the CPLEX solver executes a Branch-and-Cut algorithm for Mixed Integer Programs. This algorithm obtains upper bounds and lower bounds on the optimal solution and terminates when these two bounds coincide. When the algorithm is

terminated prematurely after 3,600 seconds, the bounds might not have coincided. This indicates that it is not definite whether the CPLEX solver found the optimal solution.

Columns 'On Hold' and 'Trips' respectively show the number of uncovered *orders* and the number of *trips* of the best obtained solution. The computation time is reported in seconds and the optimality gap in 'Gap %' is calculated with respect to the lower bound: $\frac{UB-LB}{LB}$

CPLEX is not able to find feasible solutions or lower bounds for instances with *activity* size 50 and 100. In some cases, the solver couldn't find a feasible solution within 3,600 seconds, while in some other cases the solver reports an out-of-memory error before the specified time limit. While it is unfortunate that we cannot compare the performance of the TSAMP to CPLEX for larger instances, this outcome stresses the importance of research into heuristical approaches for the RPP.

Within the predefined time limit, CPLEX finds optimal solutions for four instances with *activity* size ten, three instances are solved to an optimality gap of less than 1% and three instances report larger gaps from varying from 12% to 75%. The average optimality gap obtained by CPLEX amounts to 12% and the average solving time amounts to 2202.4 seconds. This average solving is however quite misleading. CPLEX struggles to close the gap for six instances and utilizes all available computation time, while for three other instances the optimal solution is found within one percent of the predefined time limit.

Instance	CPLEX					
Name	UB	LB	On Hold	Trips	Time (s)	Gap %
n10-1	217,425.26	123,923.60	2	3	3,600	75.45
n10-2	119,667.39	119,667.39	1	3	35	0.00
n10-3	15,559.71	13,846.43	0	6	3,600	12.37
n10-4	112,958.23	112,848.23	1	4	3,600	0.10
n10-5	104,660.73	104,660.73	1	4	29	0.00
n10-6	105,851.39	105,851.39	1	4	327	0.00
n10-7	7,874.18	7,874.18	0	5	33	0.00
n10-8	13,007.16	12,973.23	0	4	3,600	0.26
n10-9	22,222.44	16,085.06	0	6	3,600	38.16
n10-10	208,447.23	208,316.96	2	2	3,600	0.06

Table 10: Computational results for all instances in n10 obtained by CPLEX.

5.3.2 GRASP Algorithm

The solution obtained by the GRASP algorithm is used as a second benchmark. Recall that memory M is initialized with $GRASP^{iter}$ initial solutions in Algorithm 2. From these $GRASP^{iter}$ solutions, the best RPP solution is selected as benchmark. The computational results are shown in the middle block of columns in Table 11. For each instance, we report the objective, the number of uncovered *orders*, the number of *trips*, the computation time in seconds and the optimality gap with with respect to the lower bound obtained by CPLEX. Clearly, these gaps can only be calculated for instances in n10.

The obtained optimality gaps for instances in n10 vary substantially from 0.32% to 94.04% with an average gap of 45.29%. The obtained solutions are on average 34.38% worse than the objective found by CPLEX, which indicates that CPLEX outperforms the GRASP algorithm in terms of objective value. This relatively large performance disparity is mainly caused by instances n10-4 and n10-6, where the GRASP algorithm puts one *order* more on hold then the CPLEX solution. The average GRASP computation time of 1.7 seconds is however significantly less than the CPLEX solver. For instance n10-10 one could even argue that the overall performance of the GRASP algorithm is better than CPLEX. The objective value is only 0.25% worse, while the GRASP computation time is 99.94% less than the CPLEX solver computation time.

For larger instances, the running times rapidly increase in both relative and absolute terms. The average running times for the instances in n50 and n100 respectively amount to 48.3 seconds and 165.8 seconds, which is quite substantial given the fact that most construction heuristics are extremely fast. The reported running times for the TSAMP algorithm include the GRASP running time, since this is used to initialize the memory. For the instances in n10 the fraction of running time utilized by the GRASP algorithm with respect to the total TSAMP running time is 9.34% on average. For instances n50 and n100, this fraction respectively amounts to 43.88% and 47.85%. The main culprit for this increased running times could be Step 6 in Algorithm 2. For each potential *trip* merge, it is validated whether the resulting merged *trip* is *locally feasible* (Steps 4 - 15 in Algorithm 11), which is relatively time-consuming. The number of calls to this procedure in Algorithm 2 is quadratic in the instance size, since we iterate over all *order* pairs in list RL. Thus, the time utilized by this GRASP algorithm increases quadratically, while the remainder of the TSAMP is largely limited by static hyperparameters.

5.3.3 Two-Stage Adaptive Memory Procedure

The computational results for the TSAMP are presented in the left block in Table 11. We will first compare the performance of the TSAMP to respectively the CPLEX solver and the GRASP algorithm. Second, several observations regarding the behaviour of the TSAMP are discussed.

Similar to the GRASP algorithm, the optimality gaps for the instances in n10 show large variation, namely from 0.27% to 84.84%. The average optimality gap is 27.52%, which is an substantial improvement compared to the 45.29% of the GRASP algorithm. On average, the objective found by the TSAMP is 14.11% worse than the objective found by the CPLEX solver. Only for instance n10-4, the TSAMP is not able to cover the same number of *orders* as CPLEX. Without this instance, the TSAMP performs only 6.27% worse in terms of objective value. The TSAMP has an average computation time of 18.1 seconds, which is a 99.18% reduction compared to the time utilized by the CPLEX solver. When the available time is limited, the average difference of 14.11% with respect to the best found CPLEX objective is acceptable and the TSAMP is favored instead of the CPLEX solver. When planners, for example, generate next week's planning during the weekend and the available time is less limited, the CPLEX solver outperforms the TSAMP for small instances. However, in general, the TSAMP surpasses the CPLEX solver. The TSAMP provides solutions for all instances, whereas the CPLEX solver fails to produce feasible solutions for all instances with size 50 and 100.

For half of the instances in n10, the TSAMP does not improve the initial GRASP solution. This indicates

that for small instances, the GRASP algorithm is already capable of finding high-quality solutions. On the other hand, the TSAMP finds solutions covering more *orders* for instances n10-2 and n10-6 indicating that extending the algorithm after the initialization may be beneficial. All in all, the average improvement for the instances in n10 with respect to the GRASP algorithm is 11.43%, while the average running time increase is 1108.33%.

For the larger instances, the performance of the TSAMP compared to the GRASP algorithm has improved tremendously. The average improvement in objective is 39.52% (38.08%) for the instances in n50 (n100), while the average increase in running time amounts to only 137.72% (114.23%). In contrary to the instances in n10, the TSAMP finds improvements for all instances in n50 and n100. The average computation times for the instances in n50 and n100 amount to 108.9 seconds and 346.9 seconds respectively. In general, this exceeds the computation time by most state-of-the-art algorithms for related VRP problems. However, given the complexity of the RPP, the running times are within acceptable limits and contain no notable outliers. Following the findings above, we conclude that the TSAMP algorithm surpasses the GRASP algorithm in terms of overall performance. Especially for larger instances, the TSAMP significantly improves the initial GRASP solution within reasonable computation time.

Figure 4 displays the evolution of the average objective for the three different instance sets and all instances together. The red dashed line in the n10 figure reveals that for most n10 instances, only some marginal improvements are found in the early stages of the algorithm. The sudden shifts in the blue line correspond to the iterations, where the TSAMP finds a solution with increased *order* coverage for instances n10-2 and n10-6. The n50 and n100 figures show a smoother objective evolution. However, for both instance sets the most significant improvements are found in first fifty iterations of the algorithm. It is however still beneficial to continue the search, since after iteration fifty the objective value still improves 5.62% (9.58%) on average for the instances in n50 (n100).

Instance	TSAMP					GRASP					CPLEX
Name	Best Obj.	On hold	Trips	Time (s)	Gap %	Best Obj.	On Hold	Trips	Time (s)	Gap %	Best Obj.
n10-1	218,995.34	2	3	18	76.72	219,772.92	2	4	2	77.35	217,425.26
n10-2	121,550.01	1	3	18	1.57	216,619.40	2	3	1	81.02	119,667.39
n10-3	17,499.88	0	5	19	26.39	21,369.67	0	5	3	54.33	15,559.71
n10-4	208,584.80	2	2	13	84.84	208,584.80	2	2	1	84.84	112,958.23
n10-5	105,081.38	1	4	22	0.40	105,081.38	1	4	2	0.40	104,660.73
n10-6	106,136.30	1	4	15	0.27	205,390.19	2	2	1	94.04	105,851.39
n10-7	8,832.86	0	5	18	12.17	8,832.86	0	5	2	12.17	7,874.18
n10-8	14,866.38	0	4	17	14.59	14,866.38	0	4	2	14.59	13,007.16
n10-9	25,400.08	0	5	21	57.91	26,350.90	0	4	1	63.82	22,222.44
n10-10	208,974.85	2	3	20	0.32	208,974.85	2	3	2	0.32	208,447.23
n50-1	46,832.87	0	32	114	-	71,514.70	0	16	55	-	-
n50-2	59,543.76	0	31	112	-	88,169.29	0	17	56	-	-
n50-3	38,255.94	0	31	95	-	53,062.07	0	24	40	-	-
n50-4	95,373.23	0	23	117	-	121,497.54	0	15	51	-	-
n50-5	46,560.44	0	37	116	-	85,501.09	0	16	61	-	-
n50-6	95,050.84	0	24	87	-	201.986.05	1	17	28	-	-
n50-7	50,058.70	0	29	126	-	71,743.07	0	21	73	-	-
n50-8	43,169.08	0	32	99	-	65,662.80	0	20	47	-	-
n50-9	21,690.40	0	23	71	-	39,778.08	0	11	23	-	-
n50-10	78,699.93	0	24	152	-	266,058.16	2	12	49	-	-
n100-1	109,459.25	0	67	379	-	153,029.87	0	31	184	-	-
n100-2	220,462.44	1	61	366	-	557,998.84	4	27	197	-	-
n100-3	212,002.91	0	46	288	-	292,518.11	1	29	134	-	-
n100-4	175,094.57	0	47	284	-	592,639.56	4	30	165	-	-
n100-5	58,753.47	0	59	313	-	96,685.41	0	25	128	-	-
n100-6	212,289.86	0	55	449	-	439,353.29	2	31	222	-	-
n100-7	417,159.99	2	47	351	-	540,747.74	3	27	167	-	-
n100-8	181,147.11	1	61	285	-	243,978.77	1	31	122	-	-
n100-9	267,276.38	0	55	363	-	370,629.14	1	33	208	-	-
n100-10	335,820.48	1	44	391	-	456,888.26	2	34	131	-	-

Table 11: Computational results for the TSAMP and GRASP algorithms.

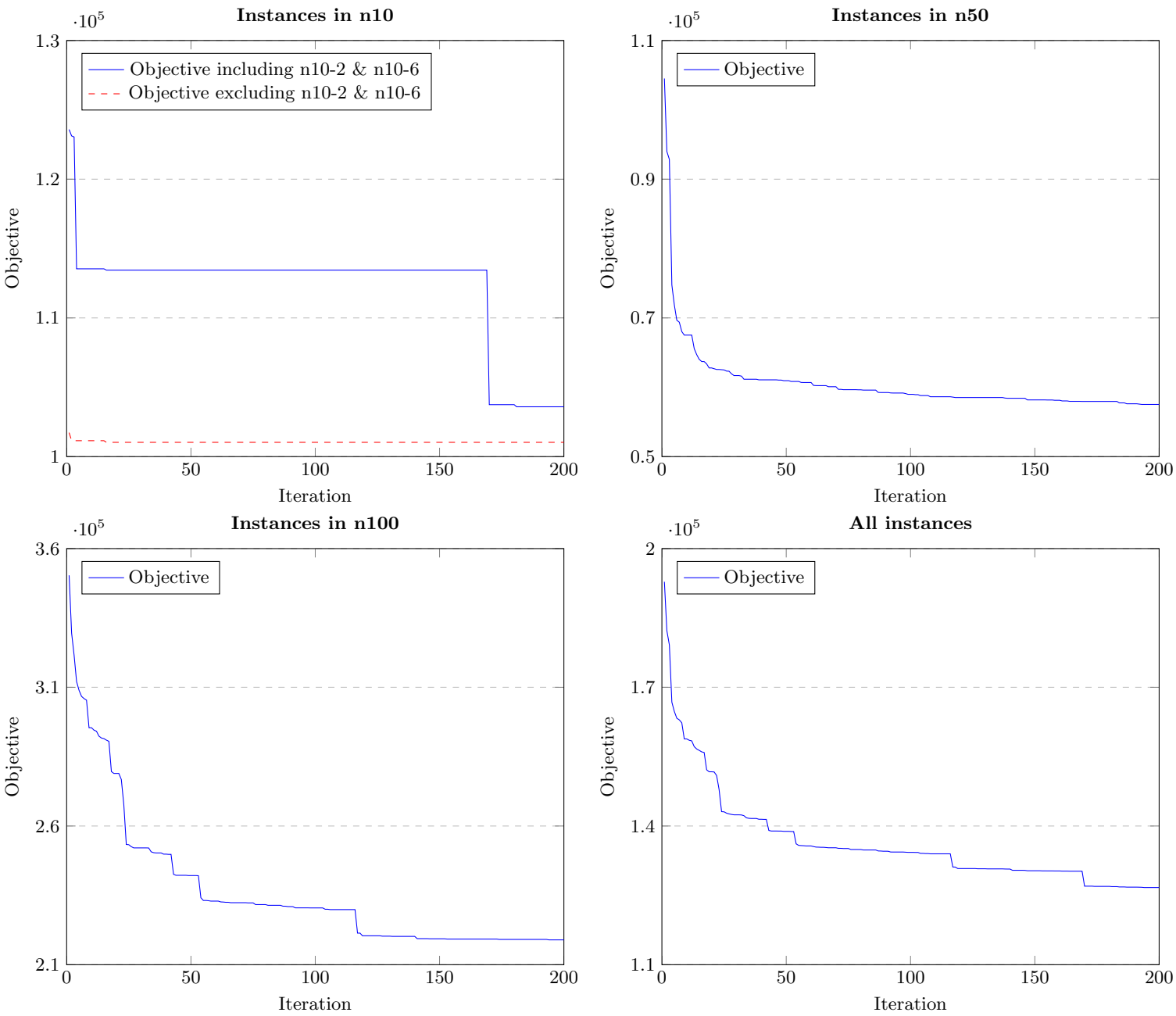

Figure 4: The average objective development over the TSAMP iterations for four different sets of instances: n10, n50, n100 and all instances.

6 Conclusion

The purpose of this research is to investigate the Resource Planning Problem (RPP): an extensive variant of the Vehicle Routing Problem (VRP) considering vehicle capacities, time windows, pickups and deliveries, allowance of multiple *trips* per resource, a heterogeneous fleet and drivers' regulations. A Mixed Integer Program (MIP) formulation capturing all aspects is presented and explained. Due to the NP-hardness of the RPP and the excessive number of constraints in the MIP formulation, the main focus is on developing an efficient heuristic. To that extent, a Two-Stage Adaptive Memory Procedure (TSAMP) is presented. The algorithm splits the problem into a *trip* construction part and a resource assignment part. The first part probabilistically selects *trips* from a memory, which is used to store *trips* from different RPP solution. The second part first assigns drivers to the selected *trips* and then iteratively assigns trucks and trailers in a greedy way. Intermediate solutions are improved using a Simulated Annealing algorithm.

The performance of the TSAMP is compared to the performance of an iterated randomized construction heuristic (GRASP) and special purpose solver CPLEX, where the CPLEX solver is given a limit of 3,600 seconds. For small instances with only ten nodes, the TSAMP performs moderately. On average, the TSAMP performs almost 15% worse then the CPLEX solver. Furthermore, the TSAMP is not able to improve the initial GRASP solution for half of the small instances, while the GRASP algorithm is more than ten times faster on average. However, for larger instances containing 50 and 100 nodes, the TSAMP significantly outperforms the GRASP algorithm. For these instances, the TSAMP improves the GRASP solution with almost 40% on average, while the average increase in computation time is limited to 140%. The CPLEX solver is not able to produce solutions for larger instances, since out-of-memory errors are raised before the time limit passes. The TSAMP can thus unfortunately not be compared to the CPLEX solver. However, this incapability of CPLEX emphasizes the relevance of research into heuristical approaches for the RPP.

The resulting TSAMP is a reliable and robust algorithm able to generate adequate solutions within several minutes for instances up to 100 nodes. The algorithm setup is highly flexible. By redefining, extending or limiting the definition of a *locally feasible trip*, multiple VRP variants can be solved using this technique. Furthermore, the driver assignment part can easily be disabled such that the problem simplifies to routing and solely assigning vehicle resources. Many transport planners facing various types of restrictions can thus adopt this TSAMP algorithm and extend or adjust it to their specific needs.

For future research, several aspects can be investigated further. First, the constraints regarding driver's regulations complicate the problem significantly. The used driver assignment heuristic in the TSAMP is extremely ad-hoc, whereas a mathematical programming based heuristic, such as Lagrangian Relaxation, might be more suitable. When researchers are not limited to a certain programming language, the use of such heuristics can be examined further. Second, the steep decrease of the average objective in the first quarter of the algorithm (see Figure 4) might indicate that the current TSAMP search is intensified too premature. Developing a new mechanism for selecting *trips* from the memory in a more diverse way has the potential to resolve this issue. Another possibility is to extend the Local Search phase with a Large Neighborhood Search, where the solution is improved by iteratively destroying an repairing the solution. Lastly, the MIP

formulation could be reviewed. The used Set Partitioning formulation is suitable for capturing all constraints in a relatively concise way. However, the number of constraints and variables is extremely large. With some thorough modelling, the formulation could be rewritten with, for example, arc flow variables instead of *trip* selection variables. Furthermore, the use of artificial nodes could be revised such that daily rest periods could be taken continuously on the interval between two *activity* nodes.

References

Adaption. (n.d.). *Logistics cloud suite*. Retrieved January 24, 2022, from https://www.adaption-it.nl/logistics-cloud-suite/

Archetti, C., & Savelsbergh, M. (2009). The Trip Scheduling Problem. *Transportation Science*, *43*(4), 417–431. https://doi.org/10.1287/trsc.1090.0278

Arnold, F., & Sorensen, K. (2019). Knowledge-guided local search for the vehicle routing problem. *Computers & Operations Research*, *105*, 32–46. https://doi.org/10.1016/j.cor.2019.01.002

Baker, B., & Ayechew, M. (2003). A genetic algorithm for the vehicle routing problem. *Computers & Operations Research*, *30*(5), 787–800. https://doi.org/10.1016/S0305-0548(02)00051-5

Baldacci, R., Hadjiconstantinou, E., & Mingozzi, A. (2004). An exact algorithm for the capacitated vehicle routing problem based on a two-commodity network flow formulation. *Operations Research*, *52*(5), 723–738. https://doi.org/10.1287/opre.1040.0111

Balinski, M. L., & Quandt, R. E. (1964). On an integer program for a delivery problem. *Operations Research*, *12*(2), 300–304. https://doi.org/10.1287/opre.12.2.300

Bell, J., & McMullen, P. (2004). Ant colony optimization techniques for the vehicle routing problem. *Advanced Engineering Informatics*, *18*(1), 41–48. https://doi.org/10.1016/j.aei.2004.07.001

Braekers, K., Ramaekers, K., & Van Nieuwenhuyse, I. (2016). The vehicle routing problem: State of the art classification and review. *Computers & Industrial Engineering*, *99*, 300–313. https://doi.org/10.1016/j.cie.2015.12.007

Braysy, I., & Gendreau, M. (2005-a). Vehicle routing problem with time windows, part 1: Route construction and local search algorithms. *Transportation Science*, *39*(1), 104–118. https://doi.org/10.1287/trsc.1030.0056

Braysy, I., & Gendreau, M. (2005-b). Vehicle routing problem with time windows, part II: Metaheuristics. *Transportation Science*, *39*(1), 119–139. https://doi.org/10.1287/trsc.1030.0057

Cattaruzza, D., Absi, N., & Feillet, D. (2016). Vehicle routing problems with multiple trips. *4OR-A Quarterly Journal of Operations Research*, *14*(3), 223–259. https://doi.org/10.1007/s10288-016-0306-2

Cattaruzza, D., Absi, N., Feillet, D., & Vidal, T. (2014). A memetic algorithm for the Multi Trip Vehicle Routing Problem. *European Journal of Operational Research*, *236*(3, SI), 833–848. https://doi.org/10.1016/j.ejor.2013.06.012

Chen, J., & Wu, T. (2006). Vehicle routing problem with simultaneous deliveries and pickups. *Journal of the Operational Research Society*, *57*(5), 579–587. https://doi.org/10.1057/palgrave.jors.2602028

Choi, E., & Tcha, D.-W. (2007). A column generation approach to the heterogeneous fleet vehicle routing problem. *Computers& Operations Research*, *34*(7), 2080–2095. https://doi.org/10.1016/j.cor.2005.08.002

Christofides, N., Mingozzi, A., & Toth, P. (1981). State-Space Relaxation Procedures for the Computation of Bounds to Routing-Problems. *Networks*, *11*(2), 145–164. https://doi.org/10.1002/net.3230110207

Clarke, G., & Wright, J. W. (1964). Scheduling of vehicles from a central depot to a number of delivery points. *Operations Research*, *12*(4), 568–581. https://doi.org/10.1287/opre.12.4.568

Cordeau, J., Laporte, G., & Mercier, A. (2001). A unified tabu search heuristic for vehicle routing problems with time windows [Annual Conference of the Operational-Research-Society, SWANSEA, WALES, SEP 12-14, 2000]. *Journal of the Operational Research Society*, *52*(8), 928–936. https://doi.org/10.1057/palgrave.jors.2601163

Dantzig, G. B., & Ramser, J. H. (1959). The truck dispatching problem. *Management Science*, *6*(1), 80–91. https://doi.org/10.1287/mnsc.6.1.80

European Transport Safety Council. (2001). *The role of driver fatigue in commercial road transport crashes.*

European Union. (2006). *Regulation (EC) No 561/2006 of the European Parliament and of the Council of 15 march 2006 on the harmonisation of certain social legislation relating to road transport and amending Council Regulations (EEC) No 3821/85 and (EC) No 2135/98 and repealing Council Regulation (EEC) no 3820/85.*

Figliozzi, M. A. (2010). An iterative route construction and improvement algorithm for the vehicle routing problem with soft time windows [10th AATT Conference, Athens, GREECE, MAY, 2008]. *Transportation Research Part C-Emerging Technologies*, *18*(5, SI), 668–679. https://doi.org/10.1016/j.trc.2009.08.005

Fleischmann, B. (1990). The vehicle routing problem with multiple use of vehicles.

Gendreau, M., Hertz, A., & Laporte, G. (1994). A Tabu Search Heuristic for the Vehicle-Routing Problem. *Management Science*, *40*(10), 1276–1290. https://doi.org/10.1287/mnsc.40.10.1276

Gendreau, M., Laporte, G., Musaraganyi, C., & Taillard, E. (1999). A tabu search heuristic for the heterogeneous fleet vehicle routing problem. *Computers & Operations Research*, *26*(12), 1153–1173. https://doi.org/10.1016/S0305-0548(98)00100-2

Goel, A. (2012). The minimum duration truck driver scheduling problem. *Euro Journal on Transportation and Logistics*, *1*(4), 285–306. https://doi.org/10.1007/s13676-012-0014-9

Goel, A. (2010). Truck Driver Scheduling in the European Union. *Transportation Science*, *44*(4), 429–441. https://doi.org/10.1287/trsc.1100.0330

Goel, A. (2009). Vehicle Scheduling and Routing with Drivers' Working Hours. *Transportation Science*, *43*(1), 17–26. https://doi.org/10.1287/trsc.1070.0226

Homberger, J., & Gehring, H. (1999). Two evolutionary metaheuristics for the vehicle routing problem with time windows. *Infor*, *37*(3), 297–318.

Koc, C., & Karaoglan, I. (2011). A branch and cut algorithm for the vehicle routing problem with multiple use of vehicles. *41st International Conference on Computers Industrial Engineering*, 554–559.

Kok, A. L., Meyer, C. M., Kopfer, H., & Schutten, J. M. J. (2010). A Dynamic Programming Heuristic for the Vehicle Routing Problem with Time Windows and European Community Social Legislation. *TRANSPORTATION SCIENCE*, *44*(4), 442–454. https://doi.org/{10.1287/trsc.1100.0331}

Laporte, G., & Nobert, Y. (1983). A Branch and Bound Algorithm for the Capacitated Vehicle-Routing Problem. *OR Spektrum*, *5*(2), 77–85. https://doi.org/10.1007/BF01720015

Laporte, G. (2009). Fifty Years of Vehicle Routing. *Transportation Science*, *43*(4), 408–416. https://doi.org/10.1287/trsc.1090.0301

Laporte, G. (2007). What you should know about the vehicle routing problem. *Naval Research Logistics*, *54*(8), 811–819. https://doi.org/10.1002/nav.20261

Lin, C. K. Y. (2008). A cooperative strategy for a vehicle routing problem with pickup and delivery time windows. *COMPUTERS & INDUSTRIAL ENGINEERING*, *55*(4), 766–782. https://doi.org/{10.1016/j.cie.2008.03.001}

Mingozzi, A., Roberti, R., & Toth, P. (2013). An exact algorithm for the multitrip vehicle routing problem. *INFORMS Journal on Computing*, *25*, 193–207. https://doi.org/10.1287/ijoc.1110.0495

Nagy, G., & Salhi, S. (2005). Heuristic algorithms for single and multiple depot Vehicle Routing Problems with Pickups and Deliveries [14th Annual Meeting of the European Chapter on Combinatorial Optimization (ECCO 14), Univ Bonn, Bonn, GERMANY, MAY 31, 2001-MAY 06, 2003]. *European Journal of Operational Research*, *162*(1), 126–141. https://doi.org/10.1016/j.ejor.2002.11.003

Olivera, A., & Viera, O. (2007). Adaptive memory programming for the vehicle routing problem with multiple trips. *Computers & Operations Research*, *34*(1), 28–47. https://doi.org/10.1016/j.cor.2005.02.044

Prins, C. (2004). A simple and effective evolutionary algorithm for the vehicle routing problem. *Computers & Operations Research*, *31*(12), 1985–2002. https://doi.org/10.1016/S0305-0548(03)00158-8

Prins, C. (2009). Two memetic algorithms for heterogeneous fleet vehicle routing problems. *Engineering Applications of Artificial Iintelligence*, *22*(6, SI), 916–928. https://doi.org/10.1016/j.engappai.2008.10.006

Rochat, Y., & Taillard, É. (1995). Taillard, e.d.: Probabilistic diversification and intensification in local search for vehicle routing. journal of heuristics 1(1), 147-167. *Journal of Heuristics*, *1*, 147–167. https://doi.org/10.1007/BF02430370

Ropke, S., & Pisinger, D. (2006). An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation Science*, *40*(4), 455–472. https://doi.org/10.1287/trsc.1050.0135

Solomon, M. (1987). Algorithms for the Vehicle-Routing and Scheduling Problems with Time Window Constraints. *Operations Research*, *35*(2), 254–265. https://doi.org/{10.1287/opre.35.2.254}

Terrazas, A. (2019). *What you need to know about empty miles in trucking.* Retrieved January 21, 2022, from https://convoy.com/blog/empty-miles-in-trucking/

Toth, P., & Vigo, D. (2002). Models, relaxations and exact approaches for the capacitated vehicle routing problem [Workshop on Discrete Optimization (DO 99), Rutgers Univ CTR Operat Res, New Brunswick, NJ, 1999]. *Discrete Applied Mathematics*, *123*(1-3), 487–512. https://doi.org/10.1016/S0166-218X(01)00351-1

Uchoa, E., Pecin, D., Pessoa, A., Poggi, M., Vidal, T., & Subramanian, A. (2017). New benchmark instances for the Capacitated Vehicle Routing Problem. *EUROPEAN JOURNAL OF OPERATIONAL RESEARCH*, *257*(3), 845–858. https://doi.org/{10.1016/j.ejor.2016.08.012}

Wang, Z., Liang, W., & Hu, X. (2014). A metaheuristic based on a pool of routes for the vehicle routing problem with multiple trips and time windows. *JOURNAL OF THE OPERATIONAL RESEARCH SOCIETY*, *65*(1), 37–48. https://doi.org/{10.1057/jors.2013.4}

Xu, H., Chen, Z., Rajagopal, S., & Arunapuram, S. (2003). Solving a practical pickup and delivery problem. *Transportatione Science*, *37*(3), 347–364. https://doi.org/10.1287/trsc.37.3.347.16044

Appendices

A Mathematical Formulation

Capturing both the fourfold objective and the complex drivers' constraints using an arc flow formulation would probably result in a cumbersome use of notation. A Set Partitioning type of formulation is, despite its exponential number of variables, more suitable for modelling all RPP aspects. For the sake of readability, we present the MIP formulation into several related parts and elaborate on the constraints one by one. First, constraints regarding vehicle resources and time variables are presented. Second, constraints regarding drivers and drivers' regulations are discussed. Finally, the objective of the MIP is shown.

A.1 Vehicle Resource Constraints

Binary variables $u_v \in \mathbb{B}$ reflect the inclusion of a particular *trip* $v \in \mathcal{V}$ in the solution, where $\mathcal{V}$ is the set of *locally feasible trips*. These *trips* satisfy the constraints described in Section 3.4.2 and are generated in a preoptimization stage. Pseudocode and explanation of the *trip* construction process are given in the appendix in Section B.3.

Each *order* o should either be covered by exactly one *trip* or put on hold, after which outsourcing can be used to deal with the uncovered *order*. Let $\mathcal{V}_o$ be the set of *trips* that cover *order* o and let $\gamma_o \in \mathbb{B}$ be a binary decision variable indicating whether *order* o is put on hold or not. The constraints in (3) reflect the *order* coverage constraints.

$$\sum_{v \in \mathcal{V}_o} u_v + \gamma_o = 1 \qquad \forall\ o \in \mathcal{O} \tag{3}$$

Every *trip* v needs to be equipped with a truck and a trailer, if and only if this *trip* is selected. Let $\mathcal{O}_v$ be the set of *orders* covered by *trip* v. The set of allowed trailers for *trip* v can then be constructed as $\mathcal{T}_v = \cap_{o \in \mathcal{O}_v} \mathcal{T}_o$. In the multi-*trip* setting, resources can be assigned to more than one *trip*. Concretely, this means that after finishing *trip* v_1, the assigned resources can be assigned to a new *trip* v_2, however, only if the start time of *trip* v_2 is after the end time of *trip* v_1. To deal with these multiple assignments we define the set of integers $\mathcal{R}_l$ ($\mathcal{R}_t$) and binary decision variables $x^r_{v,l}$ ($x^r_{v,t}$) $\in \mathbb{B}$ to indicate whether truck l (trailer t) is assigned to *trip* v using the r^{th} assignment position with $r \in \mathcal{R}_l$ ($\mathcal{R}_t$). The exact number of assigned *trips* per resource is output of the model and thus not known in advance. We will construct an upper bound on the number of assignments later in this section. The constraints regarding the assignment of vehicle resources to *trips* are given in (4) for trucks and trailers respectively.

$$\sum_{l \in \mathcal{L}} \sum_{r \in \mathcal{R}_l} x^r_{v,l} = u_v, \ \sum_{t \in \mathcal{T}_v} \sum_{r \in \mathcal{R}_t} x^r_{v,t} = u_v \qquad \forall\ v \in \mathcal{V} \tag{4}$$

Next to *trips*, vehicle resources have to be assigned to their planned external events. This may sound trivial, however it becomes important for later constraints. The constraints in (5) guarantee that exactly one assignment position r is reserved for each external event for all trucks and trailers respectively. For the sake

of conciseness, we slightly abuse notation for the vehicle assignment decision variables x and use different subscripts interchangeably (v and e).

A vehicle resource can be assigned to at most $|\mathcal{O}|$ *trips*, since this is an upper bound on the number of selected *trips* in the solution. The number of external events for truck l is known in advance and equals $|\mathcal{E}_l|$. Set R_l can thus safely be defined as $\{1, 2, ..., |\mathcal{O}| + |\mathcal{E}_l|\}$, where $|\mathcal{O}| + |\mathcal{E}_l|$ is an upper bound on the number of assignments for truck l. A similar upper bound can be obtained for trailers.

$$\sum_{r \in \mathcal{R}_l} x^r_{e,l} = 1 \ \forall \ l \in \mathcal{L}, \ e \in \mathcal{E}_l \qquad\qquad \sum_{r \in \mathcal{R}_t} x^r_{e,t} = 1 \ \forall \ t \in \mathcal{T}, \ e \in \mathcal{E}_t \tag{5}$$

To guarantee that each assignment position r can be used for at most one assignment (*trip* or external event), we include the constraints in (6) for all trucks and trailers.

$$\sum_{v \in \mathcal{V}} x^r_{v,l} + \sum_{e \in \mathcal{E}_l} x^r_{e,l} \leq 1 \ \forall \ l \in \mathcal{L}, \ r \in \mathcal{R}_l \qquad\qquad \sum_{v \in \mathcal{V}} x^r_{v,t} + \sum_{e \in \mathcal{E}_t} x^r_{e,t} \leq 1 \ \forall \ t \in \mathcal{T}, \ r \in \mathcal{R}_t \tag{6}$$

Compatibility between trucks and trailers in guaranteed in constraint set (7). Whenever the left-hand side is equal to one (trailer t is assigned to *trip* v), the constraint becomes active and a truck from compatibility set $\mathcal{L}_t$ has to be assigned to *trip* v. Together with the constraints in (4), it is ensured that an incompatible combination is never selected.

$$\sum_{r \in R_t} x^r_{v,t} \leq \sum_{l \in \mathcal{L}_t} \sum_{r \in R_l} x^r_{v,l} \qquad\qquad \forall \ v \in \mathcal{V}, \ t \in \mathcal{T}_v \tag{7}$$

We define Q^{max}_v as the maximum load weight that is carried while serving the *activities* of *trip* v. The weight capacity constraint for each truck-trailer combination is captured in (8). The constraint is only active if *trip* v is part of the solution.

$$\sum_{l \in \mathcal{L}} \sum_{r \in \mathcal{R}_l} Q_l x^r_{v,l} + \sum_{t \in \mathcal{T}_v} \sum_{r \in \mathcal{R}_t} Q_t x^r_{v,t} \geq Q^{max}_v u_v \qquad\qquad \forall \ v \in \mathcal{V} \tag{8}$$

Each *locally feasible trip* should have at least one feasible time schedule satisfying all *activity* time windows. However, the exact service start times and departure times at each *activity* or depot for a given *trip* cannot be fixed in a preprocessing phase, due to the multi-*trip* feature. It may, for example, be beneficial to start a *trip* sooner in order to be in time for the next *trip* or to create enough slack for a daily rest period during a *trip*. Thus, despite the Set-Partitioning formulation, we still need to include constraints regarding time windows.

Let $\mathcal{I}_v$ be the set of *activities* visited in *trip* v and let $\mathcal{A}_v$ be the set of arcs traversed in *trip* v. We define $t_{v,i} \geq 0$ as the start time of service at *activity* i in *trip* v and $w_{v,i} \geq 0$ as the waiting time before the start of service at *activity* i in *trip* v. Furthermore, we define $t^{startV}_v \geq 0$ and $t^{endV}_v \geq 0$ ($t^{startD}_v \geq 0$ and $t^{endD}_v \geq 0$) as the start and end time of *trip* v for the vehicle resources (drivers). The reason for distinguishing between start and end times for drivers and vehicle resources is the service time at the depot, where the driver is excluded

from. Obviously, the vehicle resources are not excluded from this service time, since they are subjected to the (un)loading process.

In constraints (9) - (11), the computation of the time variables at the depot and *activities* are modelled. In the literature (Ropke and Pisinger, 2006, Cattaruzza et al., 2016) these constraints generally appear with a '$\leq$'-sign. We use an equivalent formulation with waiting time variable $w_{v,i}$ functioning as a slack variable. In constraint set (12), the start and end times for the vehicles resources are computed. The parameter S_v^{load} (S_v^{unload}) follows from the accumulated service times of all *orders* consisting of only one delivery (pickup) *activity* and represents the service time at the depot at departure (arrival). The time window constraints are imposed in (13). When a *trip* v is not selected, (13) ensures that all time variables corresponding to this *trip* are set to zero.

$$t_v^{startD} + T_{0,j}u_v = t_{v,j} - w_{v,j} \qquad \forall\ v \in \mathcal{V},\ (0,j) \in \mathcal{A}_v \tag{9}$$

$$t_{v,i} + (S_i + T_{i,j})u_v = t_{v,j} - w_{v,j} \qquad \forall\ v \in \mathcal{V},\ (i,j) \in \mathcal{A}_v\ i,j \neq 0 \tag{10}$$

$$t_{v,i} + (S_i + T_{i,0})u_v = t_v^{endD} \qquad \forall\ v \in \mathcal{V},\ (i,0) \in \mathcal{A}_v \tag{11}$$

$$t_v^{startV} = t_v^{startD} - S_v^{load}u_v,\ t_v^{endV} = t_v^{endD} + S_v^{unload}u_v \qquad \forall\ v \in \mathcal{V} \tag{12}$$

$$E_i u_v \leq t_{v,i} \leq L_i u_v \qquad \forall\ v \in \mathcal{V},\ i \in \mathcal{I}_v \tag{13}$$

Using the computed time variables we can formulate constraints that ensure that consecutive *trips* do not overlap, when assigned to the same resource. For the sake of conciseness, we again abuse the notation of the decision variables. If j corresponds to a *trip* in $\mathcal{V}$, then t_j^{startV} is a decision variable corresponding to the start time of *trip* j for the vehicle resources, similar to the constraints in (12). However, if j corresponds to an external event in $\mathcal{E}_l$ for truck l, then t_j^{startV} is a parameter corresponding to the fixed start time of external event j. Thus, we read t_j^{startV} (and t_j^{endV}) as we need it. The constraints in (14) enforce that *trips* or external events assigned to each truck do not overlap, where M is a sufficiently large integer. The constraint is active only if j_1 and j_2 are both assigned to the same truck, with j_2 assigned at a later position than j_1. The constraints in (15) for trailers have an analogous explanation.

$$t_{j_1}^{endV} \leq t_{j_2}^{startV} + M(2 - x_{j_1,l}^r - \sum_{i=r+1}^{|R_l|} x_{j_2,l}^i) \qquad \forall\ l \in \mathcal{L},\ j_1, j_2 \in \mathcal{V} \cup \mathcal{E}_l,\ r \in \mathcal{R}_l \tag{14}$$

$$t_{j_1}^{endV} \leq t_{j_2}^{startV} + M(2 - x_{j_1,t}^r - \sum_{i=r+1}^{|R_t|} x_{j_2,t}^i) \qquad \forall\ t \in \mathcal{T},\ j_1, j_2 \in \mathcal{V} \cup \mathcal{E}_t,\ r \in \mathcal{R}_t \tag{15}$$

The RPP is usually solved sequentially for consecutive planning horizons. Therefore, *trips* starting in the previous planning horizon could overlap with the current planning horizon. Let t_l^{prev} (t_t^{prev}) be a parameter corresponding to the end time of the last assigned *trip* for truck l (trailer t). The constraints in (16) and (17) ensure that all *trips* assigned to each truck or trailer have a start time later than the end time of the last assigned *trip* from the previous planning horizon.

$$t_l^{prev} \leq t_v^{startV} + M(1 - \sum_{i=1}^{|R_l|} x_{v,l}^i) \qquad \forall\ l \in \mathcal{L},\ v \in \mathcal{V} \tag{16}$$

$$t_t^{prev} \leq t_v^{startV} + M(1 - \sum_{i=1}^{|R_t|} x_{v,t}^i) \qquad \forall\ t \in \mathcal{T},\ v \in \mathcal{V} \tag{17}$$

A.2 Drivers' Constraints

In addition to the vehicle resources, every selected *trip* needs to be equipped with a driver. Let $y_{v,d}^r \in \mathbb{B}$ be a binary variable indicating whether driver d is assigned to *trip* v as r^{th} assignment with $r \in \mathcal{R}_d$, where $\mathcal{R}_d$ is an integer set similar to sets $\mathcal{R}_l$ and $\mathcal{R}_t$. Again, we do not know the number of assignments in advance. We will construct an upper bound on $|\mathcal{R}_d|$ later in this section. The constraint set in (18) is similar to the constraint set in (4) for vehicle resources and states that each *trip* needs a driver assigned, if and only if this *trip* is selected.

$$\sum_{d \in \mathcal{D}_v} \sum_{r \in \mathcal{R}_d} y_{v,d}^r = u_v \qquad \forall\ v \in \mathcal{V} \tag{18}$$

The constraints in (19) enforce that each driver d is assigned to all its external events at exactly one assignment position r. Again, we slightly abuse notation for the binary decision variables and use subscript indices interchangeably.

$$\sum_{r \in \mathcal{R}_d} y_{e,d}^r = 1 \qquad \forall\ d \in \mathcal{D},\ e \in \mathcal{E}_d \tag{19}$$

Daily rest periods for drivers can be taken in two different ways. The first way is to take a daily rest period in between performing *trips*. In this case, the driver can return to his or her home to rest. The second way amounts to taking a daily rest period during the execution of a *trip*. In this case, costs for arranging a sleeping accommodation are incurred. In advance, we do not know the number of daily rest periods per driver exactly, since it may, for example, be beneficial to take a daily rest period sooner than necessary. We can, however, safely upper bound the number of daily rest periods by $ceil(|\mathcal{K}| * H_{day}/S_{day}^{rest})$, where H_{day} represents the number of hours (24) in a calendar day. This upper bound of daily rest periods is only met if all daily rest periods are taken directly after finishing the previous daily rest period. Incorporating more daily rest periods than necessary does not affect the solution quality, since superfluous rest periods can just be scheduled after all *trips* are performed and removed in a post-optimization stage.

Let $\mathcal{F}$ be a set of positive integers reflecting the daily rest periods and let $\rho_{d,f}$ denote the f^{th} daily rest period for driver d with $f \in \mathcal{F}$. The cardinality of $\mathcal{F}$ is fixed to the upper bound constructed in the previous paragraph: $|F| = ceil(|\mathcal{K}| * H_{day}/S_{day}^{rest})$. Let $t_{\rho_{d,f}}^{endD}$ be a fixed parameter for $f = 0$ representing the end time of the last daily rest period from the previous planning period. Lastly, we define $t_{\rho_{d,f}}^{startD} \geq 0$ and $t_{\rho_{d,f}}^{endD} \geq 0$ with $f \geq 1$ as the respective start and end times of daily rest period $\rho_{d,f}$ for driver d.

In constraint set (20), it is enforced that all daily rest periods have a duration of at least eleven hours. In (21), it is ensured that within 24 hours of the previous daily rest period a new daily rest period should be

finished. However, this constraint is formulated differently, since it states that within $24 - 11 = 13$ hours a new daily rest period should be started. This formulation is equivalent and allows rest periods to be longer than eleven hours. To illustrate, suppose that the previous daily rest period ends at $t = 50$ for driver d, then (21) enforces that the next daily rest period should start before 50 + 24 - 11 = 63. Constraint (20) now states that the end time should be at least 11 hours later, which means that $[63, 76]$ could, for example, be the new daily rest period with a duration of thirteen hours. When (21) would be formulated as $t^{endD}_{\rho_{d,f}} \leq t^{endD}_{\rho_{d,f-1}} + H_{day}$, the end time would less flexible, since the end time is now enforced to be before $50 + 24 = 74$.

Constraint set (22) is added to ensure that successive daily rest periods are taken at consecutive moments in time. Let t^{prev}_d be a parameter representing the end time of the last assigned *trip* from the previous planning horizon for driver d, then (23) ensures that the first daily rest period for driver d is taken after the end of the last assigned *trip*.

$$t^{startD}_{\rho_{d,f}} + S^{rest}_{day} \leq t^{endD}_{\rho_{d,f}} \qquad \forall\ d \in \mathcal{D},\ f = 1, ..., |\mathcal{F}| \tag{20}$$

$$t^{startD}_{\rho_{d,f}} \leq t^{endD}_{\rho_{d,f-1}} + H_{day} - S^{rest}_{day} \qquad \forall\ d \in \mathcal{D},\ f = 1, ..., |\mathcal{F}| \tag{21}$$

$$t^{startD}_{\rho_{d,f}} \geq t^{endD}_{\rho_{d,f-1}} \qquad \forall\ d \in \mathcal{D},\ f = 1, ..., |\mathcal{F}| \tag{22}$$

$$t^{startD}_{\rho_{d,1}} \geq t^{prev}_d \qquad \forall\ d \in \mathcal{D} \tag{23}$$

Similar to *trips*, drivers need to be assigned to their daily rest periods. Let $y^r_{\rho_{d,f},d} \in \mathbb{B}$ indicate whether driver d is assigned to daily rest period $\rho_{d,f}$ as r^{th} assignment. When $y^r_{\rho_{d,f},d} = 1$, driver d is explicitly assigned to rest period $\rho_{d,f}$, which is in this case taken at home between performing *trips*. Hence, no layover arrangements have to be made.

The second option is to take a daily rest period at a hotel during the execution of a *trip*, which involves layover costs. For the ease of modelling, we assume that such a daily rest period can only be taken directly after leaving an *activity* node and thus not continuously on the entire interval between two nodes. With this simplification of modelling a new problem arises. Suppose that the travel time between consecutive *activity* nodes i and $i + 1$ is 10 hours, then with the simplification, a daily rest period can only be taken directly after leaving i or directly after leaving $i + 1$ and not somewhere in the 10 hours in between, which is highly limited. To overcome this problem, we create artificial nodes between each two *activity* nodes at which daily rest periods can be taken. The procedure for creating these artificial nodes is given in Section B.2 in the appendix.

We denote $\tilde{\mathcal{I}}_v$ as the set of nodes in *trip* v extended with artificial nodes and $\tilde{\mathcal{A}}_v$ the set of arcs in *trip* v including artificial nodes. The departure times at these nodes have to be calculated in order to correctly plan daily rest periods. Hence, the time variable constraints presented in (9) - (11) are no longer correct. This is however easily fixed by replacing $\mathcal{I}_v$ by $\tilde{\mathcal{I}}_v$ and $\mathcal{A}_v$ by $\tilde{\mathcal{A}}_v$. It might seem ambiguous to rectify previously formulated constraints, however we believe that this order of constraint representation is the most logical.

Let binary variable $g^{v,i}_{\rho_{d,f},d} \in \mathbb{B}$ indicate whether driver d takes daily rest period $\rho_{d,f}$ directly after leaving node $i \in \tilde{\mathcal{I}}_v$ in *trip* v. The constraints in (24) guarantee that each daily rest period is taken, either at home and explicitly assigned or during the execution of a *trip*. In (25) it is secured that a driver can only take a

daily rest period within a *trip*, when assigned to this *trip*.

The constraints in (26) with parameter $\lambda \geq 1$ are added to put an upper bound on the length of daily rest periods. This upper bound is added for the following reason. Suppose that during a *trip*, the waiting time at a certain *activity* node is more than 50 hours. Without constraint (26), a daily rest period can be inserted before this *activity* with an start and end time such that all waiting time will be transferred to this rest period and no costs will be incurred for waiting. With a limit on the duration, multiple daily rest periods have to be inserted to reduce the waiting time and cost, however at the expense of additional layover cost incurred by multiple daily rest periods.

$$\sum_{r \in \mathcal{R}_d} y^r_{\rho_{d,f},d} + \sum_{v \in \mathcal{V}} \sum_{i \in \tilde{\mathcal{I}}_v} g^{v,i}_{\rho_{d,f},d} = 1 \qquad \forall\ d \in \mathcal{D},\ f = 1, ..., |\mathcal{F}| \tag{24}$$

$$g^{v,i}_{\rho_{d,f},d} \leq \sum_{r \in \mathcal{R}_d} y^r_{v,d} \qquad \forall\ d \in \mathcal{D},\ f = 1, ..., |\mathcal{F}|,\ v \in \mathcal{V},\ i \in \tilde{\mathcal{I}}_v \tag{25}$$

$$t^{endD}_{\rho_{d,f}} - t^{startD}_{\rho_{d,f}} \leq \lambda S^{rest}_{day} \qquad \forall\ d \in \mathcal{D},\ f = 1, ..., |\mathcal{F}| \tag{26}$$

The legislation regarding weekly rest periods is modelled similarly to the daily rest periods. However, we do not allow for weekly rest periods during the execution of a *trip*. We can safely upper bound the number of weekly rest periods by $ceil(|\mathcal{K}| * H_{day}/S^{rest}_{week})$. Let $\mathcal{G}$ be an ordered set of integers representing weekly rest periods and let $\rho_{d,g}$ correspond to the g^{th} weekly rest period for driver d with $g \in \mathcal{G}$. In this case, $y^r_{\rho_{d,g},d} \in \mathbb{B}$ represents the assignment of driver d to weekly rest period $\rho_{d,g}$ using the r^{th} assignment position, whereas $t^{startD}_{\rho_{d,g}} \geq 0$ ($t^{endD}_{\rho_{d,g}} \geq 0$) represents the start (end) time of the g^{th} weekly rest period of driver d. Similar to $t^{endD}_{\rho_{d,f}}$ with $f = 0$, $t^{endD}_{\rho_{d,g}}$ with $g = 0$ amounts to a parameter following from the previous planning period.

The weekly rest duration of S^{rest}_{week} hours is guaranteed by the constraints in (27). In constraint set (28), it is enforced that at most H^{work}_{consec} after the end of the previous weekly rest period, a new weekly rest period is started. The constraints in (29) are included to ensure that subsequent weekly rest periods are taken consecutively. Constraint set (30) for the weekly rest period is similar to (23) for daily rest periods. The last set of constraints in (31) ensures that each weekly rest period is assigned to each driver using exactly one assignment position.

$$t^{startD}_{\rho_{d,g}} + S^{rest}_{week} \leq t^{endD}_{\rho_{d,g}} \qquad \forall\ d \in \mathcal{D},\ g = 1, ..., |\mathcal{G}| \tag{27}$$

$$t^{startD}_{\rho_{d,g}} \leq t^{endD}_{\rho_{d,g-1}} + H^{work}_{consec} \qquad \forall\ d \in \mathcal{D},\ g = 1, ..., |\mathcal{G}| \tag{28}$$

$$t^{startD}_{\rho_{d,g}} \geq t^{endD}_{\rho_{d,g-1}} \qquad \forall\ d \in \mathcal{D},\ g = 1, ..., |\mathcal{G}| \tag{29}$$

$$t^{startD}_{\rho_{d,1}} \geq t^{prev}_{d} \qquad \forall\ d \in \mathcal{D} \tag{30}$$

$$\sum_{r \in \mathcal{R}_d} y^r_{\rho_{d,g},d} = 1 \qquad \forall\ d \in \mathcal{D},\ g = 1, ..., |\mathcal{G}| \tag{31}$$

We have now defined all events that can be assigned to drivers. Let $\mathcal{J}_d$ be the set of all events, excluding *trips*, that can be assigned to driver d, then $\mathcal{J}_d$ is defined as $\mathcal{E}_d \cup_{f=1}^{|\mathcal{F}|} \rho_{d,f} \cup_{g=1}^{|\mathcal{G}|} \rho_{d,g}$. Using this set an upper bound on the number of integers in $\mathcal{R}_d$ can be constructed. Namely, the set $\mathcal{R}_d$ can be defined as $\mathcal{R}_d =$

$\{1, 2, ..., |\mathcal{O}| + |\mathcal{J}_d|\}$. The constraints in (32) enforce that at most one assignment position r can be used for all types of assignments (*trip*, external event, daily rest or weekly rest) for all drivers d.

$$\sum_{v \in \mathcal{V}} y^r_{v,d} + \sum_{e \in \mathcal{E}_d} y^r_{e_i,d} + \sum_{f=1}^{|\mathcal{F}|} y^r_{\rho_{d,f},d} + \sum_{g=1}^{|\mathcal{G}|} y^r_{\rho_{d,g},d} \leq 1 \qquad \forall\ d \in \mathcal{D},\ r \in \mathcal{R}_d \tag{32}$$

Similar to the vehicle resources, we have to incorporate constraints ensuring that consecutive driver assignments do not overlap. In (33), non-overlapping constraints for consecutive *trips* are added. The constraints in (34) ensure that assignments do not overlap, when a *trip* is assigned later than a rest period or external event. Constraint set (35) is almost identical, but is enforced when a *trip* is assigned earlier than a rest period or external event. Likewise to (14) and (15), t^{startD}_j and t^{endD}_j are fixed parameters if $j \in \mathcal{E}_d$. The last constraint set (36) guarantees that all *trips* assigned to driver d have a start time later than the end time of the last assigned *trip* from the previous planning horizon.

We do not add non-overlapping constraints for consecutive assignments $j_1, j_2 \in \mathcal{J}_d$ for driver d. Those types of assignments are allowed to overlap. In this way, the model can schedule an artificial daily rest period during a weekly rest period or during an external event. Otherwise, during the S^{rest}_{week} hours of a weekly rest period, no daily rest periods could be planned, which gives an instant violation of the constraints in (21).

$$t^{endD}_v \leq t^{startD}_w + M(2 - y^r_{v,d} - \sum_{i=r+1}^{|R_d|} y^i_{w,d}) \qquad \forall\ d \in \mathcal{D},\ v, w \in \mathcal{V},\ r \in \mathcal{R}_d \tag{33}$$

$$t^{endD}_j \leq t^{startD}_w + M(2 - y^r_{j,d} - \sum_{i=r+1}^{|R_d|} y^i_{w,d}) \qquad \forall\ d \in \mathcal{D},\ w \in \mathcal{V},\ j \in \mathcal{J}_d,\ r \in \mathcal{R}_d \tag{34}$$

$$t^{endD}_v \leq t^{startD}_j + M(2 - y^r_{v,d} - \sum_{i=r+1}^{|R_d|} y^i_{j,d}) \qquad \forall\ d \in \mathcal{D},\ v \in \mathcal{V},\ j \in \mathcal{J}_d,\ r \in \mathcal{R}_d \tag{35}$$

$$t^{prev}_d \leq t^{startD}_v + M(1 - \sum_{i=1}^{|R_d|} y^i_{v,l}) \qquad \forall\ d \in \mathcal{D},\ v \in \mathcal{V} \tag{36}$$

The time variables, when taking a daily rest period during the execution of a *trip*, are computed in constraint sets (37) and (38). The constraint is only active if driver d is assigned to the corresponding *trip* v and $g^{v,i}_{\rho_{d,f},d} = 1$ meaning that a daily rest period is scheduled directly after node i in *trip* v. The constraints in (37) ensure that the start time of a daily rest period is later than the departure time from the node, after which the rest period is scheduled. The next constraints in (38) guarantee that the start time of service at the next node is later than the end time of the rest period taking into account the still remaining travel time. When active, these constraints affect the computation of the time variables in (9) - (11).

$$t_{v,i} + S_i \leq t^{startD}_{\rho_{d,f}} + M(2 - \sum_{r \in \mathcal{R}_d} y^r_{v,d} - g^{v,i}_{\rho_{d,f},d}) \qquad \forall\ d \in \mathcal{D},\ v \in \mathcal{V},\ (i,j) \in \tilde{\mathcal{A}}_v,\ f = 1, ..., |\mathcal{F}| \tag{37}$$

$$t^{endD}_{\rho_{d,f}} + T_{i,j} \leq t_{v,j} + M(2 - \sum_{r \in \mathcal{R}_d} y^r_{v,d} - g^{v,i}_{\rho_{d,f},d}) \qquad \forall\ d \in \mathcal{D},\ v \in \mathcal{V},\ (i,j) \in \tilde{\mathcal{A}}_v,\ f = 1, ..., |\mathcal{F}| \tag{38}$$

In Section 3.4.4, we assumed that for the calculation of the total driving time per day, the total driving time between two nodes i and $i+1$ is incurred at the day of the departure at node i. Therefore, we have to determine the day of departure at each node. Let $h_{v,i,k} \in \mathbb{B}$ be a binary variable indicating whether *activity* node i in *trip* v is departed from at day k and let $h^{start}_{v,i,k} \in \mathbb{B}$ ($h^{end}_{v,i,k} \in \mathbb{B}$) be an auxiliary binary variable corresponding to whether *activity* i in *trip* v is departed from after the start of day k (before the end of day k).

In (39), the auxiliary variables are calculated for the departure day from the depot and in (40) the auxiliary variables are calculated for all remaining nodes. In constraint set (40) it is necessary to compute the variables using information from the next node, since $t_{v,i} + S_i$ does not capture the possibility of a daily rest period directly after node i. Suppose that $t_{v,i} + S_i = 44$, then this would suggest that the day of departure is day 2 with interval $[24, 48]$ and thus that the driving time from i to the next node should be counted for day 2. However, when a daily rest period of 11 hours is scheduled directly after node i, then the driver departs from node i at time $44 + 11 = 55$, which is on day 3. This is not captured by $t_{v,i} + S_i$.

$h_{i,v,k}$ is defined as the product of its two auxiliary variables, which is linearized in constraint set (41). It is not necessary to force this binary variable to be zero if one of the two related auxiliary variables is zero, since it is never optimal to count driving hours more than once for several days. Hence, the model can just set all binary variables to zero that are not forced to be one.

We define $\beta_{v,d,k} \geq 0$ as the number of driving hours at day k in *trip* v for driver d and compute this variable in (42). The constraint is only active when a driver d is assigned to *trip* v. The summation over all travel times on the right hand side, only considers travel times that have a departure on the considered day. The legal upper bound of H^{drive}_{day} daily driving hours with an possibility of an H^{ext}_{day} hour extension is enforced in (43). This utilized extension at day k for driver d is reflected by the binary decision variable $z_{d,k} \in \mathbb{B}$. A legal limit on these number of allowed extensions is enforced in constraint set (44), where $\mathcal{K}_w$ denotes the set of days that reside in calendar week w.

$$t^{end}_k - M h^{end}_{v,0,k} \leq t^{startD}_v \leq M h^{start}_{v,0,k} + t^{start}_k \qquad \forall\ v \in \mathcal{V},\ k \in \mathcal{K} \qquad (39)$$

$$t^{end}_k - M h^{end}_{v,i,k} \leq t_{v,j} - w_{v,j} - T_{i,j} \leq M h^{start}_{v,i,k} + t^{start}_k \qquad \forall\ v \in \mathcal{V},\ (i,j) \in \tilde{\mathcal{A}}_v,\ k \in \mathcal{K} \qquad (40)$$

$$h_{v,i,k} \geq h^{start}_{v,i,k} + h^{end}_{v,i,k} - 1 \qquad \forall\ v \in \mathcal{V},\ i \in \tilde{\mathcal{I}}_v \cup \{0\},\ k \in \mathcal{K} \qquad (41)$$

$$\beta_{v,d,k} \geq \sum_{(i,j) \in \tilde{\mathcal{A}}_v} T_{i,j} h_{v,i,k} - M\Big(1 - \sum_{r \in \mathcal{R}_d} y^r_{v,d}\Big) \qquad \forall\ v \in \mathcal{V},\ d \in \mathcal{D},\ k \in \mathcal{K} \qquad (42)$$

$$\sum_{v \in \mathcal{V}} \beta_{v,d,k} \leq H^{drive}_{day} + H^{ext}_{day} z_{d,k} \qquad \forall\ d \in \mathcal{D},\ k \in \mathcal{K} \qquad (43)$$

$$\sum_{k \in \mathcal{K}_w} z_{d,k} \leq N_{d,w} \qquad \forall\ d \in \mathcal{D},\ w \in \mathcal{W} \qquad (44)$$

Let $\delta_{v,d,w} \geq 0$ denote the driving time for driver d in *trip* v in week w. This variable is computed in (45), which is similarly to the computation of $\beta_{v,d,k}$ in (43). However, the summation on the right-hand side in (45) includes multiple days instead of only one day. The constraints in (46) ensure that the legal maximum of driving hours per calendar week is not exceeded. The legal maximum driving hours per two calendar weeks is

enforced in constraint set (47).

$$\delta_{v,d,w} \geq \sum_{k \in \mathcal{K}_w} \sum_{(i,j) \in \tilde{\mathcal{A}}_v} T_{i,j} h_{v,i,k} - M(1 - \sum_{r \in \mathcal{R}_d} y^r_{v,d}) \qquad \forall\ v \in \mathcal{V},\ d \in \mathcal{D},\ w \in \mathcal{W} \tag{45}$$

$$\sum_{v \in \mathcal{V}} \delta_{v,d,w} \leq H^{drive}_{d,w} \qquad \forall\ d \in \mathcal{D},\ w \in \mathcal{W} \tag{46}$$

$$\sum_{v \in \mathcal{V}} (\delta_{v,d,w} + \delta_{v,d,w+1}) \leq H^{drive}_{d,w,w+1} \qquad \forall\ d \in \mathcal{D},\ w \in \mathcal{W} \setminus \{|\mathcal{W}|\} \tag{47}$$

Next, we consider legislation regarding weekly working hours, which contains driving hours, inoperative waiting before starting service and the actual service time at each *activity* node. Let $m_{v,i,k} \in \mathbb{B}$ be a binary variable representing whether service at *activity* i in *trip* v is started at day k and let $m^{start}_{v,i,k} \in \mathbb{B}$ and $m^{end}_{v,i,k} \in \mathbb{B}$ be two auxiliary variables similar to (39) - (41). The computation of the binary variables is modelled in (48) - (49).

We define $n_{v,i,k} \geq 0$ as the waiting time at *activity* i in *trip* v that is incurred on day k. The constraints computing this variable are given in (50). Note that $n_{v,i,k}$ is basically an auxiliary variable to determine at what day the waiting time $w_{v,i}$ is incurred. Let $\omega_{v,d,w} \geq 0$ denote the total waiting time in *trip* v for driver d that is incurred in week w. This variable is calculated in the constraints in (51). This constraint is only active if driver d is assigned to *trip* v. The service time in *trip* v for driver d incurred in week w is captured by variable $\sigma_{v,d,w} \geq 0$ and computed in constraint set (52). Again, the constraint is only active for *trip* v and driver d, when assigned to each other. Finally, the constraints regarding the maximum weekly working hours are modelled in (53).

Two important observations can be made regarding the binary variables $h_{v,i,k}$ and $m_{v,i,k}$. First, the binary variables are incorrectly calculated for inactive *trips*, because all time variables will be set to zero. However, this is not an issue, since the the corresponding travel, waiting and service time will not be incurred for any driver in constraints (42), (45), (51) or (52). Second, when $[E_i, L_i]$, the time window in which service should start, is fully included in a particular day k, then (48) and (49) can be replaced by $m_{v,i,k} = u_v$ and $m_{v,i,l} = 0$ for $l \neq k$.

$$t^{end}_k - M m^{end}_{v,i,k} \leq t_{v,i} \leq M m^{start}_{v,i,k} + t^{start}_k \qquad \forall\ v \in \mathcal{V},\ i \in \tilde{\mathcal{I}}_v,\ k \in \mathcal{K} \tag{48}$$

$$m_{v,i,k} \geq m^{start}_{v,i,k} + m^{end}_{v,i,k} - 1 \qquad \forall\ v \in \mathcal{V},\ i \in \tilde{\mathcal{I}}_v,\ k \in \mathcal{K} \tag{49}$$

$$n_{v,i,k} \geq w_{v,i} - M(1 - m_{v,i,k}) \qquad \forall\ v \in \mathcal{V},\ i \in \tilde{\mathcal{I}}_v,\ k \in \mathcal{K} \tag{50}$$

$$\omega_{v,d,w} \geq \sum_{k \in \mathcal{K}_w} \sum_{i \in \tilde{\mathcal{I}}_v} n_{v,i,k} - M(1 - \sum_{r \in \mathcal{R}_d} y^r_{v,d}) \qquad \forall\ v \in \mathcal{V},\ d \in \mathcal{D},\ w \in \mathcal{W} \tag{51}$$

$$\sigma_{v,d,w} \geq \sum_{k \in \mathcal{K}_w} \sum_{i \in \tilde{\mathcal{I}}_v} S_i m_{v,i,k} - M(1 - \sum_{r \in \mathcal{R}_d} y^r_{v,d}) \qquad \forall\ v \in \mathcal{V},\ d \in \mathcal{D},\ w \in \mathcal{W} \tag{52}$$

$$\sum_{v \in \mathcal{V}} (\omega_{v,d,w} + \sigma_{v,d,w} + \delta_{v,d,w}) \leq H^{work}_{d,w} \qquad \forall\ d \in \mathcal{D},\ w \in \mathcal{W} \tag{53}$$

A.3 Objective

(3) - (53) include all necessary constraints to capture all RPP restrictions. The model formulation can be finalized by the objective in (54), which attempts to minimize the sum of the total assignment cost, consisting of mileage cost and fixed cost, the total waiting cost, the total cost from uncovered *orders* and the cost arising from layovers. For the waiting cost, it is not necessary to include an additional multiplication with u_v, since waiting time variables will be set to zero for inactive *trips*. In Table 12, an overview of all cost parameters is given.

$$\min \sum_{v\in\mathcal{V}}\sum_{l\in\mathcal{L}}\sum_{r\in\mathcal{R}_l} C_{v,l}x^r_{v,l} + \sum_{v\in\mathcal{V}}\sum_{t\in\mathcal{T}}\sum_{r\in\mathcal{R}_t} C_{v,t}x^r_{v,t} + C_{wait}\sum_{v\in\mathcal{V}}\sum_{i\in\tilde{\mathcal{I}}_v} w_{v,i} + C_{omit}\sum_{o\in\mathcal{O}}\gamma_o + \tag{54}$$
$$C_{layover}\sum_{v\in\mathcal{V}}\sum_{(i,j)\in\tilde{\mathcal{A}}_v}\sum_{d\in\mathcal{D}}\sum_{b=1}^{|\mathcal{F}|} g^{v,i}_{f_{d,b},d}$$

Symbol	Description
$C_{v,l}$	Cost of assigning truck l to *trip* v. This parameter includes mileage cost and fixed cost
$C_{v,t}$	Cost of assigning trailer t to *trip* v. This parameter includes mileage cost and fixed cost
C_{wait}	Cost of waiting per time unit
C_{omit}	Cost of not covering an *order*
$C_{layover}$	Cost arising from a layover during the execution of a *trip*

Table 12: Cost parameters.

The RPP modelled in (3) - (54) includes an extremely large amount of variables and constraints. The problem is an extensive variant of VRP and can thus be classified as a NP-hard problem. Solving this problem for even moderate instances sizes is highly impractical. For this reason, we attempt to generate high-quality solution within reasonable computation times using a heuristical approach.

B Algorithms

B.1 Trip Construction

In Algorithm 11, the *trip* construction procedure is presented. Output set $\mathcal{V}$ contains all *locally feasible trips* that have the potential to be part of a RPP solution.

Steps 2 to 20 consider all permutations of all sizes, which has a complexity of $\mathcal{O}(|\mathcal{I}|! * |\mathcal{I}|)$. Steps 4 to 14 validate whether the constraints described in Section 3.4.2 are satisfied or not. If a violating constraint is found, the considered permutation is *locally infeasible* and the algorithm proceeds to the next permutation.

The precedence constraints and full *order* coverage are verified in Step 4. The sets of allowed trailers and drivers are constructed in Step 8. Measures Q_p^{max} and LDM_p^{max} can easily be obtained by updating the

LDM and weight while traversing all *activity* nodes and storing the maximum values. In Steps 10 - 11 all truck-trailer combinations with sufficient capacity are stored in $(\mathcal{T}_p, \mathcal{L}_p)$. Step 12 checks whether the sets of allowed resources are empty. A feasible time schedule, if existing, is constructed in Step 15. Finally, if still feasible, the permutation is added to the set of *trips* in Step 17.

Algorithm 11 *Trip* construction process

Input: set of *activities* $\mathcal{I}$

1: $\mathcal{V} \leftarrow \emptyset$
2: **for** $n = 1$ to $|\mathcal{I}|$ **do**
3: **for all** permutations p of $\mathcal{I}$ with size n **do**
4: **if** precedence constraints are violated **or** an *order* is only partly fulfilled **then**
5: **continue**
6: **end if**
7: $\mathcal{O}_p \leftarrow$ set of all *orders* served in p
8: $\mathcal{T}_p \leftarrow \cap_{o \in \mathcal{O}_p} \mathcal{T}_o$ (set of allowed trailers), $\mathcal{D}_p \leftarrow \cap_{o \in \mathcal{O}_p} \mathcal{D}_o$ (set of allowed drivers)
9: $Q_p^{max} \leftarrow$ max weight during visitation p, $LDM_p^{max} \leftarrow$ max LDM during visitation p
10: $\mathcal{T}_p \leftarrow \{t \in \mathcal{T}_p : LDM_t \geq LDM_p^{max}\}$
11: $(\mathcal{T}_p, \mathcal{L}_p) \leftarrow \{t \in \mathcal{T}_p, l \in \mathcal{L}_t : Q_t + Q_l \geq Q_p^{max}\}$
12: **if** $(\mathcal{T}_p, \mathcal{L}_p) = \emptyset$ **or** $\mathcal{D}_p = \emptyset$ **then**
13: **continue**
14: **end if**
15: validate whether there exists a feasible time schedule (Algorithm 13 in Section B.3)
16: **if** feasible schedule exists **then**
17: $\mathcal{V} \leftarrow \mathcal{V} \cup \{p\}$
18: **end if**
19: **end for**
20: **end for**

Output: set of *locally feasible trips* $\mathcal{V}$

B.2 Inclusion of Artificial Nodes

In Algorithm 12, the pseudocode for inserting artificial nodes is given. The main idea behind this procedure is that lengthy arcs can be split into smaller parts such that daily rest periods can be taken somewhere along the original arc. In this way, we simplify the problem by only considering possible rest locations at discrete points and not continuously on the the interval between two *activity* nodes. Parameter K functions as a splitting interval and determines how many artificial nodes are added between two *activity* nodes.

The procedure can be explained best by means of an example. Suppose that in a certain iteration of the for-loop in Steps 1 - 10 we are considering arc (i, j). Let K be five hours and let travel time $T_{i,j}$ be thirteen hours. We have that $n = 3$, since $10 \leq 13 < 15$ (Step 2). This means that three artificial nodes are added

between i and j and that the travel times of the first three new segments are set to $13/3 \approx 4.33$ (Step 3). The last artificial node is added at the same geographical location as the next *activity* node j, hence the travel time is set to zero in Step 9. This last node is added such that a daily rest period can be planned either directly before serving *activity* j or directly after serving *activity* j.

All artificial nodes receive wide-enough time window (Step 5) such that these have no effect on the feasibility of the problem.

Algorithm 12 Procedure for adding artificial nodes

Input: set of *activities* $\mathcal{I}$ **and** size of splitting interval K (hours)

1: **for all** arcs (i, j) between two *activities* or between an *activity* and the depot **do**
2: find $n \in \mathbb{N}$ such that $(n-1)K \leq T_{i,j} < nK$
3: $splitTravelTime \leftarrow T_{i,j}/n$
4: add n auxiliary *nodes* a_i for $i = 1, ..., n-1$ between i, j with following properties:
5: time windows $E_{a_i} \leftarrow \min\{E_i, E_j\}$ and $L_{a_i} \leftarrow \max\{L_i, L_j\}$ for $i = 1, ..., n$
6: service time $S_{a_i} \leftarrow 0$ for $i = 1, ..., n$
7: travel time $T_{i,a_1} \leftarrow splitTravelTime$
8: travel time $T_{a_i,a_{i+1}} \leftarrow splitTravelTime$ for $i = 1, ..n-1$
9: travel time $T_{a_n,j} \leftarrow 0$
10: **end for**

Output: set of *activities* $\mathcal{I}$ extended with auxiliary nodes

B.3 Construction of a Feasible Time Schedule

A feasible time schedule for a given sequence of *activities* is constructed in Algorithm 13. The algorithm generates a time schedule based on arriving at *activity* v_1 at the earliest possible start time E_{v_1}. In Steps 2 and 3, the arrival time, start time, waiting time, slack and departure time are initialized for the first *activity*. The slack variable can be interpreted as the amount of time that is left before violating the time window. In 4 - 12, these five variables are computed for the other *activities*. If the time schedule results in a time window violation, the *trip* is considered infeasible. The expression on the left hand side in Step 13 is an lower bound on the total time span of the *trip* excluding service at the depot. If this lower bound exceeds H^{work}_{consec}, the constraint of a weekly rest period at the depot will never be met. Hence, the *trip* is considered infeasible.

The algorithm is demonstrated with the following example. The resulting feasible schedule for a *trip* visiting three nodes is shown in Table 13. The column corresponding to node 1 is obtained by the initialization in Steps 2 and 3. The arrival time at node 2 can be obtained by the departure time from node 1 incremented by the travel time: $Arrive_2 \leftarrow 31 + 5 = 36$. The arrival is before the deadline $L_2 = 39$, hence we proceed with the algorithm. All other time variables are computed in Steps 10 and 11. The computation of the time variables for node 3 is analogous.

	Node 1	Node 2	Node 3
S_i	1	1	1
$[E_i, L_i]$	[30,34]	[38,39]	[50,55]
$T_{i,j}$	5	9	-
$Arrive_i$	30	36	48
$Wait_i$	0	2	2
$Start_i$	30	38	50
$Leave_i$	31	39	51
$Slack_i$	4	3	7

Table 13: feasible schedule of a *trip* visiting nodes 1, 2 and 3.

Algorithm 13 Procedure for creating a feasible time schedule

Input: a potential *trip* $v = \{v_1, ..., v_n\} \subseteq \mathcal{I}$

1: $feasible \leftarrow$ **true**
2: $Arrive_{v_1} \leftarrow E_{v_1}$, $Start_{v_1} \leftarrow E_{v_1}$, $Wait_{v_1} \leftarrow 0$
3: $Slack_{v_1} \leftarrow L_{v_1} - E_{v_1} + Wait_{v_1}$, $Leave_{v_1} \leftarrow Start_{v_1} + S_{v_1}$
4: **for** $i = 2$ to n **do**
5: $\quad Arrive_{v_i} \leftarrow Leave_{v_{i-1}} + T_{v_{i-1},v_i}$
6: $\quad$ **if** $Arrive_{v_i} > L_{vi}$ **then**
7: $\quad\quad feasible \leftarrow$ **false**
8: $\quad\quad$ **break** algorithm
9: $\quad$ **end if**
10: $\quad Start_{v_i} \leftarrow \max\{Arrive_{vi}, E_{vi}\}$, $Wait_{vi} \leftarrow E_{vi} - Arrive_{vi}$
11: $\quad Slack_{vi} \leftarrow L_{v_i} - E_{v_i} + Wait_{v_i}$, $Leave_{v_i} \leftarrow Start_{v_i} + S_{v_i}$
12: **end for**
13: **if** $(E_{v_n} + S_{v_n} + T_{v_n,0})$ - $(L_{v_1} - T_{0,v_1}) > H^{work}_{consec}$ **then**
14: $\quad feasible \leftarrow$ **false**
15: $\quad$ **break** algorithm
16: **end if**

Output: feasible time schedule, if existing

B.4 Construction of the Tightest Time Schedule

The procedure in Algorithm 14 describes how to construct the tightest time schedule for a given *trip* and feasible time schedule. Variable *maxLater* denotes how much time the arrival time at the first node can be extended. Following the example in Table 13, this variable is initialized with $L_1 - Start_1 = 34 - 30 = 4$. Auxiliary variable $flow$ is used to determine how much of variable *maxLater* is transferred to later *activities*. This variable is initialized with the initial value of *maxLater*.

In the first iteration with $i = 2$, we find in the if-statement in Step 4 that $flow = 4 > Slack_2 = 3$. Both $maxLater$ and $flow$ are then decreased with the difference between $flow$ and $Slack_2$: $maxLater \leftarrow 4 - 1 = 3$ and $flow \leftarrow 4 - 1 = 3$. The flow forward that is incurred at node 2 is thus three hours. In Step 8, variable $flow$ is updated. Since the waiting time at node 2 is two hours, two hours of the flow forward are consumed by this waiting time. So, only one hour of flow is transferred to node 3: $flow \leftarrow 3 - \max\{0, 2\} = 1$

In the second iteration for $i = 3$, we have that $flow = 1 < Slack_3 = 7$. Hence, variables $maxLater$ and $flow$ do not have to be updated in this iteration. In Step 8, $flow$ is updated to -6, after which the for-loop is terminated in Step 10. Note that this was the last iteration of the for-loop anyway.

The tightest schedule is ultimately obtained by extending the arrival time at the first node with $maxLater$ hours and calculating all time variables for the remaining nodes using the for-loop in Steps 4 - 12 from Algorithm 13. Variable $maxLater$ was only changed in the first iteration from four to three, which results in arrival time $Arrive_1 \leftarrow 30 + 3 = 33$. The resulting tightest schedule is showed in Table 14.

	Node 1	Node 2	Node 3
S_i	1	1	1
$[E_i, L_i]$	[30,34]	[38,39]	[50,55]
$T_{i,j}$	5	9	-
$Arrive_i$	33	39	49
$Wait_i$	0	0	1
$Start_i$	33	39	50
$Leave_i$	34	40	51
$Slack_i$	0	0	6

Table 14: Feasible schedule of a *trip* visiting nodes 1, 2 and 3.

Algorithm 14 Procedure for calculating the tightest time schedule

Input: *trip* $v = \{v_1, ..., v_n\}$ & feasible time schedule

$maxLater \leftarrow L_{v_1} - Arrive_{v_1}$
$flow \leftarrow maxLater$
for $i = 2$ to $|\mathcal{I}|$ **do**
  if $flow > Slack_{v_i}$ **then**
    $reduce \leftarrow flow - Slack_{v_i}$
    $maxLater \leftarrow maxLater - reduce,\ flow \leftarrow flow - reduce$
  end if
  $flow \leftarrow flow - \max\{0, Wait_{v_i}\}$
  if $flow \leq 0$ **then**
    break
  end if
end for
tightest schedule is ultimately obtained by setting $Arrive_{v_1} \leftarrow Arrive_{v_1} + \max\{0, maxLater\}$

Output: the tightest feasible time schedule

C Tables

C.1 Resource Types

Resource	Type	Weight	LDM		Resource Type	Weight	LDM
Trailer	Tiny Type 1	25,000	5	Trailer	Large Type 1	25,000	11
Trailer	Tiny Type 2	69,000	7	Trailer	Large Type 2	69,000	13
Trailer	Tiny Type 3	113,000	9	Trailer	Large Type 3	113,000	15
Trailer	Tiny Type 4	157,000	11	Trailer	Large Type 4	157,000	17
Trailer	Tiny Type 5	201,000	13	Trailer	Large Type 5	201,000	19
Trailer	Small Type 1	25,000	7	Trailer	Heavy Type 1	25,000	13
Trailer	Small Type 2	69,000	9	Trailer	Heavy Type 2	69,000	15
Trailer	Small Type 3	113,000	11	Trailer	Heavy Type 3	113,000	17
Trailer	Small Type 4	157,000	13	Trailer	Heavy Type 4	157,000	19
Trailer	Small Type 5	201,000	15	Trailer	Heavy Type 5	201,000	21
Trailer	Normal Type 1	25,000	9	Truck	Truck Type 1	10,000	-
Trailer	Normal Type 2	69,000	11	Truck	Truck Type 2	20,000	-
Trailer	Normal Type 3	113,000	13	Truck	Truck Type 3	40,000	-
Trailer	Normal Type 4	157,000	15	Truck	Truck Type 4	60,000	-
Trailer	Normal Type 5	201,000	17	Truck	Truck Type 5	80,000	-

Table 15: Weight and LDM capacities of all trailer and truck types.

C.2 Instance Features

Table 16 contains all instance features. The first three columns exhibit the number of *orders* with respectively one pickup *activity*, one delivery *activity* and both a pickup and delivery *activity*. The first element in the two-tuple under 'Node Positioning' contains the positioning of the depot as explained in Section 5.1.2. The second element in the two-tuple contains the used positioning mechanism for the *activity* nodes. The sixth column corresponds to the used demand distribution, which is explained in Section 5.1.3. The last three columns contain the number of available trucks, trailers and drivers respectively.

Name	#P	#D	#PD	Node Positioning	Demand	Trucks	Trailers	Drivers
n10-1	3	1	3	[CE, R]	U	2	2	3
n10-2	2	0	4	[R, R]	U	3	3	2
n10-3	5	3	1	[CE, CR]	SL	3	4	2
n10-4	0	2	4	[R, C]	R	3	2	2
n10-5	2	2	3	[CE, C]	SL	3	3	3
n10-6	3	1	3	[CO, C]	R	2	2	3
n10-7	2	2	3	[CE, CR]	SL	3	4	4
n10-8	2	2	3	[CO, C]	SL	3	2	4
n10-9	4	0	3	[CE, R]	U	3	4	4
n10-10	3	1	3	[R, R]	U	3	2	4
n50-1	13	17	10	[R, C]	U	15	19	13
n50-2	17	17	8	[CE, R]	SL	14	16	15
n50-3	17	13	10	[CE, C]	R	14	18	10
n50-4	11	17	11	[R, RC]	SL	17	17	15
n50-5	16	16	9	[CE, R]	U	13	13	17
n50-6	8	10	16	[CO, RC]	SL	16	14	14
n50-7	9	19	11	[CE, C]	R	11	20	12
n50-8	14	16	10	[CE, R]	R	12	16	14
n50-9	10	8	16	[R, C]	U	13	11	17
n50-10	11	15	12	[CO, C]	SL	17	11	15
n100-1	33	27	20	[R, C]	SL	37	27	32
n100-2	25	27	24	[CE, R]	R	27	18	22
n100-3	26	24	25	[R, C]	R	29	24	22
n100-4	19	27	27	[CE, RC]	R	17	26	29
n100-5	28	18	27	[R, C]	U	30	34	28
n100-6	28	34	19	[CE, C]	SL	29	39	38
n100-7	26	28	23	[CO, RC]	SL	30	28	24
n100-8	22	20	29	[CE, RC]	SL	34	17	15
n100-9	25	35	10	[CO, R]	R	27	17	35
n100-10	14	24	31	[CO, C]	U	16	25	24

Table 16: Instance features for all generated instances following the mechanisms described in Section 5.1

D Adaption Logistics Cloud Suite

Adaption (www.adaption-it.nl) is a specialized software company offering solutions to businesses that experience and maintain logistic processes. The founders realized that tailored software can be perfectly tuned to the customer's needs. This setup is however relatively costly in terms of support and maintenance, since several products for each specific customer have to be maintained. This tailored setup would therefore result in expensive software programs for both Adaption and its customers. Static standard software on the other hand does not meet the customers' requirements, but is relatively easy to maintain, since each customer uses the same product. Adaption therefore adopted the golden mean. The result is a hybrid standard application with a highly flexible setup, which is adjustable to the business' needs and maintainable, which makes it affordable for all sizes of logistic companies.

The software application, the Logistics Cloud Suite, consists of several modules aiming at all different parts of the supply chain. It provides businesses with a complete solution, which can support all business units and necessary scenarios (Adaption, n.d.). The resource planning tool is incorporated in the Transport Management System module, which provides a complete insight in a business' transportation activities.

In the three figures below, screenshots of the application are presented. All figures exhibit edit pages, where customers can access, edit and save data.

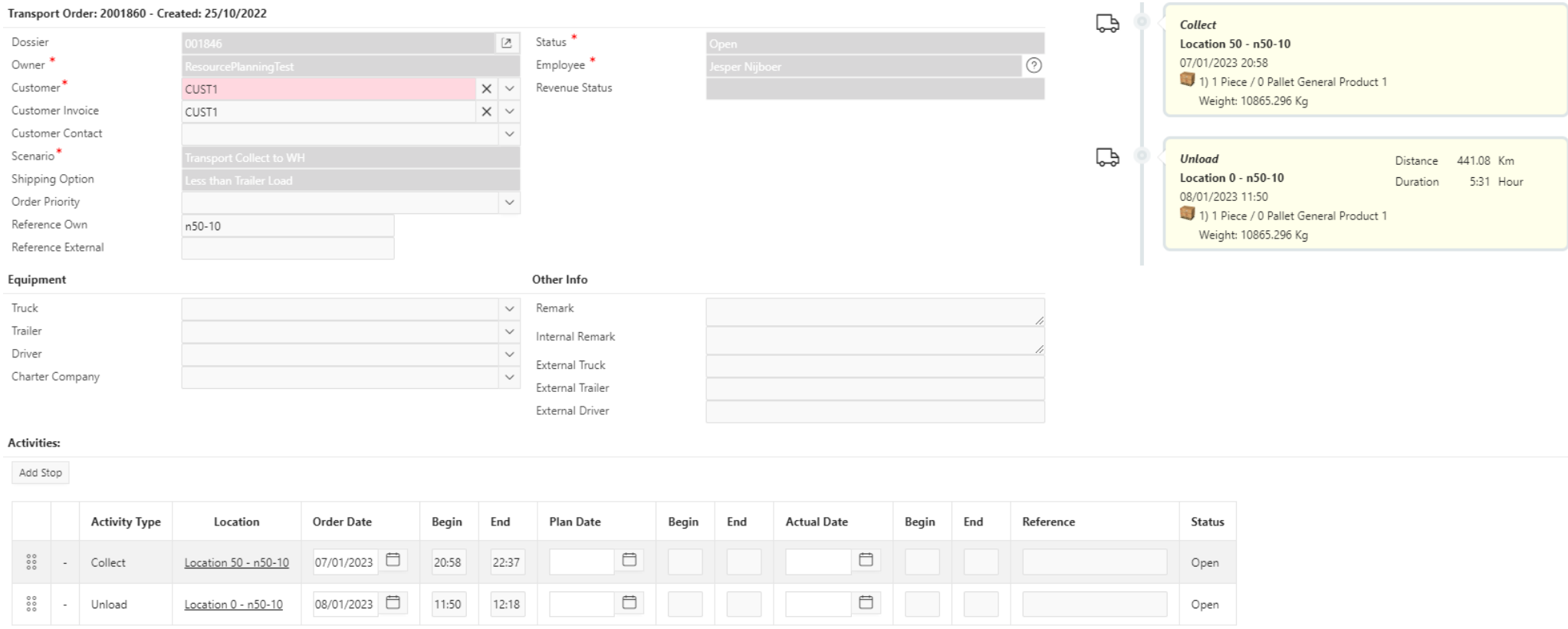

Figure 5: Screenshot of the *order* edit page. The yellow rectangles in the right upper corner correspond to the *activities* and contain information on distance, duration and weight of the goods. The time windows can be edited in the two rows at the bottom of the page.

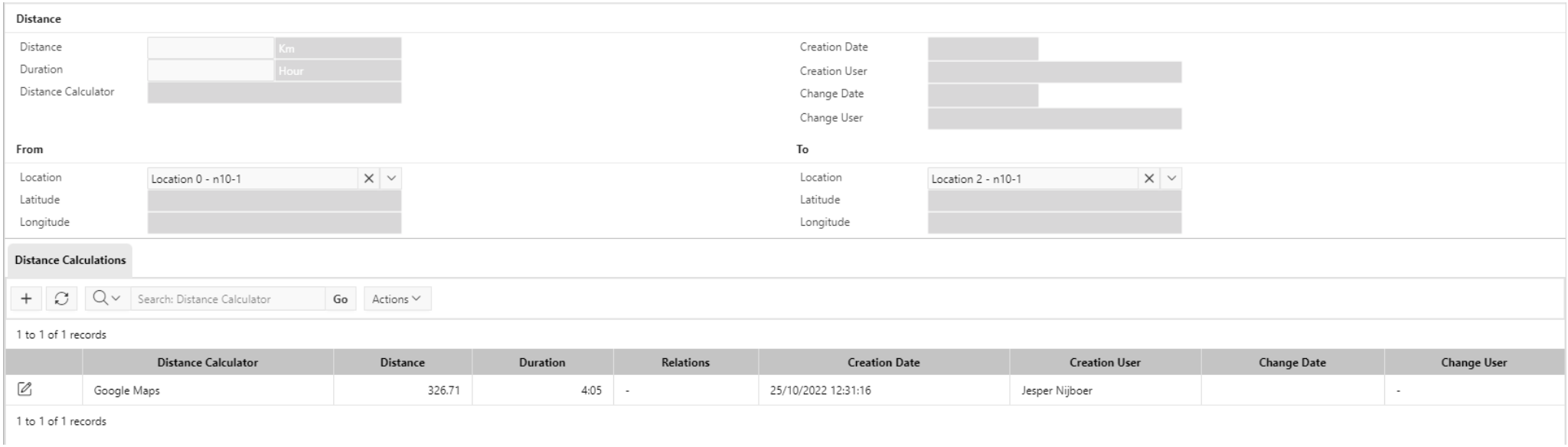

Figure 6: Screenshot of the distance edit page. For the purpose of this research, the distances and duration are manually generated and inserted. For locations with actual geocodes, Google Maps could be consulted to calculate distances and durations.

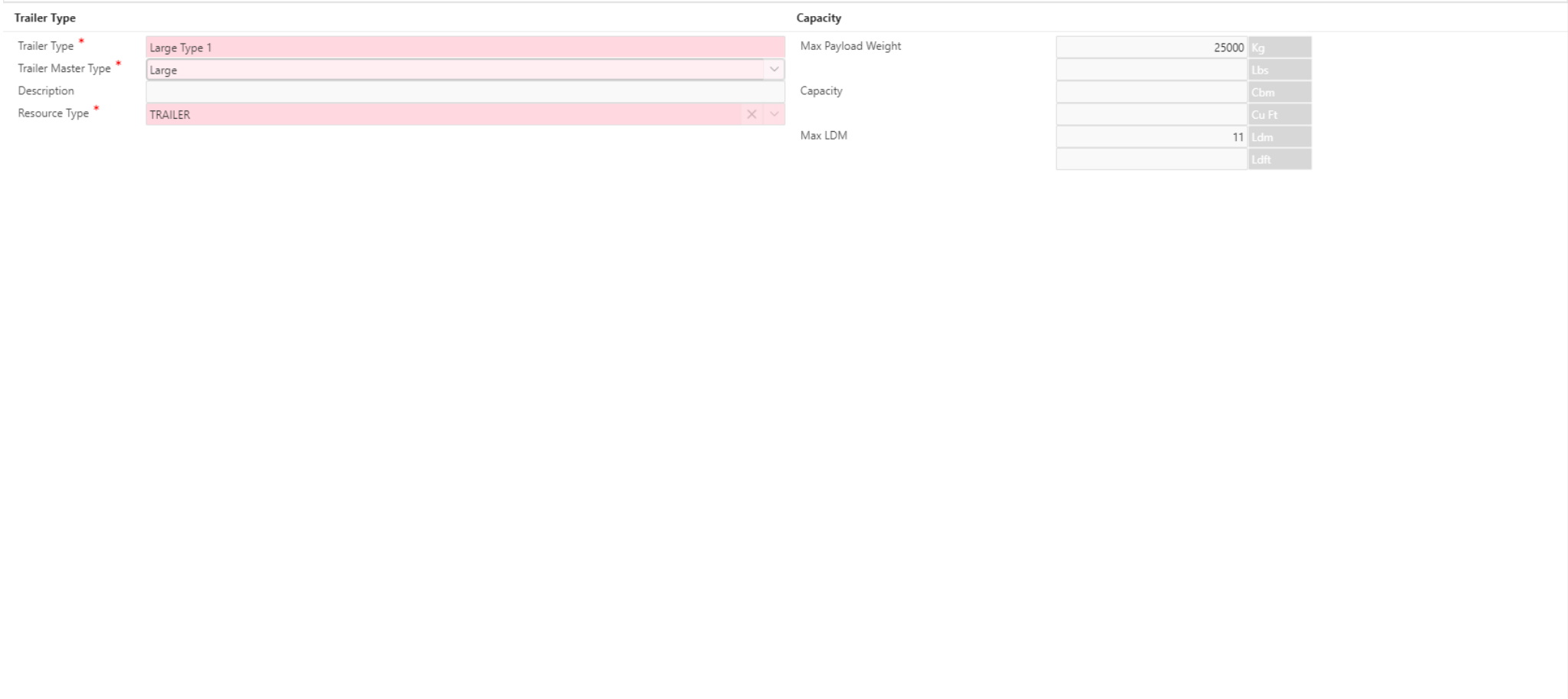

Figure 7: Screenshot of the trailer type edit page. The capacity measures of this trailer type are displayed in the fields on the right.