Scheduling and Routing of Truck Drivers Considering Regulations on Drivers' Working Hours

Zur Erlangung des akademischen Grades

Doktor der Ingenieurwissenschaften

von der KIT-Fakultät für Wirtschaftswissenschaften
des Karlsruher Instituts für Technologie (KIT)

genehmigte
Dissertation

von
Dipl.-Inf. Alexander Kleff

Tag der mündlichen Prüfung:	29. Mai 2019
Referent:	Prof. Dr. Stefan Nickel
Korreferent:	Prof. Dr.-Ing. Kai Furmans

Karlsruhe, 2019

Abstract

Scheduling and Routing of Truck Drivers Considering Regulations on Drivers' Working Hours

by Alexander KLEFF

In many countries, truck drivers are obliged by law to take a break or a rest regularly. In the European Union, for example, this is governed by Regulation (EC) No. 561/2006. It states that, after 4.5 hours of driving a truck, it is prohibited to continue driving until a 45-minute break is taken. After accumulating a driving time of 9 hours, a rest of 11 hours is mandatory. These are only two rules of a considerably longer list of break rules set out in this regulation, and it is only one of many regulations there are worldwide. Such breaks and rests have to be planned into the work schedules of the drivers. In general, the task of a dispatcher is to find routes and schedules for the truck drivers such that every customer is served in time. With the regulations on drivers' working hours, both the routing and the scheduling parts of the task become more challenging.

In this thesis, we study several optimization problems that arise in the context of drivers' working hours. One is known as the *truck driver scheduling problem*. Here, a sequence of customers is given, and the task is to find a schedule for a driver such that every customer is visited within one of the customer's time windows and the applicable break rules are complied with. Depending on the regarded break rules, we get different variants of the truck driver scheduling problem. Little is known about the complexity of the individual problem variants. One of the two focal points of this thesis is to present polynomial-time algorithms for different variants of the problem, for which polynomial-time algorithms are not yet known. With this, we can falsify the *NP*-hardness conjecture of Xu et al. (2003) for an important special case of their considered problem variant.

But this thesis is not only about scheduling, it is also about routing. This constitutes the second focal point of this thesis. We present an integrated approach for the *vehicle routing and truck driver scheduling problem*. Here, a route refers to the order in which the customers are visited. However, the meaning of route is twofold. In another studied problem, the *truck driver scheduling and routing problem*, it means the sequence of road segments that the driver takes to drive from one customer to the other. In this problem, we take into account that, before taking a break, truck drivers need to head for a rest area or at least a spot where their vehicle can be parked. We even consider the time-dependent scenario in which driving times on road segments vary over the day due to rush hours. Both an exact approach and a heuristic for this problem are presented, and both are evaluated on a recent road network instance of Germany. It turns out that the heuristic is at least two orders of magnitude faster but still hardly worse than the exact approach.

Our main motivation is the application in practice. It is our aim – and this is especially true for the second focal point – to provide helpful algorithms that may find their way into software products for dispatchers, like the described approach for the vehicle routing and truck driver scheduling problem is already integrated into the vehicle route planning tools of a commercial provider of logistics optimization software.

Contents

List of Figures

List of Tables

List of Algorithms

List of Abbreviations

CH	**C**ontraction **H**ierachies
CMV	**C**ommercial **M**otor **V**ehicle
EF-TDSP	**E**arliest **F**inish (Time) **T**ruck **D**river **S**cheduling **P**roblem
FIFO	**F**irst **I**n **F**irst **O**ut
HOS	**H**ours **O**f **S**ervice
MD-TDSP	**M**inimum **D**uration **T**ruck **D**river **S**cheduling **P**roblem
TCH	**T**ime-Dependent **C**ontraction **H**ierachies
TDSP	**T**ruck **D**river **S**cheduling **P**roblem
TDSP-2B	**T**ruck **D**river **S**cheduling **P**roblem with **2** Types of **B**reaks
VRP	**V**ehicle **R**outing **P**roblem
VRTDSP	**V**ehicle **R**outing and **T**ruck **D**river **S**cheduling **P**roblem

List of Symbols

n	number of customers
w_i	number of time windows of customer i
$\mathcal{W}_i^j$	j-th time window of customer i
$service_i$	service time at customer i
$drive_i$	driving time from customer i to customer $i+1$
$\mathcal{H}$	planning horizon
$break$	minimum break period after which a driver is rested
$break^{1st}$	minimum duration of a first split break
$break^{2nd}$	minimum duration of a second split break
$limitD$	limit on accumulated driving time
$limitT$	limit on accumulated travel time
w	total number of time windows
$\mathcal{W}_i$	time windows of customer i
$\overline{\mathcal{W}}_i^j$	j-th waiting interval at customer i
W_i	waiting time function of customer i
$\perp$	special value to be read as *undefined*
α	maps an interval to its beginning or a time-dependent function to its first defined point
ω	maps an interval to its end or a time-dependent function to its last defined point
id	identity function that maps a time to itself
$\odot$	link operation for driving time profiles
$\oplus$	merge operation for driving time profiles

Für Simeon, der es nicht abwarten wollte

Chapter 1

Introduction

"We're only here for so long. Be happy, man. You could get hit by a truck tomorrow."
— Timothee Chalamet[1]

Every traffic accident is one too many. This is all the more true in view of accidents involving trucks. Due to their size and weight the consequences are particularly severe. The risk of dying in a truck crash is significantly higher for the other road users than for the truck occupants. On German roads in the year 2016, there were 29 353 traffic accidents with the participation of a *goods road transport vehicle* in which persons were harmed, according to the German Federal Statistical Office (Statistisches Bundesamt (Destatis), 2017). 9 483 occupants of such trucks and 30 774 other road users were injured in these accidents. Of the 745 who died, only 133 were truck occupants.

One of the causal factors for accidents is fatigue of the drivers. In their effort to counteract driver fatigue and eventually increase road safety, many governments have established legal limits on *drivers' working hours*. Simply put, truck drivers are obligated to take breaks on a regular basis. For instance, the Regulation (EC) 561/2006 of the European Union stipulates a break of at least 45 minutes after at most 4.5 hours of driving (European Parliament and Council of the European Union, 2006). But this rule is only the tip of the iceberg. The complete set of rules that European truck drivers have to abide by is long and complex. (A more comprehensive description can be found in section 2.1.)

Generally speaking, such legal provisions restrict the time the driver is allowed to drive or work. For instance, this may mean a limit on the continuous driving time without break. Or it may mean a limit on the total driving time within a certain time span such as a day, a week, or even longer periods of time. Besides constraints on maximum driving or working times, they demand minimum requirements in respect of breaks like their minimum duration in order to make sure the truck driver can rest sufficiently.

These provisions leave the truck drivers and dispatchers alike with the problem of planning breaks into the drivers' work schedules such that the applicable break rules are respected. In this thesis, we investigate the question of how to help them out. To this end, we develop algorithms tailored to different variants of the planning problems that emerge in the context of drivers' working hours.

In order to enable drivers and dispatchers to make use of these algorithms, they need to be implemented and integrated into some software suite. At PTV[2], we create various software products for logistics optimization. It is this practical application of the algorithms that is a driving motivation behind our research.

[1] `https://www.wmagazine.com/story/actor-timothee-chalamet-prodigal-son` (accessed on 2018-06-28)

[2] `https://www.ptvgroup.com/`

1.1 Problem Descriptions

Let us have a deeper look at some of the planning problems that drivers and dispatchers are confronted with.

Truck Driver Scheduling Truck drivers deliver goods to customers and/or collect goods from them. Typically, everyone of these customers demands this to happen within certain time windows. For a given sequence of customers, a truck driver needs to find a schedule such that not only every customer is visited within one of these time windows but also the break rules stated in all applicable regulations concerning drivers' working hours are respected. This problem is referred to as the *truck driver scheduling problem* (inter alia, Goel (2010)).

Since such regulations may vary from country to country, it is rather a family of problems. Apart from this, we find different problem variants in the literature as not always all relevant regulations and not always all break rules specified therein are taken account of. Unfortunately, there is no agreed upon basic version of the problem. The simplest variants that are still relevant in practice are characterized by a single break rule. As an example, a break rule may enforce a break of some minimum duration (say 45 minutes) after the driver has accumulated a certain driving time since the last break (say 4.5 hours). This break rule (and its parameter setting) is derived from the Regulation (EC) 561/2006 mentioned above.

In the literature, truck driver scheduling problems are sometimes formulated as *decision problems*, deciding whether or not a *feasible* truck driver schedule exists within a given planning horizon. A truck driver schedule is deemed feasible if it is in line with the regarded break rules and all customers are served in time. If they are formulated as *optimization problems*, the goal is usually to find the minimum duration of a feasible schedule. But also other optimization goals with a focus on actual costs have been studied (Xu et al., 2003; Koç et al., 2016; Bernhardt et al., 2017).

The truck driver scheduling problem is about determining the succession and duration of driver activities like when to drive, when to load/unload the vehicle at a customer, or when to take a break. It may happen that a break becomes due when the driver is en route between two customers. In this case, it is not part of the problem to also decide *where* to take that break. So it is assumed implicitly that the location of the break plays a negligible role.

Truck Driver Routing But in how far is this the case in practice? What are the implications when we ignore the break locations? Certainly, the driver needs to park the truck in order to take a break, and such wide and long vehicles cannot be parked everywhere. Let us assume for the moment that, whenever a truck driver arrives at a parking area, he[3] finds an available parking spot there. Still two things are not taken account of when break locations are ignored: One is the detour that it takes to get there and back again. Parking areas may not be located directly aside the computed optimal route between two consecutive customers. For instance, a so-called "Autohof" may be up to a kilometer away from the junction of the autobahn in Germany.

The other issue is that there may simply not be a suitable parking area around when it is needed. In such a case, the break has to be taken some time before it becomes due. The implications can be severe when the total driving time on a route

[3]For the sake of a simpler notation, we only speak of male drivers throughout this thesis. The legal provisions and the mathematical findings in this thesis also hold for female drivers.

is close to a multiple of the maximum driving time without break. Let us take the above mentioned EU regulation as example: Suppose the total driving time is close to 9 hours. Then one break may suffice, but only as long as it can be taken after not much less than exactly 4.5 hours of driving. That is, there is only little margin, it is crucial that there is a parking spot available right when it is needed. Should there be no parking spot available at that time, then the break must be scheduled earlier, and thus a second break becomes mandatory, shortly before the end of the route. And that additional second break has presumably an even greater impact on the schedule than the detour. In both cases, time windows may be missed or some provisions regarding working hours may be violated in practice if the break locations are disregarded and there is not enough buffer in the schedule.

Taking the break locations into account becomes all the more relevant in practice, the more limited the truck driver is when in search of a spot to park the truck. For instance, this is the case when a secure, monitored parking space is needed; or when hazardous goods are transported; or when the driver has certain demands regarding the offered amenities (showers, restaurants, ...); or when information about the occupancy status of parking areas becomes available.

We learn that it is worthwhile to take parking locations and the underlying road network into account. This motivates us to introduce the *truck driver routing problem*. In this setting, we assume a long-haul driver who needs to drive from a source to a destination, and we want to find a shortest feasible route in the road network that leads from the source to the destination via some parking locations (if necessary or beneficial). Here, a route is considered as feasible if a feasible schedule exists where the breaks are only taken at parking locations.

Truck Driver Scheduling and Routing While parking locations and the underlying road network are disregarded in the truck driver scheduling problem, the truck driver routing problem is to find a feasible route in the road network only between a source and a destination. Here, more than two customers and their time windows are not an issue. In the *truck driver scheduling and routing problem*, these two problems are combined. On the one hand, we want to find a route in the road network that leads from the first to the last customer via the other customers of a given sequence and, if need be, some parking locations. On the other hand, we want to find a corresponding schedule such that every service starts within a time window of the respective customer, the provisions on working hours are respected, and every break is taken either at a parking location or at a customer.

Vehicle Routing In the truck driver scheduling problem, the sequence of customers is supposed to be given. However, the problem of finding a good if not optimal sequence is everything but simple. In the *vehicle routing problem* (here: with multiple time windows), we are given a fleet of vehicles and a large number of customers that all demand a certain service. Every customer states one or several time windows in which the service is allowed to begin. The problem is now to assign every customer to a vehicle on the one hand, and determine the sequence of customers for every vehicle on the other. Typically, the objective is to use as few vehicles as possible and, as second criterion, to minimize the mileage.

There is a tremendous amount of literature on the vehicle routing problem and seemingly every facet of this problem has already been explored (see Braekers, Ramaekers, and Nieuwenhuyse (2016) for classification and review, as well as the surveys of Vidal et al. (2013) and Lahyani, Khemakhem, and Semet (2015) particularly

on *rich* or *multi-attribute* vehicle routing problems). Just like the truck driver scheduling problem, the vehicle routing problem is rather a family of problems as it has been extended into so many different directions.

Vehicle Routing and Truck Driver Scheduling In one of these directions, we find the *vehicle routing and truck driver scheduling problem*. As the name suggests, it combines the vehicle routing problem and the truck driver scheduling problem. That is, one wants to find vehicle routes that also comply with the applicable rules on drivers' working hours, though under the simplifying assumption that breaks can be taken anywhere.

We might as well view the truck driver scheduling problem as a (non-trivial) subproblem of the vehicle routing and truck driver scheduling problem in which the subproblem has to be solved for every vehicle route. And yet, the truck driver scheduling problem also arises as a standalone problem in practice. Then, the sequence of customers is determined externally, for instance by the dispatcher. This is true especially in a real-time scenario. Here, driving and service times can only be estimated at the beginning of the route, and better estimates become available in the course of time. Even though the real-time scenario is not directly addressed in this thesis, it is a constant motivation for our studies.

It should be noted that the term *route* is ambiguous. In the truck driver scheduling and routing problem, the sequence of customers is fixed, and so the term refers to the sequence of road segments between consecutive customers. On the contrary, in the vehicle routing and truck driver scheduling problem, the sequence of road segments between two customers is supposed to be known, and so the term refers to the sequence of customers. For the sake of clarity, we may use the terms *truck driver route* and *vehicle route*, respectively – unless there is no danger of confusion. Going to the extreme, we could even think of a *vehicle routing and truck driver scheduling and routing problem*, which combines both problems and in which both definitions of routes are unified. But this would be far beyond the scope of this thesis.

1.2 Scope of Thesis

In this thesis, we analyze the aforementioned *combinatorial optimization problems* and develop algorithms to solve them. Before we expand upon these problems in detail, we outline the scope of this thesis and raise three major research questions.

At first, let us be clear that it is beyond the scope to question the legal provisions. We assume throughout this thesis that a driver is never fatigued when he is on a schedule that complies with the rules. Hence, a model of fatigue itself is not incorporated by our problem definitions. A variant of the truck driver scheduling problem that includes fatigue monitoring is introduced by Bowden and Ragsdale (2018). An optimization-based assessment of international regulations can be found in the paper of Goel and Vidal (2014).

There are two general notes regarding the input data. One is that we assume *complete* knowledge of all relevant information, that is, we only deal with *offline problems*. In a real-time scenario, it may happen that a problem instance changes over time. This fact remains disregarded in this thesis. However, a rather trivial way of coping with this would be to re-optimize the problem instance every time new information is gained. Apart from the assumption of complete knowledge, all relevant information needed to solve the problem at hand is modeled as *deterministic* in this thesis. For instance, driving times between customers or on segments of the road

network are assumed to be known exactly. But this does not mean that we ignore the fact that driving times may vary during the day. *Time-dependent driving times* are considered in connection with the truck driver routing problem as well as the truck driver scheduling and routing problem.

Regarding the working hours of drivers, we restrict our studies to a single driver behind the wheel. This note is important because different rules may apply when multiple drivers take turns. Besides, we completely ignore the possibility of a driver change along a route. Our main focus are the provisions that are valid throughout the European Union. But we also shed light on the US provisions. Additional (or more restrictive) rules may apply in individual member states of the EU or other countries in the world. But such rules are rather only touched in this thesis for the sake of conciseness.

As far as the various problem definitions are concerned, we always allow multiple time windows per customer. In the literature, most often only one time window per customer is permitted (see section 2.2). We consider the break rule parameter setting (like 45 minutes and 4.5 hours) to be a part of each problem instance and not part of the problem itself. This is not self-evident as only certain parameter settings are of practical relevance. When we study the complexity of some problems, we will not make any assumptions about the setting of the break rule parameters and concentrate on statements that are independent of the setting.

Let us have a look at the scope of the individual problems and the major research questions.

1.2.1 Truck Driver Scheduling

A wide range of different truck driver scheduling problem variants has been discussed in the literature. And while both exact and heuristic solution methods have been proposed, little is known about the complexity of these problems. For rather simple problem variants it is known that they can be solved in polynomial time (Archetti and Savelsbergh, 2009; Goel and Kok, 2012b; Goel and Kok, 2012a). For far more complex problem variants, *NP*-hardness is conjectured (Xu et al., 2003; Drexl and Prescott-Gagnon, 2010). But, to the present day, these conjectures are neither proven nor falsified. Hence, a large part of this thesis is motivated by the question:

Which problem variants of the truck driver scheduling problem can (still) be solved in polynomial time - and how?

To be precise, we are interested to find algorithms that run in *strongly polynomial time*, where the run-time only depends on the number of customers and the number of time windows. In contrast, this means that their run-time does not depend on the length of the intervals (such as the customers' time windows or the planning horizon), the driving time between the customers, or the parameter setting of the break rules (such as the minimum break duration).

The problem variants that we examine are enhancements of the problems studied by Archetti and Savelsbergh (2009), Goel and Kok (2012b), and Goel and Kok (2012a). Their problems have in common that there is only one type of break and there are two conditions under which a break of this type becomes mandatory. One is that the driver has accumulated a certain driving time since the end of the last break (as regarded before). The other is that simply a certain time has elapsed since the end of the last break. For now, let us call this truck driver scheduling problem variant *basic* as it lies the foundation of several problems considered in this thesis. In the basic problem, each customer may only specify a single time window. It is a

decision problem, so the objective is to find out whether a feasible schedule exists or not. As shown by the authors just mentioned, this basic problem is solvable in polynomial time. More precisely, it is solvable in a time that is quadratic in the number of customers. In this thesis, we enhance the basic variant into four directions:

1. Truck driver scheduling with multiple time windows per customer: As mentioned, each customer may only specify a single time window in the basic problem. The first enhancement is thus to allow the customers to specify multiple time windows. We consider the variant in which the objective is to find a schedule with earliest finish time, that is, the earliest completion time of the last service.

2. Truck driver scheduling with multiple time windows per customer and break splits: In the second enhancement, it is allowed to split a break in two parts, that is, to take two shorter breaks instead of one long break. This is motivated by a corresponding rule that is laid down in the Regulation (EC) 561/2006 of the European Union. Optimization goal is again the earliest finish time.

3. Truck driver scheduling with multiple time windows per customer and minimum duration objective: As third enhancement, we are interested in finding a feasible schedule for which the duration from the beginning of the first service to the end of the last service is minimum. This optimization goal turns out to be harder than the earliest finish time.

4. Truck driver scheduling with multiple time windows per customer and two types of breaks: In the fourth enhancement, we distinguish two break types, a short break and a long break. This means there is a set of break rules regarding the short break and another set of break rules regarding the long break. Again, optimization goal is the earliest finish time.

   In general, it depends on the length of the planning horizon how many different types of breaks need to be distinguished. For a planning horizon of one day, only one type of break has to be considered. Let us call breaks of this type "lunch breaks". For a planning horizon of up to one week, both lunch breaks and "daily rest breaks" need to be scheduled, and for a planning horizon of several weeks even, this holds for lunch breaks, daily rest breaks, and "weekly rest breaks". The official terms of the break types depend on the regulation. Details follow in the next chapter.

Drexl and Prescott-Gagnon (2010) study a very complex problem variant and conjecture that their problem at hand is *NP*-complete. So there is a gap between variants that can be solved in strongly polynomial time on one hand and those that are conjectured to be *NP*-hard on the other. This raises the question where the end of the line is, that is:

Which variants can no longer be solved in strongly polynomial time?

However, this is only a subordinate research question.

1.2.2 Truck Driver Scheduling and Routing

Taking also parking locations into account can be viewed as another extension. But our research has a different focus when we turn towards the truck driver routing problem and the combined truck driver scheduling and routing problem. The motivation is no longer to find more and more problem variants that can still be solved

in strongly polynomial time. Instead, it is the practical application that motivates us. Our aim is a "proof of concept". This means we want to develop algorithms that could be the basis of a future product feature of PTV.

The main feature we are interested in is not so much the consideration of parking locations alone. It is rather the consideration of parking locations together with time-dependent driving times. In practice, it is of high importance to take the increased driving times during rush hours into account. According to the ADAC (2018), 723 000 traffic jams were recorded on German autobahns in 2017, and these had a total length of 1 448 000 km. The time the road users had to spent in these traffic jams sums up to 457 000 hours. And these numbers are increasing from year to year.

When the driving time along road segments is regarded as time-dependent, then it makes sense to also consider parking locations. Let us give an example why. Suppose a driver has to drive from customer c_1 to customer c_2, and we have no knowledge about parking locations (but breaks en route are still allowed). Instead we are only given a function over time that maps a departure time from c_1 to the driving time to c_2. In our example, the driver is forced to take a break somewhere between the two customers (regardless of the departure time). While the driver is taking a break, the roads ahead to the next customer become more an more congested. What will be the remaining driving time to the next customer after the break? We cannot tell because we only know the driving time for the case that the driver does not stop for a break between the two customers. We could tell only if we had not only scheduled the time of the break but also the place and if we had such a driving time function on every edge of the road graph.

With time-dependent driving times, the truck driver (scheduling and) routing problem is computationally a lot harder to solve (see Foschini, Hershberger, and Suri, 2014). As compensation, we restrict our studies in two ways. One is that we only consider the truck driver (scheduling and) routing problem with just one break rule (instead of two as with the truck driver scheduling problem). This break rule demands a break after the driver has accumulated a certain driving time since the last break. The other restriction is that we only take at most one parking location between two customers into consideration. In practice, both are not a limitation for a planning horizon of one day in the EU.

Here, the research question can be stated as follows:

Given real-world data like a road graph representing the German road network and information on predictable congestion, how quickly can optimal truck driver routes and schedules be computed, and what is a good trade-off between run-time and solution quality (and memory consumption)?

1.2.3 Vehicle Routing and Truck Driver Scheduling

The literature on the vehicle routing problem is so immense that even survey papers can only cover a fraction of it and typically concentrate on certain aspects or variants of the problem. Equally high is the number of suggested solution approaches, may they be exact or heuristic (or matheuristic). And so it is not surprising that also both exact and heuristic approaches have already been proposed for the vehicle routing and truck driver scheduling problem (see section 2.2 for related work). Most of the heuristic approaches are based on local search in some way or the other.

In this thesis, we do not describe yet another algorithm for this problem. At least, this is not in the focus. Instead, it is about how solving the scheduling sub-problem,

that is, checking the feasibility of routes, can be integrated into a given local search based heuristic. Formulated as research question:

How can the feasibility in respect of drivers' working hours be checked efficiently within local search based heuristics for the vehicle routing and truck driver scheduling problem?

This part is motivated to a large extent by the needs of the commercial vehicle routing software of PTV. Besides solution quality, fast response times are key in practice, and so an efficient implementation is crucial. Concerning the considered rules, we concentrate on the rules that are effective in the EU for a planning horizon of one day.

1.3 Organization and Main Contributions

Now that we have set out the scope, we give an overview of the chapters and summarize the main contributions of each chapter. The first two lay the foundations of the main parts of the thesis.

1 **Introduction**: This chapter.

2 **Fundamentals and Preliminaries**: In the second chapter, we present legal regulations affecting drivers' working hours. We give detailed descriptions of the provisions effective in the EU and in the US. Based on these, we describe our classification of the break rules and also introduce some basic terminology and some notation that is used in this thesis. In addition, we present the literature regarding the optimization problems that are covered in this thesis.

The next three chapters belong together and constitute the first major part. Here, the focus lies on the truck driver scheduling problem and its complexity. These chapters form the part on polynomial-time algorithms for various variants of this problem. To the best of our knowledge, we develop the first algorithms that are proven to run in polynomial time (polynomial in the number of customers and/or in the number of time windows) for the following variants of this problem:

3 **Truck Driver Scheduling with Multiple Time Windows**: In this chapter, we present the first polynomial-time algorithm for the truck driver scheduling problem with multiple time windows per customer. Furthermore, we show that the same bound can be achieved when it is allowed to split a break in two parts.

4 **Truck Driver Scheduling with Minimum Duration Objective**: In this chapter, we present the first polynomial-time algorithm for the truck driver scheduling problem with multiple time windows per customer and minimum duration objective. With this, we can falsify the *NP*-hardness conjecture of Xu et al. (2003) for an important special case of their problem.

5 **Truck Driver Scheduling with Two Types of Breaks**: In this chapter, we study the truck driver scheduling problem with multiple time windows per customer and two types of break. For the case that it is only allowed to take breaks at customers and not en route between them, we present the first polynomial-time algorithm for this variant. However, the general case is harder. Here, we show that the number of non-dominated states may not solely depend on the number of customers and time windows. Hence, we conjecture that a strongly polynomial-time algorithm does not exist in the general case.

The next chapter is the only one about the combined vehicle routing and truck driver scheduling problem. In principle, any algorithm for the truck driver scheduling problem could be plugged "as is" into the feasibility check of an algorithm for the vehicle routing problem. A drawback of this approach is that information about partial solutions is not cached and likely to be computed multiple times. In contrast, the next chapter is on an integrated approach, where such information is cached, making this approach more efficient.

6 **Vehicle Routing and Truck Driver Scheduling with Multiple Time Windows**: In this chapter, we present an approach how to integrate an exact feasibility check into a local search based heuristic for the vehicle routing and truck driver scheduling problem with multiple time windows. This feasibility check is tailored to the rules that are effective in the EU for a planning horizon of one day and has not been described before.

Besides the chapters 3, 4, and 5, the chapters 7 and 8 constitute the second major part, where we present our findings concerning the truck driver (scheduling and) routing problem on time-dependent road networks. To the best of our knowledge, both problems have not been studied before, apart form the bachelor thesis of Bräuer (2016).

7 **Truck Driver Routing on Time-Dependent Road Networks**: We present both an exact and heuristic solution approaches for it. For the exact method, we describe acceleration techniques and prove their effectiveness. Our proposed heuristics turn out to be much faster while preserving a very good quality level.

This chapter is an (almost) exact quote of the article by Kleff et al. (2017).

8 **Truck Driver Scheduling and Routing on Time-Dependent Road Networks**: Again, we present both exact and heuristic solution approaches. Here, the proposed heuristic is up to three orders of magnitude faster, finding the optimal solution in most of the cases.

Most chapters end with a section that sums up the findings of the respective chapter. Analogously, this thesis is concluded by a chapter that sums up the findings of all chapters.

9 **Conclusion and Outlook**: Here, we refer back to the research questions raised in the previous section.

Chapter 2

Fundamentals and Preliminaries

The most fundamental pieces of information that the reader is missing up to now are details about the regulations that affect the drivers' working hours. This information is finally given in section 2.1. With this, it is a lot easier to appraise the related work that we outline afterwards in section 2.2. In the subsequent section 2.3, we introduce our classification of the relevant break rules. In section 2.4, we present basic definitions and notation.

2.1 Regulations Affecting Drivers' Working Hours

The role of driver fatigue in road safety is complex and subject to research. Summarizing earlier research findings, the European Transport Safety Council (2001) comes to the conclusion that driver fatigue is a significant factor in approximately 20% of commercial road transport crashes and that over 50% of long haul drivers have at some time fallen asleep at the wheel. A detailed, more recent report on how increases in hours of service are linked to increases in fatigue, and how this in turn is linked to increases in crash risk for truck drivers is given by the National Academies of Sciences, Engineering, and Medicine (2016). Similarly, they assume that up to 20% of the truck and bus crashes with fatalities on US American roadways may have involved fatigued drivers.

In this thesis, we focus on regulations effective in the European Union and the United States. Both the European Union and the United States have a long history of regulations limiting the hours of work of truck (and bus) drivers. Due to the ongoing research, the rules have been adapted over the years to reflect recent findings. This is particularly true if misuse of rules is detected or if rules do not have the expected or desired effect. Since rules can change, it will be all the more important to us to describe the problems at hand as generally as possible and only as concretely as necessary.

In this section, we present the regulations affecting drivers' working hours in the European Union (section 2.1.1) and the United States (section 2.1.2) in greater but not in every detail. However, we will explicitly state where there are gaps in our presentation. There may be equivalent regulations in other countries such as Canada, Australia, or South Africa. But these are not described here for the sake of conciseness. At least for the truck driver scheduling problems in Canada and Australia, literature exists, so we refer the reader to these works (Goel and Rousseau (2012) concerning Canadian provisions and Goel, Archetti, and Savelsbergh (2012) concerning Australian provisions).

2.1.1 European Union

In the European Union, the first community-wide drivers' hours rules were introduced in 1969 with Regulation (EEC) 543/69. In 1985, this regulation was repealed by Council Regulation (EEC) 3820/85, which in turn was repealed by Regulation (EC) No 561/2006 in 2006. This is the regulation that is currently effective.

But the current regulation does not only repeal an older regulation on drivers' working hours, it also amends Council Regulation (EEC) No 3821/85 on recording equipment in road transport, in which technical matters in connection with installation and inspection of *tachographs* are settled. These tachographs make the necessary recordings that are crucial when it comes to checking compliance with the rules. Today, *digital* tachographs are mandatory.

In the following, we present the most important drivers' hours rules of Regulation (EC) No 561/2006. These rules are supplemented by those stated in Directive 2003/88/EC and Directive 2002/15/EC, which we outline thereafter. Even though our presentation of the rules is already comprehensive, it is not complete. Examples of rules that we ignore are given in the respective subsection.

2.1.1.1 Regulation (EC) No 561/2006

Regulation (EC) No 561/2006 applies "to the carriage by road of goods where the maximum permissible mass of the vehicle, including any trailer, or semi-trailer, exceeds 3.5 tonnes" (European Parliament and Council of the European Union, 2006). In special cases, vehicles and drivers are exempt from this regulation. For instance in case of *non-commercial* carriage of goods, it only applies to drivers of vehicles with a maximum permissible mass of more than 7.5 tonnes. The rules laid down in the regulation do not only apply to truck drivers but also to bus drivers whenever they are at the wheel of a bus that is large enough to carry more than nine persons including the driver. However, the carriage of passengers is not in the focus of our research.

The regulation does not only apply to routes within or between EU member states. Since the provisions of the *European Agreement Concerning the Work of Crews of Vehicles Engaged in International Road Transport* (AETR) are in line with this regulation (at least to the extent covered in this work), they also apply to routes to or through all signatory countries of the AETR. So roughly speaking, they are relevant all over Europe.

Not only drivers can be made liable for infringements but also the employing transport undertakings. It is in their responsibility to organize the work of drivers in a way such that these are able to comply with the regulation. Additionally, they have to make regular checks to ensure the compliance. But it is not only on the transport undertakings. Also "consignors, freight forwarders, tour operators, principal contractors, subcontractors and driver employment agencies shall ensure that contractually agreed transport time schedules respect this Regulation" (ibidem).

As already mentioned, we only present the most important rules. Journeys involving ferry or train transport, for instance, are beyond our scope. Here, exceptions from the rules may apply.

Breaks and Rests From a legal perspective, we need to distinguish *breaks* from *rest periods*. While a break is defined to be a period of time during which a driver may not carry out any work and which is used exclusively for recuperation, a rest period is defined to be an "uninterrupted period during which a driver may freely dispose

of his time" (ibidem). The difference in the meaning may seem petty. But it matters in a *multi-manning* scenario for instance. Suppose there are two drivers in the vehicle to do the driving, and these drivers take turns. Then one driver may take a break while the other is driving but he may not take a rest. In order to take a rest, the truck must be stationary because a driver cannot freely dispose of his time in a moving truck.

Also, there are daily and weekly rest periods. While the definition of a rest period is the same in both cases, there is a major difference between a day and a week, and it is not (only) about their time span. A day is defined to be a 24-hour period, whereas a week "means the period of time between 00.00 on Monday and 24.00 on Sunday" (ibidem), that is, in contrast to a week, a day is only characterized by its duration. Since a day does not have to be a calendar day, there is a certain degree of freedom when it comes to scheduling breaks and rests.

A *regular daily rest period* is a rest period of at least 11 hours, and a *regular weekly rest period* is a rest period of at least 45 hours. By this definition, a regular weekly rest can also be considered as a regular daily rest, and both rest periods also count as break. In this work, the difference in the definitions does not play a role as we will focus on a single driver and a planning horizon of at most a few days. So in the following, a rest period can be viewed as a very long break.

When a Break Becomes Due A driver must take an uninterrupted break of at least 45 minutes after he has accumulated a driving time of 4.5 hours since the last such break. However, a break may be split in exactly two parts which we call the *first split break* and the *second split break*. Only after the second split break, the driver is allowed to drive for another 4.5 hours. A first split break needs to be at least 15 minutes long, a second split break at least 30 minutes. That is, a break split is not penalized in terms of total break duration.

When a Daily Rest Becomes Due There are two cases in which a daily rest becomes due. One is that the driver has accumulated a certain driving time since the last daily rest, and this maximum driving time per day is 9 hours. However, twice a week, a driver is allowed to drive for up to 10 hours before he is prohibited from driving without taking a daily rest. The other case is due to the fact that there must be a daily rest every day, which is relevant if the driver has to spend a significant amount of time doing other work than driving. To be precise, not later than 24 hours after the end of a daily rest period, the driver must be rested again with respect to a daily rest period. With a regular daily rest period of 11 hours for example, this means that only 13 hours may elapse between the end of a daily rest period and the beginning of a new one.

The latter rule may seem to be defined in a rather cumbersome way. This can be explained by the fact that a driver is allowed to take only a *reduced daily rest period* of 9 hours three times a week (between any two weekly rest periods, to be precise). So if the driver is still entitled to take a reduced daily rest period, up to 15 hours may elapse between the end of a daily rest and the beginning of a new one.

Like breaks, daily rests may be split in two parts. The *first split rest* must be at least 3 hours long, whereas the *second split rest* must be at least 9 hours long. That is, unlike a break split, a rest split is penalized in terms of total rest duration. Only after the second split rest, the driver is considered as (completely) rested with respect to a daily rest.

Should two drivers team up and drive together, both drivers must be rested only 30 hours (instead of 24 hours) after the end of the last daily rest. And instead of 11 hours, a daily rest of 9 hours suffices in such a scenario.

When a Weekly Rest Becomes Due A driver must take a weekly rest period no later than at the end of six 24-hour periods from the end of the last weekly rest period. And a driver may accumulate a maximum driving time per week of 56 hours. However, the maximum driving time accumulated in any two consecutive weeks is 90 hours. After the driver has driven for the maximum time, he must no longer drive. As already mentioned, a regular weekly rest is at least 45 hours long. However, due to the definition of a week as a calendar week, the driver is still not allowed to drive before 24.00 on Sunday in case the 45 hours of a weekly rest end earlier.

The notion of a *reduced weekly rest period* exists (at least 24 hours long) but such a reduced rest must be compensated adequately. Here, we omit the details of how this shall be done. In any two consecutive weeks a driver shall take at least two weekly rests, one of which must not be reduced. A weekly rest cannot be split in two parts like a daily rest.

2.1.1.2 Directive 2003/88/EC

Unlike a regulation, a directive is not self-executing but requires implementing measures by the member states of the EU. For example, the German "Arbeitszeitgesetz" implements the provisions of Directive 2003/88/EC, also known as *working time directive* (European Parliament and Council of the European Union, 2003). It is way beyond the scope of this thesis to address the national legislation of all EU member states.

According to the working time directive, every worker is entitled to

- a rest break, provided that the working day is longer than six hours,
- a daily rest period of at least 11 consecutive hours per 24-hour period, and
- an uninterrupted rest period of at least 24 hours plus the 11 hours daily rest per each seven-day period.

In addition, the average working time for each seven-day period, including overtime, shall not exceed 48 hours. Here, we neglect that also certain aspects of night work, shift work and patterns of work are laid down in the directive. Considering night work aspects is beyond the scope of this work.

2.1.1.3 Directive 2002/15/EC

Directive 2002/15/EC is on the "organisation of the working time of persons performing mobile road transport activities" (European Parliament and Council of the European Union, 2002). It applies to all mobile workers employed by undertakings established in a EU member state as well as to self-employed drivers. It contains more specific provisions than the working time directive:

- A mobile worker shall not "work for more than six consecutive hours without a break. Working time shall be interrupted by a break of at least 30 minutes, if working hours total between six and nine hours, and of at least 45 minutes, if working hours total more than nine hours" (ibidem).

- "Breaks may be subdivided into periods of at least 15 minutes each" (ibidem).
- The average weekly working time in road transport may not exceed 48 hours. The maximum weekly working time may be extended to 60 hours only if an average of 48 hours per week is not exceeded within a period of four months.

This directive also contains night work provisions that we ignore again as it is out of scope of this thesis.

2.1.2 United States

In the United States, regulations concerning drivers' working hours are issued by the Federal Motor Carrier Safety Administration (FMCSA) as "Hours of Service of Drivers". Hence, they are commonly referred to as *HOS* (Hours Of Service) rules. Comparable to the EU equivalent, these rules limit the driving and working hours of truck drivers and anyone else operating a commercial motor vehicle (CMV) in the United States.

The first rules date back as far as 1938, whereas the most recent change to the rules is from 2014 (temporary suspension of enforcement of some provisions of the final rule of 2011). In the following, we present the most notable rules effective today, again focusing on property-carrying vehicles only. For a brief history of the HOS rules, we refer to the Federal Motor Carrier Safety Administration (2000).

2.1.2.1 Hours of Service of Drivers (Final Rule 76-FR-81134)

The most recent final rule on the hours of service of drivers was published in 2011 (Federal Motor Carrier Safety Administration, 2011). The terms and definitions of this regulation differ from the EU counterpart. Here, every part of the work schedule of a truck driver is assigned to one of four different times: *Driving time*, *on-duty time*, *off-duty time*, and *sleeper berth time*. On-duty time comprises driving time and time spent for all other work. This includes the time when the driver is waiting to be dispatched, unless he has been relieved from duty by the motor carrier. Off-duty time is any time not spent on-duty or in the sleeper berth. As the name suggests, sleeper berth time is time spent resting in the sleeper berth of a truck, given that the truck provides one. It can be thought of as more restful than off-duty time. Due to this, time spent in a sleeper berth is treated slightly different in the regulation than time being simply off-duty. However, we will ignore this and assume in the following that the truck at hand does not have a sleeper berth. This means, the driver cannot benefit from the so-called "sleeper berth provision" that relaxes some constraints and allows a certain flexibility in regard to when and how to take a daily rest.

When a Break Becomes Due A driver is required to take a break of 30 minutes. To be precise, a driver may drive a CMV only if at most 8 hours have elapsed since the end of the last off-duty period of at least half an hour. It should be noted that this "lunch break rule" is fairly new and only entered into force in 2013.

When a Daily Rest Becomes Due Here, let a daily rest denote a consecutive off-duty period of at least 10 hours. Such a daily rest becomes due after 11 hours of driving or after 14 hours have passed since the end of the last daily rest. Strictly speaking, only driving a CMV is prohibited then, doing other work is still allowed.

This is unlike in the EU, where it is enforced that drivers maintain a circadian rhythm of at most 24 hours.

When a Weekly Rest Becomes Due There is no HOS rule that directly compares to the weekly rules of the European Union. Unlike in the EU, a weekly rest never becomes due but it may well become beneficial.

There is a rule regarding the maximum on-duty time per week. For this rule, it is distinguished whether the employing motor carrier operates CMV every day of the week or not. In the former case, the maximum on-duty time per 8 day period is limited to 70 hours (average of 8.75 hours per day), whereas in the latter case, the maximum on-duty time per 7 day period is limited to 60 hours (average of 8.57 hours per day). After the maximum weekly on-duty time is exceeded, the driver is no longer allowed to drive a CMV until the next day. That means, as long as the driver is on duty for not longer than 8.57 hours per day, the rule does not prohibit driving a CMV every day of the year.

However, a concept exists that is similar to the weekly rules of the EU, and it is commonly referred to as the "34-hour restart provision". Let us call an off-duty period of at least 34 hours a weekly rest. Then, the 34-hour restart provision says that after a weekly rest, the driver is considered as completely rested, and the accumulated on-duty time is reset to zero. The history before that weekly rest no longer counts.

In the original regulation of 2011, it is stated that a weekly rest must span two periods that include 1 a.m. to 5 a.m., and that a restart after a weekly rest may only be considered once every 168 hours. This restriction is currently suspended (United States House Committee on Rules, 2014).

2.2 Related Work

This section gives an overview over the literature on planning problems related to drivers' working hours. Since the truck driver (scheduling and) routing problem is only introduced in this thesis, there is no literature on this, apart from some master theses and some rather remotely related literature. This literature will be reviewed in the respective chapters. In the following, we focus on the literature regarding the truck driver scheduling problem and the vehicle routing and truck driver scheduling problem. In each subsequent chapter, we may once again present some of the literature as far as it is relevant in the scope of the respective chapter.

Truck Driver Scheduling A polynomial-time algorithm for a truck driver scheduling problem (TDSP) is first described by Archetti and Savelsbergh (2009) (as "trip scheduling problem"). As already mentioned in section 1.2.1, they study a problem where there are two conditions under which a break becomes due. It is motivated by the US rules of that time, and the objective is simply to find a feasible schedule if one exists. The authors show that their problem at hand can be solved in $O(n^3)$ time in the case of a single time window per customer (with n being the number of customers).

Goel and Kok (2012b) improve this bound to $O(n^2)$. They also show that the same bound can be achieved in the presence of multiple time windows per customer if these time windows are at least as far apart as the minimum break duration. For the problem with multiple and arbitrarily distributed time windows, they "doubt

that a polynomial bound on the number of schedules generated" can be stated for their scheduling algorithm.

As far as a similar problem variant is concerned (motivated by EU rules this time), Goel and Kok (2012a) present an $O(n^2)$ algorithm for the single time window case. They even study the problem that arises when the standard daily driving time limit may be raised a certain number of times. This is motivated by the EU rule that allows the driver to drive for 10 hours instead of 9 hours per day, but only twice a week. The authors conclude that an $O(n^2)$ bound can still be achieved. However, they regard the setting of the break rule parameters like the daily driving time limit, the break duration, and even the planning horizon as a characteristic of the problem itself, and hence the maximum number of breaks within the planning horizon as a constant. For the general case in which the parameter setting is a part of each problem *instance*, there is no result [1].

To the best of our knowledge, these three papers are the only ones that contain a polynomial-time algorithm. The first paper to consider the hours of service of a driver in the United States is the one by Xu et al. (2003). The authors solve a practical pickup and delivery problem with a variant of the TDSP with multiple time windows as subproblem. In fact, their objective is to minimize the total cost where the cost of a single route is a linear combination of the following components: a fixed part, mileage cost, cost for the break periods, and cost for the residual waiting time. The first two components are irrelevant for the subproblem. For the case that the last two components are equally weighted, their subproblem coincides with the problem variant that asks for the minimum duration. They conjecture that their subproblem is *NP*-hard.

Goel (2010) solves a TDSP with two different break types but only with a single time window per customer. This research is motivated by the rules in the EU, where both breaks and daily rest periods have to be scheduled for a planning horizon of several days. It is also taken account of by the author that both breaks and daily rests are allowed to be split in two parts. The number of partial schedules created during the suggested algorithm may grow exponentially. Dominance rules are given to reduce the number of partial schedules in memory, but still no polynomial bound on the number of non-dominated partial schedules is stated.

Drexl and Prescott-Gagnon (2010) investigate the break rules of the EU comprehensively, though only for the single time window case as well. Since they regard a planning horizon of more than a week, three types of break must be distinguished. Their labeling algorithm finds a legal schedule if one exists. It can even be used to construct a route in case the sequence of customers is not given. But even if it is, again the number of created labels grows exponentially for the exact approach. The authors conjecture that the variant of the TDSP that they study, i.e., a variant with a comprehensive set of EU drivers' rules, is *NP*-complete.

[1] It appears that the complexity analysis in both papers (Goel and Kok, 2012a; Goel and Kok, 2012b) is incomplete. Goel and Kok (2012a) show that the number of iterations of their algorithm is in $O(n^2)$ but they omit to make a statement on the time complexity of each iteration. Here, each iteration comprises a step in which dominated schedules are removed from a set of schedules for a partial route. In their other paper (Goel and Kok, 2012b), the authors prove that the number of non-dominated schedules that their algorithm generates is in $O(n^2)$. However, they again disregard the time it takes to check for dominated schedules. In fact, if checking for dominated schedules implies that different schedules with respect to the same partial route are compared pairwise, then each iteration of the algorithm by Goel and Kok (2012a) may take linear time in the worst case. And in their other algorithm (Goel and Kok, 2012b), the dominance check that is invoked n times may take $O(n^2)$ time. So we conjecture that both algorithms have a time complexity of only $O(n^3)$. However, in the remainder of this thesis, we will assume that the authors are right and that their time complexity statements are correct.

The only approach we are aware of that tries to tackle both the US and the EU variants of the TDSP with a unified approach is developed by Goel (2012c). The author introduces a flexible model that can be configured for several types of breaks, different rulesets and a planning horizon of more than a week. Multiple time windows are also included in the model, and the objective is to find a feasible schedule with minimum duration. However, a severe limitation is that all breaks can only be taken at customers, either before or after service but never en route between customers, unless parking locations are explicitly added to the route as "dummy customers". (In chapter 5, we will call this the "no-break-en-route policy".) Both a MIP formulation and a dynamic programming approach are presented. The number of partial solutions generated by the DP algorithm may grow exponentially, even when the dominance criterion is applied.

For other MIP-based approaches, the same limitation applies: Kok, Hans, and Schutten (2011) regard a variant of the TDSP with one type of break (and for two types of breaks with an extension of the model) where the driving times are considered to be time-dependent. The objective is again to minimize the total duration. Koç et al. (2016) introduce a variant of the TDSP where the objective is to minimize operational and emissions costs. They even include real-world data of parking lots (here: interstate rest areas in the US) into their experimental analysis. A MIP model without the limitation that breaks en route are disallowed is given by Bernhardt et al. (2016). Also included in this model are "soft" time windows, that is, penalties for missed time windows are considered. Based on this work, Bernhardt et al. (2017) present a model which integrates gas stations along the route and decisions about refueling.

Besides the rulesets according to the provisions of the US and the EU, the provisions of other countries have been studied, in particular those of Canada by Goel and Rousseau (2012) and Goel (2012b) as well as those of Australia by Goel, Archetti, and Savelsbergh (2012) and Goel (2012a). Many papers that deal with the provisions in the US were written before the most recent final rule entered into force 2013. This US rule change is explicitly treated by Goel (2014), and an algorithm is proposed (single time window, minimum duration objective, two types of break). Yet again, for the suggested exact algorithms, the number of partial schedules generated may grow exponentially. In another paper, the same author focuses on the impact of night work provisions in the EU, which is not considered in previous works and also not in this thesis (Goel, 2018).

Vehicle Routing and Truck Driver Scheduling In many papers, the truck driver scheduling problem is not treated independently but seen as a subproblem of the vehicle routing problem (VRP) and therefore solved together. But due to the complexity of the combined problem, called the vehicle routing and truck driver scheduling problem (VRTDSP), the scheduling subproblem is usually only solved heuristically but often for a planning horizon of several days. Heuristic approaches for the VRTDSP have been proposed by Xu et al. (2003) and Rancourt, Cordeau, and Laporte (2013) for US provisions (and multiple time windows) as well as by Zäpfel and Bögl (2008), Goel (2009), Prescott-Gagnon et al. (2010), Kok et al. (2010), Derigs, Kurowsky, and Vogel (2011), and Drexl et al. (2013) for different subsets of EU provisions (and the single time window case).

Goel and Vidal (2014) describe how to integrate the exact scheduling approaches for US (Goel and Kok, 2012b; Goel, 2014), Canadian (Goel and Rousseau, 2012), EU (Goel, 2010), and Australian (Goel, Archetti, and Savelsbergh, 2012) provisions into the hybrid genetic vehicle routing algorithm of Vidal et al. (2012). According to their

computational experiments on benchmark instances for the EU rules, 103 of 112 best known solutions were obtained or even improved by their proposed method.

Goel and Irnich (2017) solve the VRTDSP with respect to the current US rules (single time window, two types of break) to optimality by a branch-and-price approach. Based on this, Tilk (2016) describes a faster branch-and-price-and-cut method. Koç, Jabali, and Laporte (2017) introduce a variant of the VRTDSP where - like in the paper by Koç et al. (2016) before - the objective is to minimize operational and emissions costs. Schiffer et al. (2017) study the VRTDSP for electric vehicles and investigate the impact of synchronizing breaks and recharging operations.

It should be noted that Schiffer et al. (2017) claim that their algorithm takes account of both the EU and the US provisions. However, their paper is flawed as far as the presentation of the US hours of service regulation is concerned. In the US, driving is not permitted when more than 8 hours have passed since the end of the last off-duty period of at least 30 minutes (see previous section). In contrast, their proposed algorithm only takes a break after a certain accumulated driving time into account like it is needed for the EU regulation. For instance, a break after 8 hours of accumulated driving time may be too late with respect to US rules if the driver does not only drive but also performs service or has to wait. This means, their algorithm is only correct with regard to EU rules and a planning horizon of one day.

Worth mentioning are those papers that do not relate to a legal regulation but still take lunch breaks or even night breaks into account. In some cases, such as in the paper by Ceselli, Righini, and Salani (2009), the regarded break rules resemble the ones of the EU. In other cases, the regarded break rules demand a break of a certain duration within a given interval. In case of night breaks, it is common to also consider a selection of sleeping locations. Solution approaches can be found in the works of Savelsbergh and Sol (1998), Sahoo et al. (2005), Beaudry et al. (2010), Vidal et al. (2014), and Coelho et al. (2016), amongst others. Also worth mentioning is the work of Bartodziej et al. (2009) who solve combined vehicle and crew scheduling problems with rest constraints. In the dissertation of Meyer (2011), the vehicle routing and truck driver scheduling problem is investigated from a distributed decision making perspective.

Shortly before publishing this thesis, we got to know a very recent technical report by Goel, Vidal, and Kok (2019). They give answers to the question under which circumstances it may be advantageous to assign two drivers to one vehicle.

2.3 Classification of Break Rules

There is a certain gap between a legal text on the one hand and a mathematical optimization problem on the other. And it may not be possible to bridge that gap in every respect due to the very different nature of the two sides. Nevertheless, we present some formalism in this section that we need in order to bring the legal aspects of drivers' working hours (as presented in section 2.1) and the definition of a template for different variants of the truck driver scheduling problem together (as it will be presented in section 2.4).

This section is about classifying the rules laid out in the regulations presented in section 2.1. In some respect, the rules are similar to each other, and in other respects they are very different. Due to the high complexity and diversity of the rules, this classification is far from being complete. It has a clear focus on the rules that our algorithms will be able to take into account. And it helps the reader to see through

the jungle of provisions. It could even be used for a classification and a review of the related literature, even though that is beyond the scope of this thesis.

Our model resembles the one given by Goel (2012c). His paper is the first and the only one so far to contain an approach to classify the diverse constraints. The algorithm presented therein is based on this model and flexible enough to cover both EU and US provisions on breaks.

In section 2.3.1, we first introduce our basic terminology before we then turn towards the major part of this section, the classification of break rules (section 2.3.2).

2.3.1 Basic Terminology

In the following, we present our basic terminology which is again different from both the EU and the US. It is tailored to serve our mathematical model.

Driving Time, Service Time, and Idle Time In our model, not more than three different driver activities need to be distinguished. While *driving time* does not require an explanation, *service time* is the usual term in the literature to subsume other work like loading, unloading, and any paperwork. And then there is time during which the driver is neither driving nor performing service. Let us call this time *idle time*.

For our mathematical model, we assume that whenever the driver neither drives nor performs service, then he does not carry out any other work, can freely dispose of his time, decides to use the time for recuperation, and is relieved from duty by the motor carrier. This assumption includes the definitions of a break, of a rest period (both according to EU legislation), and of off-duty time (according to US legislation). We have to make this general assumption for the sake of conciseness.

Working Time and Travel Time *Working time* and *travel time* are cumulative times. In the following, any driving time and any service time is working time. Travel time denotes the total of driving time, service time, and idle time, i.e., the time of all three driver activities. In other words, the travel time accumulated by the driver since a certain point in time coincides with the time elapsed since then.

Breaks, Types of Breaks, and Waiting Time A *break* is a period of idle time, and it always refers to a *type of break*. Due to our assumption from above, there is only a single attribute that characterizes a type of break in our model: the minimum duration of idle time after which the driver can be considered as (completely) *rested* with respect to this type of break. Should we have to distinguish more than one type of break (it is only chapter 5 in which we deal with two), we give these types names such as “lunch break” and “sleep break”, or “short break” and “long break” to be able to tell the break types apart. We refrain from being too close to the terms used in a regulation to emphasize the abstract level of the mathematical model.

A period of idle time that has (at least) the required length *qualifies* as a break of this type. For a shorter notation, a break of a certain type, say “lunch break”, is simply called a lunch break. If breaks are allowed to be split, even shorter periods of time may count as break or, to be precise, as a part of a split break. (Splitting rules are treated in the next section 2.3.2.) However, a period of idle time may be too short to even be a part of a split break. In the EU for instance, an idle time of less than 15 minutes does not qualify as a break. Such idle time is called *waiting time* in this thesis as it only occurs when the driver has to wait for a customer time window to open. Since such waiting time is not a break, it may count as working time from a

legal point of view. However, in our model, this distinction can be neglected as far as the scope of our work is concerned.

2.3.2 Types of Break Rules

Maybe the most important term is the notion of *break rules*. First of all, there are two kinds of break rules: *restricting* rules and *relaxing* rules. The restricting rules laid out in the regulations have certain similarities: there are some limit values, and whenever a limit is reached, driving (and in some cases even working) is no longer allowed. We distinguish two basic types of restricting rules, namely *recuperation-based* and *horizon-based* break rules. They differ in what the driver has to do in order to be allowed to drive (or work) again. Among the relaxing rules, we solely focus on the *break splitting rules*.

2.3.2.1 Break Splitting Rules

There are two prominent splitting rules, namely the *first-second-split* and the *minimum-length-split* rule. The latter rule allows a split of a break into arbitrarily many parts as long as each part has a minimum duration (see Directive 2002/15/EC). The *first-second-split* rule means a break may be split in exactly two parts, a *first split break* and a *second split break* (in this order). After the first split break, the driver is *partially rested*, after the second he is *completely rested* (with respect to the type of break). To qualify as a second split break, there has to be a first split break before. In the EU for instance, the minimum duration of a daily rest (that is, a break of type "daily rest") is 11 hours, and the splitting rule says that there may be a first split break of at least 3 hours and a second split break of at least 9 hours. Of the two splitting rules mentioned here, the *first-second-split* rule is the only one that we will have a closer look at in this thesis (see section 3.4).

2.3.2.2 Recuperation-Based Break Rules

A recuperation-based rule relates to a certain type of break. It states that after the driver has accumulated certain activity times since the end of the last break of the corresponding type (like accumulated driving time or travel time), the driver needs to recuperate again, i.e., another break of that type must be taken (or completed in case of break splits). Without such a break, the recuperation-based rule prohibits drivers from driving or even working. On the other hand, once the driver is considered as completely rested with respect to the corresponding type, history can be forgotten, and the accumulated times (with respect to the rule at hand) can be reset to zero.

An example from the EU: Only after a break of at least 45 minutes, the driver is allowed to drive for another 4.5 hours. By the end of a 45-minute break, the history before the break becomes irrelevant for this rule. And only after a daily rest of at least 11 hours, the driver is allowed to drive for another 9 hours and to travel for another 13 hours. By the end of a 11-hour rest, the history before the rest becomes irrelevant for this rule.

To sum up, a recuperation-based break rule is characterized by

- the activity times that are accumulated and observed,
- their limit values,
- the activities that are forbidden once a limit is reached, and

- the type of the break that resets the accumulated times.

2.3.2.3 Horizon-Based Break Rules

A horizon-based rule states that after a certain event (like the beginning of a week, the beginning of a day, or the end of a break), a period of fixed length begins in which certain activity times (like driving time or working time) are observed. They may only be accumulated up to certain limits. When a limit is reached, the horizon-based rule prohibits drivers from driving or even working until the *observation period* ends.

For one and the same rule, there can be several such observation periods, and these periods may overlap. Once an observation period has ended, we can stop accumulating times for this period. Other observation periods may still be on-going though.

Some examples: In the EU, the driving time per calendar week is limited. The observation period starts at midnight between Sunday and Monday and ends a week later. Once the maximum driving time per week is reached, the driver has to wait until the end of the observation period to be allowed to drive again. On that date, the accumulated driving time per week is reset to zero, and the history (accumulated driving time of last week) can be forgotten. Also, the driving time within two consecutive calendar weeks is limited. Here, two observation periods overlap. In the US, the on-duty time per 7 (or 8) day period needs to be observed. Accordingly, there are 7 (or 8) overlapping observation periods. The history prior to 7 (or 8) days in the past may be forgotten.

To sum up, a horizon-based break rule is characterized by

- the event that triggers the beginning of an observation period,
- the length of the observation period,
- the activity times that are accumulated and observed,
- their limit values, and
- the activities that are forbidden once a limit is reached.

2.3.2.4 Activity Combinations

Both recuperation-based and horizon-based rules share common attributes, namely the activity times that are accumulated and observed and the activities that are forbidden once a limit is reached. We use a certain short form to write these common rule attributes down. In this thesis, we distinguish the following three activity combinations:

- *drive until driven*: driving is only allowed until the driver has accumulated a certain driving time,
- *drive until traveled*: driving is only allowed until the driver has accumulated a certain travel time,
- *work until traveled*: working is only allowed until the driver has accumulated a certain travel time.

2.3.2.5 Interdependent Break Rules

We may call the recuperation-based and the horizon-based rules basic. As we have already learned from the "34-hour restart provision", rules exist that are merely a mix of the two types. This provision gives the driver the option to take a weekly rest in order to reset the accumulated times but the driver might as well wait until the next day when one of the observation periods ends. Or let us take an example from the EU: Twice a week, a driver is allowed to drive for 10 hours per day instead of only 9 hours. Again, this is rather a mix of a horizon-based rule (counting the number of times the driver has already driven for more than 9 hours a day, reset at the beginning of each week) and a recuperation-based rule (prohibiting the driver from driving as soon as the driving limit is reached).

This highlights the complexity of the provisions in both the EU and the US in general and makes clear that these two basic types of rules are not sufficient to describe all of them. However, these two types are sufficient to describe the rules studied in this thesis.

In the EU and the US, horizon-based break rules come into play when the planning horizon spans roughly a week or more. In this thesis, we study rather elementary truck driver scheduling (and routing) problems and stick to a planning horizon of only a few days at most. In this scenario, we can concentrate on recuperation-based break rules.

2.4 Basic Definitions and Notation

Now that we have presented our classification of rules, we continue with the definition of the *truck driver scheduling problem template* and its parameters (section 2.4.1). This template must be complemented by *rulesets*. Dependent on the applicable rulesets, additional problem parameters come into play. In section 2.4.2, we give examples of rulesets relevant in practice. Finally in section 2.4.3, we introduce some more definitions that will turn out to be helpful.

2.4.1 Truck Driver Scheduling Problem Template

Given is a sequence of n different *customers*. Everyone of these requests a certain service that takes $service_i$ time and has to begin within one of w_i disjoint, arbitrarily distributed *time windows* $\mathcal{W}_i^j$ (for $1 \leq j \leq w_i$ and $1 \leq i \leq n$). The service is performed by a truck driver who has to comply with provisions on breaks when visiting the customers in the given order. The *driving time* between consecutive customers is also given: Let $drive_i$ denote the driving time from customer i to $i+1$ (for $1 \leq i \leq n-1$). In addition, the driver only operates within a time interval $\mathcal{H}$, called *planning horizon*. This means the service at the last customer must be completed before the planning horizon ends. W.l.o.g., the driver is assumed to be located at the first customer when the planning horizon begins, and all customer time windows are assumed to lie within the planning horizon. A summary of parameters is given in Table 2.1.

Every service at a customer must not be interrupted. Apart from that, the driver may wait or take a break at any time. There may be multiple *types of breaks* that need to be distinguished. The conditions under which a driver is obligated to take breaks are specified by a collection of *rulesets*, one per type of break. Here, a ruleset is a collection of *break rules* that all apply to the same type of break. Different rulesets yield different variants of the truck driver scheduling problem. So such rulesets

complement this problem template and define a specific truck driver scheduling problem.

TABLE 2.1: Parameters of Truck Driver Scheduling Template.

n	number of customers
$service_i$	service time demanded by customer i ($1 \leq i \leq n$)
w_i	number of time windows of customer i ($1 \leq i \leq n$)
$\mathcal{W}_i^j$	j-th time window of customer i ($1 \leq i \leq n, 1 \leq j \leq w_i$)
$drive_i$	driving time from customer i to customer $i+1$ ($1 \leq i \leq n-1$)
$\mathcal{H}$	planning horizon

Break Rule Parameters Without break rules, a truck driver scheduling problem is incomplete. In this thesis, we study four different rules. One is the break splitting rule *first-second-split*. The other three are all *recuperation-based* rules. As already mentioned, there are three different activity combinations of interest. Accordingly, we regard three corresponding rules in this thesis. The names of the rules are adopted from those of the combinations, that is, we consider the *drive until driven* rule, the *drive until traveled* rule, and the *work until traveled* rule.

Break rules as well as the types of breaks are parameterized. In the case that there is only one type of break, let *break* be the parameter that denotes the minimum break duration until the driver is considered as rested with respect to this type of break. Should we have to distinguish several break types, then we add the name of the break type as subscript to this parameter (and all the other respective parameters). For instance, we could write $break_{short}$ and $break_{long}$ to tell the minimum length of a short break and a long break apart.

In the following, we assume only one type, so we leave out the subscript. Suppose that a break of this one type is allowed to be split according to the *first-second-split* rule. Then we need two more parameters: Let $break^{1st}$ and $break^{2nd}$ be the parameters that denote the minimum duration of the first split break and the second split break, respectively. W.l.o.g., we demand both periods of time to be smaller than *break* and, conversely, their sum $break^{1st} + break^{2nd}$ not to be smaller. However, the sum of the parts may be greater than *break* in order to penalize the split.

Now let us turn towards the recuperation-based break rules. We introduce two more parameters:

1. Rule *drive until driven*: After *limitD* accumulated driving time since the last (i.e., most recent) break, the driver must no longer drive until he is completely rested.

2. (a) Rule *drive until traveled*: If *limitT* time has elapsed since the end of the last break, i.e., the driver has traveled for *limitT* time, the driver must no longer drive until he is completely rested.

   (b) Rule *work until traveled*: If *limitT* time has elapsed since the end of the last break, i.e., the driver has traveled for *limitT* time, the driver must no longer drive or perform service until he is completely rested. Since the service must not be separated, we assume w.l.o.g. that $service_i \leq limitT$ holds for all customers i whenever this rule is applicable.

Again, should we have to distinguish several break types, then we add the name of the break type as subscript to the parameter. Even though there are three rules, it

suffices to distinguish only two different parameters *limitD* and *limitT* because, as far as the scope of this thesis is concerned, we do not need to consider the rules *drive until traveled* and *work until traveled* at the same time.

Optimization Goal Apart from the rulesets, also different optimization goals are of interest and complement the problem template. In this thesis, we investigate two different objectives:

- *Earliest finish time*: Here, we look for the earliest point in time at which the service at the last customer can be completed. We may speak of an EF-TDSP to denote a truck driver scheduling problem with this objective.
- *Minimum duration*: Here, we want to find the minimum time span between start and finish time, that is, between the beginning of service at the first customer and the end of the service at the last customer. Such a problem may be called an MD-TDSP (see Goel, 2012c).

Of course, such a time can only be returned if a feasible schedule exists after all. For the case that there is none, we introduce the special value $\perp$ that is to be read as "*undefined*". In practice, an algorithm for (a variant of) the truck driver scheduling problem should not only return a time (earliest finish time or minimum duration) but also a feasible schedule that corresponds to that time.

2.4.2 Examples of Concrete Truck Driver Scheduling Problems

Now how are the problems in practice related to this problem template or, more concretely, for which problem variant do we need an algorithm in order to solve a practical problem? Let us have a look at some meaningful examples of rulesets (and their interplay with the planning horizon and the optimization goal). In all these cases we expect the driver to be completely rested (with respect to all relevant types of breaks) at the beginning of the planning horizon.

- Suppose a dispatcher needs to plan trips for single days within the European Union. According to the Regulation (EC) No. 561/2006 of the EU (see section 2.1.1.1), a driver (of a vehicle with a total mass exceeding 3.5t) must no longer drive after an accumulated driving time of $limitD = 4.5\text{h}$. The accumulated driving time is reset after a break of at least $break = 0.75\text{h}$. It is allowed to split the break in two. The first split break must be at least $break^{1st} = 0.25\text{h}$ long and the second split break at least $break^{2nd} = 0.5\text{h}$ long. In practice, it may be okay to not consider a break split. In this case, an algorithm for the ruleset {*drive until driven* } is sufficient to find a feasible schedule. But a shorter schedule may be found by an algorithm for the ruleset {*drive until driven*, *first-second-split* }.

  However, this is only half the truth because we have not yet taken two relevant provisions of the regulation into account. And it remains in the responsibility of the dispatcher to check these as well. First, there is a limit on the maximum driving time between two daily rests. This means the dispatcher has to check beforehand whether the sum of driving times $\sum_{i=1}^{n} drive_i$ exceeds that limit or not. And second, there is a limit on the maximum travel time between two daily rests. Here, the dispatcher has two options: Either he restricts the length of the planning horizon to the limit value a priori. Or he calls an algorithm with the objective to find a schedule with minimum duration. In this case, he checks whether the minimum duration meets the limit a posteriori.

- According to the most recent hours-of-service regulation of the United States (see section 2.1.2.1), a driver must no longer drive after an accumulated *travel time* of $limitT = 8\text{h}$. The accumulated travel time is reset after a break of at least $break = 0.5\text{h}$. So for a planning horizon of a single day in the US, the ruleset {*drive until traveled* } is sufficient.

  But as before, two more checks remain in the responsibility of the dispatcher. Again, this is because there is a limit on the maximum driving time and the maximum travel time between two daily rests. The only difference is that according to US law, the driver is still allowed to perform service after the maximum travel time is reached. To exploit this, the travel time check could be adjusted a little. Let us assume that we have an algorithm for the minimum duration variant, that the driving time to the last customer is greater than zero, and that a time window at the last customer is open when the planning horizon closes (the last two assumptions are without loss of generality). In this case, the dispatcher could create a modified problem instance in which the last customer is always open and requests no service. Then there is a feasible solution for the original instance if there is one for the modified instance that meets the travel time limit.

- According to an older hours-of-service regulation of the United States (Federal Motor Carrier Safety Administration, 2008, meanwhile extended by the provision on rest breaks as mentioned above), a driver must no longer drive after an accumulated driving time of $limitD = 11\text{h}$ or an accumulated travel time of $limitT = 14\text{h}$. The accumulated times are reset after a break of at least $break = 10\text{h}$. So for a planning horizon of one week in the US, the ruleset {*drive until driven*, *drive until traveled* } used to be sufficient. We mention this problem variant because it was studied in the literature before the "rest break provision" entered into force in 2013. Again, there remains an additional a priori check for the dispatcher. He has to make sure that no more than 60 or 70 hours of on-duty time are assigned to the driver within the planning horizon.

- In case of multi-manning, two drivers take turns. One driver may take a lunch break while the other is driving. According to the Regulation (EC) No. 561/2006 of the European Union (see again section 2.1.1.1), both drivers must no longer drive after an accumulated driving time of $limitD = 18\text{h}$ and not even work after an accumulated travel time of $limitT = 21\text{h}$. The accumulated times are reset after a break of at least $break = 9\text{h}$. So for a planning horizon of roughly a week in the EU, an algorithm for the ruleset {*drive until driven*, *work until traveled* } suffices in case of a doubly-manned vehicle.

  However, it is up to the dispatcher to take care that the maximum driving time in two consecutive weeks is not exceeded and that the planning horizon is no longer than $6 \cdot 24$ hours such that enough time for a weekly rest remains.

- Let us stick to a planning horizon of at most $6 \cdot 24$ hours and the provisions of the EU regulation but suppose that there is only a single driver on board. Then we have to distinguish two kinds of breaks which we call "short" and "long" in the following. At least, we need an algorithm for the two rulesets $\{\textit{drive until driven}\}_{short}$ and $\{\textit{drive until driven}, \textit{work until traveled}\}_{long}$, where we set the two minimum break periods to be $break_{short} = 0.75\text{h}$ and $break_{long} = 11\text{h}$, and set the limit values as follows: $limitD_{short} = 4.5\text{h}$, $limitD_{long} = 9\text{h}$, and $limitT_{long} = 13\text{h}$.

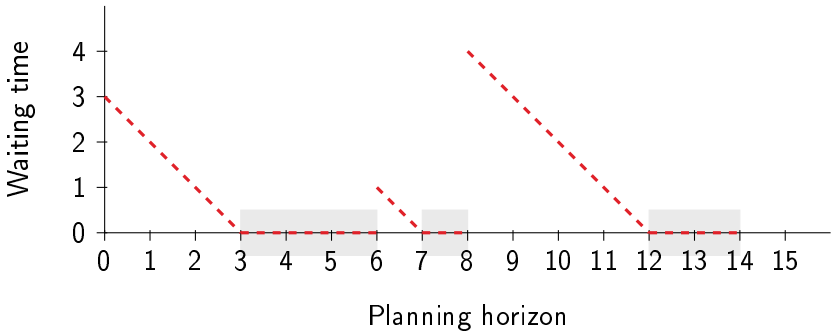


FIGURE 2.1: Example of a waiting time function W. Function in red and dashed. Time windows in the background.

Since both the short and the long break are allowed to be split, it would be even better to have an algorithm for the two rulesets $\{$*drive until driven*, *first-second-split* $\}_{short}$ and $\{$*drive until driven*, *work until traveled*, *first-second-split*$\}_{long}$. In this case, we have to set $break^{1st}_{short} = 0.25$h and $break^{2nd}_{short} = 0.5$h as well as $break^{1st}_{long} = 3$h and $break^{2nd}_{long} = 9$h in addition. The check of the maximum bi-weekly driving time can be conducted beforehand.

2.4.3 Convenient Definitions

Waiting Intervals For the sake of simplicity and w.l.o.g., we assume that time elapses in discrete steps (e.g., in seconds). For the presentation of the algorithms and the examples, we suppose that the step size is 0.1 time units. This means we expect all input time values like the driving and service times to be (non-negative) multiples of 0.1. It also means that for two points in time t and t', the open interval (t, t') and the closed interval $[t + 0.1, t' - 0.1]$ contain the same points in time. We expect two consecutive time windows of the same customer to be at least 0.2 time units apart, because otherwise they could be aggregated. For any time interval, let α and ω map it to the first and the last contained point in time, respectively. For instance, $\alpha(\mathcal{H})$ denotes the beginning of the planning horizon.

For every time window, there is a time interval that precedes it. Precisely, for the sequence of time windows of customer i, let

$$\overline{\mathcal{W}}_i^1 := [\alpha(\mathcal{H}), \alpha(\mathcal{W}_i^1)) \text{ and } \overline{\mathcal{W}}_i^j := (\omega(\mathcal{W}_i^{j-1}), \alpha(\mathcal{W}_i^j)) \text{ for } 2 \leq j \leq w_i$$

define the sequence of the time periods before and between the time windows of customer i. We call these *waiting intervals* as the driver has to wait for a time window to open when he arrives at a customer within such an interval, unless he takes a break.

Waiting Time Functions It is convenient to let $\mathcal{W}_i$ denote two things: an interval set and a multi-interval. In both cases, $\mathcal{W}_i$ contains the time windows of customer i. But depending on context, it allows us to write $\mathcal{W}_i^j \in \mathcal{W}_i$ or $\mathcal{W}_i^j \subset \mathcal{W}_i$ for $1 \leq j \leq w_i$, whichever feels more natural in the context. An analogue statement holds for $\overline{\mathcal{W}}_i$, the interval set (or multi-interval) containing the waiting intervals.

With this, we introduce another notation: For a point in time t, let $\mathcal{W}_i[t]$ be the time window that contains t if $t \in \mathcal{W}_i$, and let $\mathcal{W}_i[t]$ be the time window that follows t if $t \in \overline{\mathcal{W}}_i$. We use this notation to define the time-dependent function W_i that can

be derived from the time windows of customer i. The *waiting time function* $W_i(t)$ maps a time t inside a waiting interval to the duration until the next time window begins, a time within a time window to 0, and a time after the last time window to ∞. Precisely, we set

$$W_i(t) := \begin{cases} 0, & t \in \mathcal{W}_i \\ \alpha(\mathcal{W}_i[t]) - t, & t \in \overline{\mathcal{W}}_i \\ \infty, & \text{otherwise} \end{cases}$$

An example of a waiting time function that is derived from three time windows is shown in Figure 2.1.

Chapter 3

Truck Driver Scheduling with Multiple Time Windows

3.1 Introduction

The *truck driver scheduling problem* (TDSP) is the problem of scheduling a given route of a driver in a way that all customers along the route are visited within their time windows and the provisions in the appropriate regulations are observed. This chapter is the first of three in each of which we present a polynomial-time algorithm for some variants of this problem. In this chapter, we regard three variants of a TDSP with respect to one type of break and multiple, arbitrarily distributed time windows per customer. To be precise, we assume that there are two conditions under which a break becomes mandatory. One is that the driver has accumulated a certain driving time since the end of the last break. The other is that simply a certain time has elapsed since the end of the last break. In a third variant, we allow a break to be split in two parts. For all three variants, we present a polynomial-time algorithm.

Let us shortly recall the most relevant literature (see also section 2.2) to assess our contribution. Archetti and Savelsbergh (2009), Goel and Kok (2012b), and Goel and Kok (2012a) each describe a truck driver scheduling problem with one type of break. To the best of our knowledge, these variants are the only truck driver scheduling problems currently known to be solvable in polynomial time – in the single time window case. The considered rulesets differ only slightly among each other. Archetti and Savelsbergh (2009) and Goel and Kok (2012b) study the ruleset {*drive until driven*, *drive until traveled* } (please recall our notation introduced in section 2.4). Their research is motivated by the provisions effective in the United States before the year 2013. At that time, a 30-minute lunch break was not mandatory, and thus the considered ruleset was sufficient for a planning horizon of several days. In the paper by Goel and Kok (2012a), the ruleset {*drive until driven*, *work until traveled* } is regarded. This, in turn, is motivated by those provisions of the European Union that are applicable in a multi-manning scenario. If two drivers take turns, one driver can take the mandatory 45-minute lunch break while the other is driving. Again, the planning horizon could be several days long.

In the following, we also consider these two rulesets. We will refer to the ruleset {*drive until driven*, *drive until traveled* } as *RulesetUS* and to the ruleset {*drive until driven*, *work until traveled* } as *RulesetEU*. As a third variant, we discuss the ruleset *RulesetEU* together with the splitting rule *first-second-split*. For this variant, a polynomial-time algorithm has not been proposed so far, to the best of our knowledge.

In the above mentioned papers, the authors introduce their respective problem as a decision problem. The question to be answered is whether a schedule exists that is in accordance with the respective ruleset. However, we regard an optimization

problem, where the goal is to find a feasible solution with *earliest finish time*, that is, a feasible schedule with the earliest completion time of the last service, if a feasible schedule exists after all. It should be noted that the algorithms described in the three mentioned papers can easily be adjusted accordingly. The regarded optimization problem is not (asymptotically) harder than the decision problem variant.

Table 3.1 summarizes the findings from the literature in respect of a polynomial time bound for the *RulesetUS*-EF-TDSP and the *RulesetEU*-EF-TDSP (see also section 2.2). The bound found by Archetti and Savelsbergh (2009) was later improved by Goel and Kok (2012b) and is therefore not in the list. As we can conclude from this table, there is currently no result in case of multiple, arbitrarily distributed time windows per customer, whereas there is a result in the special case that the time windows are at least *break* apart. However, we question this result. For a given point in time at some customer i, it takes $O(\log w_i)$ time to find a matching time window in a sorted list. This time is disregarded here. We doubt that a time bound can be given that is completely independent from the number of time windows, at least without further assumptions. However, if we assumed a maximum length of the planning horizon, then this would imply – together with the assumption that consecutive time windows are at least *break* apart – that the number of time windows of each customer is bounded from above.

TABLE 3.1: Results from literature on polynomial time bounds (EF-TDSP).

Source	Ruleset	Limitation	Bound
Goel and Kok, 2012b	*RulesetUS*	single time window per customer	$O(n^2)$
Goel and Kok, 2012b	*RulesetUS*	time windows are at least *break* apart	$O(n^2)$
Goel and Kok, 2012a	*RulesetEU*	single time window per customer	$O(n^2)$

Contribution and Outline We present the first polynomial-time algorithm for both the *RulesetUS*-EF-TDSP and the *RulesetEU*-EF-TDSP as well as the *RulesetEU*+-EF-TDSP in the presence of multiple (and arbitrarily distributed) time windows per customer. Besides the use cases described in the original papers, there are some more use cases of interest. For instance, this algorithm can cover the truck driver scheduling problem as it occurs both in the EU and in the US for a planning horizon of one day (see the examples given in section 2.4.2). Thus, it is the only polynomial-time algorithm that is designed to handle the provisions of more than just one regulation.

The problem definition is given in section 3.2. The description of our solution approach follows in sections 3.3 and 3.4 (for the extension by break splits), before we conclude in section 3.5.

3.2 Problem Definition

The problem definition is based on the template introduced in section 2.4.1. Since we only deal with one type of break, we can leave out a subscript on the break rule parameters that would otherwise be necessary to denote the type. There are three break rule parameters with which we can cover the three break rules *drive until driven*, *drive until traveled*, and *work until traveled*: The parameter *break* denotes the minimum break duration after which the driver is considered as (completely) rested. The two limit values *limitD* and *limitT* denote the maximum accumulated driving

time and the maximum accumulated travel time without a break, respectively. It is w.l.o.g. when we assume that $limitD \leq limitT$ holds. When the planning horizon begins, we expect the driver to be rested.

The only thing missing now is a formal definition of a truck driver schedule. It is time to catch up on this in the next section 3.2.1. In section 3.2.2, we investigate the characteristics of this problem.

3.2.1 Definition of a Truck Driver Schedule

In this section, we give a formal definition of a truck driver schedule and state constraints that a feasible schedule needs to satisfy. In the following, let a *truck driver schedule* be a sequence

$$\left((t_i^{arr@c}, t_i^{start}, t_i^{dep@c}, t_i^{fb@r}, b_i, t_i^{lb@r})\right)_{1 \leq i \leq n}$$

of n tuples, one for each customer. The first three values of the i-th tuple refer to points in time at customer i. The arrival time at the customer is denoted by $t_i^{arr@c}$, the start time of the service by t_i^{start}, and the time of departure by $t_i^{dep@c}$. The other three values refer to the route between customer i and customer $i+1$. The time when the first break en route begins is $t_i^{fb@r}$, the number of breaks scheduled en route is b_i, and finally, $t_i^{lb@r}$ is the time when the last of the b_i breaks ends. For the sake of consistency, we expect the time values $t_i^{fb@r}$ and $t_i^{lb@r}$ to be given even if $b_i = 0$ and also for $i = n$. For instance, they could be set equal to $t_i^{dep@c}$ in these cases. Overall, a truck driver schedule is feasible only if the following basic conditions hold:

$$\alpha(\mathcal{H}) \leq t_1^{arr@c} \tag{3.1}$$

$$t_i^{arr@c} \leq t_i^{start} \qquad \text{for all } i \leq n \tag{3.2}$$

$$t_i^{start} \in \mathcal{W}_i \qquad \text{for all } i \leq n \tag{3.3}$$

$$t_i^{dep@c} - t_i^{start} \geq service_i \qquad \text{for all } i \leq n \tag{3.4}$$

$$0 \leq t_i^{fb@r} - t_i^{dep@c} \leq limitD \qquad \text{for all } i \leq n \tag{3.5}$$

$$t_i^{lb@r} - t_i^{fb@r} \geq b_i \cdot break \qquad \text{for all } i \leq n \tag{3.6}$$

$$b_i = 0 \Rightarrow t_i^{lb@r} - t_i^{fb@r} < break \qquad \text{for all } i \leq n \tag{3.7}$$

$$0 \leq t_{i+1}^{arr@c} - t_i^{lb@r} \leq limitD \qquad \text{for all } i \leq n-1 \tag{3.8}$$

$$b_i \leq 1 \Rightarrow (t_i^{fb@r} - t_i^{dep@c}) + (t_{i+1}^{arr@c} - t_i^{lb@r}) \geq drive_i \qquad \text{for all } i \leq n-1 \tag{3.9}$$

$$b_i \geq 2 \Rightarrow (t_i^{fb@r} - t_i^{dep@c}) + (t_{i+1}^{arr@c} - t_i^{lb@r}) + \min\{t_i^{lb@r} - t_i^{fb@r} - b_i \cdot break, (b_i - 1) \cdot limitD\} \geq drive_i \qquad \text{for all } i \leq n-1 \tag{3.10}$$

$$t_n^{lb@r} \leq \omega(\mathcal{H}) \tag{3.11}$$

Constraints 3.6 and 3.7 ensure consistency of $t_i^{fb@r}$ and $t_i^{lb@r}$ with b_i. With Constraint 3.7, it is guaranteed that the waiting time between $t_i^{fb@r}$ and $t_i^{lb@r}$ counts as break (that is, $b_i > 0$) when it is at least as long as *break*. Constraints 3.9 and 3.10 enforce that enough driving time is scheduled. If there is at most one break en route, the scheduled driving time between two customers is the time between departure and arrival less the time spent for the break. In case of at least two breaks, we have to add the maximum possible driving time between those breaks. Between $t_i^{fb@r}$ and

$t_i^{lb@r}$, the driver cannot drive for more than $t_i^{lb@r} - t_i^{fb@r} - b_i \cdot break$ and not more than $(b_i - 1) \cdot limitD$ without violating rule *drive until driven*.

The constraints above are not sufficient to check compliance with the rules completely. Before we present the remaining constraints, we introduce three sequences in order to simplify notation. The first one is the sequence of *arrival times* $\mathcal{A}$. It contains not only the arrival times $t_i^{arr@c}$ for every i but also the break start times $t_i^{fb@r}$ en route, which can be thought of as the arrival times at some implicit parking areas. These are all points in time at which a break may begin (apart from those implicit breaks en route if $b_i > 1$). The sequence of *departure times* $\mathcal{D}$ is defined in an analogue way as $\mathcal{A}$. It contains not only the departure times $t_i^{dep@c}$ for every i but also the break end times $t_i^{lb@r}$ en route, so to say the departure times from some implicit parking areas. The third sequence $\mathcal{B}$ contains all t_i^{start} and all $t_i^{lb@r}$. These are all points in time at which a break may end (apart from those implicit breaks en route if $b_i > 1$). Should there be a break scheduled right after service, we treat this break as a break en route. So in this case $t_i^{fb@r}$ equals $t_i^{dep@c}$. This also means that the period from t_i^{start} to $t_i^{fb@r}$ is considered to never contain a break. All these sequences contain $2n$ points in time each.

$$\begin{aligned}
\mathcal{A} &:= (t_1^{arr@c}, t_1^{fb@r}, t_2^{arr@c}, t_2^{fb@r}, \ldots) \\
\mathcal{B} &:= (t_1^{start}, t_1^{lb@r}, t_2^{start}, t_2^{lb@r}, \ldots) \\
\mathcal{D} &:= (t_1^{dep@c}, t_1^{lb@r}, t_2^{dep@c}, t_2^{lb@r}, \ldots)
\end{aligned}$$

For the sake of simplicity, let the truck driver be completely rested at the beginning of the planning horizon. Then, for the rule *drive until driven*, we constrain in addition to the basic conditions:

$$\begin{aligned}
&\sum_{k'=j}^{\ell} \mathcal{A}[k'+1] - \mathcal{D}[k'] > limitD \Rightarrow \\
&\quad \exists k : j < k \leq \ell \wedge \mathcal{B}[k] - \mathcal{A}[k] \geq break
\end{aligned} \qquad \text{for all } j, \ell : j < \ell < 2n \qquad (3.12)$$

The time from $\mathcal{D}[k']$ to $\mathcal{A}[k'+1]$ for some index $k' < 2n$ is a driving period. Whenever the sum of consecutive driving periods, say from some index j to some index ℓ, exceeds the driving time limit, then there must be a break scheduled in between, that is, there must be an index k with $j < k \leq \ell$ such that there is enough buffer for a break between $\mathcal{A}[k]$ and $\mathcal{B}[k]$.

For the rule *drive until traveled*, we have to make sure that the driver does not drive when the travel time limit is exceeded. This time we need two more constraints:

$$t_i^{dep@c} < t_i^{fb@r} \Rightarrow t_i^{fb@r} - t_i^{start} \leq limitT \qquad \text{for all } i \leq n \qquad (3.13)$$

$$\begin{aligned}
&\mathcal{D}[\ell] < \mathcal{A}[\ell+1] \wedge \mathcal{A}[\ell+1] - \mathcal{B}[j] > limitT \Rightarrow \\
&\quad \exists k : j < k \leq \ell \wedge \mathcal{B}[k] - \mathcal{A}[k] \geq break
\end{aligned} \qquad \text{for all } j, \ell : j < \ell < 2n \qquad (3.14)$$

Constraint 3.13 is an analogue of constraint 3.5. Since there cannot be a break between t_i^{start} and $t_i^{fb@r}$, the time in between must not exceed the travel time limit unless there is no driving time before $t_i^{fb@r}$. Constraint 3.14 resembles Constraint 3.12. Whenever there is a non-degenerate driving period from $\mathcal{D}[\ell]$ to $\mathcal{A}[\ell+1]$ for some $\ell < 2n$ and, for some earlier $j < \ell$, the travel time from the (potential) end of a break $\mathcal{B}[j]$ to the (potential) beginning of a break $\mathcal{A}[\ell+1]$ exceeds the travel time limit, then there

must be a break scheduled in between.

For the rule *work until traveled*, we again need two more constraints in order to ensure that the driver does not work when the travel time limit is exceeded:

$$t_i^{fb@r} - t_i^{start} \leq limitT \qquad \text{for all } i \leq n \qquad (3.15)$$

$$\mathcal{B}[\ell] < \mathcal{A}[\ell+1] \wedge \mathcal{A}[\ell+1] - \mathcal{B}[j] > limitT \Rightarrow \exists k : j < k \leq \ell \wedge \mathcal{B}[k] - \mathcal{A}[k] \geq break \qquad \text{for all } j, \ell : j < \ell < 2n \qquad (3.16)$$

In contrast to the Constraints 3.13 and 3.14, the Constraints 3.15 and 3.16 must hold even if the driving time immediately before some arrival is 0. Similar to Constraint 3.14, Constraint 3.16 says that whenever there is work scheduled before $\mathcal{A}[\ell+1]$ and the travel time from $\mathcal{B}[j]$ to $\mathcal{A}[\ell+1]$ for some $j < \ell$ exceeds the travel time limit, then there must be a break scheduled in between.

With our definition, only the beginning of the first break and the end of the last break en route is determined. If more than one break becomes due en route, the start and end times of all these breaks can be computed recursively: 1. Every break other than the last ends *break* time units later than it begins. 2. Every break other than the first begins *limitD* time units later than the previous break ends unless $t_i^{lb@r} - t_i^{fb@r} < b_i \cdot break + (b_i - 1) \cdot limitD$. In this case, the next break may have to begin earlier than after *limitD* time.

It is our aim to describe an algorithm that is polynomial time bound regardless of the setting of the parameters *limitD*, *limitT*, *break*, and $\mathcal{H}$. To this end, it is crucial to only determine $t_i^{lb@r}, t_i^{fb@r}$, and b_i and not the beginning and end of each and every break en route because the recursion above may not be polynomial in the input size. Suppose we had $break = 1$ and $limitD = limitT = 1$ and $\mathcal{H} = [0, \infty)$, and let $drive_1$ tend towards infinity. Then the run-time of an algorithm that specifies all start and end times of breaks is not polynomial in the input size. Accordingly, the algorithm presented by Goel and Kok (2012b) is only polynomial if we regard the setting of the parameters *limitD*, *limitT*, *break* and $\mathcal{H}$ as problem immanent constants so that there is a maximum number of breaks that can be scheduled en route.

3.2.2 Problem Characteristics

Before we describe an algorithm, we shed light on the characteristics of the problem. To this end, we look at the problem from the perspective of a single driver and consider one important question: What are the decisions that the driver has to make and, on the other hand, when is it clear for the driver what to do next because any alternative action would have no advantage? Here, we characterize the state of a driver by the following attributes:

- the progress along the route, i.e., how much of the work (driving and service) has been accomplished,
- the current point in time,
- the accumulated driving time and the accumulated travel time since the end of the last break.

Let us accompany the driver from one customer to another. For a start, let us suppose the driver has just arrived at a customer. Then he has to decide whether to take a break immediately or not, even if a break is not yet due. We call such a break an *early break*. The decision for an early break is certainly sensible if otherwise the

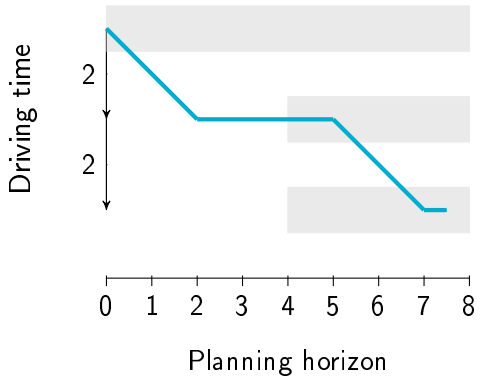


FIGURE 3.1: In this example, taking an early break is beneficial.

driver had to wait for the opening of a time window and the waiting time was at least as long as the minimum break period. But it may also be beneficial even if the waiting time was shorter than the minimum break period.

An example is depicted in Figure 3.1. Let $limitD = limitT = 3$, $break = 3$, both driving times be 2, the service times at the first and second customer be 0, and the service time at the third customer be 0.5. The driver arrives at the second customer at time 2, begins a break at the same time, and arrives at the third customer before the time window closes. If the driver had waited until time 4 and scheduled the break only when it becomes due, he would have missed that time window. But on the other hand, an early break would not be beneficial at all if the driving time from the second to the third customer was only 0.5 because then a break would simply be unnecessary.

If the driver decides to take an early break, there is another decision to be made. Due to the rules *drive until traveled* and *work until traveled*, it is always preferable to schedule the end of a break as late as possible, that is, it may be a good idea to not only take a break of minimum duration but to prolong the break. It is always best to not end a break as long as no time window is open. But even if a time window is already open, it may still be beneficial to extend the break even further.

Figure 3.2 shows an example. Let the parameters be the same as in the previous example. The driver arrives at the second customer at time 3, begins a break at the same time, but ends this break only at time 8 in order to arrive at the third customer at time 10. If the driver had left the second customer immediately after the minimum break period, the rule *work until traveled* would have forced him to take a second break before the service at the third customer due to inevitable waiting time, so the service could not start before time 11.

As a next step, the driver has to wait for the beginning of the next time window if there is currently no time window open. It is not worthwhile to wait even longer. Should there be no next time window, the partial schedule of the driver cannot be continued in a feasible way. After waiting, the driver is ready to perform the requested service. But in case of the rule *work until traveled*, he has to make sure that the service can be completed before a break becomes due. And if it cannot be completed, the partial schedule cannot feasibly be continued. In this case, the driver should have decided to take an early break before. In all other cases, the driver begins with the service immediately.

After service, the driver departs from the customer. En route from one customer to the next, the driver takes a break only when it becomes due. Before arrival at the next customer, it is not advantageous to wait or to take an early break because it is

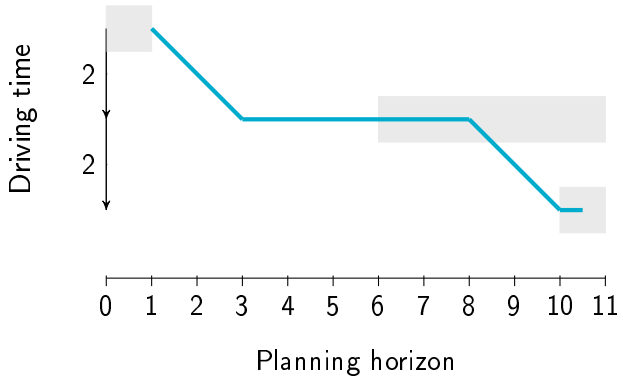


FIGURE 3.2: In this example, prolonging a break is beneficial.

sufficient to take all that into account when the driver arrives there. But as before, it may be beneficial to prolong a break, once it must be taken. So again, the driver has then to decide when to end the break.

Let us summarize our findings: If we want to find a feasible schedule for the driver, we have to take an early break immediately on arrival at the next customer into account, and we have to consider every possible (or at least every relevant) end of a scheduled break. Apart from that, the next action of the driver can simply be derived from his current state. But still, the number of nodes in a decision tree may explode, so a sophisticated approach is crucial.

3.3 Solution Approach

We now turn towards the EF-TDSP, i.e., we seek to find a feasible truck driver schedule with earliest finish time. In 3.3.1, we introduce the notion of a driver states label and define its content. The algorithm is then presented in two subsequent sections. In 3.3.2, we give an overview on the algorithm that finds the earliest finish time of a feasible truck driver schedule. In 3.3.3, we describe the main part of the algorithm in greater detail. In 3.3.4, we show how to derive a schedule that accomplishes this finish time. In 3.3.5, we analyze the algorithm and prove the polynomial time bound.

3.3.1 Driver States Label

In many truck driver scheduling algorithms from the literature, there is a label for every feasible state of a driver. In our algorithm, this is different as we store all feasible driver states with the same progress in the same label. To be precise, a *driver states label* $\mathcal{L}$ comprises two *time-dependent functions* D and T. For a point in time t, $D(t)$ denotes the *minimum* accumulated driving time and $T(t)$ the *minimum* accumulated travel time since the end of the last (i.e., most recent) break among all feasible driver states with the same progress. As we will see, the functions D and T are piecewise constant.

For a point in time t, it is sufficient to store only one pair of values $(D(t), T(t))$. Suppose there are two drivers with the same progress at the same time t but one of the drivers has a lower accumulated driving time and a higher accumulated travel time compared to the other driver. This situation is depicted in Figure 3.3. Here, the wavy lines indicate that the exact course of the schedule does not matter. The end of the last break in the blue schedule is at time t_1, whereas the end of the last break in the red schedule is at time t_2. But the break in the blue schedule may be prolonged

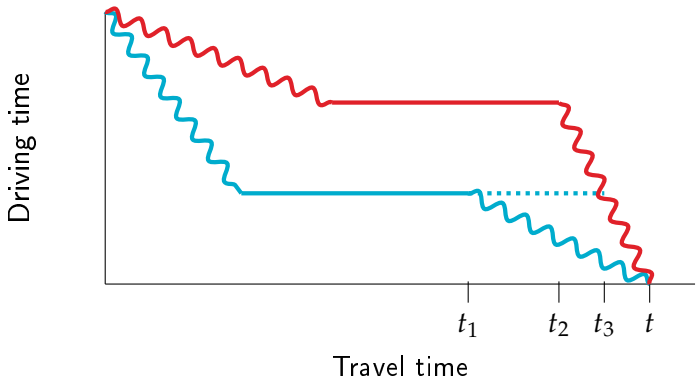


FIGURE 3.3: Two schedules, each with a different end of the last break.

until t_3. This means, a blue-red schedule exists that coincides with the beginning of the blue schedule until t_1 and the end of the red schedule from t_3. In between, the break is extended. This schedule is feasible because the driver is completely rested at time t_3. And the state of a driver that follows the blue-red schedule dominates the states of the other two drivers, so it is sufficient to store the accumulated times of only that driver at time t.

On the other hand, for one and the same progress, it is not necessary to store a pair of values for every point in time. This is certainly true for every point in time before the planning horizon begins, but not only. Since the driver is always allowed to wait and the accumulated times are supposed to be minimum, it follows that $D(t_1) \geq D(t_2)$ and $T(t_1) + t_2 - t_1 \geq T(t_2)$ must hold for two points in time $t_1 < t_2$. But if equality holds in both cases, there is no need to store the accumulated times for t_2 because they could be derived from those of t_1. So we introduce the special value $\perp$ to indicate this. This special value is to be read as "*undefined*". That is, we say a time-dependent function like D or T is undefined at time t if this function is in fact defined to be $\perp$ at this time. In our algorithm, $D(t) = \perp$ if and only if $T(t) = \perp$. For a driver states label $\mathcal{L} = (D, T)$, let $\alpha(\mathcal{L})$ be the first and $\omega(\mathcal{L})$ be the last point in time for which the two functions D and T are defined, or $\perp$ if there is none.

3.3.2 Outline and Initialization of the Algorithm

Our algorithm works in a breadth-first-like manner and goes through as many iterations as there are customers, so n iterations. Then again we subdivide every iteration into four steps, based on our observations in section 3.2.2:

1. In step *Setup*, we consider an early break and its prolongation.
2. In step *Wait*, we regard the time windows of the current customer and schedule waiting time if necessary.
3. In step *Serve*, we make sure that the service can be completed in time.
4. In step *Drive*, we schedule due breaks en route to the next customer if necessary. We also take their prolongation into account.

So the progress of a driver can be characterized by the iteration and the completed step. Accordingly, our algorithm computes labels $\mathcal{L}_i^{step} = (D_i^{step}, T_i^{step})$ in every iteration i, where the superscript $step$ is one of the following: $setup, waited, served,$

or *driven*. How this is done is described in section 3.3.3. Table 3.2 summarizes the contents of a label in case of the EF-TDSP.

Algorithm 1: Generic truck driver scheduling algorithm

Input : $\mathcal{L}_0^{driven}$
Output: $\texttt{obj}(\mathcal{L}_n^{served})$
forall $i = 1 \dots n$ **do**
  $\mathcal{L}_i^{setup} := \texttt{Setup}\,(\mathcal{L}_{i-1}^{driven})$;
  $\mathcal{L}_i^{waited} := \texttt{Wait}\,(\mathcal{L}_i^{setup})$;
  $\mathcal{L}_i^{served} := \texttt{Serve}\,(\mathcal{L}_i^{waited})$;
  if $i = n$ **then**
    return $\texttt{obj}\,(\mathcal{L}_n^{served})$;
  $\mathcal{L}_i^{driven} := \texttt{Drive}\,(\mathcal{L}_i^{served})$;

In order to convey the big picture, we first present pseudo-code of the *generic truck driver scheduling algorithm* in Algorithm 1. This algorithm requests the driver states label $\mathcal{L}_0^{driven}$ as input. We need to set this input parameter according to the state of the driver when the planning horizon begins. For the sake of simplicity and without loss of generality, let us assume that the driver is completely rested at that time, that is, the accumulated times are then both 0. But not only then. Our assumption implies that the driver must have started a break at least *break* time before. And this implicit break can be prolonged beyond the beginning of the planning horizon. To take that into account, we set the accumulated times to 0 for every point in time within the planning horizon:

$$\mathcal{L}_0^{driven}(t) := \begin{cases} (0,0), & \text{for } t \in \mathcal{H} \\ (\bot,\bot), & \text{otherwise} \end{cases}$$

Having set $\mathcal{L}_0^{driven}$, we then apply Algorithm 1. In case of the EF-TDSP, the objective of this algorithm is to return the earliest finish time or the information that there is no feasible schedule. To this end, we simply set $obj(\mathcal{L}_n^{served}) := \alpha(\mathcal{L}_n^{served})$, that is, we set it to the earliest point in time for which a feasible driver state after the service at the last customer exists (or $\bot$ if there is none). Algorithm 1 does not provide an actual driver schedule that corresponds to the earliest finish time. How to deduce one from the computed driver states labels is described in section 3.3.4.

TABLE 3.2: Driver states label summary (EF-TDSP).

$step$	one of {*Setup*, *Wait*, *Serve*, *Drive* }
D_i^{step}	time-dependent function mapping points in time at the end of step $step$ in iteration i to minimum accumulated *driving times* since the end of the last break
T_i^{step}	time-dependent function mapping points in time at the end of step $step$ in iteration i to minimum accumulated *travel times* since the end of the last break
$\mathcal{L}_i^{step}$	the pair (D_i^{step}, T_i^{step}) of two functions comprising all feasible (and non-dominated) driver states at the end of step $step$ in iteration i

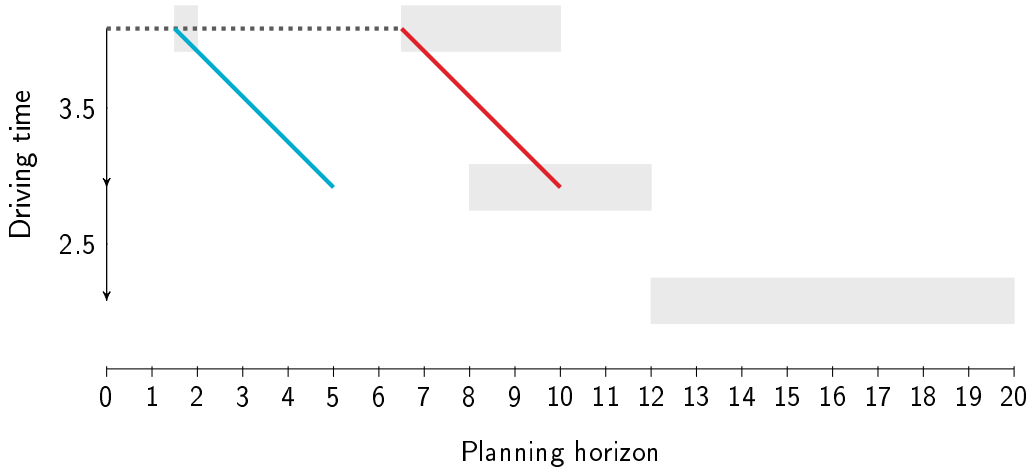


FIGURE 3.4: Example instance and two significant (partial) schedules.

3.3.3 Steps of Algorithm in Detail

In this section, we describe the steps of the algorithm that finds the earliest finish time and illustrate these steps by means of an example. Let the example instance be as follows: The driver starts completely rested at the first customer and needs to visit two more customers within the planning horizon $[0, 20]$. The driving times are $drive_1 = 3.5$ and $drive_2 = 2.5$, and for the sake of simplicity, the service times are all 0. Let the time windows be $\mathcal{W}_1 = ([1.5, 2], [6.5, 10]), \mathcal{W}_2 = ([8, 12])$, and $\mathcal{W}_3 = ([12, 20])$. This instance is depicted in Figure 3.4. For the example we choose the break parameters to be $break = 5.5, limitD = 4.5$, and $limitT = 6.5$. (This means half of the usual values in hours in the EU.) Since service times are 0, the rules *drive until traveled* and *work until traveled* coincide. Figure 3.4 shows not only the example instance but also two different (partial) schedules: The driver departs from customer 1 as soon as either the first or the second time window opens. A break does not become due en route.

Figure 3.5 shows the driver states label $\mathcal{L}_1^{driven}$ in case of this example instance. It contains the minimum accumulated driving and travel times for a driver that has just arrived at customer 2. Since the driving time from customer 1 is 3.5, so are the minimum accumulated times. And the label is only defined over those intervals that correspond to the time windows of the first customer shifted by 3.5 to the right.

In the following sections, we present every step of the algorithm in two different ways. One is a rather mathematical formulation that describes a driver states label as a pair of two time-dependent functions (see section 3.3.1). The other formulation is closer to an implementation. Here, a driver states label is represented by a sorted sequence of *pieces*, where a piece comprises a time interval and a pair of two values that correspond to the values of the functions over that interval. For the k-th piece of a label, we use the short notation $D[k]$ and $T[k]$ to denote the two stored values, the minimum accumulated driving time and the minimum accumulated travel time of the piece, respectively. Furthermore, let $\alpha[k]$ and $\omega[k]$ denote the beginning and the end of the interval of the k-th piece of the label. The intervals of the pieces do not overlap so the chronological sorting of the pieces is well-defined. For the pseudo-code we do not need the special value $\perp$ because a piece is simply missing where the value of the two functions is $\perp$. Using this alternative formulation, the label $\mathcal{L}_0^{driven}$ is initialized by adding exactly one piece to the (initially empty) sequence of pieces, and that piece is defined over the same interval as the planning horizon and has the

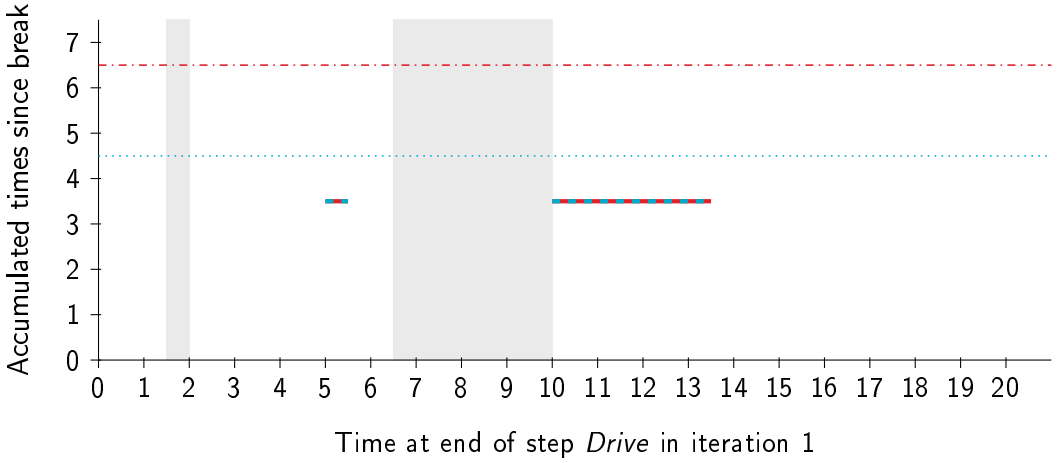


FIGURE 3.5: Driver states label $\mathcal{L}_1^{driven}$. Functions D_1^{driven} (blue) and T_1^{driven} (red) are both the same. Limits *limitD* (blue, dotted) and *limitT* (red, dash-dotted), and time windows of current customer in the background.

value $(0, 0)$.

3.3.3.1 Step *Setup*

In this step, $\mathcal{L}_i^{setup}$ is computed from $\mathcal{L}_{i-1}^{driven}$. So initially, we know the driver states for every (significant) point in time at which the driver may arrive at customer i in a feasible way. Immediately after arrival, the driver may take an early break before he starts with the service in order to avoid unproductive waiting time. This break may even be prolonged if beneficial. In this step, we take such an early break into account.

The earliest time of arrival at customer i is $\alpha(\mathcal{L}_{i-1}^{driven})$. If the driver starts a break immediately, the driver can be considered as completely rested *break* later at time $t^* := \alpha(\mathcal{L}_{i-1}^{driven}) + break$. So we can set the minimum accumulated times since the end of the last break to $(0, 0)$ for that point in time. And since a break may be prolonged, we set the driver states label to $(0, 0)$ for every point in time thereafter until the planning horizon ends:

$$D_i^{setup}(t) := \begin{cases} D_{i-1}^{driven}(t), & \text{for } t < t^* \\ 0, & \text{for } t^* \leq t \leq \omega(\mathcal{H}) \\ \bot, & \text{otherwise} \end{cases}$$

$$T_i^{setup}(t) := \begin{cases} T_{i-1}^{driven}(t), & \text{for } t < t^* \\ 0, & \text{for } t^* \leq t \leq \omega(\mathcal{H}) \\ \bot, & \text{otherwise} \end{cases}$$

This label gives us the minimum accumulated times for every point in time at which the driver can start with the service at customer i, not regarding the customer's time windows though.

Function Setup shows pseudo-code to compute the pieces of $\mathcal{L}_i^{setup}$. Like $\alpha(\mathcal{L}_{i-1}^{driven})$, $\alpha[1]$ is when the first piece of the input label begins. Due to the assumption that time elapses in discrete steps of 0.1 time units, *endOfFirstPeriod* is the last point in time

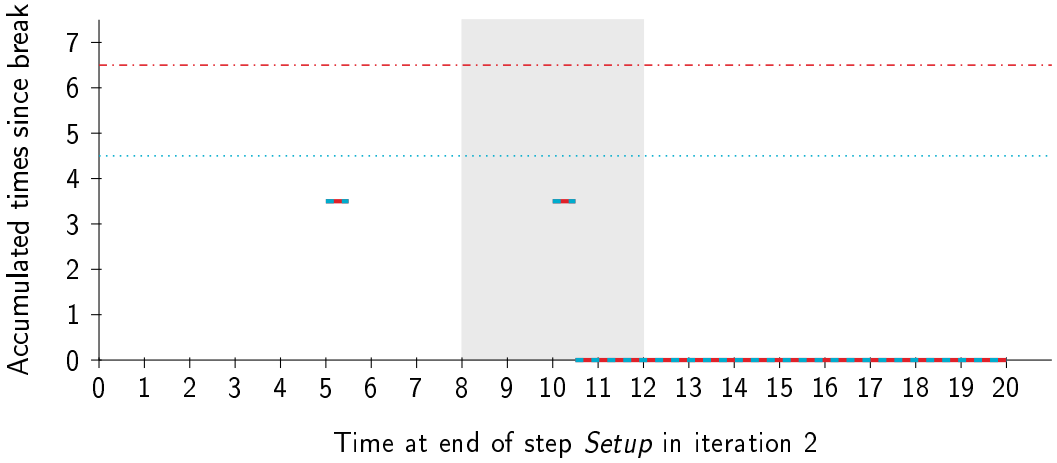


FIGURE 3.6: Driver states label $\mathcal{L}_2^{setup}$.

up to which pieces of $\mathcal{L}_{i-1}^{driven}$ are copied. After that, another piece with value $(0,0)$ is appended, provided that the planning horizon does not end earlier.

As far as our example instance is concerned, the input label $\mathcal{L}_{i-1}^{driven}$ is depicted in Figure 3.5, and the output label $\mathcal{L}_2^{setup}$ looks like as in Figure 3.6. The only difference is that both functions are now 0 from time 10.5 onwards.

Function Setup($\mathcal{L}_{i-1}^{driven}$)

$\mathcal{L}_i^{setup} := \emptyset$;
$endOfFirstPeriod := \min\{\alpha[1] + break - 0.1, \omega(\mathcal{H})\}$;
forall *pieces k of* $\mathcal{L}_{i-1}^{driven}$ *in chronological order* **do**
    if $endOfFirstPeriod < \alpha[k]$ **then**
        break;
    append piece from $\alpha[k]$ to $\min\{\omega[k], endOfFirstPeriod\}$ with value $(D[k], T[k])$ to $\mathcal{L}_i^{setup}$;
if $\alpha[1] + break \leq \omega(\mathcal{H})$ **then**
    append piece from $\alpha[1] + break$ to $\omega(\mathcal{H})$ with value $(0,0)$ to $\mathcal{L}_i^{setup}$;
return $\mathcal{L}_i^{setup}$;

3.3.3.2 Step *Wait*

In this step, $\mathcal{L}_i^{waited}$ is computed from $\mathcal{L}_i^{setup}$. As described, the time windows of customer i are ignored in the previous step. Considering them in this step means two things: First, we make sure that $\mathcal{L}_i^{waited}$ is undefined for every point in time outside of the time windows, so that it is only defined for those points in time at which the driver can actually start with the service at customer i. Second, we add a waiting time to the accumulated travel time if waiting for the next time window at customer i may be beneficial.

Let t be a point in time for which $\mathcal{L}_i^{setup}$ is defined. If t is inside a time window, there is no need to wait. If it is outside, $W_i(t)$ gives the waiting time until the next time window opens (please recall the definition of the waiting time function $W_i(t)$

in section 2.4.3). It is not worthwhile to wait longer than that. But is waiting advantageous at all? It is not if there is a later point in time $t' > t$ for which $\mathcal{L}_i^{setup}$ is also defined and that is mapped to the beginning of the same time window, i.e., $t + W_i(t) = t' + W_i(t')$, provided that $\mathcal{L}_i^{setup}(t')$ is minimum. Because from this, it follows that $D_i^{setup}(t) \geq D_i^{setup}(t')$ and $T_i^{setup}(t) + (t' - t) \geq T_i^{setup}(t')$ holds. So waiting from time t has no advantage over waiting from the later time t'.

With this in mind, we introduce an auxiliary function H^{shift} by means of which we shift the pieces accordingly. It has the following properties: If a given point in time t is inside a time window and also $\mathcal{L}_i^{setup}(t)$ is defined, H^{shift} simply maps t to itself. If a given t is the beginning of a time window, say $t = \alpha(\mathcal{W}_i^j)$ for some j, but $\mathcal{L}_i^{setup}(t)$ is undefined, then H^{shift} maps to the latest t' in the waiting interval $\overline{\mathcal{W}}_i^j$ for which $\mathcal{L}_i^{setup}(t')$ is defined, if such a t' exists. If not and in all other cases, H^{shift} maps to $\perp$. Especially, $H^{shift}(t)$ is $\perp$ for all t outside of the time windows of the current customer. So we define H^{shift} as follows:

$$H^{shift}(t) := \max\{t' \mid t = t' + W_i(t') \wedge \mathcal{L}_i^{setup}(t') \neq (\perp, \perp)\}$$

for all t for which the set $\{t' \mid t = t' + W_i(t') \wedge \mathcal{L}_i^{setup}(t') \neq (\perp, \perp)\}$ is not empty, and $\perp$ for all other t. We observe that $H^{shift}(t) \leq t$ holds unless $H^{shift}(t)$ is undefined.

With this auxiliary function, we shift $\mathcal{L}_i^{setup} = (D_i^{setup}, T_i^{setup})$ accordingly, and also add the waiting time $t - H^{shift}(t)$ to the travel time. So we set

$$\begin{aligned} D_i^{waited}(t) &:= D_i^{setup}\left(H^{shift}(t)\right) \\ T_i^{waited}(t) &:= T_i^{setup}\left(H^{shift}(t)\right) + (t - H^{shift}(t)) \end{aligned}$$

for all t with $H^{shift}(t) \neq \perp$, and $\mathcal{L}_i^{waited}(t) := (\perp, \perp)$ for all other t.

Let us have a look at the driver states label $\mathcal{L}_2^{waited}$ in Figure 3.7. Compared to the label $\mathcal{L}_2^{setup}$ in Figure 3.6, there are two changes: One is that the piece over the interval $[10.5, 20]$ is shortened to $[10.5, 12]$ to fit the time window of the customer. The other is that the piece over the interval $[5, 5.5]$ is both shifted according to the auxiliary function and shortened to the degenerate interval $[8, 8]$. This is because $\mathcal{L}_2^{setup}$ is not defined at time 8 when the time window of the second customer opens. The latest point in time in the waiting interval, for which $\mathcal{L}_2^{setup}$ is defined, is 5.5, so $H^{shift}(8) = 5.5$, and hence $D_2^{waited}(8) = D_2^{setup}(5.5) = 3.5$ and $T_2^{waited}(8) = T_2^{setup}(5.5) + (8 - 5.5) = 6$.

Function Wait shows pseudo-code to compute the pieces of $\mathcal{L}_i^{waited}$. It resembles a sweep-line algorithm to find the intersections of two (sorted) interval sets, yet with one small but important modification. Whenever the interval of the current piece k ends before the current time window $\mathcal{W}_i^j$ opens, we store the waiting time between the end of piece k and the beginning of $\mathcal{W}_i^j$ in the variable *waiting*. If the next piece $k + 1$ also happens to end before the still current time window $\mathcal{W}_i^j$ opens, we update the waiting time. Otherwise, if the next piece does not end before, the beginning $\alpha(\mathcal{W}_i^j)$ of the current time window may be either covered by the piece $k + 1$ or not. If it is covered, we reset the waiting time to 0. But if it is not covered, we have to append a piece that is only defined at $\alpha(\mathcal{W}_i^j)$ and has the driving time of the previous piece k and the travel time of this piece plus the stored waiting time.

Function Wait($\mathcal{L}_i^{setup}$)

$\mathcal{L}_i^{waited} := \emptyset$;
$j = k = 1$;
$waiting = 0$;
while *k-th piece of label* $\mathcal{L}_i^{setup}$ *and time window* $\mathcal{W}_i^j$ *exist* **do**
  if $\alpha[k] \leq \alpha(\mathcal{W}_i^j)$ **then**
    if $\omega[k] \geq \alpha(\mathcal{W}_i^j)$ **then**
      $waiting = 0$;
      append piece from $\alpha(\mathcal{W}_i^j)$ to $\min\{\omega[k], \omega(\mathcal{W}_i^j)\}$ with value $(D[k], T[k])$ to $\mathcal{L}_i^{waited}$;
      if $\omega(\mathcal{W}_i^j) \leq \omega[k]$ **then**
        j++;
      if $\omega(\mathcal{W}_i^j) \geq \omega[k]$ **then**
        k++;
    else
      $waiting = \alpha(\mathcal{W}_i^j) - \omega[k]$;
      k++;
  else
    if $waiting > 0$ **then**
      append piece from $\alpha(\mathcal{W}_i^j)$ to $\alpha(\mathcal{W}_i^j)$ with value $(D[k-1], T[k-1] + waiting)$ to $\mathcal{L}_i^{waited}$;
      $waiting = 0$;
    if $\alpha[k] \leq \omega(\mathcal{W}_i^j)$ **then**
      append piece from $\alpha[k]$ to $\min\{\omega[k], \omega(\mathcal{W}_i^j)\}$ with value $(D[k], T[k])$ to $\mathcal{L}_i^{waited}$;
      if $\omega(\mathcal{W}_i^j) \leq \omega[k]$ **then**
        j++;
      if $\omega(\mathcal{W}_i^j) \geq \omega[k]$ **then**
        k++;
    else
      j++;
if $\mathcal{W}_i^j$ *exists and* $waiting > 0$ **then**
  append piece from $\alpha(\mathcal{W}_i^j)$ to $\alpha(\mathcal{W}_i^j)$ with value $(D[k-1], T[k-1] + waiting)$ to $\mathcal{L}_i^{waited}$;
return $\mathcal{L}_i^{waited}$;

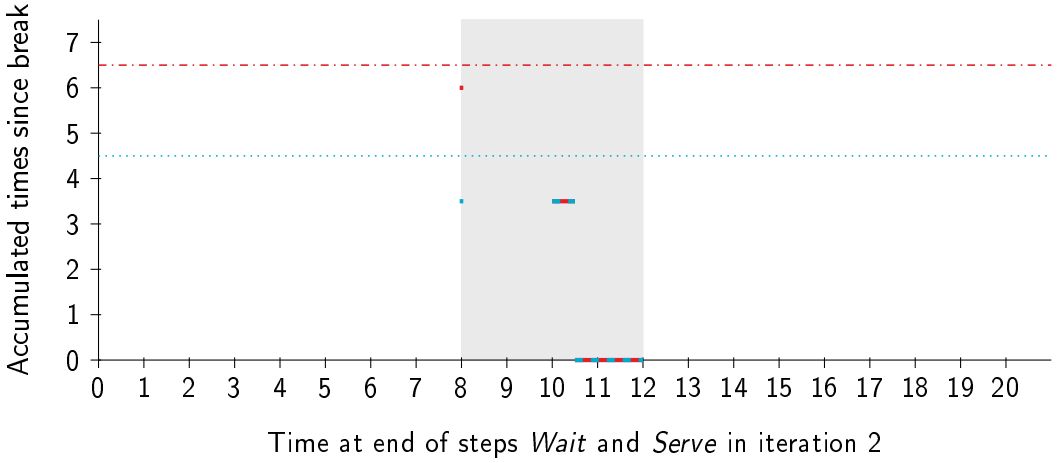


FIGURE 3.7: Driver states label $\mathcal{L}_2^{waited} = \mathcal{L}_2^{served}$. Functions are - if at all - only defined within the time window of the second customer.

3.3.3.3 Step *Serve*

In this step, we take the duration of the service at customer i and its indivisibility into account. The label $\mathcal{L}_i^{served}$ is computed from $\mathcal{L}_i^{waited}$ and shall hold the minimum accumulated times for every point in time at which the driver may finish the service at customer i. In this step, we have to distinguish the two rules *drive until traveled* and *work until traveled*.

In case of the rule *drive until traveled* (as in the ruleset *RulesetUS*), performing service is allowed even if the travel time since last break has exceeded the limit. In this special case, we stop further accumulating travel time. So we set

$$D_i^{served}(t) := D_i^{waited}(t - service_i)$$
$$T_i^{served}(t) := \min\{T_i^{waited}(t - service_i) + service_i, limitT\}$$

for all $t \in \mathcal{H}$ with $\mathcal{L}_i^{waited}(t - service_i) \neq (\bot, \bot)$ and $\mathcal{L}_i^{served}(t) := (\bot, \bot)$ for all other t.

In case of the rule *work until traveled* (as in the ruleset *RulesetEU*), service must not be performed if it cannot be finished before the travel time limit is exceeded. So we set

$$D_i^{served}(t) := D_i^{waited}(t - service_i)$$
$$T_i^{served}(t) := T_i^{waited}(t - service_i) + service_i$$

for all $t \in \mathcal{H}$ with both $\mathcal{L}_i^{waited}(t - service_i) \neq (\bot, \bot)$ and $T_i^{waited}(t - service_i) + service_i \leq limitT$, and $\mathcal{L}_i^{served}(t) := (\bot, \bot)$ for all other t. It should be noted that the travel time limit may already be exceeded while waiting before service. So strictly speaking, we allow this limit to be exceeded in step *Wait*. We do so for the sake of not having to distinguish both rules in the previous step, too.

For the sake of simplicity, all service times are 0 in our example instance, so $\mathcal{L}_2^{served}$ equals $\mathcal{L}_2^{waited}$ here. Again, we present pseudo-code. Function Serve shows how to compute the pieces of $\mathcal{L}_i^{served}$.

Function Serve($\mathcal{L}_i^{waited}$)

$\mathcal{L}_i^{served} := \emptyset$;
if *drive until traveled* **then**
  forall *pieces k of* $\mathcal{L}_i^{waited}$ *in chronological order* **do**
    if $\alpha[k] + service_i \leq \omega(\mathcal{H})$ **then**
      append piece from $\alpha[k] + service_i$ to $\min\{\omega[k] + service_i, \omega(\mathcal{H})\}$
        with value $(D[k], \min\{T[k] + service_i, limitT\})$ to $\mathcal{L}_i^{served}$;
else if *work until traveled* **then**
  forall *pieces k of* $\mathcal{L}_i^{waited}$ *in chronological order* **do**
    if $\alpha[k] + service_i \leq \omega(\mathcal{H}) \wedge T[k] + service_i \leq limitT$ **then**
      append piece from $\alpha[k] + service_i$ to $\min\{\omega[k] + service_i, \omega(\mathcal{H})\}$
        with value $(D[k], T[k] + service_i)$ to $\mathcal{L}_i^{served}$;
return $\mathcal{L}_i^{served}$;

3.3.3.4 Step *Drive*

In this step, we compute $\mathcal{L}_i^{driven}$ from $\mathcal{L}_i^{served}$ by adding the driving time $drive_i$ to the minimum accumulated driving and travel times and by scheduling inevitable breaks. For instance, if $D_i^{served}(t) + drive_i > limitD$ holds for a point in time t for which $\mathcal{L}_i^{served}(t)$ is defined, a break becomes due en route after $limitD - D_i^{served}(t)$ driving time in order to not violate rule *drive until driven*. An analogue statement holds for the rule *drive until traveled* or the rule *work until traveled*. Finally, the label $\mathcal{L}_i^{driven}$ shall hold the minimum accumulated times for every (significant) point in time at which the driver may feasibly arrive at the next customer $i + 1$. And these times should never exceed their respective limit $limitD$ or $limitT$. Be reminded that we do not need to take an early break en route into account because it is simply not advantageous. It is sufficient to consider it on arrival at the next customer, as we do in step *Setup*.

Let us assume for now that $drive_i$ is shorter than $limitD$ and $limitT$, so that at most one break can become due en route. The general case is treated at the end of this section. We need two auxiliary functions. One is the *excess function* H^{exc}. For a given departure time t from customer i, $H^{exc}(t)$ tells us by how much the driving time $drive_i$ exceeds the time that the driver is still allowed to drive at that time. H^{exc} is defined as follows:

$$H^{exc}(t) := drive_i - \min\{limitD - D_i^{served}(t), limitT - T_i^{served}(t)\}$$

for all $t \leq \omega(\mathcal{H}) - drive_i$ for which $\mathcal{L}_i^{served}(t)$ is defined, and $H^{exc}(t) := \perp$ for all other t. If $H^{exc}(t) \leq 0$ for some point in time t (this shall imply that $H^{exc}(t)$ is defined), the driver who leaves customer i at time t could arrive at customer $i + 1$ at time $t + drive_i$ without having to take a break en route, so having accumulated a driving and a travel time of another $drive_i$ time. On the other hand, if $H^{exc}(t)$ is positive, the same driver could arrive at customer $i + 1$ only at time $t + drive_i + break$ but with an accumulated driving and travel time of only $H^{exc}(t) \leq drive_i$.

Let t^+ be the first point in time for which H^{exc} is positive, or ∞ if this is never the case. And let $t^* := t^+ + drive_i + break$ be the earliest point in time at which a driver that has to take a due break arrives at the next customer. The driver may arrive at time t^* with an accumulated driving and travel time of $H^{exc}(t^+)$ each. But

the driver may also arrive at any later point in time with the same accumulated times by prolonging the due break en route. In order to take the prolongation of due breaks into account, we introduce a second auxiliary function H^{acc} that specifies the accumulated driving and travel time on arrival at the next customer:

$$H^{acc}(t) := \min\{H^{exc}(t') \mid t' \leq t - break - drive_i \wedge H^{exc}(t') > 0\}$$

for all t from t^* to $\omega(\mathcal{H})$, and $H^{acc}(t) := \perp$ for all other t. The function H^{acc} can be decomposed into two periods over $\mathcal{H}$: It is undefined in the first period until t^*, and it is a monotonically decreasing, positive step function in the second period from t^*. Just like H^{acc}, we may also see $\mathcal{L}_i^{driven}$ as separated into two periods:

1. In the first period, i.e., for all $t < t^*$, $H^{acc}(t)$ is undefined. If $H^{exc}(t - drive_i) \leq 0$ for some time $t < t^*$, we set

   $$D_i^{driven}(t) := D_i^{served}(t - drive_i) + drive_i$$
   $$T_i^{driven}(t) := T_i^{served}(t - drive_i) + drive_i$$

   This is the case if and only if customer $i + 1$ can be reached at time t without having to take a due break en route when leaving customer i at time $t - drive_i$ or any point before.

   In every other case, i.e. if $H^{exc}(t) > 0$ or undefined for some time $t < t^*$, $\mathcal{L}_i^{driven}(t)$ remains undefined:

   $$D_i^{driven}(t) := T_i^{driven}(t) := \perp$$

2. If $H^{acc}(t)$ is defined for some point in time t, this means some time $t' \leq t - break - drive_i$ exists such that a break becomes due en route when the driver leaves customer i at time t'. Then, $H^{acc}(t)$ is the minimum accumulated driving and travel time since the end of that due break, taking a prolongation of it into account. So once $H^{acc}(t)$ is defined for some t, we set for this t and every point in time thereafter:

   $$D_i^{driven}(t) := T_i^{driven}(t) := H^{acc}(t)$$

   So the second period of D_i^{driven} and T_i^{driven} is identical to the one of H^{acc}.

We summarize:

$$D_i^{driven}(t) := \begin{cases} D_i^{served}(t - drive_i) + drive_i, & t < t^* \wedge H^{exc}(t - drive_i) \leq 0 \\ H^{acc}(t), & t \geq t^* \\ \perp, & \text{otherwise} \end{cases}$$

$$T_i^{driven}(t) := \begin{cases} T_i^{served}(t - drive_i) + drive_i, & t < t^* \wedge H^{exc}(t - drive_i) \leq 0 \\ H^{acc}(t), & t \geq t^* \\ \perp, & \text{otherwise} \end{cases}$$

Again, we present pseudo-code to compute the pieces of $\mathcal{L}_i^{driven}$, see Function Drive on page 46. And again, let us have a look at the example instance and how the driver states label $\mathcal{L}_2^{driven}$ in Figure 3.8 is created from the label $\mathcal{L}_2^{served}$ in Figure 3.7. We observe that $H^{exc}(8) = 2.5 - 0.5 = 2 > 0$ and hence $t^+ = 8$. So a driver that leaves customer 2 at time 8 has to take a break en route. But on the other hand,

a driver who departs within the interval $[10.5, 12]$ does not because that driver is completely rested and $H^{exc}(10.5) < 0$. And so the piece over the interval $[10.5, 12]$ from $\mathcal{L}_2^{served}$ is shifted to the right and raised by the driving time $drive_2 = 2.5$ alone. At time $t^* = t^+ + drive_i + break = 8 + 2.5 + 5.5 = 16$, the first period ends.

In the second period of $\mathcal{L}_2^{driven}$, there are two pieces. These originate from the other two pieces of $\mathcal{L}_2^{served}$, shifted to the right by $drive_2 + break = 8$. Both accumulated times are set to the corresponding value of H^{acc}: For $16 \leq t < 18$, $H^{acc}(t) = 2$, and for $18 \leq t \leq \omega(\mathcal{H})$, $H^{acc}(t) = 1.5$. Both pieces are longer than the original pieces to take account of a prolongation of the due break. The last piece ends together with the planning horizon at time 20.

Since there is no service time at customer 3 and all three pieces are within the time window, there is nothing to do in the next steps. So we have finally computed the minimum accumulated driving and travel times at the end of the route. Figure 3.9 depicts the example instance again but also three different schedules. We observe that the three pieces in Figure 3.8 correspond to the three schedules in Figure 3.9. For the first schedule (blue), the driver departs as soon as possible from customer 1, takes an early break before the service at customer 2, and arrives earliest possible at customer 3. For the second schedule (green), the driver departs when the first time window of customer 1 closes, waits for the opening of the time window at customer 2, and takes a due break en route to customer 3. It is the rule *drive until traveled* that enforces this break. For the third schedule (red), the driver departs when the second time window of customer 1 opens and also takes a due break en route from customer 2 to customer 3. But here, it is the rule *drive until driven* that demands this break.

Function Drive($\mathcal{L}_i^{served}$)

$\mathcal{L}_i^{driven} := \emptyset$;
$endOfFirstPeriod := \omega(\mathcal{H})$;
forall *pieces k of* $\mathcal{L}_i^{served}$ *in chronological order* **do**
  if $\alpha[k] + drive_i > endOfFirstPeriod$ **then**
    break;
  $excess := drive_i - \min\{limitD - D[k], limitT - T[k]\}$;
  if $excess \leq 0$ **then**
    append piece from $\alpha[k] + drive_i$ to $\min\{\omega[k] + drive_i, endOfFirstPeriod\}$ with value $(D[k] + drive_i, T[k] + drive_i)$ to $\mathcal{L}_i^{driven}$;
  else if $\alpha[k] + drive_i + break - 0.1 < endOfFirstPeriod$ **then**
    $endOfFirstPeriod := \alpha[k] + drive_i + break - 0.1$;
$excess^{min} = \infty$;
forall *pieces k of* $\mathcal{L}_i^{served}$ *in chronological order* **do**
  if $\alpha[k] + drive_i + break > \omega(\mathcal{H})$ **then**
    break;
  $excess := drive_i - \min\{limitD - D[k], limitT - T[k]\}$;
  if $0 < excess < excess^{min}$ **then**
    $excess^{min} := excess$;
    set end of last piece of $\mathcal{L}_i^{driven}$ to at most $\alpha[k] + drive_i + break - 0.1$;
    append piece from $\alpha[k] + drive_i + break$ to $\omega(\mathcal{H})$ with value $(excess, excess)$ to $\mathcal{L}_i^{driven}$;
return $\mathcal{L}_i^{driven}$;

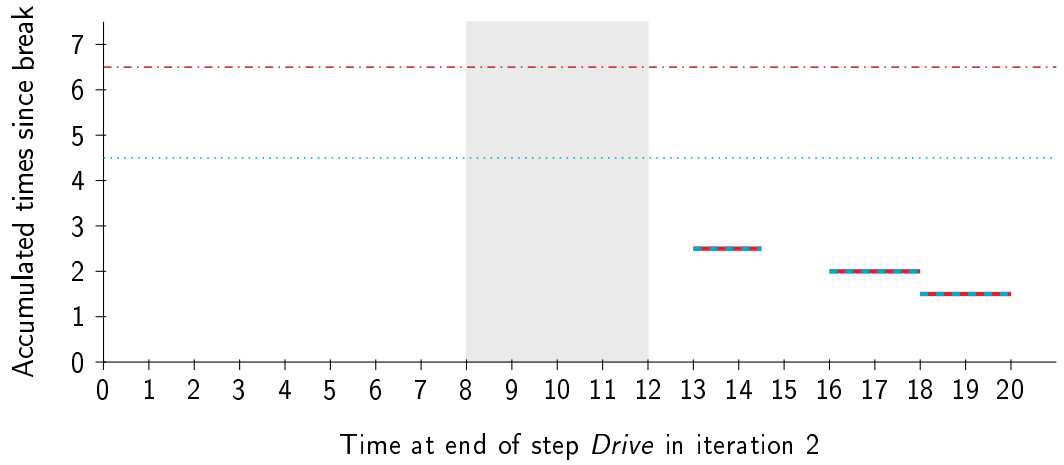


FIGURE 3.8: Driver states label $\mathcal{L}_2^{driven}$. Prolongation of the due breaks en route have been taken into account.

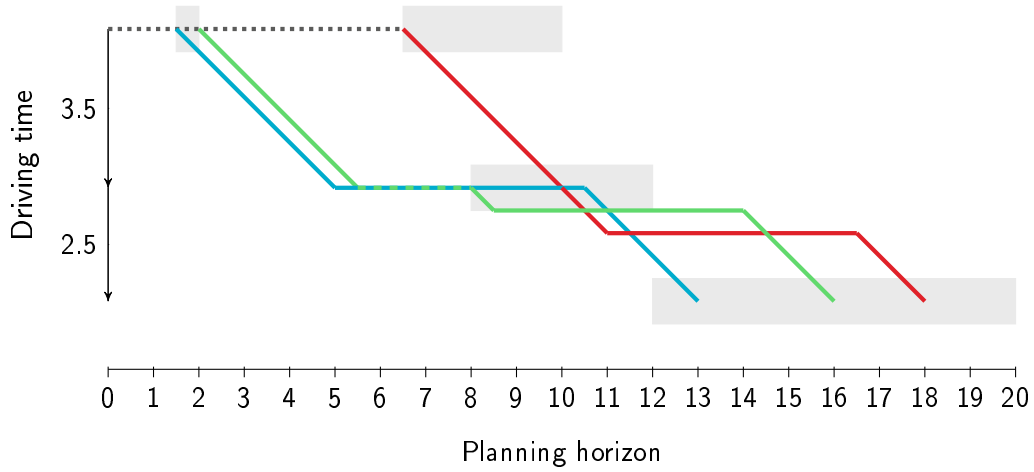


FIGURE 3.9: Example instance and three significant schedules.

Now what if the driving time $drive_i$ exceeds the driving time limit $limitD$ (assuming w.l.o.g. that $limitD \leq limitT$)? In this case, H^{exc} is positive wherever it is defined. We introduce another auxiliary function $H^{\#}$ that counts the number of additional mandatory breaks en route:

$$H^{\#}(t) := \lceil H^{exc}(t) \ / \ limitD \rceil - 1$$

for the same t for which H^{exc} is defined. We observe that $H^{exc}(t) > H^{\#}(t) \cdot limitD \geq 0$ holds. With this, we re-define the function H^{acc} and set:

$$H^{acc}(t) := \min\{H^{exc}(t') - H^{\#}(t') \cdot limitD \mid t' + drive_i + (H^{\#}(t') + 1) \cdot break \leq t \\ \wedge H^{exc}(t') > 0\}$$

for all t from $t^* + H^{\#}(t^*) \cdot break$ to $\omega(\mathcal{H})$, and $H^{acc}(t) := \perp$ for all other t. The rest of step *Drive* remains as before. This approach is based on the observation that it is sufficient to prolong at most one of the breaks en route. All other breaks adhere to the minimum break period.

3.3.4 Deriving a Schedule

Before we explain how we derive a schedule from the computed labels, we recall the definition of a truck driver schedule.

Algorithm 2 creates such a truck driver schedule, given the computed driver states labels $\mathcal{L}_i^{step}$ for all iterations i and all steps. Although the earliest finish time can be computed even if some driving and service times are zero, we expect them to be strictly positive for Algorithm 2 to always work correctly. This is just for the sake of a simpler notation.

Input of the algorithm is a time $t_n^{dep@c}$ for which $\mathcal{L}_n^{served}$ is defined. For every iteration i, time t_i^{start} is computed from $t_i^{dep@c}$ by subtracting the service time $service_i$. Since $\mathcal{L}_i^{served}$ is defined at time $t_i^{dep@c}$, so are $\mathcal{L}_i^{waited}$ and $\mathcal{L}_i^{setup}$ at time $t_i^{start} := t_i^{dep@c} - service_i$. If $\mathcal{L}_i^{setup}(t_i^{start})$ is $(0,0)$, there must be an early break scheduled before service at customer i, exploiting the assumption that the driving time to customer i is greater than zero. Due to a possible prolongation of this early break, $\mathcal{L}_{i-1}^{driven}$ might not be defined at time $t_i^{start} - break$, in which case we go even further back in time until we find a time for which $\mathcal{L}_{i-1}^{driven}$ is defined. If otherwise $\mathcal{L}_i^{setup}(t_i^{start})$ is not $(0,0)$, $\mathcal{L}_i^{waited}(t_i^{start})$ may incorporate waiting time at customer i. So to derive $t_i^{arr@c}$ from t_i^{start}, we might need to go back in time until we find a time for which $\mathcal{L}_{i-1}^{driven}$ is defined.

If and only if $T_{i-1}^{driven}(t_i^{arr@c})$ is greater than $drive_{i-1}$, there is no break scheduled en route, given that the service time at customer $i-1$ is positive. So $b_{i-1} := 0$. Since there is no appropriate point in time to be set for $t_{i-1}^{fb@r}$ and $t_{i-1}^{lb@r}$, we set both to the same value $t_{i-1}^{dep@c}$. In case $T_{i-1}^{driven}(t_i^{arr@c}) \leq drive_{i-1}$, at least one break is scheduled after service at customer $i-1$, and so $D_{i-1}^{driven}(t_i^{arr@c})$ and $T_{i-1}^{driven}(t_i^{arr@c})$ are equal. The end of the last break must be $D_{i-1}^{driven}(t_i^{arr@c})$ earlier than $t_i^{arr@c}$ and the total number of breaks b_{i-1} scheduled after service at customer i and before arrival at customer $i+1$ must be $\lfloor (drive_{i-1} - D_{i-1}^{driven}(t_i^{arr@c})) \;/\; limitD \rfloor + 1$ (assuming w.l.o.g. $limitD \leq limitT$). When deriving $t_{i-1}^{dep@c}$ from $t_i^{arr@c}$, we need to go back in time by at least $drive_{i-1} + b_{i-1} \cdot break$. But again due to a possible prolongation of the due break(s), we might need to go even further back in time until we find a time for which $\mathcal{L}_{i-1}^{served}$ is defined. Finally, we set $t_{i-1}^{fb@r}$ by adding the maximum allowed driving time at time $t_{i-1}^{dep@c}$ to $t_{i-1}^{dep@c}$. This concludes one loop of Algorithm 2.

Let us have a look at our example of section 3.3.3 again. We apply Algorithm 2 for $t_3^{dep@c} \in \{13, 16, 18\}$, even though the service times are zero here. We get the results as shown in Table 3.3.

3.3.5 Complexity Analysis

Let us repeat the arguments that constitute the correctness of the algorithm. First, we repeat some observations already discussed in section 3.2.2. In our algorithm, we do not have to consider every possible action of the driver at any point in time. Even though the driver may wait at any time, it does not need to be taken account of most of the time. It is sufficient to consider waiting only on arrival at a customer, only if the arrival time is within a waiting interval, and only until the next time window opens. And even though the driver may take a break at any time, it is sufficient to regard one in only two situations: Either when it becomes due on the way from one customer to another, i.e., after service at the one and before arrival at the other. Or

TABLE 3.3: Derived schedules for $t_3^{dep@c} \in \{13, 16, 18\}$.

Customer i	$t_i^{arr@c}$	t_i^{start}	$t_i^{dep@c}$	$t_i^{fb@r}$	b_i	$t_i^{lb@r}$
1	1.5	1.5	1.5	1.5	0	1.5
2	5	10.5	10.5	10.5	0	10.5
3	13	13	13	13	0	13
1	2	2	2	2	0	2
2	5.5	8	8	8.5	1	14
3	16	16	16	16	0	16
1	6.5	6.5	6.5	6.5	0	6.5
2	10	10	10	11	1	16.5
3	18	18	18	18	0	18

Algorithm 2: Schedule deduction (EF-TDSP)

Input : $t_n^{dep@c}$ for which $\mathcal{L}_n^{served}(t_n^{dep@c})$ is defined

Output: $\left((t_i^{arr@c}, t_i^{start}, t_i^{dep@c}, t_i^{fb@r}, b_i, t_i^{lb@r})\right)_{i \le n}$

$t_n^{lb@r} := t_n^{fb@r} := t_n^{dep@c}$;
$b_n := 0$;
$t_n^{start} := t_n^{dep@c} - service_n$;
forall $i = n-1 \dots 1$ **do**
  if $\mathcal{L}_{i+1}^{setup}(t_{i+1}^{start}) = (0,0)$ **then**
    $t_{i+1}^{arr@c} := \max\{t \mid t \le t_{i+1}^{start} - break \wedge \mathcal{L}_i^{driven}(t) \neq (\bot, \bot)\}$;
  else
    $t_{i+1}^{arr@c} := \max\{t \mid t \le t_{i+1}^{start} \wedge \mathcal{L}_i^{driven}(t) \neq (\bot, \bot)\}$;
  if $T_i^{driven}(t_{i+1}^{arr@c}) > drive_i$ **then**
    $t_i^{dep@c} := t_{i+1}^{arr@c} - drive_i$;
    $t_i^{lb@r} := t_i^{fb@r} := t_i^{dep@c}$;
    $b_i := 0$;
  else
    $t_i^{lb@r} := t_{i+1}^{arr@c} - D_i^{driven}(t_{i+1}^{arr@c})$;
    $b_i := \left\lfloor (drive_i - D_i^{driven}(t_{i+1}^{arr@c})) \ / \ limitD \right\rfloor + 1$;
    $t_i^{dep@c} := \max\{t \mid t \le t_{i+1}^{arr@c} - drive_i - b_i \cdot break \wedge \mathcal{L}_i^{served}(t) \neq (\bot, \bot)\}$;
    $t_i^{fb@r} := t_i^{dep@c} + \min\{limitD - D_i^{served}(t_i^{dep@c}), limitT - T_i^{served}(t_i^{dep@c})\}$;
  $t_i^{start} := t_i^{dep@c} - service_i$;
$t_1^{arr@c} := t_1^{start}$;

just when the driver arrives at a customer, before beginning the service, even if it is not yet due. However, a prolongation of a break needs to be considered in any way.

We observe that the steps of the algorithm (section 3.3.3) are laid out to reflect these observations. It follows from the construction of the algorithm that the customers' time windows, the service and driving times as well as the limits on accumulated driving and travel times are respected. In section 3.3.1, we have argued why it is sufficient to only store one pair of minimum accumulated times for a given point in time, and why we do not need to store such a pair for every point in time within the planning horizon. We observe that the accumulated times stored in a driver states label after each step are indeed minimum (leaving early breaks en route aside as mentioned above).

We turn towards proving that this algorithm has a polynomial time bound.

Lemma 3.3.1. *After iteration i, the two functions of a driver states label consist of at most $1 + \sum_{j=1}^{i} w_j$ pieces each.*

Proof. Initially, both functions consist of one piece each (constant zero). In step *Setup*, one constant zero piece may be added to each function. In step *Wait*, the number of pieces added depends on the number of time windows w_i at customer i. Suppose there is a piece in $\mathcal{L}_i^{setup}$ that spans parts of two time windows of customer i. Then the shift function H^{shift} ensures that $\mathcal{L}_i^{waited}$ is undefined over the waiting interval between the two time windows. And so two pieces are created from one. For the general case, suppose $H^{shift}(\alpha(\mathcal{W}_i^j))$ is defined for the beginning of a time window other than the first, i.e., $2 \leq j \leq w_i$. Then an additional piece is created if $\mathcal{L}_i^{setup}$ is defined over the end of the previous time window and both points in time belong to the same piece in $\mathcal{L}_i^{setup}$. In total, the number of pieces in both functions may grow at most by $w_i - 1$ each in iteration i.

In steps *Serve* and *Drive*, the number of pieces cannot increase: In step *Serve*, the pieces are only shifted and raised. In step *Drive*, every piece is shifted by either the driving time only or the driving time plus the break period. The function values on these pieces and their length may change but their number cannot grow. So altogether, up to w_i many pieces might be added to each function in iteration i. □

So the space complexity is $O(nw)$ where $w := \sum_{j=1}^{n} w_j$ is the total number of all time windows. But $O(nw)$ is also the time complexity because all function operations described can be implemented in a time that is linear in the number of pieces. For the special case of one time window per customer, we get the $O(n^2)$ bound already known from the literature.

Theorem 3.3.1. *The time complexity of the two considered variants of the EF-TDSP is in $O(nw)$.*

3.4 Discussion of the Extension by Break Splits

In this section, we discuss how the algorithm could be enhanced to also support break splits according to the break splitting rule *first-second-split* (see section 2.3.2.1). Such a rule applies in the European Union. In the following, we regard the ruleset {*drive until driven*, *work until traveled*, *first-second-split* }, or *RulesetEU+* in short, and show how the algorithm could be adapted accordingly. From the literature, it is not known how the additional relaxing rule affects the complexity of the problem. We claim that the enhanced version remains a polynomial-time algorithm and outline

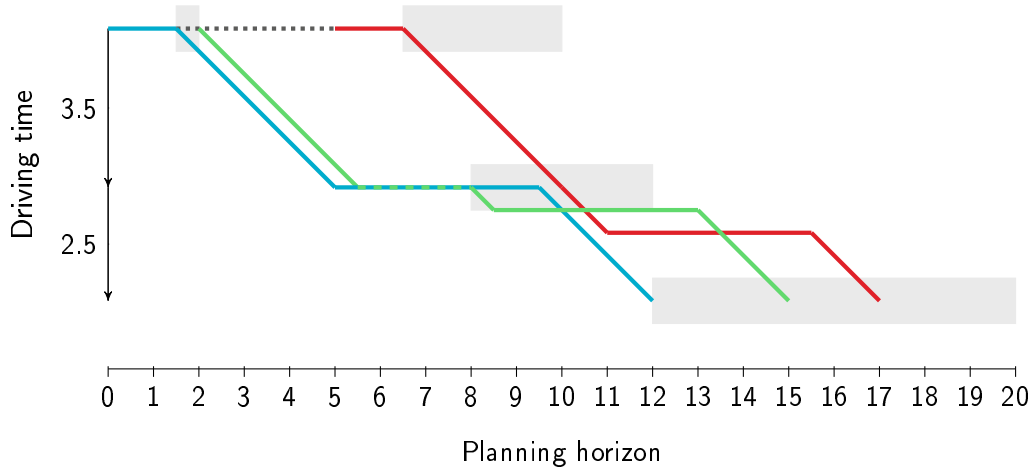


FIGURE 3.10: Example instance and three significant schedules in case of allowed break splits.

a proof of this claim. In general, we keep the presentation short and do not go into every detail.

From Regulation (EC) No. 561/2006 of the EU, it is not clear if a completely rested driver has to drive for a certain period of time before he is allowed to take the first part of a split break, or if this period may be arbitrarily small (e.g., 0 seconds) so that a first split break may follow a (second split) break directly. In the remainder of this section, we will assume the latter for the sake of a concise mathematical problem formulation. The same assumption can be found in the papers by Goel (2010) and Goel (2012c). It should be noted that this interpretation may not be correct or may be considered as abusive from a legal perspective. In the end, this thesis is about mathematical problems and not legal issues.

As a motivating example, we refer to Figure 3.10. It shows the same problem instance as in Figure 3.9 but this time, break splits are allowed with $break^{1st} = 1.5$ and $break^{2nd} = 4.5$. Since $break = 5.5$ as before, this means a split is penalized by 0.5 time units. We find that with split breaks, the earliest finish time of each of the three schedules is 1 time unit earlier than before. In case of the green schedule, the first split break can be taken before the service at the second customer. In the other two cases, it is taken before the service at the first customer.

3.4.1 Driver States Label Extension

For every point in time, we need to distinguish and store exactly two states of the driver. Either the driver is already partially rested or not. So in the following, let the driver states label $\mathcal{L}_i^{step}$ in step $step$ of iteration i be a pair $(\mathcal{F}_i^{step}, \hat{\mathcal{F}}_i^{step})$ of two function pairs. The function pair $\mathcal{F}_i^{step}$ stands for the case that no first split break is taken since the last time the driver was fully rested. In contrast, the function pair $\hat{\mathcal{F}}_i^{step}$ stands for the case that the first split break is already taken since then, so the driver is considered as partially rested. Both $\mathcal{F}_i^{step} := (D_i^{step}, T_i^{step})$ and $\hat{\mathcal{F}}_i^{step} := (\hat{D}_i^{step}, \hat{T}_i^{step})$ are each a pair of two time-dependent functions as before. That is, D_i^{step} and $\hat{D}_i^{step}$ each map a time to the minimum accumulated driving time since the last time the driver was completely rested, and T_i^{step} and $\hat{T}_i^{step}$ each map a time to the minimum accumulated travel time since the last time the driver was completely rested.

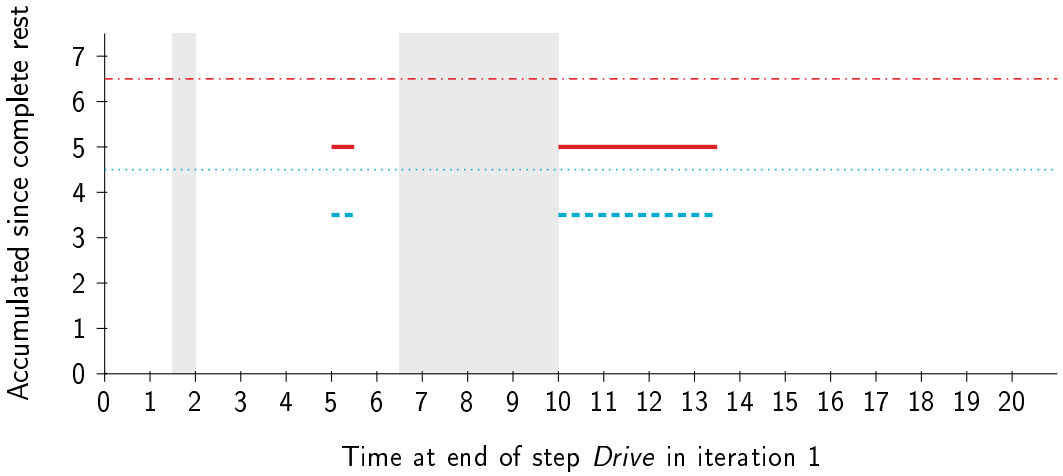


FIGURE 3.11: $\hat{D}_1^{driven}$ (blue, dashed), $\hat{T}_1^{driven}$ (red, solid).

3.4.2 Outline and Initialization of the Algorithm

The basic algorithm is the same as before, so for an outline, we refer to Algorithm 1. This algorithm returns $\texttt{obj}(\mathcal{L}_n^{served})$. In the break split case here, we set $\texttt{obj}(\mathcal{L}_n^{served}) = \alpha(\mathcal{L}_n^{served})$, which is the same as $\alpha(\mathcal{F}_n^{served})$. We again assume that the driver is completely rested when the planning horizon begins. So the first function pair $\mathcal{F}_0^{driven} = (D_0^{driven}, T_0^{driven})$ of the label $\mathcal{L}_0^{driven}$ is initialized by setting

$$D_0^{driven}(t) := 0$$
$$T_0^{driven}(t) := 0$$

for every $t \in \mathcal{H}$. For every other t, we set $\mathcal{F}_0^{driven}(t) := (\bot, \bot)$.

In the current case with break splits, we have to be a little more precise with respect to when exactly the driver becomes completely rested. Here, we assume that the planning horizon begins as soon as the driver becomes completely rested. This implies that the driver can be considered as partially rested $break^{1st}$ later. So the second function pair $\hat{\mathcal{F}}_0^{driven} = (\hat{D}_0^{driven}, \hat{T}_0^{driven})$ of the label $\mathcal{L}_0^{driven}$ is initialized as follows:

$$\hat{D}_0^{driven}(t) := 0$$
$$\hat{T}_0^{driven}(t) := break^{1st}$$

for every t with $\alpha(\mathcal{H}) + break^{1st} \leq t \leq \omega(\mathcal{H})$. For every other t, we again set $\hat{\mathcal{F}}_0^{driven}(t) := (\bot, \bot)$.

3.4.3 Steps of Algorithm in Detail

We start with the description of step *Setup*. $\mathcal{F}_1^{driven}$ is the same as in Figure 3.5. $\hat{\mathcal{F}}_1^{driven}$ is depicted in Figure 3.11.

3.4.3.1 Step *Setup*

In step *Setup*, we again take account of an early break. This may mean to take either a complete break of length *break* or – in case the driver is already partially rested – a second split break of length $break^{2nd}$. Let $t^* := \min\{\alpha(\mathcal{F}_{i-1}^{driven}) + break, \alpha(\hat{\mathcal{F}}_{i-1}^{driven}) +$

$break^{2nd}\}$ be the first point in time when the driver can be completely rested. With this re-definition of t^*, the function pair $\mathcal{F}_i^{setup}$ is defined just like in section 3.3.3.1. Precisely, we set for every point in time t:

$$\mathcal{F}_i^{setup}(t) := \begin{cases} \mathcal{F}_{i-1}^{driven}(t), & \text{for } t < t^* \\ (0,0), & \text{for } t^* \leq t \leq \omega(\mathcal{H}) \\ (\bot,\bot), & \text{otherwise} \end{cases}$$

Before we show how to set $\hat{\mathcal{F}}_i^{setup}$, we introduce the auxiliary function pair $\mathcal{F}_i' = (D_i', T_i')$. It consists of the two functions that we get from $\mathcal{F}_{i-1}^{driven}$ when we take an early first split break into account. For every t, it is defined as follows:

$$D_i'(t) := D_{i-1}^{driven}(t - break^{1st})$$
$$T_i'(t) := T_{i-1}^{driven}(t - break^{1st}) + break^{1st}$$

Now in order to set $\hat{\mathcal{F}}_i^{setup}$, we merge the function pairs $\mathcal{F}_i'$ and $\hat{\mathcal{F}}_{i-1}^{driven}$:

$$\hat{\mathcal{F}}_i^{setup}(t) := \begin{cases} \min\{\mathcal{F}_i'(t), \hat{\mathcal{F}}_{i-1}^{driven}(t)\}, & \text{for } t < t^* + break^{1st} \\ (0, break^{1st}), & \text{for } t^* + break^{1st} \leq t \leq \omega(\mathcal{H}) \\ (\bot,\bot), & \text{otherwise} \end{cases}$$

That minimum operation is not yet defined. Here, let the minimum operation $\min\{\mathcal{F}(t), \hat{\mathcal{F}}(t)\}$ for two function pairs $\mathcal{F} = (D,T)$ and $\hat{\mathcal{F}} = (\hat{D},\hat{T})$ as well as a point in time t be defined as follows:

$$\min\{\mathcal{F}(t), \hat{\mathcal{F}}(t)\} := \begin{cases} \mathcal{F}(t), & \text{if } T(t) < \hat{T}(t) \\ \hat{\mathcal{F}}(t), & \text{otherwise} \end{cases}$$

Here, let $\bot$ be treated just like ∞. That is, if one of the two functions T and $\hat{T}$ is defined at time t and the other one is not, the function defined at t is considered to be less. However, if both are undefined, neither of the two is less than the other.

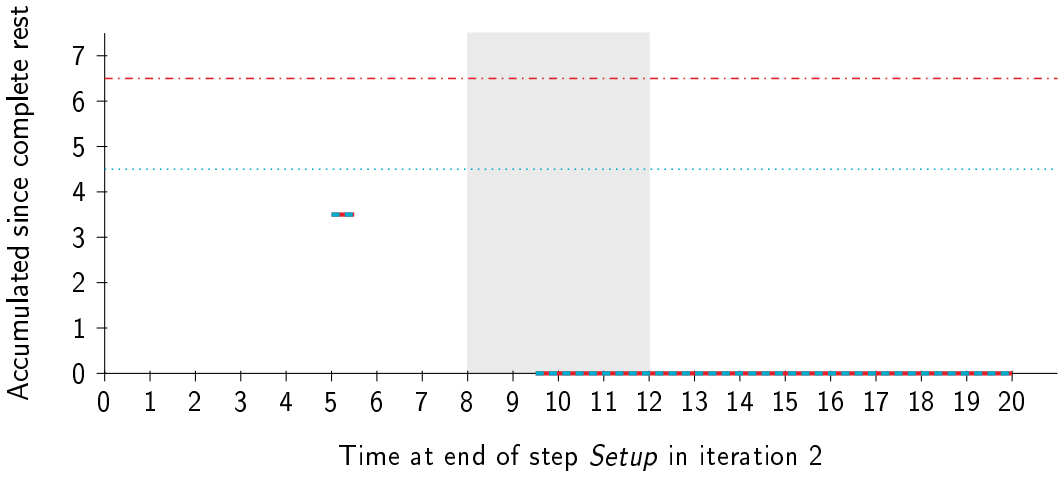
By definition, we only compare the accumulated travel times, not the accumulated driving times. For a justification of this definition, let us have a look at Figure 3.3 again. We have learned from this figure that for one point in time and the same progress, we only need to store one driver state. And we can conclude from this figure that $D(t) < \hat{D}(t)$ implies $T(t) < \hat{T}(t)$ and, conversely, $D(t) > \hat{D}(t)$ implies $T(t) > \hat{T}(t)$.

As far as the example is concerned, the function pairs of the label $\mathcal{L}_2^{setup}$ are depicted in Figure 3.12.

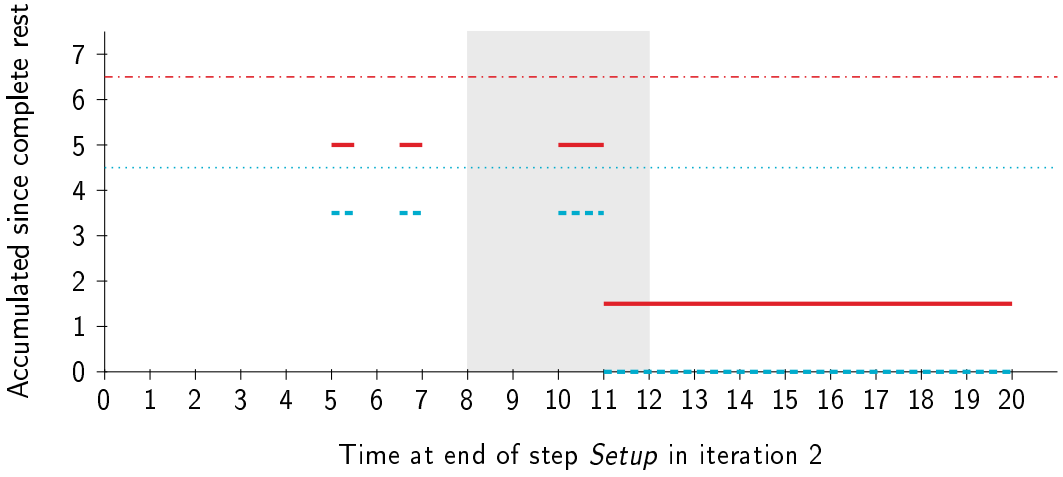
3.4.3.2 Steps *Wait* and *Serve*

The steps *Wait* and *Serve* are more or less unchanged compared to the same steps in sections 3.3.3.2 and 3.3.3.3. The only difference is that we now deal with two function pairs. But both pairs $\mathcal{F}$ and $\hat{\mathcal{F}}$ can be treated separately.

Since the service time is 0 at the second customer, the function pairs after step *Wait* and after *Serve* are the same. They are depicted in Figure 3.13.

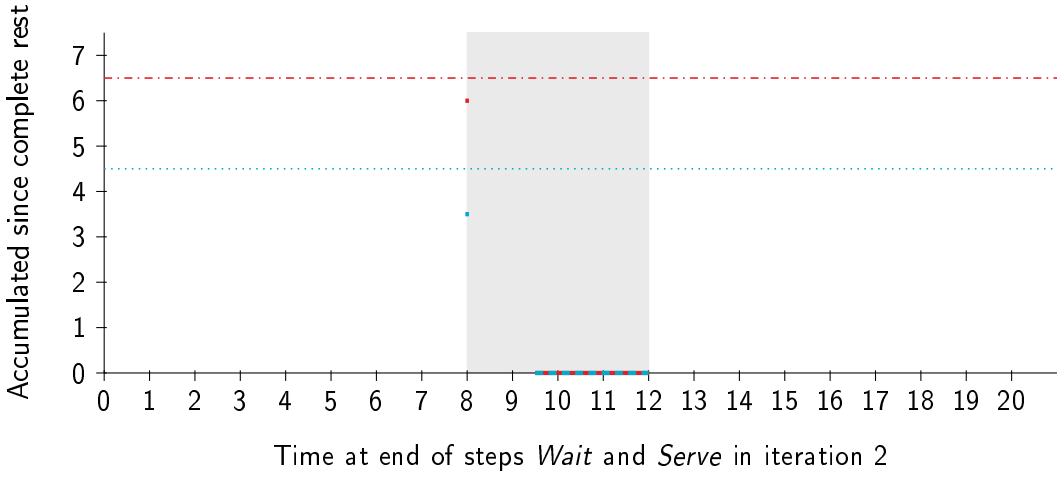


(A) $\mathcal{F}_2^{setup} = (D_2^{setup}, T_2^{setup})$.

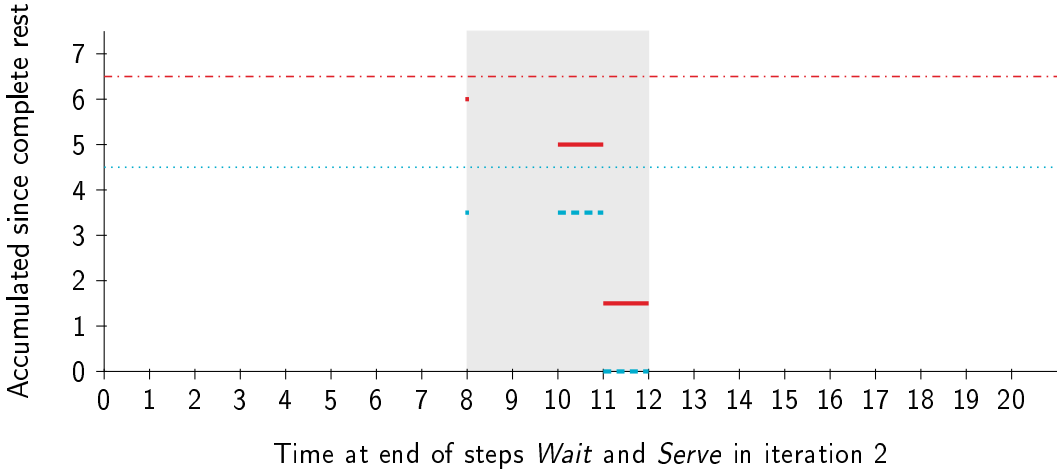


(B) $\hat{\mathcal{F}}_2^{setup} = (\hat{D}_2^{setup}, \hat{T}_2^{setup})$.

FIGURE 3.12: Driver states label $\mathcal{L}_2^{setup}$.



(A) $\mathcal{F}_2^{waited} = \mathcal{F}_2^{served}$.



(B) $\hat{\mathcal{F}}_2^{waited} = \hat{\mathcal{F}}_2^{served}$.

FIGURE 3.13: Driver states label $\mathcal{L}_2^{waited} = \mathcal{L}_2^{served}$.

3.4.3.3 Step *Drive*

Step *Drive* is again slightly different. The driver may have to take a break en route between the customers, and that break may either be a full break or only the second part of a split break. In this presentation, we focus on the case in which the driving time between two customers does not exceed *limitD* because then, it is never mandatory to schedule more than one break en route. An important observation is that it is never beneficial to schedule a break en route of which the period is longer than necessary. Especially, a first split break en route has no advantage over a first split break at the next customer.

Analogously to section 3.3.3.4, we first compute the excess functions H^{exc} and $\hat{H}^{exc}$ before we then set the points in time $t^+, \hat{t}^+, t^*$, and $\hat{t}^*$. Finally, we determine the auxiliary functions H^{acc} and $\hat{H}^{acc}$. The only adjustment worth mentioning is that we have to replace *break* with $break^{2nd}$ in the definitions of $\hat{t}^*$ and $\hat{H}^{acc}$, that is,

$$\hat{t}^* := \hat{t}^+ + drive_i + break^{2nd} \text{ and}$$

$$\hat{H}^{acc}(t) := \min\{\hat{H}^{exc}(t') \mid t' \leq t - break^{2nd} - drive_i \wedge \hat{H}^{exc}(t') > 0\}.$$

With all these auxiliary functions we have everything together to set the function pairs $\mathcal{F}_i^{driven} = (D_i^{driven}, T_i^{driven})$ and $\hat{\mathcal{F}}_i^{driven} = (\hat{D}_i^{driven}, \hat{T}_i^{driven})$. In step *Setup*, some pieces of $\mathcal{F}_{i-1}^{driven}$ were merged together with $\hat{\mathcal{F}}_{i-1}^{driven}$ to create $\hat{\mathcal{F}}_i^{setup}$. Now in step *Drive*, some pieces of $\hat{\mathcal{F}}_i^{served}$ are merged together with $\mathcal{F}_i^{served}$ to create $\mathcal{F}_i^{driven}$. This is because when a driver takes a due second split break en route, the driver is completely rested after the break, and an early first split break right after that is not beneficial. So pieces that need due breaks en route are moved from $\hat{\mathcal{F}}_i^{served}$ to $\mathcal{F}_i^{driven}$. The function pair $\hat{\mathcal{F}}_i^{driven}$ only contains pieces without due breaks en route. No additional pieces are created in this step. The four functions are set as follows:

$$D_i^{driven}(t) := \begin{cases} D_i^{served}(t - drive_i) + drive_i, & t < \min\{t^*, \hat{t}^*\} \wedge H^{exc}(t - drive_i) \leq 0 \\ \min\{H^{acc}(t), \hat{H}^{acc}(t)\}, & t \geq \min\{t^*, \hat{t}^*\} \\ \bot, & \text{otherwise} \end{cases}$$

$$T_i^{driven}(t) := \begin{cases} T_i^{served}(t - drive_i) + drive_i, & t < \min\{t^*, \hat{t}^*\} \wedge H^{exc}(t - drive_i) \leq 0 \\ \min\{H^{acc}(t), \hat{H}^{acc}(t)\}, & t \geq \min\{t^*, \hat{t}^*\} \\ \bot, & \text{otherwise} \end{cases}$$

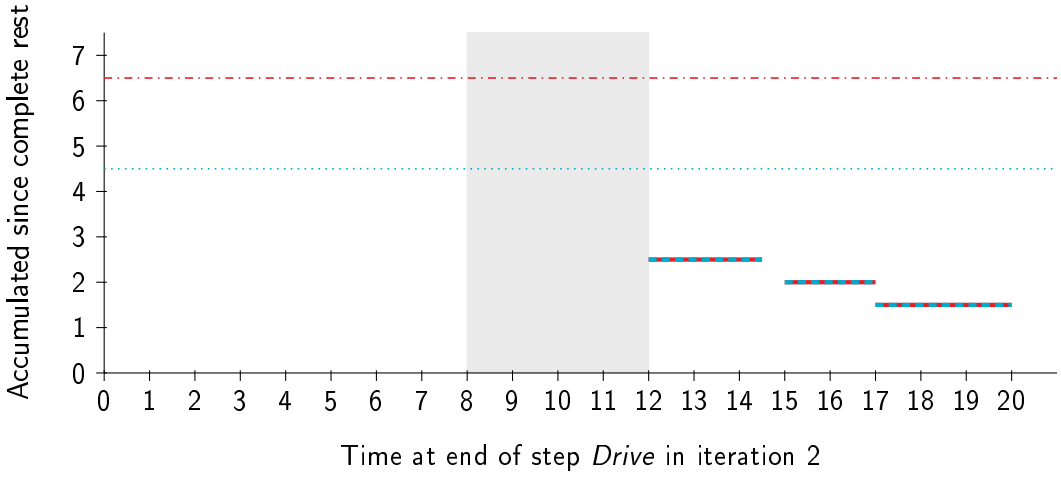
$$\hat{D}_i^{driven}(t) := \begin{cases} \hat{D}_i^{served}(t - drive_i) + drive_i, & \hat{H}^{exc}(t - drive_i) \leq 0 \\ \bot, & \text{otherwise} \end{cases}$$

$$\hat{T}_i^{driven}(t) := \begin{cases} \hat{T}_i^{served}(t - drive_i) + drive_i, & \hat{H}^{exc}(t - drive_i) \leq 0 \\ \bot, & \text{otherwise} \end{cases}$$

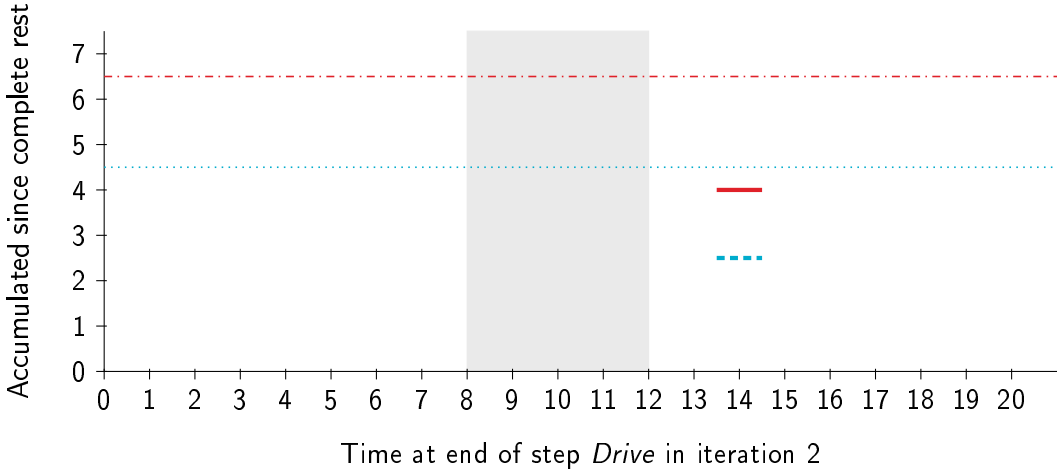
After this step, the function pairs from the example are as shown in Figure 3.14.

3.4.4 Complexity Analysis

What is the number of pieces of the function pairs? In section 3.3.5, we have analyzed their number step by step, literally. But with break splits, there are interdependencies between the two function pairs of a label that make this approach rather unpromising. Particularly in step *Setup*, we compute $\min\{\mathcal{F}_i'(t), \hat{\mathcal{F}}_{i-1}^{driven}(t)\}$ over a



(A) $\mathcal{F}_2^{driven}$.



(B) $\hat{\mathcal{F}}_2^{driven}$.

FIGURE 3.14: Driver states label $\mathcal{L}_2^{driven}$.

period of time, and this function pair may have as many pieces as the two input pairs together. This means for some pieces of $\mathcal{F}_{i-1}^{driven}$, a copy may be created in $\hat{\mathcal{F}}_i^{setup}$ – a "twin" of a piece in $\mathcal{F}_i^{setup}$. Even though it is true that no additional pieces are created in step *Drive*, some pieces – those with $\hat{H}^{exc}(t - drive_i) > 0$ – are moved from $\hat{\mathcal{F}}_i^{served}$ to $\mathcal{F}_i^{driven}$, where they may then induce twin pieces in $\hat{\mathcal{F}}_{i+1}^{setup}$ in the worst case. Due to these interdepedencies, a polynomial bound on the number of pieces is not obvious.

A more promising approach to show such a bound is the one that we are going to pursue in the complexity analysis section of chapter 5. The main idea is that every piece can be assigned to a time window. To be precise, every piece can be assigned to a partial schedule that starts at the end of the last break, and this partial schedule, in turn, can be assigned to the end of a time window. It is the end of the time window that the partial schedule hits on when we prolong the last break and shift the partial schedule further to the right. We leave out the details here. They can be found in section 5.4.5, albeit in the context of two types of breaks and not break splits.

We claim that no more than two pieces are assigned to the same end of a time window. This is because there are exactly two different driver states that we need to distinguish at the end of an assigned time window: Either the driver is partially rested at its end or not. We conclude that the number of pieces per label is asymptotically the same as in the case without break splits, namely it is in $O(w)$. Again, we claim that the algorithm can be implemented to run in linear time in the number of created pieces. It follows that the enhanced algorithm for the *RulesetEU+* still runs in $O(nw)$ time.

3.5 Conclusion and Outlook

We have studied three variants of a truck driver scheduling problem with one type of break and multiple time windows per customer. These variants differ in the considered rulesets. As the names suggest, the rulesets *RulesetEU* and *RulesetEU+* as well as *RulesetUS* are relevant in the European Union and the United States, respectively, at least for a planning horizon of a day. In contrast to the ruleset *RulesetEU*, the ruleset *RulesetEU+* even allows a break to be taken in two parts.

We have presented an algorithm for all three variants. As it is shown, it runs in polynomial time, independent of the break rule parameter setting. In case of the rulesets *RulesetEU* and *RulesetUS*, this is the first polynomial-time algorithm in presence of multiple (and arbitrarily distributed) time windows per customer. To the best of our knowledge, a polynomial-time algorithm for the ruleset *RulesetEU+* has not been known before, even in the single time window case.

This is the only chapter in which we regard a break splitting rule. In section 2.3.2.1, we have introduced the *minimum-length-split* rule. One of the questions we leave open is how this rule could be integrated into an algorithm for the TDSP and how this rule affects the complexity of the problem.

Chapter 4

Truck Driver Scheduling with Minimum Duration Objective

4.1 Introduction

Different optimization goals have been proposed for the truck driver scheduling problem (TDSP). It is most common to look for the earliest finish time (EF-TDSP) or the minimum duration (MD-TDSP) of a feasible schedule (see also section 2.4.1). An algorithm for the EF-TDSP has already been described in the previous chapter 3. In this chapter, we study the MD-TDSP. To be precise, we study the same two problem variants as in the previous chapter – apart from the optimization goal. That is, there is one type of break and there are two different rulesets, denoted as *RulesetUS* and *RulesetEU*. As throughout this thesis, customers may each have multiple (and arbitrarily distributed) time windows. In the next sections, we present an algorithm for the corresponding variants of the MD-TDSP and prove a polynomial-time bound. It is based on the algorithm that we have described previously in chapter 3.

Related Work Apart from the earliest finish time and the minimum duration, also other optimization goals have been regarded in the literature. For instance, in the case that a feasible schedule does not exist, it is worthwhile to consider two lexicographically ordered objectives, minimizing lateness as the first criterion, and the earliest finish time or the minimum duration as second criterion (Bernhardt et al., 2016). Another example is the optimization goal considered by Xu et al. (2003). They distinguish the time spent solely for waiting and the time spent for taking a break. In their model, these times may induce different costs. The authors regard a linear combination of these costs as objective function. For the case that both times are equally weighted in the function, their problem coincides with the variant that asks for the minimum duration because the sum of driving and service times are a constant. They conjecture that their problem - in the presence of multiple time windows per customer - is *NP*-hard.

For a very special case of a TDSP, namely the unrestricted case without any break rules, a polynomial time bound is known. It is proven by Jong, Kant, and Vliet (1996) that the minimum duration in the case with multiple time windows can be computed in $O(w \log n)$ time if there are no break rules that need to be respected (where w is the total number of time windows and n, as before, is the number of customers). However, no polynomial-time bound is known in case there are break rules to be observed.

Contribution and Outline Our algorithm is the first for the *RulesetUS*-MD-TDSP and the *RulesetEU*-MD-TDSP that runs in polynomial time. After all, it is the first

polynomial-time algorithm for any variant of an MD-TDSP (except for the trivial case without any rules). With this, the conjecture of Xu et al. (2003) can be falsified, at least when waiting cost and cost for the break periods are equally weighted.

The problem at hand is defined in section 4.2. Our solution approach including its analysis is described in section 4.3. In section 4.4, we shortly discuss the minimum idle cost objective. Finally, a conclusion follows in section 4.5.

4.2 Problem Definition

The problem at hand is defined almost exactly as in section 3.2 of the previous chapter. So again, we study a problem with one type of break and the rulesets *RulesetUS* = {*drive until driven*, *drive until traveled* } and *RulesetEU* = {*drive until driven*, *work until traveled* }. Still, the minimum break duration is denoted by *break*, and the two limits on the accumulated driving and travel time are *limitD* and *limitT*, respectively. The conditions to identify a feasible truck driver schedule are the same as in section 3.2.1. We again assume that the driver is rested when the planning horizon $\mathcal{H}$ begins, that is, there is an implicit break right before $\mathcal{H}$.

The only difference between the two problem variants is the optimization goal. Instead of looking for a feasible schedule with earliest finish time $t_n^{dep@r}$, we would now like to find one with shortest duration $t_n^{dep@r} - t_1^{arr@c}$. In section 4.2.1, we argue why the MD-TDSP is harder than the EF-TDSP.

4.2.1 Problem Characteristics

An algorithm for the MD-TDSP can also solve the EF-TDSP by modifying the original instance. However, we have to be careful. For instance, we can not just fix the start time, that is, the start of service at the first customer, because then we would be unable to further prolong the implicit break at the beginning, even though this may be necessary. Instead, we add a customer 0 to the front of the route that is *limitD* away from customer 1 and needs no service. W.l.o.g. the first time window of the originally first customer opens at the beginning of the planning horizon. We shift the beginning of the planning horizon by $limitD + break$ backward in time. Let the only time window of the newly created customer 0 open and close at the new beginning of the planning horizon, so $limitD + break$ before the first time window of the next customer opens. With this modification, a feasible schedule allows for a break before the first time window of customer 1. The new beginning of the planning horizon plus the minimum duration of a feasible truck driver schedule is equal to the earliest finish time of both the original and the modified instance. There is no corresponding transformation the other way round.

Finding the minimum duration is harder than finding the earliest finish time. This is already true in the single time window case, as is shown by the example in Figure 4.1. Let $break = 2$, $limitD = 2$, $limitT = \infty$, all driving times be 1 and all service times be 0. Here, the minimum duration of 5 is only achievable when departing at time 1 (blue schedule). For a schedule with the earliest finish time of 5.5, a waiting time of 0.5 is inevitable (red schedule). For symmetry reasons, this also holds for a schedule with the latest start time of 1.5. So calculating the earliest finish time (or analogously the latest start time) does not directly help find the minimum duration, even in the single time window case.

In the case with multiple time windows per customer, the problem of finding the minimum duration becomes even more challenging. With this problem, it is

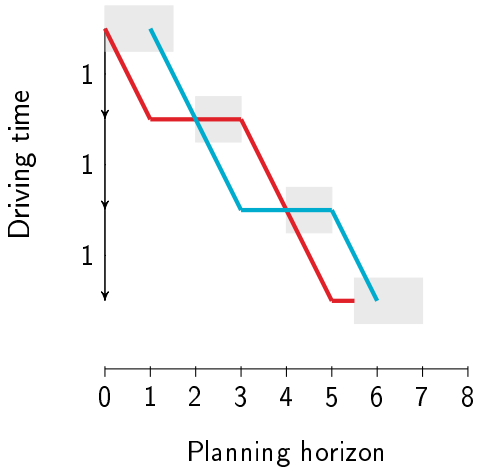


FIGURE 4.1: Calculating the earliest finish time does not directly help find the minimum duration. Here, $break = 2$, $limitD = 2$, $limitT = \infty$, all driving times are 1 and all service times are 0.

no longer sufficient to store one driver state per point in time. To illustrate this, imagine the situation that is depicted in Figure 4.2. It shows the time windows of the first four customers of a route and four partial schedules. Let all driving times be 1 and all service times be 0, let the minimum break period be 5 and $limitD > 3$ non-restrictive. The driver has (at least) four options when to leave the first customer. One is to start the route at time 7 and to service all customers without ever taking a break. Other options are to depart right away at time 0 or at time 1 or at time 2. In all of these cases, the driver takes an early break before service at a customer.

To find the earliest finish time, we only have to keep the information that it is possible to arrive at customer 4 at time 10 and that the minimum accumulated driving and travel time at that time is 0. We achieve this when the driver starts at time 0, so there is no need to consider the later start times.

This is different in the MD-TDSP case. Here, there are two criteria of interest, the (total) duration and the accumulated times since the end of the last break as before. A schedule *dominates* another with the same finish time only if it is better in terms of one of these two criteria and at least as good in terms of the other criterion compared to the other schedule. In Figure 4.2, the partial schedules and their corresponding driver states do not dominate each other because either a schedule is better in terms of duration (the later the start time, the shorter the duration) or better in terms of accumulated times since the end of the last break (the earlier the start time, the lower the accumulated times). So we have to store the information of all four partial schedules in a driver states label.

Now let us have a look at the example in Figure 4.3. Let again all driving times be 1 and all service times be 0, but let the minimum break period be 3 and $limitD = limitT = 1.5$. In this example, the earliest finish time is 9. To achieve this finish time, the driver has to depart at time 0 and to take an early break at customer 2. Without the early break, the driver could not service customer 3 within the first time window. But due to the early break, a second break becomes inevitable en route between customers 3 and 4. The second schedule proves that a feasible schedule with only one break exists. It has the minimum duration. However, the finish time of the second schedule is not earliest possible. So this is another example where computing the earliest finish time does not help find the minimum duration.

But there is more to this example. A dotted arrow in Figure 4.3 indicates how

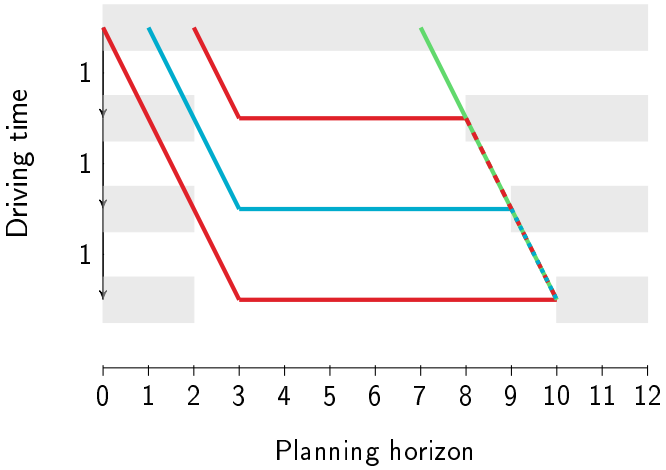


FIGURE 4.2: Four significant start times. Time $t = 10$ is covered by four different, non-dominated schedules.

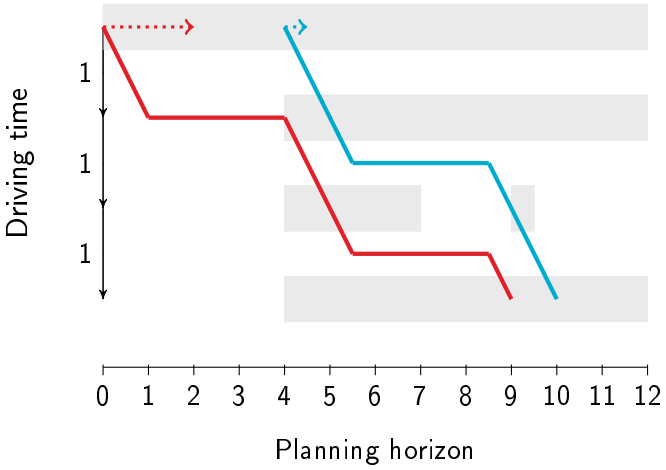


FIGURE 4.3: Example. Driving time between customers is 1 each, *break* $= 3$ and *limitD* $= 1.5$. Both schedules do not dominate each other. The red schedule has slack of two time units, the blue schedule only 0.5 time units. The red schedule could be shifted to the right so that it still starts earlier but ends later than the blue schedule.

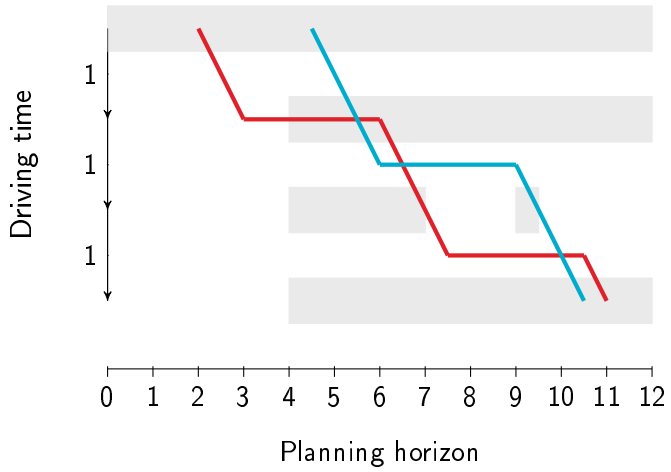


FIGURE 4.4: Example of Figure 4.3 with both schedules shifted to the right by the respective maximum slack value.

much *slack* there is in the schedule, i.e., how far the schedule could be shifted to the right such that every service remains within its associated time window. In the example, the slack values are 2 and 0.5, and it is the time windows of customer 3 that limit the slack to these values. Figure 4.4 shows both schedules again, shifted to the right by the respective slack values. That is, all time values of a schedule are increased by the respective slack value.

Let us go back to Figure 4.3 and recall the EF-TDSP algorithm: The minimum accumulated driving and travel times on arrival at customer 4 are stored in $\mathcal{L}_3^{driven}$. The value of both functions is 0.5 over the interval from 9 to the end of the planning horizon because the due break en route could be prolonged indefinitely. Now with the minimum duration objective and in contrast to before, there is no reason to prolong the break as long as there is still slack in the schedule. Instead, it is always better to shift the schedule accordingly. This is because shifting it does not change its duration as opposed to prolonging the break. So in a MD-TDSP algorithm, we need to keep track of how much slack there is in a schedule and make use of it as much as possible.

There are some more important observations. One is that there is no more slack in a schedule if and only if some service starts exactly at the end of a time window. Another is that whenever a schedule comprises slack, it is disadvantageous to let the driver wait. Shifting a schedule is neutral to accumulated times since last break and the schedule duration, whereas waiting lets both the accumulated travel time and the schedule duration increase. A third observation is that a driver who starts earlier may still finish later. That is, the so-called *first-in-first-out* property does not hold. An example of this is given in Figure 4.4.

4.3 Solution Approach

Now we address a solution approach and show how to find the minimum duration of a feasible schedule if one exists. In section 4.3.1, we redefine the contents of a driver states label, before we then turn towards the description of an algorithm for this problem (sections 4.3.2 through 4.3.4). Finally, it is shown in section 4.3.5 that this algorithm has a polynomial time bound.

4.3.1 Driver States Label

The rough idea of the MD-TDSP algorithm is to call the EF-TDSP algorithm for every *significant* point in time. We call a start time significant if a non-dominated schedule starts at that time, where the schedule must have one of the following properties, additionally:

- At least one service commences exactly at the beginning of the associated time window.
- At least one service commences exactly at the end of the associated time window.
- At least one early break commences exactly at the beginning of a waiting interval.

For every schedule without that property, there is an equivalent schedule with that property. We obtain this by shifting it to the left or to the right until the boundary of a time window (or the beginning of a waiting interval) is reached. For the algorithm, it is sufficient to consider only the start times of such representative schedules.

However, we do not know these points in time beforehand. So we will have to find out during the course of the algorithm. For the MD-TDSP algorithm, let the set of the significant start times known in iteration i be denoted by $\mathcal{S}_i$. This set may only change from iteration to iteration but remains the same between steps.

Since the algorithm distinguishes schedules with slack from those without, the set $\mathcal{S}_i = \mathcal{S}_i^+ \cup \mathcal{S}_i^0$ is decomposed into two subsets, the set of *shiftable* start times $\mathcal{S}_i^+$ and the set of *fixed* start times $\mathcal{S}_i^0$. A start time s is contained in $\mathcal{S}_i^+$ and considered as *shiftable* if there is slack in the schedule that begins at time s and that makes the start time significant. This means no service commences at the end of a time window (otherwise there would be no slack) whereas some service starts at the beginning of a time window or an early break commences at the beginning of a waiting interval (otherwise the start time would not be significant). Accordingly, $\mathcal{S}_i^0$ comprises the start times of non-dominated schedules where some service starts at the end of the respective time window, so there is no slack. Thus it turns out that the two sets are disjoint. In the example of Figure 4.2, the start times 2 and 7 are contained in $\mathcal{S}_4^+$, whereas 0 and 1 are contained in $\mathcal{S}_4^0$.

Like in the driver states label of the EF-TDSP algorithm, we again store a pair of two time-dependent functions in a label, but not just one. This time, we need to store such a function pair for every significant start time. So let $\mathcal{F}_{i,s}^{step} := (D_{i,s}^{step}, T_{i,s}^{step})$ be a pair of two time-dependent functions that hold the minimum accumulated driving time and the minimum accumulated travel time since the end of the last break with respect to start time s. For the MD-TDSP algorithm, a driver states label is defined as follows:

$$\mathcal{L}_i^{step} = \left(\mathcal{S}_i^+, \mathcal{S}_i^0, (\mathcal{F}_{i,s}^{step})_{s \in \mathcal{S}_i}\right)$$

where i is the current iteration and $step$ is one of $\{setup, waited, served, driven\}$ and $(\mathcal{F}_{i,s}^{step})_{s \in \mathcal{S}_i}$ is a sequence of function pairs.

By construction of the algorithm, the function pair $\mathcal{F}_{i,s}^{step}$ has different properties depending on whether the start time s is from $\mathcal{S}_i^+$ or from $\mathcal{S}_i^0$. If $s \in \mathcal{S}_i^+$, there is still slack in the corresponding schedule, that is, we could add an $\varepsilon > 0$ to every time value of the schedule without making the schedule infeasible. In this case, the functions in $\mathcal{F}_{i,s}^{step}$ comprise exactly one piece each, and the length of that piece

equals the slack value. If $s \in \mathcal{S}_i^0$, the functions are like in the EF-TDSP algorithm and can comprise several pieces (at most $1 + w$). And there is no slack, that is, if the driver started an ε later, he would miss a time window.

But the difference between the function pairs for s from $\mathcal{S}_i^+$ or from $\mathcal{S}_i^0$ is not only about the number of pieces, it is also about how the minimum accumulated times are related to a start time. As opposed to the EF-TDSP algorithm, the start time of a schedule that achieves the minimum accumulated times at a given point in time is relevant now. Suppose we had a function pair for a start time s that is defined over the interval from α_s to ω_s. If s is in $\mathcal{S}_i^0$ and hence considered as fixed, the minimum accumulated times stored at time α_s and ω_s and any time in between refer to a schedule that starts at time s. However, if s is from $\mathcal{S}_i^+$, then the minimum accumulated times stored at some time $\alpha_s \leq t \leq \omega_s$ refer to a schedule that starts at time $s + (t - \alpha_s)$ because the slack is exploited.

To give an example even before we describe the algorithm, let us use Figure 4.2 again. One of the significant start times in $\mathcal{S}_4^+$ is $s = 7$. For this start time, the minimum accumulated driving and travel time is 3 at time $\alpha_s = 10$ in iteration 4. Since the schedule that corresponds to start time 7 still has slack, the minimum accumulated driving and travel time is also 3 at time $t = 11$. But an accumulated travel time of 3 at time 11 is only possible if the driver starts at time $s + (t - \alpha_s) = 8$.

The algorithm not only ensures that the sets $\mathcal{S}_i^+$ and $\mathcal{S}_i^0$ alone are disjoint, but also that the start time intervals $[s, s + \omega_s - \alpha_s)$ for all $s \in \mathcal{S}_i^+$ and $[s, s]$ for all $s \in \mathcal{S}_i^0$ are all disjoint.

4.3.2 Outline and Initialization of the Algorithm

Before the first iteration, no significant start times are known. For the initialization of the algorithm, we consider the beginning of the planning horizon as significant, and so we set $\mathcal{S}_0^+ := \{\alpha(\mathcal{H})\}$ and $\mathcal{S}_0^0 := \emptyset$. For the start at $\alpha(\mathcal{H})$, we again assume that the driver is completely rested and so the function pair $\mathcal{F}_{0,\alpha(\mathcal{H})}^{driven}$ is set to

$$\mathcal{F}_{0,\alpha(\mathcal{H})}^{driven}(t) := \begin{cases} (0,0), & \text{for } t \in \mathcal{H} \\ (\bot,\bot), & \text{otherwise} \end{cases}$$

Having initialized $\mathcal{L}_0^{driven}$, we then apply Algorithm 1 again and iterate over the number of customers. The main differences between the EF-TDSP and the MD-TDSP algorithm are limited to step *Setup*. It is explained in section 4.3.3. The differences in the other steps are only minor and treated in section 4.3.4.

Finally, we check if at least one of the function pairs in the sequence $(\mathcal{F}_{n,s}^{served})_{s \in \mathcal{S}_n}$ is defined for some point in time, so if a feasible schedule exists. If it does, we let the function $\texttt{obj}(\mathcal{L}_n^{served})$ of Algorithm 1 return a pair of the minimum duration among all feasible schedules on the one hand, and the set of start times to achieve this minimum duration on the other:

$$\texttt{obj}(\mathcal{L}_n^{served}) := \left(\min_{s \in \mathcal{S}_n : \alpha(\mathcal{F}_{n,s}^{served}) \neq \bot} \alpha(\mathcal{F}_{n,s}^{served}) - s, \quad \operatorname*{arg\,min}_{s \in \mathcal{S}_n : \alpha(\mathcal{F}_{n,s}^{served}) \neq \bot} \left(\alpha(\mathcal{F}_{n,s}^{served}) - s \right) \right)$$

Should there be no feasible schedule, then $\texttt{obj}(\mathcal{L}_n^{served})$ is set to $(\bot, \emptyset)$.

The simplest way to find a corresponding schedule is to pick one of the optimal start times from the set and call the EF-TDSP algorithm of section 3.3 on a problem

instance where the time windows of the first customer do not begin before the chosen optimal start time.

TABLE 4.1: Driver states label summary (MD-TDSP).

$step$	one of {*Setup*, *Wait*, *Serve*, *Drive* }
$\mathcal{S}_i$	disjoint union of shiftable start time set $\mathcal{S}_i^+$ and fixed start time set $\mathcal{S}_i^0$ in iteration i
$\mathcal{F}_{i,s}^{step}$	the pair $(D_{i,s}^{step}, T_{i,s}^{step})$ of two functions comprising all feasible (and non-dominated) driver states at the end of step $step$ in iteration i for start time $s \in \mathcal{S}_i$
$\mathcal{L}_i^{step}$	the tupel $\left(\mathcal{S}_i^+, \mathcal{S}_i^0, (\mathcal{F}_{i,s}^{step})_{s\in\mathcal{S}_i}\right)$ at the end of step $step$ in iteration i

4.3.3 Step *Setup* in Detail

At the beginning of this step in iteration i, the start time sets $\mathcal{S}_i^+$ and $\mathcal{S}_i^0$ are empty. In order to fill them (and step *Setup* is the only step in which these sets are filled), the algorithm iterates once over all start times in $\mathcal{S}_{i-1}^+$ (section 4.3.3.1) and once over those in $\mathcal{S}_{i-1}^0$ (section 4.3.3.2).

As before, we want to go through this step by means of an example. Figure 4.5 shows the time windows of the first two customers of a route. Let the service time at the first customer be 0, the driving time to the second customer be 1, the break rule parameters be $break = 0.5$ and $limitT = limitD > 1$ be non-restrictive, and the planning horizon be $[0, 25]$. Initially, $\mathcal{S}_0^+ := \{0\}$ and $\mathcal{S}_0^0 := \emptyset$. Since some parts of the step *Setup* cannot be exemplified if we only look at the first iteration, our example covers the first two iterations, even if that means that we preempt the results of the first iteration of the algorithm.

The situation at the end of the first iteration is depicted in the figure. The set $\mathcal{S}_1^+$ contains the start times $\{1, 6, 10, 18\}$, which correspond to the times when the time windows of the first customer open. Analogously, $\mathcal{S}_1^0$ contains the start times $\{5, 8, 13, 19\}$, which correspond to the times when these time windows close. To every start time s in the sets, there is an associated function pair $\mathcal{F}_{1,s}^{driven}$. All these functions have exactly one piece. While the functions in $\mathcal{F}_{1,s}^{driven}$ for any s in $\mathcal{S}_i^0$ have a degenerate piece of length zero, the length of each piece of the functions in $\mathcal{F}_{1,s}^{driven}$ for any s in $\mathcal{S}_i^+$ corresponds to the length of the time window that starts at time s. This length is indicated by the dotted arrows in the figure.

For this section, we need some more definitions to simplify notation. From the previous iteration we are given $\mathcal{F}_{i-1,s}^{driven}$, a pair of two functions. In this section we just write α_s and ω_s in short for $\alpha(\mathcal{F}_{i-1,s}^{driven})$ and $\omega(\mathcal{F}_{i-1,s}^{driven})$, respectively. That is, for a given start time $s \in \mathcal{S}_{i-1}$, α_s (ω_s) denotes the first (last) defined point of the functions in $\mathcal{F}_{i-1,s}^{driven}$ (and $\perp$ if there is none). In the example, α_1 is 2 and ω_1 is 6 in step *Setup* of iteration 2. We call α_s (ω_s) the *earliest (latest) arrival time* with respect to s. Besides this short notation, we introduce a new function η_s that maps a start time s to the minimum of the earliest arrival times with respect to all later start times in $\mathcal{S}_{i-1}$, which we call the *next earliest arrival time* with respect to s:

$$\eta_s := \min_{s'\in\mathcal{S}_{i-1}: s'>s \wedge \alpha_{s'}\neq\perp} \alpha_{s'}$$

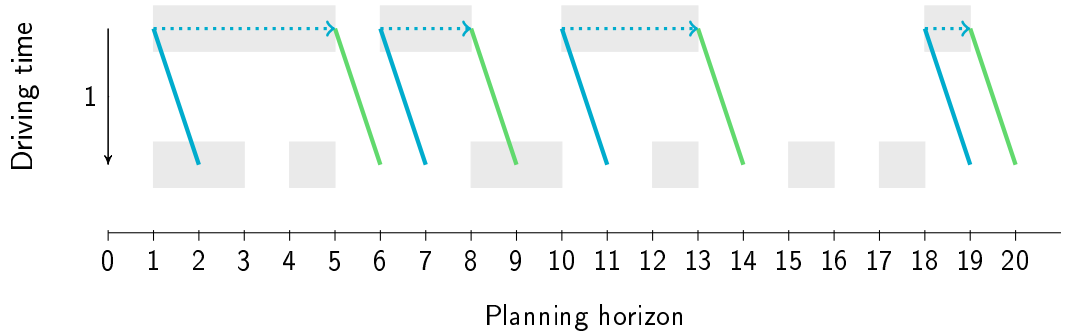


FIGURE 4.5: Example of section 4.3.3. Situation after step *Drive* of first iteration. Blue schedules relate to the set $\mathcal{S}_1^+ = \{1, 6, 10, 18\}$, green schedules relate to the set $\mathcal{S}_1^0 = \{5, 8, 13, 19\}$.

where the minimum over the empty set is supposed to be ∞. In our example, the earliest arrival time with respect to $s = 1$ is $\alpha_1 = 2$ and the next earliest arrival time is $\eta_1 = 6$, which we achieve when departing at $s' = 5$. Be aware that due to early breaks, η_s may be earlier than α_s. We have already seen this in the example of Figure 4.4, where $\alpha_2 = 11$ and $\eta_2 = 10.5$.

The definition of the next earliest arrival time helps us to not store dominated driver states. A driver state $\mathcal{F}_{i-1,s}^{driven}(t)$ for a start time s is dominated at time t if a later start time yields the same (or better) accumulated times since last break. So $\mathcal{F}_{i-1,s}^{driven}(t)$ does not have to be defined for any $t \geq \eta_s + break$ because we know that it is possible to start later and be completely rested at time $\eta_s + break$.

We present the details of step *Setup* in two parts. In the next subsection 4.3.3.1, we have a closer look at the shiftable start times only. We describe how we can make use of the slack and shift the start times profitably. In section 4.3.3.2, when we iterate over the fixed start times, the algorithm behaves very much like in step *Setup* of the EF-TDSP algorithm for each of the fixed start times. Functions SetupMDshiftable (page 72) and SetupMDfixed (page 71) show pseudo-code of this step.

4.3.3.1 Loop over Shiftable Start Times

We begin with the loop over $\mathcal{S}_{i-1}^+$. In the following, let s be an arbitrary start time from $\mathcal{S}_{i-1}^+$. As a preprocessing step is necessary in case of the ruleset *RulesetEU*, we focus on the ruleset *RulesetUS* for the time being, and defer the description of that step until the end of this section. Both functions in $\mathcal{F}_{i-1,s}^{driven}$ have exactly one piece each, and that piece is defined over the (non-degenerate) interval from α_s to ω_s. For $s = 1$ from $\mathcal{S}_1^+$ in the example, this is the interval from 2 to 6. This means the driver may arrive at the current customer i at every point in time between α_s and ω_s. But different from start times in $\mathcal{S}_{i-1}^0$, we can defer the start time depending on how much later than α_s we want the driver to arrive. That is, in order to arrive at time t^* sometime between α_s and ω_s, it is best (and feasible) for the driver to depart at time $s + t^* - \alpha_s$.

The algorithm finds all significant start times within the interval $[s, s + \omega_s - \alpha_s)$ and adds them to either $\mathcal{S}_i^+$ or $\mathcal{S}_i^0$. By construction of the algorithm, the start time $s + \omega_s - \alpha_s$ is already contained in $\mathcal{S}_i^0$ and therefore treated within the loop over the fixed start times (section 4.3.3.2). In the example, the set $\mathcal{S}_1^0$ contains the start time 5, and the corresponding function pair is defined over the degenerate interval from 6 to 6.

We differentiate two cases: The arrival of the driver within a time window and the arrival within a waiting interval, i.e., outside of a time window. Accordingly, the first thing we do (see Identifying the Significant Arrival Times) is to fill two disjoint sets $\mathcal{T}_{i,s}^{in}$ and $\mathcal{T}_{i,s}^{out}$ of arrival times that are *significant* with respect to start time s, where $\mathcal{T}_{i,s}^{in}$ contains the significant arrival times within the time windows of customer i, and $\mathcal{T}_{i,s}^{out}$ those within the waiting intervals. Once we know all the significant arrival times, there is a second round (see Loop over Arrivals within Time Windows and Loop over Arrivals outside of Time Windows) in which we loop over these significant arrival times in order to find significant start times.

Identifying the Significant Arrival Times We first describe the significant arrival times within the time windows. For our purposes, an arrival time t^{in} within the interval $[\alpha_s, \omega_s)$ is called significant if one of the following conditions is satisfied:

- $t^{in} = \alpha_s$ and t^{in} is within a time window
- t^{in} is the beginning of a time window

Precisely, we set

$$\mathcal{T}_{i,s}^{in} := \bigcup_{j=1}^{w_i} \{\min\{\alpha_s \leq t^{in} < \omega_s \mid t^{in} \in \mathcal{W}_i^j\}\} \setminus \{\infty\}$$

where the minimum over the empty set is supposed to be ∞. In the example, $\mathcal{T}_{2,1}^{in} = \{2, 4\}$, $\mathcal{T}_{2,6}^{in} = \{8\}$, $\mathcal{T}_{2,10}^{in} = \{12\}$, and $\mathcal{T}_{2,18}^{in} = \{\}$.

It may only be beneficial to arrive within a waiting interval if the driver starts an early break immediately on arrival and if that break ends within one of the time windows. Remember that a break is not prolonged as long as there is slack. Also keep in mind that for dominance reasons, we want the break to end before $\eta_s + \mathit{break}$. So an arrival time t^{out} within the interval $[\alpha_s, \min\{\omega_s, \eta_s\})$ is also called significant if one of the following conditions is fulfilled:

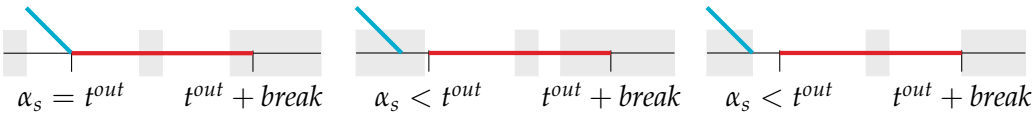
- $t^{out} = \alpha_s$ and t^{out} is within a waiting interval, and $t^{out} + \mathit{break}$ is within a time window
- t^{out} is the beginning of a waiting interval, and $t^{out} + \mathit{break}$ is within a time window
- t^{out} is within a waiting interval, and $t^{out} + \mathit{break}$ is the beginning of a time window

These three cases are also illustrated in Figure 4.6.

If the value of the (defined) piece of $\mathcal{F}_{i-1,s}^{driven}$ is $(0, 0)$, then there is no need to take a break and no reason to arrive outside of a time window. So we set $\mathcal{T}_{i,s}^{out} := \emptyset$ in this case. Otherwise we set

$$\mathcal{T}_{i,s}^{out} := \bigcup_{j=1}^{w_i} \bigcup_{k=j}^{w_i} \{\min\{\alpha_s \leq t^{out} < \min\{\omega_s, \eta_s\} \mid t^{out} \in \overline{\mathcal{W}}_i^j \wedge t^{out} + \mathit{break} \in \mathcal{W}_i^k\}\} \setminus \{\infty\}$$

where $\overline{\mathcal{W}}_i$ are the waiting intervals as defined in section 2.4.3. In the example, $\mathcal{T}_{2,1}^{out} = \{3.5\}$, $\mathcal{T}_{2,6}^{out} = \{7.5\}$, $\mathcal{T}_{2,10}^{out} = \{11.5\}$, and $\mathcal{T}_{2,18}^{out} = \{\}$.



(A) Case 1 (B) Case 2 (C) Case 3

FIGURE 4.6: The three cases in which t^{out} is considered as a significant arrival time.

Loop over Arrivals within Time Windows We do the following for every $t^{in} \in \mathcal{T}^{in}_{i,s}$. By construction, t^{in} is within a time window, namely $\mathcal{W}_i[t^{in}]$, and $\omega(\mathcal{W}_i[t^{in}])$ denotes the end of that time window (recall definition of $\mathcal{W}_i[t^{in}]$ in section 2.4.3). If, in turn, this end is before ω_s, we learn in this iteration that $s' := s + \omega(\mathcal{W}_i[t^{in}]) - \alpha_s$ must be a significant start time because a (non-dominated) schedule exists such that some service begins at the end of a time window. And so we add s' to $\mathcal{S}^0_i$ and set

$$\mathcal{F}^{setup}_{i,s'}(t) := \begin{cases} \mathcal{F}^{driven}_{i-1,s}(t), & \text{for } t = \omega(\mathcal{W}_i[t^{in}]) \\ (\bot, \bot), & \text{otherwise} \end{cases}$$

so the function pair is only defined at the end of that time window. In the example, the start times 2, 4 and 12 are added to $\mathcal{S}^0_2$ in this case (green schedules in Figure 4.7). Table 4.2 helps comprehend the auxiliary calculations.

If $\omega(\mathcal{W}_i[t^{in}]) < \omega_s$ and in addition $t^{in} < \omega(\mathcal{W}_i[t^{in}])$, i.e., t^{in} is not at the end of its associated time window, $s' := s + t^{in} - \alpha_s$ must be a significant start time, too. If t^{in} is the beginning of a time window, this follows directly from the definition of a significant start time. However, if t^{in} equals α_s, then the claim follows by induction. So we add s' to $\mathcal{S}^+_i$ and set

$$\mathcal{F}^{setup}_{i,s'}(t) := \begin{cases} \mathcal{F}^{driven}_{i-1,s}(t), & \text{for } t^{in} \leq t \leq \omega(\mathcal{W}_i[t^{in}]) \\ (\bot, \bot), & \text{otherwise} \end{cases}$$

In the example, the start times 1, 3 and 11 are added to $\mathcal{S}^+_2$ in this case (blue schedules, lengths of pieces ($\equiv$ slack) are indicated by dotted arrows as before).

Otherwise, if $\omega(\mathcal{W}_i[t^{in}]) \geq \omega_s$, it is ω_s that limits the length of the function pair pieces. By the same argument as before, we add $s' := s + t^{in} - \alpha_s$ to $\mathcal{S}^+_i$ and set

$$\mathcal{F}^{setup}_{i,s'}(t) := \begin{cases} \mathcal{F}^{driven}_{i-1,s}(t), & \text{for } t^{in} \leq t \leq \omega_s \\ (\bot, \bot), & \text{otherwise} \end{cases}$$

In the example, the start time 7 is added to $\mathcal{S}^+_2$ in this case (blue schedule).

Also $s + \omega_s - \alpha_s$ may be a significant start time. But even if it is, there is no need to consider it here because this would have already been learned in a previous iteration, and so it would have been added to the set of fixed start times before.

Let us summarize: In this loop, $\mathcal{F}^{setup}_{i,s'}$ is copied from $\mathcal{F}^{driven}_{i-1,s}$ but only over a subinterval from $[\alpha_s, \omega_s]$. The start time s' is deferred by as much as we truncate in the front of the piece of $\mathcal{F}^{driven}_{i-1,s}$. By construction, the intervals $[\alpha(\mathcal{F}^{setup}_{i,s'}), \omega(\mathcal{F}^{setup}_{i,s'})]$ with $s' := s + t^{in} - \alpha_s$ are pairwise disjoint for all $t^{in} \in \mathcal{T}^{in}_{i,s}$ because they are all subsets of different time windows. For every t^{in}, at most one start time is added to $\mathcal{S}^+_i$, at most one to $\mathcal{S}^0_i$, and at least one to either of the two sets.

TABLE 4.2: Start times added during loop over arrival times within time windows.

s	$[\alpha_s, \omega_s)$	t^{in}	$\omega(\mathcal{W}_i[t^{in}])$	$\mathcal{S}_i^+$	$\mathcal{S}_i^0$
1	[2,6)	2	3	1	2
		4	5	3	4
6	[7,9)	8	10	7	-
10	[11,14)	12	13	11	12

Loop over Arrivals outside of Time Windows We do the following for every $t^{out} \in \mathcal{T}_{i,s}^{out}$. By definition, t^{out} is outside of a time window, and $t^{out} + break$ is inside a time window. To emphasize this, we set $t^{in} := t^{out} + break$ as short notation, even though this t^{in} may not be contained in $\mathcal{T}_{i,s}^{in}$. As before, $\omega(\mathcal{W}_i[t^{in}])$ is the time when the time window of t^{in} closes. As far as the arrival time t^{out} is concerned, we want the early break to begin not only before the latest arrival time ω_s and before the next earliest arrival time η_s but also before the next time window opens at time $\alpha(\mathcal{W}_i[t^{out}])$. For convenience, we set $slack(t^{in}) := \omega(\mathcal{W}_i[t^{in}]) - t^{in}$ and $slack(t^{out}) := \min\{\omega_s, \eta_s, \alpha(\mathcal{W}_i[t^{out}])\} - t^{out}$, and we observe that $0 < slack(t^{out}) \leq break$ holds.

If $slack(t^{in}) < slack(t^{out})$, we can see that $s' := s + t^{out} - \alpha_s + slack(t^{in})$ is a significant start time because a (non-dominated) schedule exists such that - after an early break - the service at customer i begins at the end of time window $\mathcal{W}_i[t^{in}]$. Accordingly, we add s' to $\mathcal{S}_i^0$ and set

$$\mathcal{F}_{i,s'}^{setup}(t) := \begin{cases} (0,0), & \text{for } t = \omega(\mathcal{W}_i[t^{in}]) \\ (\perp,\perp), & \text{otherwise} \end{cases}$$

If besides $slack(t^{in}) < slack(t^{out})$ also $slack(t^{in}) > 0$ holds, $s' := s + t^{out} - \alpha_s$ is a significant start time as well. This results from the definition of a significant start time and the construction of $\mathcal{T}_{i,s}^{out}$ because either t^{in} is when a time window opens, or t^{out} is when a waiting interval begins, or t^{out} equals α_s, in which case the claim follows by induction. So we add s' to $\mathcal{S}_i^+$ and set

$$\mathcal{F}_{i,s'}^{setup}(t) := \begin{cases} (0,0), & \text{for } t^{in} \leq t \leq \omega(\mathcal{W}_i[t^{in}]) \\ (\perp,\perp), & \text{otherwise} \end{cases}$$

If $slack(t^{in}) \geq slack(t^{out})$, we add $s' := s + t^{out} - \alpha_s$ to $\mathcal{S}_i^+$, too (by the same argument as above). However, we set

$$\mathcal{F}_{i,s'}^{setup}(t) := \begin{cases} (0,0), & \text{for } t^{in} \leq t \leq t^{in} + slack(t^{out}) \\ (\perp,\perp), & \text{otherwise} \end{cases}$$

In the example, the start times $\{2.5, 6.5, 10.5\}$ are added to $\mathcal{S}_2^+$ in this case (red schedules in Figure 4.7). Again, the auxiliary calculations can be reproduced more easily with the help of Table 4.3.

Let us summarize: In this loop, $\mathcal{F}_{i,s'}^{setup}$ is set to $(0,0)$ over a subinterval from $[\alpha_s, \omega_s]$, shifted to the right by the minimum break period. Analogously to $\mathcal{T}_{i,s}^{in}$, the intervals $[\alpha(\mathcal{F}_{i,s'}^{setup}), \omega(\mathcal{F}_{i,s'}^{setup}))$ with $s' := s + t^{out} - \alpha_s$ are pairwise disjoint for all $t^{out} \in \mathcal{T}_{i,s}^{out}$. This is because for two such intervals either the two intervals are

TABLE 4.3: Start times added during loop over arrivals outside of time windows.

s	$[\alpha_s, \omega_s)$	t^{out}	t^{in}	$slack(t^{out})$	$slack(t^{in})$	$\mathcal{S}_i^+$	$\mathcal{S}_i^0$
1	[2,6)	3.5	4	0.5	1	2.5	-
6	[7,9)	7.5	8	0.5	2	6.5	-
10	[11,14)	11.5	12	0.5	1	10.5	-

subsets of different time windows, or the two intervals are, if shifted to the left by *break*, subsets of different waiting intervals, or both. In fact, even the intervals $[\alpha(\mathcal{F}_{i,s'}^{setup}), \omega(\mathcal{F}_{i,s'}^{setup})]$ are pairwise disjoint.

But there is more to the half-open intervals $[\alpha(\mathcal{F}_{i,s'}^{setup}), \omega(\mathcal{F}_{i,s'}^{setup}))$: They are pairwise disjoint not only for all $t^{out} \in \mathcal{T}_{i,s}^{out}$ and one start time s but also for all $s \in \mathcal{S}_{i-1}^+$. For an explanation, let s_1' and s_2' be the associated shifted start times for two start times $s_1 < s_2$ from $\mathcal{S}_{i-1}^+$. It is ensured that the interval for s_1' ends before $\eta_{s_1} + break$, whereas the interval for s_2' does not begin earlier than $\alpha_{s_2} + break \geq \eta_{s_1} + break$. This observation will later be exploited in the analysis.

As mentioned in the beginning of section 4.3.3.1, a preprocessing step is necessary in case of the ruleset *RulesetEU*. This is because the rule *work until traveled* of this ruleset necessitates to check whether the service at customer i can be completed before a break becomes due. A compulsory break (in contrast to an early break) may begin inside a time window. To take account of this, we check whether $T_{i-1,s}^{driven}(\alpha_s) + service_i \leq limitT$ holds at the very beginning of the loop. And if this is not the case, we shift the piece of both functions of $\mathcal{F}_{i-1,s}^{driven}$ by the break period and reset the values to $(0,0)$ before we continue with the loop, provided that at least $service_i \leq limitT$.

4.3.3.2 Loop over Fixed Start Times

We continue with the loop over $\mathcal{S}_{i-1}^0$. So let s be an arbitrary start time from $\mathcal{S}_{i-1}^0$ in the following. If α_s is defined, we set $\mathcal{S}_i^0 := \mathcal{S}_i^0 \cup \{s\}$ and

$$\mathcal{F}_{i,s}^{setup}(t) := \begin{cases} \mathcal{F}_{i-1,s}^{driven}(t), & \text{for } \alpha_s \leq t < \min\{\alpha_s, \eta_s\} + break \\ (0,0), & \text{for } \alpha_s + break \leq t < \eta_s + break \wedge t \leq \omega(\mathcal{H}) \\ (\perp, \perp), & \text{otherwise} \end{cases}$$

If $\alpha_s = \perp$, we can simply discard s. In the example, the start times $\{5, 8, 13, 19\}$ are added to $\mathcal{S}_2^0$ (green schedules). Of these, the start times 5 and 19 will be discarded in the next iteration because $\mathcal{F}_{2,5}^{setup}(t) = \mathcal{F}_{2,19}^{setup}(t) = (\perp, \perp)$ for all t.

Function SetupMDfixed($\mathcal{L}_{i-1}^{driven}$)

1 **forall** *start times s in* $\mathcal{S}_{i-1}^0$ **do**
2 if $\alpha_s \neq \perp$, then add s to $\mathcal{S}_i^0$, copy pieces in range from α_s to $\min\{\alpha_s, \eta_s\} + break - 0.1$ from $\mathcal{F}_{i-1,s}^{setup}$ to $\mathcal{F}_{i,s}^{setup}$, and append piece from $\alpha_s + break$ to $\min\{\eta_s + break - 0.1, \omega(\mathcal{H})\}$ with value $(0,0)$ to $\mathcal{F}_{i,s}^{setup}$;
3 **return** $\left(\mathcal{S}_i^0, (\mathcal{F}_{i,s}^{setup})_{s \in \mathcal{S}_i^0}\right)$;

Function SetupMDshiftable($\mathcal{L}_{i-1}^{driven}$)

forall *start times s in* $\mathcal{S}_{i-1}^{+}$ **do**
  $k = 1$;
  forall *time windows* $\mathcal{W}_i^j$ *of customer i in chronological order* **do**
    if $\alpha_s > \omega(\mathcal{W}_i^j)$ **then**
      continue;
    $shift := \max\{\alpha(\mathcal{W}_i^j) - \alpha_s, 0\}$;
    if $shift < \omega_s - \alpha_s$ **then**
      if $\omega(\mathcal{W}_i^j) < \omega_s$ **then**
        add $s + \omega(\mathcal{W}_i^j) - \alpha_s$ to $\mathcal{S}_i^0$, and append piece from $\omega(\mathcal{W}_i^j)$ to $\omega(\mathcal{W}_i^j)$ with value $\mathcal{F}_{i-1,s}^{setup}(\alpha_s)$ to $\mathcal{F}_{i,s+\omega(\mathcal{W}_i^j)-\alpha_s}^{setup}$;
        if $\alpha_s + shift < \omega(\mathcal{W}_i^j)$ **then**
          add $s + shift$ to $\mathcal{S}_i^+$, and append piece from $\alpha_s + shift$ to $\omega(\mathcal{W}_i^j)$ with value $\mathcal{F}_{i-1,s}^{setup}(\alpha_s)$ to $\mathcal{F}_{i,s+shift}^{setup}$;
      else
        add $s + shift$ to $\mathcal{S}_i^+$, and append piece from $\alpha_s + shift$ to ω_s with value $\mathcal{F}_{i-1,s}^{setup}(\alpha_s)$ to $\mathcal{F}_{i,s+shift}^{setup}$;
    while $k \leq j$ **do**
      $minShift := \max\{\alpha(\mathcal{W}_i^j) - (\alpha_s + break), \alpha(\overline{\mathcal{W}}_i^k) - \alpha_s, 0\}$;
      if $\alpha_s + minShift > \omega(\overline{\mathcal{W}}_i^k)$ **then**
        k++;
        continue;
      $maxShift := \omega(\mathcal{W}_i^j) - (\alpha_s + break)$;
      if $minShift > maxShift \vee minShift \geq \omega_s - \alpha_s$ **then**
        break;
      $latest := \min\{\omega_s + break, \eta_s + break, \alpha(\mathcal{W}_i^k) + break\}$;
      if $\omega(\mathcal{W}_i^j) < latest$ **then**
        add $s + maxShift$ to $\mathcal{S}_i^0$, and append piece from $\omega(\mathcal{W}_i^j)$ to $\omega(\mathcal{W}_i^j)$ with value $(0,0)$ to $\mathcal{F}_{i,s+maxShift}^{setup}$;
        if $minShift < maxShift$ **then**
          add $s + minShift$ to $\mathcal{S}_i^+$, and append piece from $\alpha_s + break + minShift$ to $\omega(\mathcal{W}_i^j)$ with value $(0,0)$ to $\mathcal{F}_{i,s+minShift}^{setup}$;
      else
        add $s + minShift$ to $\mathcal{S}_i^+$, and append piece from $\alpha_s + break + minShift$ to $latest$ with value $(0,0)$ to $\mathcal{F}_{i,s+minShift}^{setup}$;
      if $\omega(\overline{\mathcal{W}}_i^k) - \alpha_s > maxShift$ **then**
        break;
      k++;
return $\left(\mathcal{S}_i^+, \mathcal{S}_i^0, (\mathcal{F}_{i,s}^{setup})_{s \in \mathcal{S}_i^+ \cup \mathcal{S}_i^0}\right)$;

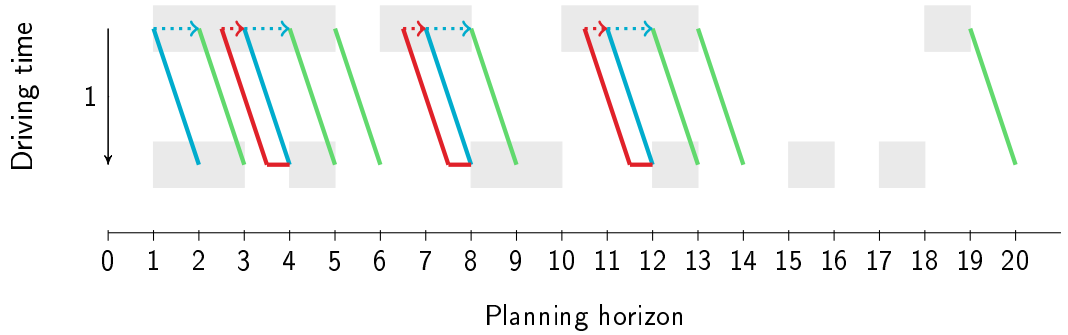


FIGURE 4.7: Example of section 4.3.3. Situation after step *Setup* of second iteration.

4.3.4 Other Steps in Detail

In step *Wait*, hardly any changes need to be made compared to the same step of the EF-TDSP algorithm in section 3.3.3.2. For all $s \in \mathcal{S}_i^+$, there is nothing to do because by construction, each piece is only defined within a time window. For all $s \in \mathcal{S}_i^0$, we proceed very similar to before. The only difference is that we do not want the driver to wait until after $\omega(\mathcal{F}_{i,s}^{setup})$ for dominance reasons. So when we loop over all $s \in \mathcal{S}_i^0$, we redefine H^{shift} each time:

$$H^{shift}(t) := \max\{t' \mid t = t' + W_i(t') \wedge \mathcal{F}_{i,s}^{setup}(t') \neq (\bot, \bot)\}$$

only for all $t \leq \omega(\mathcal{F}_{i,s}^{setup})$ for which the maximum is taken from a non-empty set, and $\bot$ for all other t. The remainder of each loop is as before.

In step *Serve*, nothing has to be changed compared to section 3.3.3.3. For all $s \in \mathcal{S}_i$, we proceed as before.

In step *Drive*, we compute $\mathcal{F}_{i,s}^{driven}$ for all $s \in \mathcal{S}_i^0$ as in section 3.3.3.4. For all $s \in \mathcal{S}_i^+$, the calculation is similar to before, but as long as there is slack in a schedule, we do not prolong a break. So let s be from $\mathcal{S}_i^+$ in the following, and let t be a point in time for which the piece of $\mathcal{F}_{i,s}^{served}$ is defined. Then

$$h := drive_i - \min\{limitD - D_{i,s}^{served}(t - drive_i), limitT - T_{i,s}^{served}(t - drive_i)\}$$

is the part of the driving time that exceeds the time that the driver is allowed to drive. If $h \leq 0$, a break en route is not necessary, so we set

$$\mathcal{F}_{i,s}^{driven}(t) := \left(D_{i,s}^{served}(t - drive_i) + drive_i, T_{i,s}^{served}(t - drive_i) + drive_i\right)$$

for all $t \in \mathcal{H}$ for which $\mathcal{F}_{i,s}^{served}(t - drive_i) \neq (\bot, \bot)$ holds, and $\mathcal{F}_{i,s}^{driven}(t) := (\bot, \bot)$ for all other t. On the other hand, if $0 < h \leq limitD$, we set

$$\mathcal{F}_{i,s}^{driven}(t) := (h, h)$$

for all $t \in \mathcal{H}$ for which $\mathcal{F}_{i,s}^{served}(t - drive_i - break) \neq (\bot, \bot)$ holds, and $\mathcal{F}_{i,s}^{driven}(t) := (\bot, \bot)$ for all other t. Should h even exceed $limitD$, more than one break becomes mandatory en route, so we have to calculate the number of additional breaks first, before we shift and raise $\mathcal{F}_{i,s}^{driven}$ accordingly (analogue to the approach in section 3.3.3.4).

4.3.5 Complexity Analysis

We claim that the MD-TDSP algorithm has a polynomial time bound. For the proof, we find that some arguments of the corresponding proof for the EF-TDSP algorithm (section 3.3.5) can be re-used. Again, we observe that the algorithm can be implemented in a way such that it runs in a time that is linear in the overall number of pieces stored in the function pairs of the labels created during the algorithm. And we can also re-use the argument why the number of pieces per function pair (and thus per significant start time) is in $O(w)$, where w is the total number of time windows. So a proof mainly boils down to answering one question: How can we know that the number of significant start times found during the algorithm is bounded by a polynomial?

Before we go into the details of the proof, we give an example of how the set of significant start times evolves from iteration to iteration (section 4.3.5.1). With the help of this example, it is a lot easier to comprehend the idea behind the proof set out in section 4.3.5.2. But even before that example, we need to introduce some more notation.

Function Intervals and Start Time Intervals The characteristic of the shiftable start times in $\mathcal{S}_i^+$ is that the associated functions are only defined over a single interval. For this section, let $\mathcal{I}_{i,s}^{step} := [\alpha(\mathcal{F}_{i,s}^{step}), \omega(\mathcal{F}_{i,s}^{step}))$ be the (half-open) *function interval* over which the function pair $\mathcal{F}_{i,s}^{step}$ is defined for a step in iteration i and a start time $s \in \mathcal{S}_i^+$. Moreover, let $\mathcal{J}_{i,s}^{step} := [s, s + \omega(\mathcal{F}_{i,s}^{step}) - \alpha(\mathcal{F}_{i,s}^{step}))$ be the *start time interval* for iteration i and start time $s \in \mathcal{S}_i^+$. The following observations are essential:

- The start time intervals $\mathcal{J}_{i,s}^{step}$ do not change from step to step within the same iteration, so we can leave the superscript out and simply write $\mathcal{J}_{i,s}$.
- By construction, the start time intervals $\mathcal{J}_{i,s}$ are pairwise disjoint for any two start times from $\mathcal{S}_i^+$ in iteration i.
- However, the function intervals $\mathcal{I}_{i,s}^{step}$ may overlap for two start times from $\mathcal{S}_i^+$, and not only partially. There are cases in which a function interval overlaps another entirely, as is the case in Figures 4.3 and 4.4. Here, the function interval for start time 0 spans from 9 to 11 and the function interval for start time 4 spans from 10 to 10.5.

For the proof of the polynomial time bound, it is crucial that not all function intervals overlap. In fact, for an iteration i, a partition of the set of all function intervals $\mathcal{I}_{i,s}^{step}$ exists such that the intervals in the subsets are pairwise disjoint. Before we present the proof(s) in section 4.3.5.2, we turn towards an example in the next section 4.3.5.1. Here, we also introduce the start time tree for illustration purposes.

4.3.5.1 How the Set of Significant Start Times Evolves

A key to understanding the algorithm and our analysis is to have a look at the *start time tree*. It visualizes how the sets of significant start times evolve from iteration to iteration. We first introduce another example instance, one with more customers but fewer time windows. Figure 4.8 shows the time windows of the first five customers of a route that are all a driving time of 1 apart. Only customer 4 has two time windows, all other customers have a single time window. The service times are all 0, the planning horizon is from 0 to "open end", $break = 5$, and $limitD = 2.5$. So customer

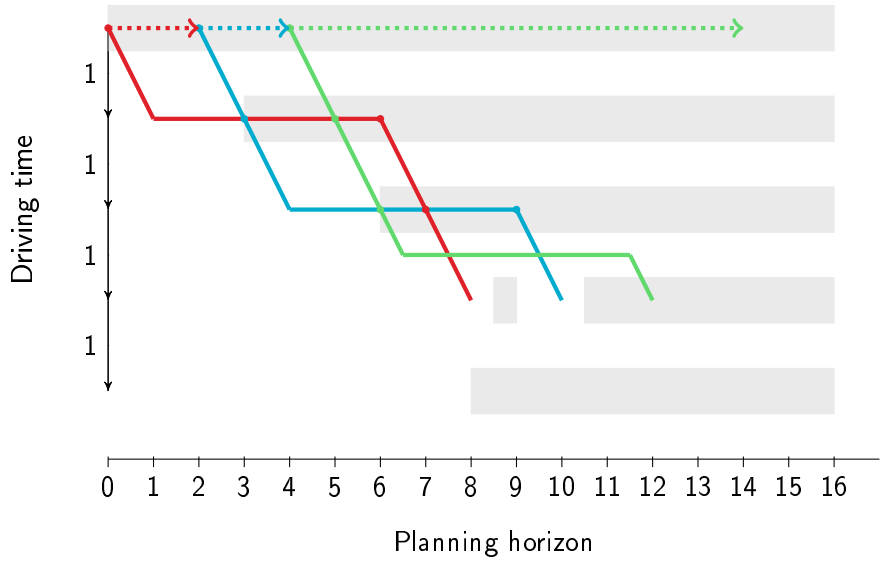


FIGURE 4.8: Example instance. Significant schedules created in the first three iterations.

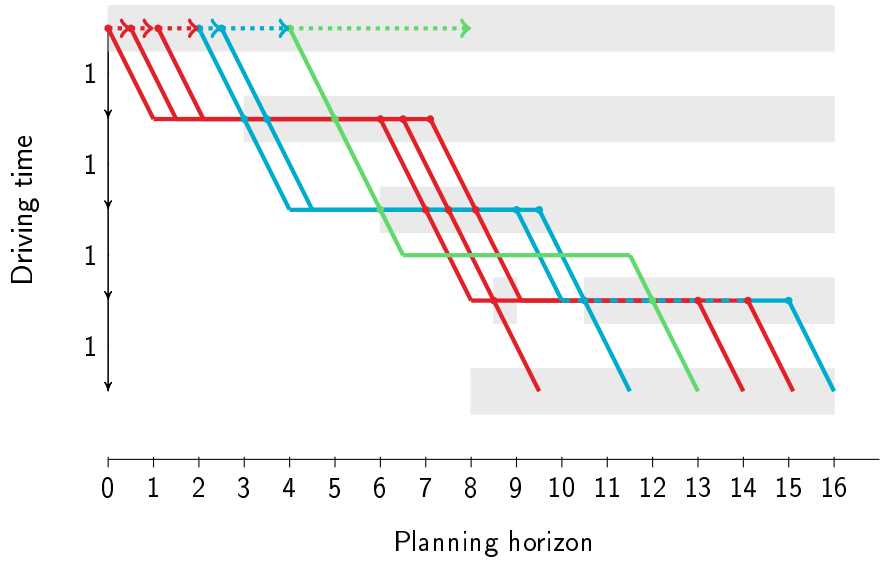


FIGURE 4.9: Example instance. Significant schedules created in the first four iterations.

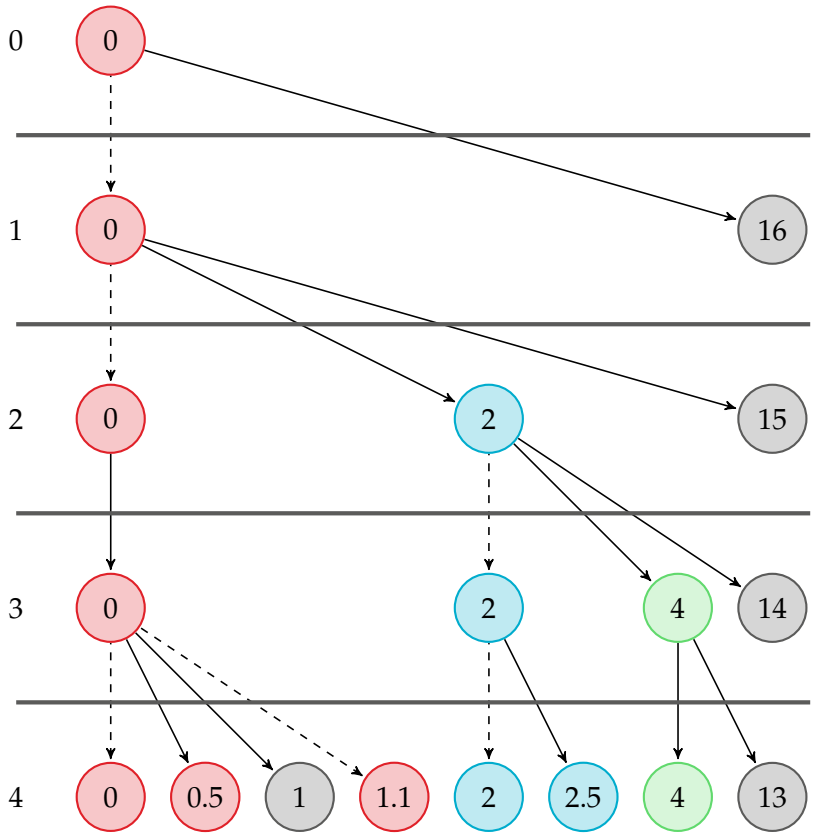


FIGURE 4.10: Start time tree after the first four iterations corresponding to the instance from Figures 4.8 and 4.9.

4 cannot be reached without a break. Also three schedules are depicted in Figure 4.8. They correspond to the start times 0, 2, and 4 in $\mathcal{S}_3^+$. In addition, Figure 4.9 displays the schedules for the start times in $\mathcal{S}_4^+$.

The start time tree that refers to the instance of Figures 4.8 and 4.9 is depicted in Figure 4.10. The tree has several levels, one for each iteration. The numbers in the nodes of level i represent the significant start times in the start time set $\mathcal{S}_i$. So every node is identified by the iteration/level and the start time. On every level, the nodes for the start times in $\mathcal{S}_i^0$ are gray, those for the start times in $\mathcal{S}_i^+$ are colored. The different colors correspond to the schedules in Figures 4.8 and 4.9 of the same color. The schedules for the start times in $\mathcal{S}_i^0$ are not shown in these figures for better clarity. There is an edge from node v_1 on level $i-1$ to node v_2 on level i if the start time of v_2 emerges from the start time of v_1, i.e., v_2 is inserted into $\mathcal{S}_i$ when the loop has reached the start time of v_1. Such an edge is dashed if the start times of the incident nodes are shiftable, and an early break (or that implicit break before the planning horizon begins) is taken right before service at customer i.

Gray nodes of fixed start times can have at most one successor node, and that successor node is gray again and has the same start time. Colored nodes of shiftable start times can have multiple successors. So in order to know how many nodes are at most on one level (i.e. how many significant start times are at most contained in the corresponding set), it is sufficient to find out how many successor nodes the colored nodes (i.e. shiftable start times) can have. For the sake of simplicity, we may also speak of successors (and predecessors) of start times instead of nodes in the following.

It is notable that the number of leaves in the tree doubles from iteration 3 to iteration 4. Likewise, the number of schedules in Figure 4.9 are twice as many as in Figure 4.8. It is obvious that this increase has to do with the fact that customer 4 has two time windows instead of one like the other customers. But it is not obvious how and why it is polynomial bounded.

Table 4.4 shows the start times, the start time intervals, and the function intervals computed in the first four iterations of the algorithm. It shows the start time intervals $\mathcal{J}_{i,s}$ for each start time in $\mathcal{S}_i^+$, the start times in $\mathcal{S}_i^0$, and the function intervals $\mathcal{I}_{i,s}^{step}$ for the steps *Setup* and *Drive* and each start time in $\mathcal{S}_i^+$.

TABLE 4.4: Start time intervals and function intervals computed in the first four iterations.

Iteration i	$\mathcal{J}_{i,s} : s \in \mathcal{S}_i^+$	$s \in \mathcal{S}_i^0$	$\mathcal{I}_{i,s}^{setup} : s \in \mathcal{S}_i^+$	$\mathcal{I}_{i,s}^{driven} : s \in \mathcal{S}_i^+$
0	[0,∞)	-	-	[0,∞)
1	[0,16)	16	[0,16)	[1,17)
2	[0,2)	15	[6,8)	[7,9)
	[2,15)		[3,16)	[4,17)
3	[0,2)	14	[7,9)	[8,10)
	[2,4)		[9,11)	[10,12)
	[4,14)		[6,16)	[12,22)
4	[0,0.5)	1, 13	[13,13.5)	[14,14.5)
	[0.5,1)		[8.5,9)	[9.5,10),
	[1.1,2)		[14.1,15)	[15.1,16)
	[2,2.5)		[15,15.5)	[16,16.5),
	[2.5,4)		[10.5,12)	[11.5,13)
	[4,8)		[12,16)	[13,17)

Now let us go through the example instance iteration by iteration.

0. Initially, only 0 is considered as a significant start time and added to $\mathcal{S}_0^+$. Since the planning horizon has an open end, both $\mathcal{J}_{0,0}$ and $\mathcal{I}_{0,0}^{driven}$ are $[0, \infty)$.
1. In iteration 1, the time window $[0, 16]$ of the first customer is regarded. 0 is again added to the shiftable start times, 16 is added to the fixed start times.
2. In iteration 2, start times 0 (arrival outside of time window) and 2 (arrival within time window) are added to $\mathcal{S}_2^+$. Start time 15 is added to $\mathcal{S}_2^0$, start time 16 is discarded.
3. In iteration 3, the number of shiftable start times grows by 1 again. Start time 2 is added to $\mathcal{S}_3^+$ (arrival outside of time window), as well as 0 and 4 (arrival within time window). 14 is added to $\mathcal{S}_3^0$, 15 is discarded.
4. In Iteration 4, it is the first time that two time windows ($[8.5, 9]$ and $[10.5, 16]$) need to be regarded. We learn from the start time tree which start times emerge from which start times on level $i - 1$. Start times 0, 1.1 (assuming that time elapses in discrete steps of 0.1 time units), and 2 are added to $\mathcal{S}_4^+$ (arrival outside of time window, hence the dashed edges), as well as 0.5, 2.5, and 4 (arrival within time window). 1 and 13 are added to $\mathcal{S}_4^0$, 14 is discarded.

4.3.5.2 Partitioning the Set of Shiftable Start Times

At first, let us give an outline of the proof because some parts will be rather technical. As we have already pointed out in the beginning, the time complexity is the same as the space complexity, so we again count the number of created pieces. We already know that the functions associated with a shiftable start time have exactly one piece whereas those associated with a fixed start time can have $O(w)$ many pieces. How many fixed start times can there be?

A fixed start time can have only one child in the start time tree, and it is itself always a descendant (not necessarily direct descendant) of a shiftable start time. So to prove a polynomial bound on the number of fixed start times, it is sufficient to focus on the number of shiftable start times.

Here, a myopic approach does not work. When the function interval associated with a shiftable start time from $\mathcal{S}_{i-1}^+$ is intersected with the w_i time windows of customer i, this may lead to $O(w_i)$ new shiftable start times. That would not be a polynomial bound. The problem is that function intervals may overlap. We have to find a partition of the shiftable start times such that the associated function intervals in every subset of the partition are pairwise disjoint. Such a partition with at most i subsets exists. This leads to $O(w_i)$ new shiftable start times per subset of the partition (and not per each and every shiftable start time). Putting everything together then results in a polynomial time bound.

Notation It is convenient to introduce the following notation: Let $\mathcal{S}_i^{rested} \subset \mathcal{S}_i^+$ be the shiftable start times such that the driver is completely rested before service at customer i, i.e., for an $s \in \mathcal{S}_i^{rested}$, $\mathcal{F}_{i,s}^{setup}(t) := (0, 0)$ for all t for which the function pair is defined. Initially, $\mathcal{S}_0^{rested} := \mathcal{S}_0^+$ and also $\mathcal{S}_1^{rested} := \mathcal{S}_1^+$. Whenever a start time is added to $\mathcal{S}_i^+$ in the loop over the arrivals within waiting intervals (section 4.3.3.1) in a subsequent iteration i, this start time is also inserted into $\mathcal{S}_i^{rested}$.

For a start time $s \in \mathcal{S}_i^+$ in iteration i, let $pred(s) \in \mathcal{S}_{i-1}^+$ be the *predecessor start time* of s in iteration $i-1$. We enhance this definition recursively and also write $pred^0(s) := s$ and $pred^k(s) := pred^{k-1}(pred(s)) \in \mathcal{S}_{i-k}^+$ for an $s \in \mathcal{S}_i^+$ and a k from 1 to i. For convenience, we introduce $s(k)$ as short notation for the predecessor start time $pred^{i-k}(s)$ in iteration $k \leq i$ of a start time $s \in \mathcal{S}_i^+$.

We find that for an $s \in \mathcal{S}_i^+$, $s(1)$ is included in $\mathcal{S}_1^{rested}$ and so there is always a k such that $s(k) \in \mathcal{S}_k^{rested}$ holds. We make use of this when we define the set $\mathcal{S}_{i,j}^+$ for every $j \leq i$ to be the set of shiftable start times in iteration i with the property that there is no larger value than j such that $s(j)$ is included in $\mathcal{S}_j^{rested}$, i.e.,

$$\mathcal{S}_{i,j}^+ := \{s \in \mathcal{S}_i^+ \mid \max\{k \leq i \mid s(k) \in \mathcal{S}_k^{rested}\} = j\}$$

We observe that $\mathcal{S}_{i,i}^+ = \mathcal{S}_i^{rested}$, and that the $\mathcal{S}_{i,j}^+$ are pairwise disjoint. In fact, the set $\mathcal{S}_i^+$ is a disjoint union of $\mathcal{S}_{i,j}^+$ for all $j \leq i$. In the example, $\mathcal{S}_{4,4}^+ := \{0, 1.1, 2\}, \mathcal{S}_{4,3}^+ := \{2.5\}, \mathcal{S}_{4,2}^+ := \{0.5\}$, and $\mathcal{S}_{4,1}^+ := \{4\}$.

Now, let us recall two important findings of previous sections.

1. For a shiftable start time s from $\mathcal{S}_{i-1}^+$, the intervals $\mathcal{I}_{i,s'}^{setup}$ with $s' := s + t^{in} - \alpha(\mathcal{F}_{i-1,s}^{driven})$ are pairwise disjoint for all $t^{in} \in \mathcal{T}_{i,s}^{in}$ (see section 4.3.3.1). In other words: The intervals $\mathcal{I}_{i,s'}^{setup}$ are pairwise disjoint for all shiftable start times s' from $\mathcal{S}_i^+ \setminus \mathcal{S}_i^{rested}$ for which the predecessor start time is the same.

2. The intervals $\mathcal{I}_{i,s'}^{setup}$ with $s' := s + t^{out} - \alpha(\mathcal{F}_{i-1,s}^{driven})$ are pairwise disjoint for all $t^{out} \in \mathcal{T}_{i,s}^{out}$ and all shiftable start times s from $\mathcal{S}_{i-1}^+$ (see section 4.3.3.1). In other words: The intervals $\mathcal{I}_{i,s'}^{setup}$ are pairwise disjoint for all shiftable start times s' from $\mathcal{S}_i^{rested}$.

We complement these by some observations that help us prove Lemma 4.3.1.

3. If for two shiftable start times s and t from $\mathcal{S}_i^+$ the intervals $\mathcal{I}_{i,s}^{setup}$ and $\mathcal{I}_{i,t}^{setup}$ are disjoint but the accumulated times of $\mathcal{F}_{i,s}^{setup}$ and $\mathcal{F}_{i,t}^{setup}$ are equal, then the intervals $\mathcal{I}_{i,s}^{step}$ and $\mathcal{I}_{i,t}^{step}$ are also disjoint and the accumulated times from $\mathcal{F}_{i,s}^{step}$ and $\mathcal{F}_{i,t}^{step}$ are still equal. This holds for all $step \in \{Wait, Serve, Drive\}$. In step *Wait*, nothing is to be done for start times from $\mathcal{S}_i^+$. And in step *Serve* resp. *Drive*, both intervals are shifted by the service time resp. the driving time and, if need be, equally by the minimum break duration because the accumulated times are the same.

4. If for two shiftable start times s and t from $\mathcal{S}_i^+ \setminus \mathcal{S}_i^{rested}$ the intervals $\mathcal{I}_{i-1,s(i-1)}^{driven}$ and $\mathcal{I}_{i-1,t(i-1)}^{driven}$ are disjoint, then by construction $\mathcal{I}_{i,s}^{setup} \subset \mathcal{I}_{i-1,s(i-1)}^{driven}$ and $\mathcal{I}_{i,t}^{setup} \subset \mathcal{I}_{i-1,t(i-1)}^{driven}$, so the intervals $\mathcal{I}_{i,s}^{setup}$ and $\mathcal{I}_{i,t}^{setup}$ are also disjoint. The accumulated times of $\mathcal{F}_{i-1,s(i-1)}^{driven}$ and $\mathcal{F}_{i,s}^{setup}$ (as well as $\mathcal{F}_{i-1,t(i-1)}^{driven}$ and $\mathcal{F}_{i,t}^{setup}$) are the same.

With the help of the observations 1 through 4, we can easily prove the following lemma.

Lemma 4.3.1. *For every iteration i, every previous iteration $j \leq i$, and every $step \in \{setup, waited, served, driven\}$, the intervals $\mathcal{I}_{i,s}^{step}$ and $\mathcal{I}_{i,t}^{step}$ are disjoint and the accumulated times of $\mathcal{F}_{i,s}^{step}$ and $\mathcal{F}_{i,t}^{step}$ are the same for any two distinct start times s and t from the set $\mathcal{S}_{i,j}^+$.*

Proof. For some i and $j \leq i$, let two distinct start times s and t from $\mathcal{S}^+_{i,j}$ be given. The predecessor start times of s and t in iteration j are denoted by $s(j)$ and $t(j)$. By definition of $\mathcal{S}^+_{i,j}$, $s(j)$ and $t(j)$ are contained in $\mathcal{S}^{rested}_j$, so the accumulated times of $\mathcal{F}^{setup}_{j,s(j)}$ and $\mathcal{F}^{setup}_{j,t(j)}$ are the same, namely $(0,0)$ wherever the functions are defined. If $s(j) \neq t(j)$, it follows from observation 2 that the intervals $\mathcal{I}^{setup}_{j,s(j)}$ and $\mathcal{I}^{setup}_{j,t(j)}$ are disjoint. However, if $s(j) = t(j)$, let k be the earliest iteration after j such that $s(k) \neq t(k)$. Then it follows from observation 1 that the intervals $\mathcal{I}^{setup}_{k,s(k)}$ and $\mathcal{I}^{setup}_{k,t(k)}$ are disjoint. In both cases, the accumulated times associated with the start times are the same, i.e., the accumulated times of $\mathcal{F}^{setup}_{j,s(j)}$ and $\mathcal{F}^{setup}_{j,t(j)}$ as well as $\mathcal{F}^{setup}_{k,s(k)}$ and $\mathcal{F}^{setup}_{k,t(k)}$ are the same. So it follows from observations 3 and 4 that also $\mathcal{I}^{step}_{i,s}$ and $\mathcal{I}^{step}_{i,t}$ must be disjoint. □

For two interval sets $\mathcal{I}$ and $\mathcal{J}$ that are each pairwise disjoint, let the intersection $\mathcal{I} \Cap \mathcal{J}$ be defined elementwise: $\mathcal{I} \Cap \mathcal{J} := \{I \cap J \mid I \in \mathcal{I}, J \in \mathcal{J}, I \cap J \neq \emptyset\}$. $\mathcal{I} \Cap \mathcal{J}$ is again a pairwise disjoint interval set. To prove Lemma 4.3.3, the following lemma is very helpful:

Lemma 4.3.2. *For two pairwise disjoint interval sets $\mathcal{I}$ and $\mathcal{J}$, $\mathcal{I} \Cap \mathcal{J}$ contains at most $|\mathcal{I}| + |\mathcal{J}| - 1$ intervals, unless both sets are empty.*

Proof. W.l.o.g. let $\mathcal{J}$ be non-empty. $\mathcal{I} \Cap \mathcal{J}$ contains an element for every element in $\mathcal{I}$ that covers at least a part of an interval from $\mathcal{J}$ (at most $|\mathcal{I}|$ many). In addition, $\mathcal{I} \Cap \mathcal{J}$ contains an element for every gap (including the endpoints of the enclosing intervals) between two intervals from $\mathcal{J}$ that is covered in full by an element from $\mathcal{I}$ (at most $|\mathcal{J}| - 1$ many). Further elements are not contained in $\mathcal{I} \Cap \mathcal{J}$. □

For the following lemma, we again introduce a short notation and just write $\mathcal{I}(\mathcal{S}^+_{i,j})$ instead of $\{\mathcal{I}^{driven}_{i,s} \mid s \in \mathcal{S}^+_{i,j}\}$ for some $j \leq i$. Trivially, $|\mathcal{I}(\mathcal{S}^+_{i,j})| = |\mathcal{S}^+_{i,j}|$. According to Lemma 4.3.1, $\mathcal{I}(\mathcal{S}^+_{i,j})$ is a set of pairwise disjoint intervals.

We are interested in at most how many more start times can be contained in $\mathcal{S}^+_i$ compared to $\mathcal{S}^+_{i-1}$.

Lemma 4.3.3. *For every $j \leq i-1$, at most $|\mathcal{S}^+_{i-1,j}| + 3w_i - 2$ shiftable (or fixed) start times with a predecessor start time in $\mathcal{S}^+_{i-1,j}$ are added to $\mathcal{S}^+_i$ (or $\mathcal{S}^0_i$) in step Setup of some iteration $i > 1$.*

Proof. On one hand, a start time is added to $\mathcal{S}^+_i$ (or $\mathcal{S}^0_i$) for every interval in the intersection of $\mathcal{I}(\mathcal{S}^+_{i-1,j})$ with the w_i time windows of customer i, that is, for every interval in $\mathcal{I}(\mathcal{S}^+_{i-1,j}) \Cap \mathcal{W}_i$ (arrival within time windows). On the other hand, a start time is added to $\mathcal{S}^+_i$ (or $\mathcal{S}^0_i$) for every interval in the set that we get when we first intersect $\mathcal{I}(\mathcal{S}^+_{i-1,j})$ with the w_i waiting intervals $\overline{\mathcal{W}}_i$, shift the outcome to the right by the minimum break duration, and then intersect it again with the time windows $\mathcal{W}_i$ (arrival within waiting intervals). In this case, the final interval set contains at most $w_i - 1$ more intervals than there are elements in $\mathcal{I}(\mathcal{S}^+_{i-1,j}) \Cap \overline{\mathcal{W}}_i$, according to Lemma 4.3.2. Since the sets $\mathcal{W}_i$ and $\overline{\mathcal{W}}_i$ are disjoint, $|\mathcal{I}(\mathcal{S}^+_{i-1,j}) \Cap \mathcal{W}_i| + |\mathcal{I}(\mathcal{S}^+_{i-1,j}) \Cap \overline{\mathcal{W}}_i| + w_i - 1 = |\mathcal{I}(\mathcal{S}^+_{i-1,j}) \Cap (\mathcal{W}_i \cup \overline{\mathcal{W}}_i)| + w_i - 1$ holds. And since the interval set $\mathcal{W}_i \cup \overline{\mathcal{W}}_i$ contains exactly $2w_i$ elements, we conclude that $|\mathcal{I}(\mathcal{S}^+_{i-1,j}) \Cap (\mathcal{W}_i \cup \overline{\mathcal{W}}_i)| + w_i - 1 \leq |\mathcal{I}(\mathcal{S}^+_{i-1,j})| + 3w_i - 2$, once again according to Lemma 4.3.2. □

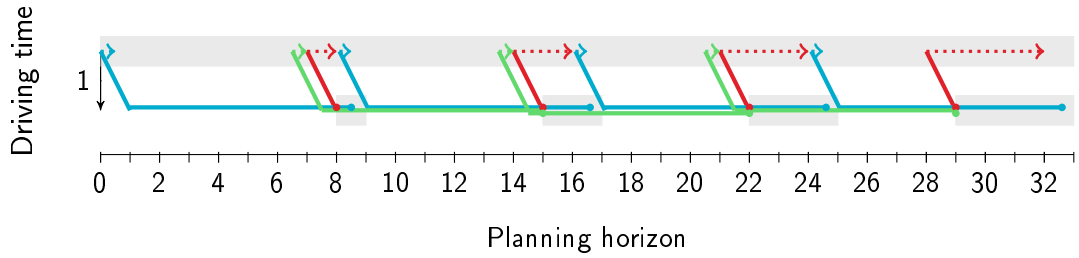


FIGURE 4.11: Worst case. The number of significant start times grows from 1 to 11.

Figure 4.11 shows a worst case example that proves that this bound is tight. Here, the first customer has only one very long time window, while the second customer has four short time windows. The minimum break duration is 7.5. In step *Setup* of the second iteration, the number of significant start times grows from $|\mathcal{S}_1^+| = 1$ to 11 in accordance with the formula $|\mathcal{S}_{1,1}^+| + 3w_2 - 2$ of Lemma 4.3.3 with $w_2 = 4$ and $|\mathcal{S}_{1,1}^+| = 1$.

Theorem 4.3.1. *The time and space complexity of the MD-TDSP algorithm is in $O(n^3w^2)$, where n is the number of customers and $w := \sum_{j=1}^{n} w_j$ is the total number of all time windows.*

Proof. In step *Setup* of some iteration $i > 1$, at most $\sum_{j=1}^{i-1} |\mathcal{S}_{i-1,j}^+| + 3w_i - 2 = |\mathcal{S}_{i-1}^+| + (i-1)(3w_i - 2)$ shiftable start times are added to $\mathcal{S}_i^+$. This follows from Lemma 4.3.3 and the fact that the set $\{\mathcal{S}_{i-1,1}^+, \ldots, \mathcal{S}_{i-1,i-1}^+\}$ is a partition of $\mathcal{S}_{i-1}^+$. We conclude that the cardinality of $\mathcal{S}_n^+$ is in $O(nw)$. During the loop over the shiftable start times, also at most $|\mathcal{S}_{i-1}^+| + (i-1)(3w_i - 2)$ fixed start times are inserted into $\mathcal{S}_i^0$, according to Lemma 4.3.3. During the loop over the fixed start times, at most another $|\mathcal{S}_{i-1}^0|$ start times are added. So $|\mathcal{S}_i^0| \leq |\mathcal{S}_{i-1}^0| + |\mathcal{S}_{i-1}^+| + (i-1)(3w_i - 2)$. This means that the cardinality of $\mathcal{S}_n^0$ is in $O(n^2w)$. From Lemma 3.3.1 we can conclude that the number of pieces stored in all function pairs in all n iterations is in $O(n^3w^2)$, and so is the space complexity of the MD-TDSP algorithm. The algorithm can be implemented in a way that it takes a time that is linear in the number of pieces. □

Some more remarks on an implementation: As the pseudo-code of SetupMDshiftable suggests, it is possible to iterate over the time windows (and waiting intervals) just once for a start time in $\mathcal{S}_i^+$ in step *Setup*. We could further improve the asymptotic run-time of this part by exploiting the idea of Lemma 4.3.3: Instead of iterating over all start times in $\mathcal{S}_i^+$ independently, we could process all start times in $\mathcal{S}_{i,j}^+$ within the same loop over all $j \leq i$. However, this does not affect the total asymptotic run-time as it is dominated by the loop over the fixed start times.

The start time sets could be maintained in sorted order at no additional costs. As side effect of sorted start times, determining the next earliest arrival times η_s for all start times $s \in \mathcal{S}_i$ in every iteration i takes only linear time in the number of start times.

We do not claim that the bound given in Theorem 4.3.1 is best possible. A more thorough analysis of the algorithm (or even an improved version thereof) may reveal a lower asymptotic bound for the MD-TDSP.

4.4 Discussion of a Problem Variant with Minimum Idle Cost Objective

The MD-TDSP is a special case of the following problem: Suppose two non-negative cost coefficients c_b and c_w are given with which the total idle time that counts as break (break time in the following) and the total idle time that does not count as break (waiting time) are weighted, respectively. Now we seek to find a feasible schedule such that the sum of costs for waiting and breaks are minimum.

This objective function corresponds to the one regarded by Xu et al. (2003) for a given sequence of customers. For easier reference, we will call it the *minimum idle cost* objective and speak of the MIC-TDSP, accordingly. If the coefficients c_w and c_b are the same, it coincides with the MD-TDSP. As already shown, this problem can be solved in polynomial time. But what if $c_w > c_b$ or $c_w < c_b$ holds?

Prefer Break over Waiting In fact, the driver may be paid less for taking a break than for waiting (if at all), so the case that break time costs less than waiting time may be relevant in practice. In this case, we claim that the MIC-TDSP is not harder (asymptotically) than the MD-TDSP. According to our terminology (see section 2.3.1), an idle time that is at least as long as *break* is deemed to be a break. This is because *not* considering it as a break would not have any benefit. Our algorithm for the MD-TDSP is based on the principle to always turn waiting time into break time if this is possible, i.e., to rather prolong a previous break instead of waiting for the opening of a time window. With $c_w > c_b$, this is still beneficial.

So all we have to do to turn our MD-TDSP algorithm into an MIC-TDSP algorithm is to add another function to $\mathcal{F}^{step}_{i,s}$. Precisely let $\mathcal{F}^{step}_{i,s}$ now be a function triple $(D^{step}_{i,s}, T^{step}_{i,s}, C^{step}_{i,s})$, where $C^{step}_{i,s}$ is a piecewise linear function that maps a time t to the *accumulated idle costs since start of the route* for given i, s, and $step$. This additional function is for reporting only. The algorithm itself does not change significantly. The same driver states are computed. But this time, we know the idle costs for every driver state in addition.

We leave out the details here. Finally, the algorithm returns a pair of the minimum idle cost itself and those start times that yield this cost:

$$\texttt{obj}(\mathcal{L}^{served}_n) := \left(\min_{s \in \mathcal{S}_n : \alpha(\mathcal{F}^{served}_{n,s}) \neq \bot} \min_t \{C^{served}_{n,s}(t)\}, \operatorname*{arg\,min}_{s \in \mathcal{S}_n : \alpha(\mathcal{F}^{served}_{n,s}) \neq \bot} \min_t \{C^{served}_{n,s}(t)\} \right)$$

So in this setting, the optimal finish time given a start time may not be the earliest finish time but one for which a schedule with more break time and less waiting time exists, even if that means a longer total duration.

Since the number of pieces is the same for all functions in the triple $\mathcal{F}$, the MIC-TDSP is not harder (asymptotically) than the MD-TDSP. So for the special case $c_w \geq c_b$, we have falsified the conjecture by Xu et al. (2003) who claimed that this problem could be *NP*-hard.

In the end, we may not only be interested in the idle costs but also the times themselves. The total waiting time can simply be calculated as the sum of the waiting times $t^{start}_i - t^{arr@c}_i$ over all customers i for which $t^{start}_i - t^{arr@c}_i$ is less than *break*, plus the sum of waiting times en route $t^{arr@c}_{i+1} - t^{dep@c}_i - drive_i$ over all $i < n$ for which $t^{arr@c}_{i+1} - t^{dep@c}_i - drive_i$ is less than *break*. On the other hand, the total break time is the sum of break times $t^{start}_i - t^{arr@c}_i$ over all customers i for which $t^{start}_i - t^{arr@c}_i \geq break$,

plus the sum of break times en route $t_{i+1}^{arr@c} - t_i^{dep@c} - drive_i$ over all $i < n$ for which $t_{i+1}^{arr@c} - t_i^{dep@c} - drive_i \geq break$.

Prefer Waiting over Break However if $c_w < c_b$, a different approach is necessary because the basic principle "taking a break is better than waiting" is no longer valid. We can no longer tell by the duration of an idle period whether it is meant to be a break or not. We have to leave the question concerning the complexity of this problem open.

4.5 Conclusion and Outlook

We have described an algorithm for a truck driver scheduling problem that is able to find a schedule with minimum duration if a feasible schedule exists. It contains the algorithm presented in the previous chapter as a subroutine and calls it for every significant start time. These significant start times are not known in advance but found as the algorithm progresses. As in the previous chapter, we have considered a single type of break and the two corresponding rulesets *RulesetUS* and *RulesetEU*. Furthermore, we have regarded the unrestricted break policy, that is, breaks en route between two customers are allowed (compare Goel (2012c)). As in the whole thesis, we have studied the case with multiple time windows per customer. The main contribution is the proof that it is a polynomial-time algorithm. Even in the single time window case, a polynomial-time algorithm for the two considered problem variants had not been proposed before.

We have shortly discussed the minimum idle cost objective and outlined how to adjust the algorithm such that breaks can be preferred over waiting time even more. So if $c_w \geq c_b$ holds for the cost coefficients, the MIC-TDSP can be solved in polynomial time. This falsifies the conjecture of Xu et al. (2003) in this case. If $c_w < c_b$ holds though, the complexity of the problem remains an open question. However, it is at least not apparent to us in how far a longer schedule would be preferred over a shorter schedule, just because it contains less break time but more waiting time. Unfortunately, Xu et al. (2003) do not motivate this scenario. For future research, it may be an interesting question how to consider an objective function that is a linear combination of three terms, two for the idle time costs as before and another that induces a fixed cost for every break (to take account of the cost of overnight stays, for example).

Chapter 5

Truck Driver Scheduling with Two Types of Breaks

"A change is as good as a rest."
— British saying

5.1 Introduction

For a planning horizon of one day, it is sufficient to schedule a lunch break and make sure that the total driving time and the total travel time do not exceed their respective limit. However, a long-haul truck driver may be en route for several days or even longer than a week. For such a driver, sufficiently long rest breaks must be scheduled at the end of each day in order to ensure that the driver can get enough sleep. In general, the longer the planning horizon is, the more types of breaks and the more rules in regard to driving and travel time limits come into play and need to be respected. In the European Union for instance, a truck driver has to take a so-called "daily rest period" of 11 hours after at most 9 hours of driving or 13 hours have elapsed since the end of the last daily rest period (European Parliament and Council of the European Union, 2006). In the United States, similar rules apply. Here, a truck driver must be off-duty for at least 10 hours after an accumulated driving time of 11 hours or being on-duty for at most 14 hours (Federal Motor Carrier Safety Administration, 2011). In both cases, these rules are effective in addition to the lunch break rules already considered in previous chapters.

In this chapter, we investigate the complexity of truck driver scheduling problems with two types of breaks. Roughly speaking, this means that the driver has to take a short break after a short period of driving or traveling and a long break after a long period of driving or traveling. In our model, we suppose that short and long breaks only differ in their duration, and thus every long break also counts as short break. As in the two previous chapters, we regard two different rulesets per break type, one that is derived from the rules in the US and one that is derived from the rules in the EU. We propose a polynomial-time algorithm for two problem variants where taking a break of any of the two types is only allowed at customers. This algorithm is again based on the one that we have described before in chapter 3.

To decide whether a feasible schedule exists or not constitutes the *truck driver scheduling problem with two types of breaks* (TDSP-2B). In this chapter, we consider the earliest finish time objective, i.e., we search for a feasible schedule with the earliest finish time if a feasible schedule exists (EF-TDSP-2B).

No-break-en-route policy We obtain an important special case of the problem if we do not allow the driver to take a break when he is en route between two customers. We call this the *no-break-en-route policy*. This policy leaves the driver only the two options to either take a break *before* or *after* the service at a customer. It is for the sake of simplicity and without loss of generality when we allow the driver to take a break *only before* the service. If we wanted to also allow the other option, we could simply adapt the problem instance by inserting a "shadow" customer right after each (original) customer other than the last. These dummy customers demand zero service, have a non-restrictive time window, and the driving times to their preceding customers are all zero. In an analogue way, we could even add dummy customers that do not represent real customers but rest areas along the route (see Goel (2012c) or Kok, Hans, and Schutten (2011)). This may alleviate the negative effects that the restriction to breaks only at customers may have.

Related Work In this chapter, we focus on the EF-TDSP-2B under the no-break-en-route policy. This problem could be solved by the algorithm presented by Goel (2012c), for instance. However, to the best of our knowledge, no polynomial-time algorithm is known for this problem, so we are the first to present one. For the general EF-TDSP-2B with a non-restrictive break policy, we know of no result – or conjecture even – regarding the complexity of this problem. In a sense, it can be thought of as a subproblem of the problem studied by Drexl and Prescott-Gagnon (2010). And these authors conjecture that their problem at hand is *NP*-complete. However, they consider only a single time window per customer and only the parameter setting as it is valid in the EU, not an arbitrary break rule parameter setting as we do.

Contribution We regard both the variant with respect to the no-break-en-route policy and the variant with respect to a non-restrictive break policy. The main contribution is to present a polynomial-time algorithm for the EF-TDSP-2B under the no-break-en-route policy. In addition, we show that in the general case – due to early long breaks en route – the number of non-dominated driver states does not only depend on the number of customers and time windows but also depends on the rule parameter setting and the driving times. This brings us to conjecture that a strongly polynomial-time algorithm does not exist in the general case.

Outline In section 5.2, we give a formal definition of the EF-TDSP-2B under the no-break-en-route policy. In section 5.3, we describe characteristics of this problem. A polynomial-time algorithm for it is presented in section 5.4. In section 5.5, we discuss the general case in which breaks en route are allowed. Finally, in section 5.6, we conclude and give an outlook.

5.2 Problem Definition

This chapter deals with the *truck driver scheduling problem with two types of breaks*. The general setting of the problem is as before: A sequence of n customers is given, and each customer i may have $w_i \geq 1$ time windows $\mathcal{W}_i^1, \ldots, \mathcal{W}_i^{w_i}$ in which the requested service of length $service_i$ may start. The driving time between consecutive customers i and $i+1$ is known to be $drive_i$. The driver works only within a given planning horizon $\mathcal{H}$. Without loss of generality, we expect all time windows to lie within this horizon. Furthermore, we distinguish two types of breaks. It is also without loss of generality when we assume that the duration of one type is shorter than the

duration of the other, so we can call them *short break* and *long break*, respectively. Let the duration of these be $break_{short}$ and $break_{long}$, respectively.

For each of the two break types, there is a corresponding ruleset. In this work, we discuss two problem variants, differing in these rulesets. In one variant, the two corresponding rulesets are $RulesetUS_{short}$ and $RulesetUS_{long}$, respectively. In the other variant, the corresponding ruleset of the short break is $RulesetEU_{short}$ and the one corresponding to the long break is $RulesetEU_{long}$. Here, we use the short notation *RulesetUS* for the ruleset {*drive until driven*, *drive until traveled* } and *RulesetEU* for the ruleset {*drive until driven*, *work until traveled* }, as introduced in chapter 3. Accordingly, we are given four time limits:

- $limitD_{short}$, the maximum accumulated driving time since the end of the last short break
- $limitD_{long}$, the maximum accumulated driving time since the end of the last long break
- $limitT_{short}$, the maximum accumulated travel time since the end of the last short break
- $limitT_{long}$, the maximum accumulated travel time since the end of the last long break

It is again without loss of generality when we assume that $limitD_{short} \leq limitD_{long}$ and $limitT_{short} \leq limitT_{long}$ as well as $limitD_{short} \leq limitT_{short}$ and $limitD_{long} \leq limitT_{long}$ holds.

We only allow breaks, no matter if short or long, before service at customers and not en route between them. In this scenario, let a *truck driver schedule* be a sequence

$$\left((t_i^{arr}, t_i^{start}, t_i^{dep}) \right)_{i \leq n}$$

of n triples of points in time, one triple for each customer. The three values denote the *arrival time*, the *start time* of service, and the *departure time* from customer i, respectively. A truck driver schedule is feasible only if the following basic conditions hold:

$$\alpha(\mathcal{H}) \leq t_1^{arr} \tag{5.1}$$

$$t_i^{arr} \leq t_i^{start} \qquad \text{for all } i \leq n \tag{5.2}$$

$$t_i^{start} \in \mathcal{W}_i \qquad \text{for all } i \leq n \tag{5.3}$$

$$t_i^{dep} - t_i^{start} \geq service_i \qquad \text{for all } i \leq n \tag{5.4}$$

$$drive_i \leq t_{i+1}^{arr} - t_i^{dep} \leq limitD_{short} \qquad \text{for all } i \leq n-1 \tag{5.5}$$

$$t_n^{dep} \leq \omega(\mathcal{H}) \tag{5.6}$$

Conditions 5.2 and 5.4 and 5.5 ensure that the sequence of points in time is monotonously increasing. Conditions 5.1 and 5.6 make sure that all points in time are within the planning horizon. Condition 5.3 demands that each service must begin within one of the respective customer's time windows. Conditions 5.4 and 5.5 ensure that enough time is scheduled for service and driving, respectively.

Apart from condition 5.5, there is no constraint in respect of the two rulesets. In addition to the basic constraints above and in order to consider the *drive until driven*

rules, we also demand that whenever the sum of consecutive driving times exceeds a limit, the respective break is scheduled at one of the customers in between:

$$\sum_{k=j}^{\ell} t_{k+1}^{arr} - t_k^{dep} > limitD_{type} \Rightarrow \quad \exists k : j < k \leq \ell \wedge t_k^{start} - t_k^{arr} \geq break_{type} \qquad j < \ell, \ell < n, type \in Types \qquad (5.7)$$

where $Types := \{short, long\}$.

In the first problem variant, i.e., in respect of the two *drive until traveled* rules, we have to make sure that the driver does not drive (be aware that a driving time of zero is allowed in the input) when the travel time limit is exceeded unless the respective break is scheduled at one of the customers in between:

$$t_i^{dep} < t_{i+1}^{arr} \Rightarrow t_{i+1}^{arr} - t_i^{start} \leq limitT_{short} \qquad i < n \qquad (5.8)$$

$$t_\ell^{dep} < t_{\ell+1}^{arr} \wedge t_{\ell+1}^{arr} - t_j^{start} > limitT_{type} \Rightarrow \quad \exists k : j < k \leq \ell \wedge t_k^{start} - t_k^{arr} \geq break_{type} \qquad j < \ell, \ell < n, type \in Types \qquad (5.9)$$

This means that even if a travel time limit is exceeded, the driver is still permitted to drive for zero time units.

In the second problem variant, i.e., in respect of the two *work until traveled* rules, we have to make sure that the driver does neither drive nor perform service when the travel time limit is exceeded unless the respective break is scheduled at one of the customers in between:

$$t_{i+1}^{arr} - t_i^{start} \leq limitT_{short} \qquad i < n \qquad (5.10)$$

$$t_n^{dep} - t_n^{start} \leq limitT_{short} \qquad (5.11)$$

$$t_\ell^{start} < t_{\ell+1}^{arr} \wedge t_{\ell+1}^{arr} - t_j^{start} > limitT_{type} \Rightarrow \quad \exists k : j < k \leq \ell \wedge t_k^{start} - t_k^{arr} \geq break_{type} \qquad j < \ell, \ell < n, type \in Types \qquad (5.12)$$

$$t_n^{dep} - t_j^{start} > limitT_{type} \Rightarrow \quad \exists k : j < k \leq n \wedge t_k^{start} - t_k^{arr} \geq break_{type} \qquad j < n, type \in Types \qquad (5.13)$$

This concludes our constraint based problem formulation (for a MIP formulation of a related problem, see for instance Goel (2012c)). In this chapter, the objective is to find a feasible schedule with earliest finish time, i.e., with t_n^{dep} being minimum (EF-TDSP-2B).

Now, which practical problems can we solve with this? We give two examples.

- For a planning horizon of several days in the United States, we need the subsets $\{drive\ until\ traveled\}_{short}$ and $\{drive\ until\ driven, drive\ until\ traveled\}_{long}$ of the rulesets of the first problem variant. We set the break rule parameters according to the first row of Table 5.1. The parameter of the rule we do not need (*drive until driven* with respect to a short break) is set to a non-restrictive value.
- For a planning horizon of several days in the European Union, we need the subsets $\{drive\ until\ driven\}_{short}$ and $\{drive\ until\ driven, work\ until\ traveled\}_{long}$ of the rulesets of the second problem variant if we want to take account of Regulation (EC) No. 561/2006 (European Parliament and Council of the European Union, 2006). The break rule parameters can be set as in the second row of Table 5.1.

TABLE 5.1: Example parameter sets. Values in hours.

Example	$break_{short}$	$limitD_{short}$	$limitT_{short}$	$break_{long}$	$limitD_{long}$	$limitT_{long}$
US	0.5	8	8	10	11	14
EU	0.75	4.5	13	11	9	13

5.3 Problem Characteristics

With two types of breaks, a *driver state* can the thought of as a collection of values of these attributes:

- the progress along the route, i.e., how much of the work (driving and service) has been accomplished by the driver,
- the current point in time,
- the driving time and travel time accumulated since the end of the last *short* break,
- the driving time and travel time accumulated since the end of the last *long* break.

For one and the same progress, how many driver states do we need to distinguish per point in time? Figure 5.1 gives a hint to an answer. In this example, let $break_{short} = 1$ and $break_{long} = 3$, and let all limits be non-restrictive. It shows four different schedules. The wavy line shall imply that the exact course of the schedules before time 1 is irrelevant. At time 1, a break starts, so at time 4, the driver is completely rested with respect to the long and thus also the short break. From this point on, the schedules begin to differ. In the left-most schedule, the driver leaves the first (top-most) customer at time 4 and takes a (prolonged but still) short break at the fourth customer. In contrast, the long break at the first customer is further extended in the other three schedules. In the right-most schedule, the driver departs from the first customer only at time 6 and does not take a break at any subsequent customer.

We notice that the four driver states at time 9 that correspond to these four schedules do not dominate each other. That is, no matter which two schedules we compare, either the accumulated travel time since last *short* break is lower, or this holds for the accumulated travel time since last *long* break. And so we need to store four driver states for time 9. Precisely, the four pairs of accumulated travel time since the end of the last (short, long) break that we need to store for time 9 are $(0, 5)$, $(1, 4.5)$, $(2, 4)$, and $(3, 3)$.

In fact, it is sufficient to store at most as many driver states per point in time as there are customers. For a justification of this claim, suppose there are two schedules with the same finish time t and with the last short break at the same customer. This is illustrated in Figure 5.2. Instead of wavy lines, here, thin dashdotted lines imply that the exact course does not matter. The two schedules do not dominate each other because the end of the last long break is later in case of the red schedule, and the end of the last short break is later in case of the blue schedule (solid lines refer to the breaks). However, another schedule exists that dominates both. At first, it follows the blue schedule until the end of the last long break, but then this break is further prolonged (dotted) until it hits on the red schedule. The middle part until the end of the last short break is adopted from the red schedule, and the final part is again taken from the blue schedule. This new blue-red-blue schedule is feasible if the original

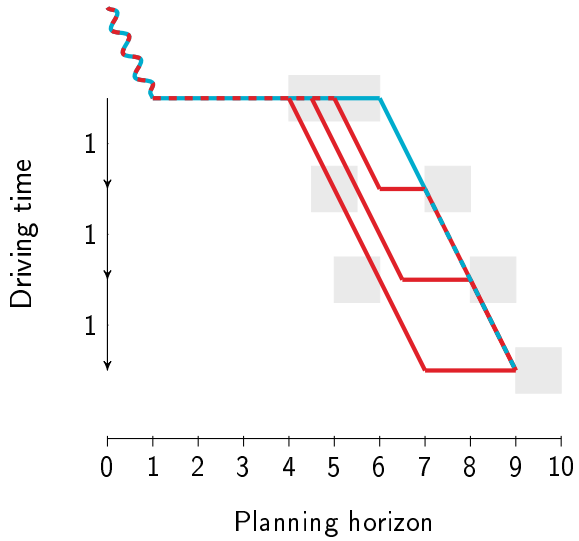


FIGURE 5.1: Four different schedules, not dominating each other.

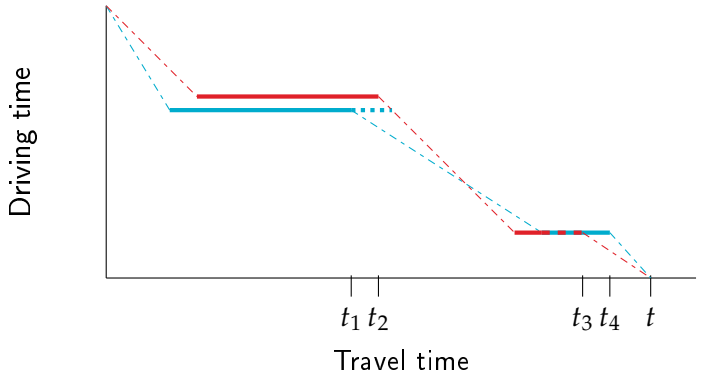


FIGURE 5.2: Two schedules, each with the last short break at the same customer.

schedules are because the accumulated times can never be worse – neither when we switch from the blue to the red schedule nor when we switch back.

Generally speaking, whenever there are two different schedules that end at the same point in time and where the last short break is taken at the same customer, then either one schedule dominates the other or a third schedule must exist that dominates both. In other words, there is only one non-dominated driver state per point in time and per customer already visited.

There is another important observation that we are going to exploit. Suppose the driver will begin another short break at time t. In this case, the accumulated times since last short break become irrelevant. If we are thus only interested in the accumulated times since last long break, then there is only one driver state that is relevant for us.

5.4 Solution Approach

We begin with the definition of the contents of a driver states label in section 5.4.1. In section 5.4.2, we outline the algorithm. We present it in more detail thereafter in

section 5.4.3. We show how to derive a schedule in section 5.4.4. Finally, in section 5.4.5, we analyze the complexity of the algorithm.

5.4.1 Driver States Label

As in previous chapters, our algorithm for the EF-TDSP-2B computes a label $\mathcal{L}_i^{step}$ of driver states in every iteration i and every step *step* within the current iteration, where *step* is from the set $STEP := \{setup, waited, served, driven\}$. Based on the latter observation (section 5.3), we conclude that it suffices to store the information about minimum accumulated times for every point in time and for every customer already visited. Accordingly, a driver states label $\mathcal{L}_i^{step}$ is a sequence of i tuples $(\mathcal{F}_{i,j}^{step})_{j \leq i}$. Here, the second index $j \leq i$ indicates at which customer the last – short or long – break was taken. Every such tuple $\mathcal{F}_{i,j}^{step}$ is a quadruple $(\dot{D}_{i,j}^{step}, \dot{T}_{i,j}^{step}, \overline{D}_{i,j}^{step}, \overline{T}_{i,j}^{step})$ of four time-dependent, piecewise linear functions that may be "undefined" – that is, defined to be $\perp$ – over some points in time.

$\dot{D}_{i,j}^{step}$ and $\dot{T}_{i,j}^{step}$ contain the minimum accumulated driving times and travel times since the end of the last *short* break, respectively. Analogously, $\overline{D}_{i,j}^{step}$ and $\overline{T}_{i,j}^{step}$ state the minimum accumulated driving times and travel times since the end of the last *long* break, respectively. Recall that every long break also counts as short break. The two functions with the minimum accumulated driving times must be monotonously decreasing because the driver is allowed to wait. The two functions with the minimum accumulated travel times are monotonously decreasing only if we subtract the identity function before. This is because the time that the driver waits counts towards the travel time. In other words, $T_{i,j}^{step}(t_1) - t_1 \geq T_{i,j}^{step}(t_2) - t_2$ must hold for any two points in time $t_1 < t_2$ over which the travel time function is defined.

5.4.2 Outline and Initialization of the Algorithm

An outline of the algorithm is given with Algorithm 3. It is a variant of Algorithm 1 introduced in chapter 3. The only difference is the sequence of the four steps.

Algorithm 3: Generic truck driver scheduling algorithm - variant

Input : $\mathcal{L}_1^{setup}$
Output: $\texttt{obj}(\mathcal{L}_n^{served})$
1 **forall** $i = 1 \dots n$ **do**
2 $\mathcal{L}_i^{waited} := \texttt{Wait}\ (\mathcal{L}_i^{setup})$;
3 $\mathcal{L}_i^{served} := \texttt{Serve}\ (\mathcal{L}_i^{waited})$;
4 **if** $i = n$ **then**
5 **return** $\texttt{obj}\ (\mathcal{L}_n^{served})$;
6 $\mathcal{L}_i^{driven} := \texttt{Drive}\ (\mathcal{L}_i^{served})$;
7 $\mathcal{L}_{i+1}^{setup} := \texttt{Setup}\ (\mathcal{L}_i^{driven})$;

Dependency Graph As already explained, a driver states label $\mathcal{L}_i^{step}$ is a sequence of i tuples $(\mathcal{F}_{i,j}^{step})_{j \leq i}$, which implies there is a tuple more with every iteration i. To illustrate the course of the algorithm in general and the dependencies of the tuples among each other in particular, a (rooted) directed acyclic graph is pictured in Figure 5.3, called *dependency graph* in the following. There is a pair (i, j) of two indices

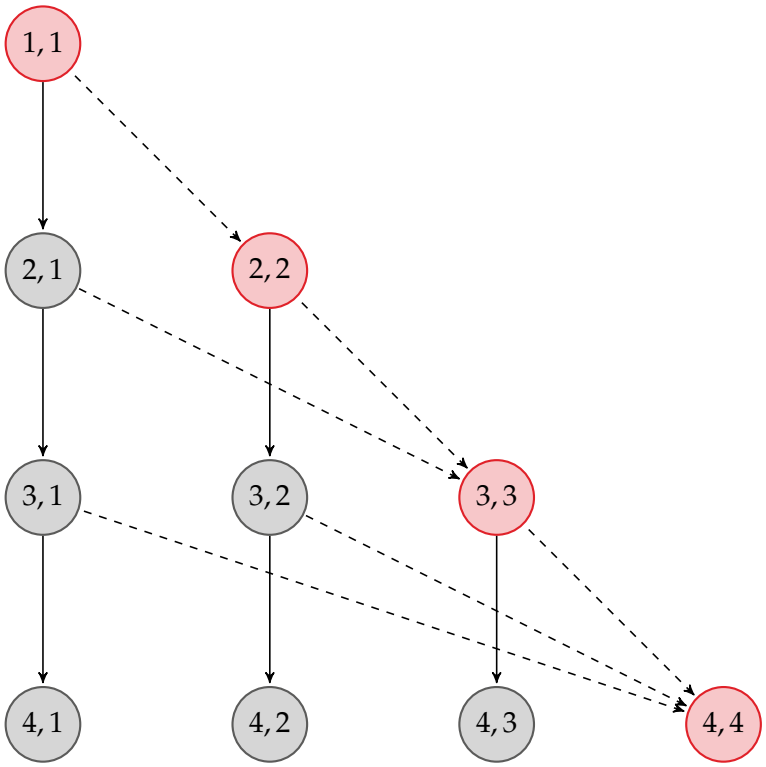


FIGURE 5.3: Dependency graph as introduced in section 5.4.2. The red highlighted nodes are the break nodes.

written inside every node, the first of which specifies the current customer (equals the current iteration). The second index $j \leq i$ indicates the customer at which the last break was taken. Every node (i, j) in the graph represents all tuples $\mathcal{F}^{step}_{i,j}$, that is, we subsume the tuples regarding the steps for the sake of simplicity. In total, the number of nodes is $\sum_{i=1}^{n} i \in O(n^2)$.

Every inner node (i, j) has exactly two outgoing edges. These correlate with the two options the driver has on arrival at a customer: to take a break before service (dashed edge to $(i+1, i+1)$) or *not* to take a break before service (solid edge to $(i+1, j)$). The nodes (i, j) with $j = i$ are special and therefore highlighted. These are the "break nodes". They have $i - 1$ (dashed) incoming edges, whereas all other nodes (with $j < i$) have only one (solid) incoming edge. The edges indicate dependencies. The tuples represented by node (i, j) with $j < i$ can be calculated from those represented by node $(i - 1, j)$ alone. However, the computation of the tuples represented by node (i, j) with $j = i$ needs the knowledge of the tuples represented by the nodes $(i - 1, k)$ for all k from 1 to $i - 1$.

What does this tell us? For instance, $\dot{D}^{driven}_{i,j}(t)$ must be exactly $\sum_{k=j}^{i} drive_k$ for every t for which it is defined because we know that the last short break is taken at customer j. And $\overline{D}^{driven}_{i,j}(t)$ must have at least that value because we know that there is no break after leaving customer j. However, the value may be higher if there is only a short break scheduled immediately before service at that customer.

Initialization We assume that the driver is completely rested with respect to the long break (and thus also the short break) when the planning horizon begins. The driver has thus not accumulated any driving or travel time since the end of the last

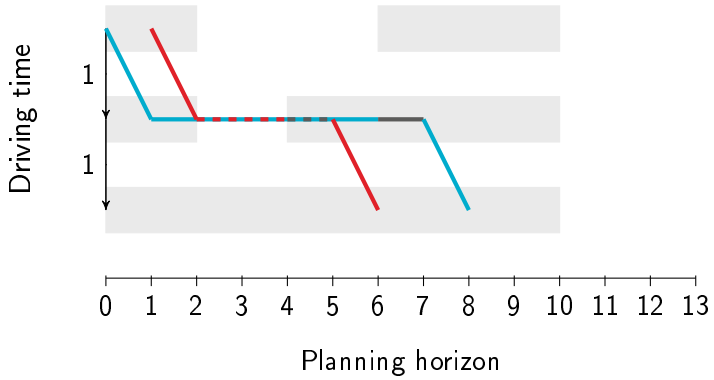


FIGURE 5.4: Schedule view of example instance.

long break. And so we initialize all four functions with zero over the whole planning horizon $\mathcal{H}$.

$$\mathcal{F}_{1,1}^{setup}(t) := \begin{cases} (0,0,0,0), & \text{for } t \in \mathcal{H} \\ (\perp,\perp,\perp,\perp), & \text{otherwise} \end{cases}$$

Finalization Finally, after running the algorithm, we need to compute $\texttt{obj}(\mathcal{L}_n^{served})$. To this end, we check whether at least one of the tuples of label $\mathcal{L}_n^{served}$ has a defined point, i.e., whether a feasible schedule exists. If it does, we let the function $\texttt{obj}(\mathcal{L}_n^{served})$ of Algorithm 3 return a pair of the first defined point over all tuples on the one hand, and the set of tuple indices to achieve this earliest finish time on the other:

$$\texttt{obj}(\mathcal{L}_n^{served}) := \left(\min_{j \le n} \alpha(\mathcal{F}_{n,j}^{served}), \arg\min_{j \le n} \alpha(\mathcal{F}_{n,j}^{served}) \right)$$

Should there be no feasible schedule, then $\texttt{obj}(\mathcal{L}_n^{served})$ is set to $(\perp, \emptyset)$.

5.4.3 Steps of Algorithm in Detail

When we explain the four steps in detail, we will always refer to the same example. It is depicted in Figure 5.4. We see the first three customers of a possibly longer route. The driving time between consecutive customers is 1, as well as the service time at the second customer. All other service times are 0. Let the break parameters be $limitD_{short} = 1.5$, $break_{short} = 2$, $limitD_{long} = 3$, $limitT_{long} = 5.5$, and $break_{long} = 5$. So by construction, the third customer cannot be reached without at least a short break. In our example, let $limitT_{short}$ be non-restrictive. Besides the customers' time windows, Figure 5.4 also shows two feasible schedules. In one case, the driver takes a long break (blue), in the other, the driver takes a short break (red) before the service at the second customer (service is highlighted in gray).

We do not describe the steps in the same order as in Algorithm 3. Instead, we start with step *Setup* in section 5.4.3.1. We do so because of our example instance. Here, not much of interest is happening to the functions of the label in the very first steps of the algorithm. So let us jump right to the end of the step *Drive* of the first iteration. In Figure 5.5, we present the functions of $\mathcal{F}_{1,1}^{driven}$. We see the time windows of the first customer in the background and two horizontal lines representing the pieces of $\dot{D}_{1,1}^{driven}$, $\dot{T}_{1,1}^{driven}$, $\overline{D}_{1,1}^{driven}$, and $\overline{T}_{1,1}^{driven}$ in the foreground. In fact, all four functions are the same. The minimum accumulated driving time as well as the accumulated

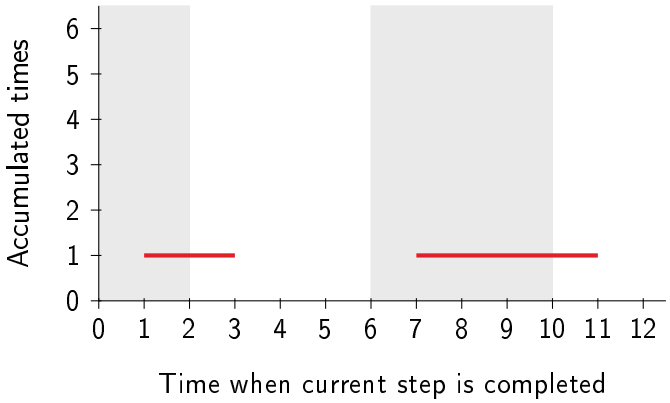


FIGURE 5.5: Function view: After step *Drive* in iteration 1.

travel time since the end of both the short and the long break is 1. And the functions are only defined over those intervals that correspond to the time windows when shifted to the right by 1.

5.4.3.1 Step *Setup*

When the driver arrives at a customer, he may take a break. In step *Setup*, we take account of this. Taking a break may be beneficial because otherwise one of the next customers may not be reachable without exceeding one of the time limits. But it also may be disadvantageous, either because a favorable time window is missed this way or because it leads to a later finish time of the route. So we keep the two options separate. In our example, we create two function quadruples $\mathcal{F}^{setup}_{2,1}$ and $\mathcal{F}^{setup}_{2,2}$ from $\mathcal{F}^{driven}_{1,1}$. In general, we create a $\mathcal{F}^{setup}_{i+1,j}$ for every $j \leq i$, and another tuple $\mathcal{F}^{setup}_{i+1,i+1}$ in addition, where i is the current iteration. The latter function quadruple stands for the case that a break is taken before the service at customer $i+1$.

Let us begin with the other case, that is, we create a $\mathcal{F}^{setup}_{i+1,j}$ for every $j \leq i$. For this case, not much has to be done. We introduce three significant points in time. Let $t_0 := \min_{j \leq i} \alpha(\mathcal{F}^{driven}_{i,j})$ be the first defined point over all function quadruples $\mathcal{F}^{driven}_{i,j}$. With this, let $t_1 := t_0 + break_{short}$ be a short break later, and let $t_2 := t_0 + break_{long}$ be a long break later. Since taking a break is disregarded when $j \leq i$, we prune the functions and cut off everything later than t_2 because at that time, the driver could be completely rested with respect to the long break. And so these driver states would be dominated by those in $\mathcal{F}^{setup}_{i+1,i+1}$. Specifically, we set:

$$\dot{D}^{setup}_{i+1,j}(t) := \dot{D}^{driven}_{i,j}(t)$$
$$\dot{T}^{setup}_{i+1,j}(t) := \dot{T}^{driven}_{i,j}(t)$$
$$\overline{D}^{setup}_{i+1,j}(t) := \overline{D}^{driven}_{i,j}(t)$$
$$\overline{T}^{setup}_{i+1,j}(t) := \overline{T}^{driven}_{i,j}(t)$$

for all $t < t_2$, and $\mathcal{F}^{setup}_{i+1,j}(t) := (\bot, \bot, \bot, \bot)$ for all other t.

Now let us turn to how $\mathcal{F}^{setup}_{i+1,i+1}$ is created. Taking a long break into account is rather simple. By definition of t_2, the driver may be completely rested at that time. And he is also rested at any time later because the long break is further prolonged

accordingly. So we set all four functions to 0 from time t_2 onwards until the planning horizon ends.

But how do we take account of a short break, that is, how do we set the functions between t_1 and t_2? In regard of the minimum accumulated times since the end of the last short break, this is also not too difficult. The functions $\dot{D}^{setup}_{i+1,i+1}(t)$ and $\dot{T}^{setup}_{i+1,i+1}(t)$ can be set to 0 between t_1 and t_2, too. By doing so, we again exploit that a short break may be prolonged. However, the case of the minimum accumulated times since the end of the last long break, that is, how to set $\overline{D}^{setup}_{i+1,i+1}(t)$ and $\overline{T}^{setup}_{i+1,i+1}(t)$ for $t_1 \leq t < t_2$, needs a little more thinking because these functions are not reset to 0 when only a short break is taken.

What if we just set $\overline{D}^{setup}_{i+1,i+1}(t) := \min_{j \leq i}\{\overline{D}^{driven}_{i,j}(t - break_{short})\}$ and $\overline{T}^{setup}_{i+1,i+1}(t) := \min_{j \leq i}\{\overline{T}^{driven}_{i,j}(t - break_{short})\} + break_{short}$? In general, the four functions of each quadruple may contain gaps, that is, intervals over which the functions are not defined (as in Figure 5.5). Hence, for some $j \leq i$ and a time $t_1 \leq t < t_2$, the four functions of $\mathcal{F}^{driven}_{i,j}$ may be undefined at time $t - break_{short}$. However, since a break may be prolonged, we may be able to close these gaps. To this end, we define an auxiliary function $H^{sp}_{i,j}$ for every $j \leq i$ that maps a time t to the most recent point in time t' that is at least $break_{short}$ earlier than t and at which $\mathcal{F}^{setup}_{i,j}(t')$ is defined:

$$H^{sp}_{i,j}(t) := \max\{t' \mid t' \leq t - break_{short} \wedge \mathcal{F}^{setup}_{i,j}(t') \neq (\perp, \perp, \perp, \perp)\}$$

Here, let the maximum over the empty set be $\perp$.

Then, $\overline{D}^{setup}_{i+1,i+1}(t)$ is the minimum of $\overline{D}^{driven}_{i,j}$ at time $H^{sp}_{i,j}(t)$ over all $j \leq i$ for which $H^{sp}_{i,j}(t)$ is defined. For a time $t \geq t_1$, such a j must exist (by the definition of t_1). In case of the minimum accumulated travel times $\overline{T}^{setup}_{i+1,i+1}(t)$, we proceed analogously, though we need to add the time for the (possibly prolonged) break to the travel time. Putting all this together, we set the functions as follows:

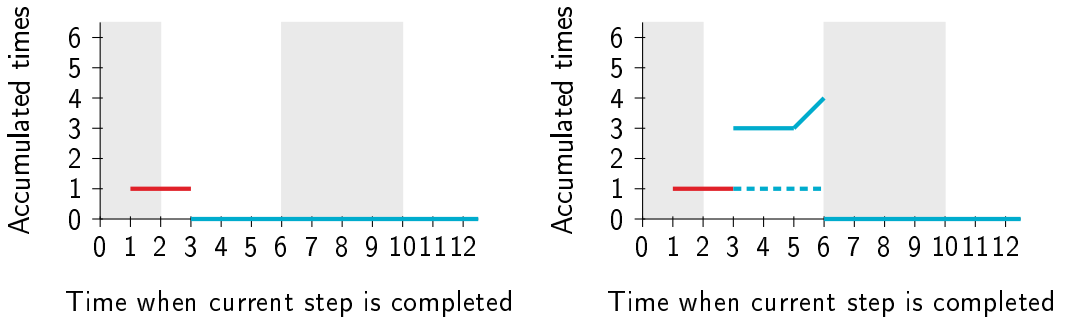
$$\dot{D}^{setup}_{i+1,i+1}(t) := \begin{cases} 0, & \text{for } t_1 \leq t < t_2 \\ 0, & \text{for } t_2 \leq t \leq \omega(\mathcal{H}) \\ \perp, & \text{otherwise} \end{cases}$$

$$\dot{T}^{setup}_{i+1,i+1}(t) := \begin{cases} 0, & \text{for } t_1 \leq t < t_2 \\ 0, & \text{for } t_2 \leq t \leq \omega(\mathcal{H}) \\ \perp, & \text{otherwise} \end{cases}$$

$$\overline{D}^{setup}_{i+1,i+1}(t) := \begin{cases} \min\limits_{j:j \leq i, H^{sp}_{i,j}(t) \neq \perp} \{\overline{D}^{driven}_{i,j}\left(H^{sp}_{i,j}(t)\right)\}, & \text{for } t_1 \leq t < t_2 \\ 0, & \text{for } t_2 \leq t \leq \omega(\mathcal{H}) \\ \perp, & \text{otherwise} \end{cases}$$

$$\overline{T}^{setup}_{i+1,i+1}(t) := \begin{cases} \min\limits_{j:j \leq i, H^{sp}_{i,j}(t) \neq \perp} \{\overline{T}^{driven}_{i,j}\left(H^{sp}_{i,j}(t)\right) + \left(t - H^{sp}_{i,j}(t)\right)\}, & \text{for } t_1 \leq t < t_2 \\ 0, & \text{for } t_2 \leq t \leq \omega(\mathcal{H}) \\ \perp, & \text{otherwise} \end{cases}$$

By construction, all four functions of $\mathcal{F}^{setup}_{i+1,i+1}$ never contain any gaps between t_1 and $\omega(\mathcal{H})$, i.e., $\mathcal{F}^{setup}_{i+1,i+1}(t) \neq (\perp, \perp, \perp, \perp)$ for all $t_1 \leq t \leq \omega(\mathcal{H})$. Over all steps of an



(A) $\dot{D}^{setup}_{2,1}$ and $\dot{T}^{setup}_{2,1}$ red, $\dot{D}^{setup}_{2,2}$ and $\dot{T}^{setup}_{2,2}$ blue

(B) $\overline{D}^{setup}_{2,1}$ and $\overline{T}^{setup}_{2,1}$ red, $\overline{D}^{setup}_{2,2}$ blue dashed, $\overline{T}^{setup}_{2,2}$ blue solid

FIGURE 5.6: Function view: After step *Setup* in iteration 1.

iteration, it is an invariant that, for one and the same point in time, all four functions are either defined or undefined. (For correctness, please recall the observation mentioned at the very end of section 5.3, referring to Figure 5.2.)

Figure 5.6 shows the functions of both $\mathcal{F}^{setup}_{2,1}$ (red) and $\mathcal{F}^{setup}_{2,2}$ (blue). In our example, $t_0 = 1$, $t_1 = 3$, and $t_2 = 6$. When we compare this figure with Figure 5.5, we notice that the second (red) piece is pruned. So the functions $\dot{D}^{setup}_{2,1}$, $\overline{D}^{setup}_{2,1}$, $\dot{T}^{setup}_{2,1}$, and $\overline{T}^{setup}_{2,1}$ contain exactly one (red) piece each. They are still all the same and constantly 1 where they are defined.

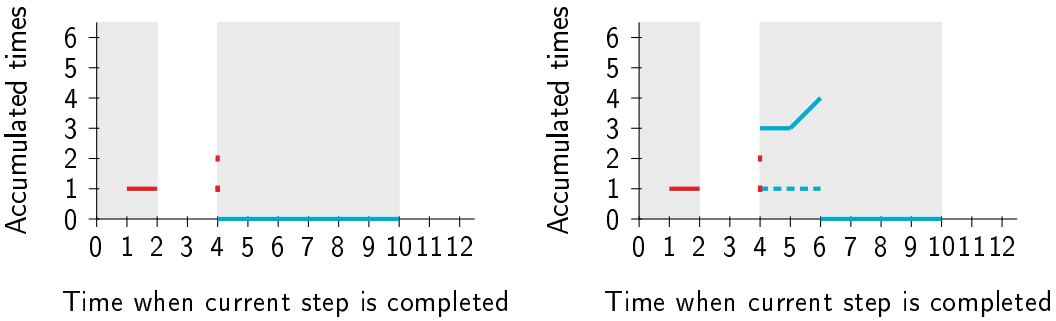
For $\mathcal{F}^{setup}_{2,2}$, the minimum accumulated driving and travel times since the end of the last short break $\dot{D}^{setup}_{2,2}$ and $\dot{T}^{setup}_{2,2}$ are 0 from $t_1 = 3$ until the end of $\mathcal{H}$ (blue, left figure). In the right figure, we see that $\overline{D}^{setup}_{2,2}$ (dashed, blue) is 1 not only from 3 to 5 but also from 5 to $6 (= t_2)$ because the gap is closed. And $\overline{T}^{setup}_{2,2}$ (solid, blue) is 3 (driving time plus break time) until time 5. When closing the gap between 5 and 6, inevitable waiting time must be added to the travel time, thus a diagonal piece with gradient 1 is introduced. From 6 until the planning horizon ends, all functions of $\mathcal{F}^{setup}_{2,2}$ are zero, where the dashed blue line is hidden by the solid blue line.

5.4.3.2 Step *Wait*

In the following, let i be the current iteration. The first step of each iteration is *Wait*. In this step, the time windows of customer i are taken into account. As in earlier chapters, this means three things:

- We ensure that every function stored in label $\mathcal{L}^{waited}_i$ is undefined outside of the time windows of the current customer.
- We shift the pieces of the functions by the waiting time whenever necessary.
- We add waiting time to the accumulated travel times whenever necessary.

But when exactly is waiting necessary? It is necessary if the driver is not able to arrive right at the beginning of a time window but is at least able to arrive within the waiting interval before that time window.



(A) $\dot{D}^{waited}_{2,1}$ and $\dot{T}^{waited}_{2,1}$ red, $\dot{D}^{waited}_{2,2}$ and $\dot{T}^{waited}_{2,2}$ blue

(B) $\overline{D}^{waited}_{2,1}$ and $\overline{T}^{waited}_{2,1}$ red, $\overline{D}^{waited}_{2,2}$ blue dashed, $\overline{T}^{waited}_{2,2}$ blue solid

FIGURE 5.7: Function view: After step *Wait* in iteration 2.

Accordingly, we define an auxiliary function $H^{shift}_{i,j}$ for every $j \leq i$ as follows (cp. section 3.3.3.2):

$$H^{shift}_{i,j}(t) := \max\{t' \mid t = t' + W_i(t') \wedge \mathcal{F}^{setup}_{i,j}(t') \neq (\bot, \bot, \bot, \bot)\}$$

Here, let the maximum over the empty set be $\bot$. (The waiting time function $W_i(t)$ is introduced in section 2.4.3.)

With this auxiliary function, we set

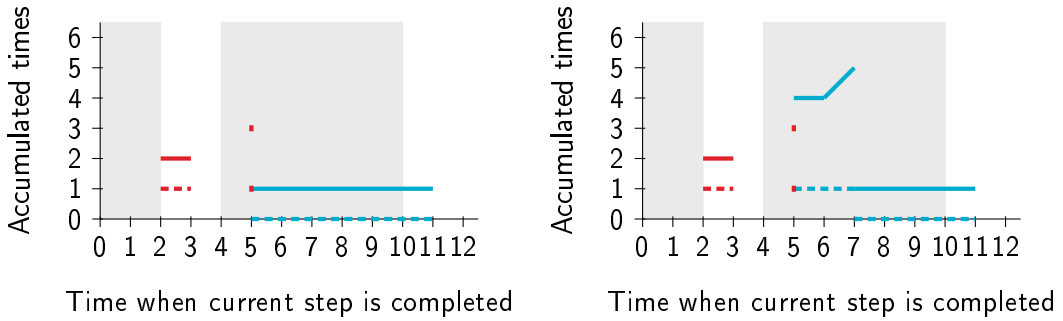
$$\begin{aligned}
\dot{D}^{waited}_{i,j}(t) &:= \dot{D}^{setup}_{i,j}\left(H^{shift}_{i,j}(t)\right) \\
\dot{T}^{waited}_{i,j}(t) &:= \dot{T}^{setup}_{i,j}\left(H^{shift}_{i,j}(t)\right) + (t - H^{shift}_{i,j}(t)) \\
\overline{D}^{waited}_{i,j}(t) &:= \overline{D}^{setup}_{i,j}\left(H^{shift}_{i,j}(t)\right) \\
\overline{T}^{waited}_{i,j}(t) &:= \overline{T}^{setup}_{i,j}\left(H^{shift}_{i,j}(t)\right) + (t - H^{shift}_{i,j}(t))
\end{aligned}$$

for all t with $H^{shift}_{i,j}(t) \neq \bot$, and $\mathcal{F}^{waited}_{i,j}(t) := (\bot, \bot, \bot, \bot)$ for all other t.

Now let us turn towards the example. In the background of Figure 5.7, we see the time windows of the second customer. First of all, we find that both tuples $\mathcal{F}^{waited}_{2,1}$ (red) and $\mathcal{F}^{waited}_{2,2}$ (blue) are indeed undefined outside of the time windows. When comparing this figure with Figure 5.6, we notice that $\dot{D}^{waited}_{2,1}(t) = \overline{D}^{waited}_{2,1}(t)$ and $\dot{T}^{waited}_{2,1}(t) = \overline{T}^{waited}_{2,1}(t)$ are defined for $t = 4$. Here, waiting from 3 to 4 is considered, and also one unit of waiting time is added to $\dot{T}^{waited}_{2,1}(4) = \overline{T}^{waited}_{2,1}(4) = 2$. This illustrates that in step *Wait*, gaps between time windows may induce gaps in functions.

5.4.3.3 Step *Serve*

This step is the only one in which we have to distinguish the two problem variants that we study in this chapter. In case both types of breaks are subject to the rule *drive*



(A) $\dot{D}^{served}_{2,1}$ red dashed, $\dot{T}^{served}_{2,1}$ red solid, $\dot{D}^{served}_{2,2}$ blue dashed, $\dot{T}^{served}_{2,2}$ blue solid

(B) $\overline{D}^{served}_{2,1}$ red dashed, $\overline{T}^{served}_{2,1}$ red solid, $\overline{D}^{served}_{2,2}$ blue dashed, $\overline{T}^{served}_{2,2}$ blue solid

FIGURE 5.8: Function view: After step *Serve* in iteration 2.

until traveled, we cap the minimum accumulated travel times. We set

$$\begin{aligned}
\dot{D}^{served}_{i,j}(t) &:= \dot{D}^{waited}_{i,j}(t - service_i) \\
\dot{T}^{served}_{i,j}(t) &:= \min\{\dot{T}^{waited}_{i,j}(t - service_i) + service_i, limitT_{short}\} \\
\overline{D}^{served}_{i,j}(t) &:= \overline{D}^{waited}_{i,j}(t - service_i) \\
\overline{T}^{served}_{i,j}(t) &:= \min\{\overline{T}^{waited}_{i,j}(t - service_i) + service_i, limitT_{long}\}
\end{aligned}$$

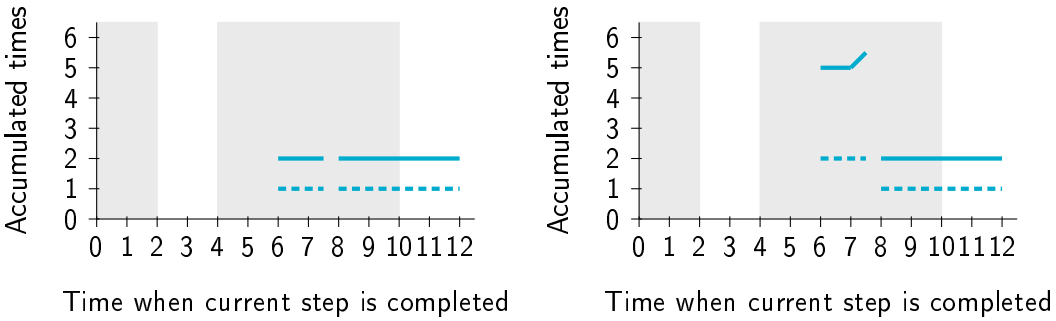
for all $t \in \mathcal{H}$ with $\mathcal{F}^{waited}_{i,j}(t - service_i) \neq (\bot, \bot, \bot, \bot)$, and $\mathcal{F}^{served}_{i,j}(t) := (\bot, \bot, \bot, \bot)$ for all other t.

In case of the rule *work until traveled*, we make sure that service is only performed when allowed, and set

$$\begin{aligned}
\dot{D}^{served}_{i,j}(t) &:= \dot{D}^{waited}_{i,j}(t - service_i) \\
\dot{T}^{served}_{i,j}(t) &:= \dot{T}^{waited}_{i,j}(t - service_i) + service_i \\
\overline{D}^{served}_{i,j}(t) &:= \overline{D}^{waited}_{i,j}(t - service_i) \\
\overline{T}^{served}_{i,j}(t) &:= \overline{T}^{waited}_{i,j}(t - service_i) + service_i
\end{aligned}$$

for all $t \in \mathcal{H}$ with $\mathcal{F}^{waited}_{i,j}(t - service_i) \neq (\bot, \bot, \bot, \bot)$ as well as both $\dot{T}^{waited}_{i,j}(t - service_i) + service_i \leq limitT_{short}$ and $\overline{T}^{waited}_{i,j}(t - service_i) + service_i \leq limitT_{long}$; for all other t, we set $\mathcal{F}^{served}_{i,j}(t) := (\bot, \bot, \bot, \bot)$.

Figure 5.8 helps comprehend how the functions in the example change in this step. Every function piece is shifted to the right by the service time, which is 1 in the example. The pieces of the travel time functions are also raised by this service time. In the figure, the function pieces of $\mathcal{F}^{served}_{2,1}$ are red, and those of $\mathcal{F}^{served}_{2,2}$ are blue. The pieces of the driving time functions like $\dot{D}^{served}_{2,j}$ and $\overline{D}^{served}_{2,j}$ are displayed as dashed lines, whereas those of the travel time functions $\dot{T}^{served}_{2,j}$ and $\overline{T}^{served}_{2,j}$ are displayed as solid lines ($1 \leq j \leq 2$).



(A) $\dot{D}^{driven}_{2,2}$ dashed, $\dot{T}^{driven}_{2,2}$ solid

(B) $\overline{D}^{driven}_{2,2}$ dashed, $\overline{T}^{driven}_{2,2}$ solid

FIGURE 5.9: Function view: After step *Drive* in iteration 2.

5.4.3.4 Step *Drive*

Either the next customer $i+1$ can be reached without taking a break - or not. To tell whether the former or the latter is the case, we introduce an auxiliary function $H^{exc}_{i,j}(t)$, mapping a time t to the maximum allowed driving time when departing from customer i at time t:

$$H^{exc}_{i,j}(t) := \min\{ \begin{array}{l} limitD_{short} - \dot{D}^{served}_{i,j}(t), limitT_{short} - \dot{T}^{served}_{i,j}(t), \\ limitD_{long} - \overline{D}^{served}_{i,j}(t), limitT_{long} - \overline{T}^{served}_{i,j}(t) \end{array} \}$$

Here, we again treat $\perp$ like ∞, i.e., we define $H^{exc}_{i,j}(t)$ to be $-\infty$ whenever $\mathcal{F}^{served}_{i,j}(t)$ is undefined.

With this auxiliary function set up, it is easy to define the four functions of the tuple $\mathcal{F}^{driven}_{i,j}$:

$$\begin{aligned} \dot{D}^{driven}_{i,j}(t) &:= \dot{D}^{served}_{i,j}(t - drive_i) + drive_i \\ \dot{T}^{driven}_{i,j}(t) &:= \dot{T}^{served}_{i,j}(t - drive_i) + drive_i \\ \overline{D}^{driven}_{i,j}(t) &:= \overline{D}^{served}_{i,j}(t - drive_i) + drive_i \\ \overline{T}^{driven}_{i,j}(t) &:= \overline{T}^{served}_{i,j}(t - drive_i) + drive_i \end{aligned}$$

for all $t \in \mathcal{H}$ with $H^{exc}_{i,j}(t - drive_i) \geq drive_i$, and we set $\mathcal{F}^{driven}_{i,j}(t) := (\perp, \perp, \perp, \perp)$ for all other t.

Figure 5.9 shows the pieces of the functions in $\mathcal{F}^{driven}_{2,1}$ and $\mathcal{F}^{driven}_{2,2}$. By construction of the example, it is not possible to reach customer 3 without taking at least a short break at the second customer. Accordingly, the auxiliary function $H^{exc}_{2,1}(t)$ is less than $drive_i$ for all t, and hence $\mathcal{F}^{driven}_{2,1}(t)$ is undefined for all t.

The pieces of the functions of $\mathcal{F}^{driven}_{2,2}$ are shifted to the right and raised by the driving time $drive_2 = 1$. In addition, a gap between 7.5 and 8 is introduced in all four functions because it is not possible for the driver to arrive in that interval without violating the limit $limitT_{long} = 5.5$. The pieces of the driving time functions $\dot{D}^{driven}_{2,2}$ and $\overline{D}^{driven}_{2,2}$ are displayed as dashed lines, whereas those of the travel time functions $\dot{T}^{driven}_{2,2}$ and $\overline{T}^{driven}_{2,2}$ are displayed as solid lines.

5.4.4 Deriving a Schedule

To finally derive a schedule $\left((t_i^{arr}, t_i^{start}, t_i^{dep})\right)_{i \leq n}$ from the computed labels, we can use Algorithm 4. For this algorithm to work correctly, we expect the service times and the driving times to be positive values. Input to the algorithm is the earliest finish time t_n^{dep} and a node index j such that $\mathcal{F}_{n,j}^{served}(t_n^{dep})$ is defined at that time. Beginning at the bottom of the dependency graph with the finish time t_n^{dep}, we derive all the other points in time as we make our way upwards in the graph. Deriving t_i^{dep} from t_{i+1}^{arr} and t_i^{start} from t_i^{dep} is simple – we just have to subtract the driving time and the service time, respectively. Deriving t_{i+1}^{arr} from t_{i+1}^{start} is a little harder, so let us focus on this.

At the beginning of each iteration i of Algorithm 4, we are given a start time of service t_{i+1}^{start} and a node index j such that $\mathcal{F}_{i+1,j}^{waited}(t_{i+1}^{start})$ is defined. If $j < i+1$, then $\mathcal{F}_{i+1,j}^{setup}$ is not necessarily defined at time t_{i+1}^{start} because there may be waiting time before the service at customer $i+1$. So t_{i+1}^{arr} is set to the latest point in time that is not later than t_{i+1}^{start} such that $\mathcal{F}_{i+1,j}^{setup}(t_{i+1}^{arr})$ and thus $\mathcal{F}_{i,j}^{driven}(t_{i+1}^{arr})$ is defined.

However, if $j = i+1$, we have reached a break node. $\mathcal{F}_{i+1,j}^{setup}$ must be defined at time t_{i+1}^{start} because there are no gaps in $\mathcal{F}_{i+1,j}^{setup}$ in this case. We need to determine not only t_{i+1}^{arr} but also a new node index k such that $\mathcal{F}_{i,k}^{driven}(t_{i+1}^{arr})$ is defined and matches with $\mathcal{F}_{i+1,j}^{setup}(t_{i+1}^{start})$. We distinguish the short break case from the long break case. In the latter case, a long break is scheduled before the service at customer $i+1$. It suffices to look for a $t \leq t_{i+1}^{start} - break_{long}$ and a node index k such that $\mathcal{F}_{i,k}^{driven}(t)$ is defined. In the short break case, we look for a $t \leq t_{i+1}^{start} - break_{short}$ and a node index k such that $\mathcal{F}_{i,k}^{driven}(t)$ is defined and $\overline{T}_{i,k}^{driven}(t) + (t_{i+1}^{start} - t) = \overline{T}_{i+1,i+1}^{setup}(t_{i+1}^{start})$ holds.

In both cases, the general idea is to always choose the latest such time in order to eventually find a rather late start time t_1^{arr} of the schedule. But as with Algorithm 2, there is no guarantee that we find the latest possible start time for the given finish time t_n^{dep}.

5.4.5 Complexity Analysis

We claim that the presented algorithm runs in polynomial time. As in other chapters of this thesis, a major step towards a proof is counting the number of pieces of the piecewise linear functions in each label. At first, we concentrate on the function $\overline{T}_{i+1,i+1}^{setup}$ and the number of pieces it consists of. Counting its pieces is not straightforward as the dependency graph (Figure 5.3) indicates. Just like every node has two outgoing edges in this graph, a piece may induce another piece in each of the two successor nodes. In step *Setup* of iteration $i+1$, the function $\overline{T}_{i+1,i+1}^{setup}$ is calculated from i different functions $\overline{T}_{i,j}^{driven}$ ($j \leq i$). It is not obvious how many pieces are dominated when computing the minimum of these functions. Even if we proved that the minimum cannot have more pieces than the input functions together, then this would still not lead to a polynomial bound on the number of pieces. Hence, we need a different approach in order to show that the number of pieces of $\overline{T}_{i+1,i+1}^{setup}$ is indeed bounded by a polynomial.

In this thesis, we speak of piecewise linear functions even though we assume that time passes in discrete time steps. As mentioned in section 2.4.3, we expect all input values to be multiples of 0.1 time units. In this chapter, we assume that the time step

Algorithm 4: Schedule deduction (EF-TDSP-2B)

Input : t_n^{dep} and an index j for which $\mathcal{F}_{n,j}^{served}(t_n^{dep})$ is defined

Output: $\left((t_i^{arr}, t_i^{start}, t_i^{dep})\right)_{i \leq n}$

$t_n^{start} := t_n^{dep} - service_n$;
forall $i = n-1 \dots 1$ **do**
  if $j < i+1$ **then**
    $t_{i+1}^{arr} := \max\{t \mid t \leq t_{i+1}^{start} \wedge \mathcal{F}_{i+1,j}^{setup}(t) \neq (\bot, \bot, \bot, \bot)\}$;
  else if $\mathcal{F}_{i+1,j}^{setup}(t_{i+1}^{start}) = (0,0,0,0)$ **then**
    set t_{i+1}^{arr} to the maximum $t \leq t_{i+1}^{start} - break_{long}$ such that an index k exists with $\mathcal{F}_{i,k}^{driven}(t)$ is defined;
    set j to such a k;
  else
    set t_{i+1}^{arr} to the maximum $t \leq t_{i+1}^{start} - break_{short}$ such that an index k exists with $\overline{T}_{i,k}^{driven}(t) + (t_{i+1}^{start} - t) = \overline{T}_{i+1,i+1}^{setup}(t_{i+1}^{start})$;
    set j to such a k;
  $t_i^{dep} := t_{i+1}^{arr} - drive_i$;
  $t_i^{start} := t_i^{dep} - service_i$;
$t_1^{arr} := t_1^{start}$;

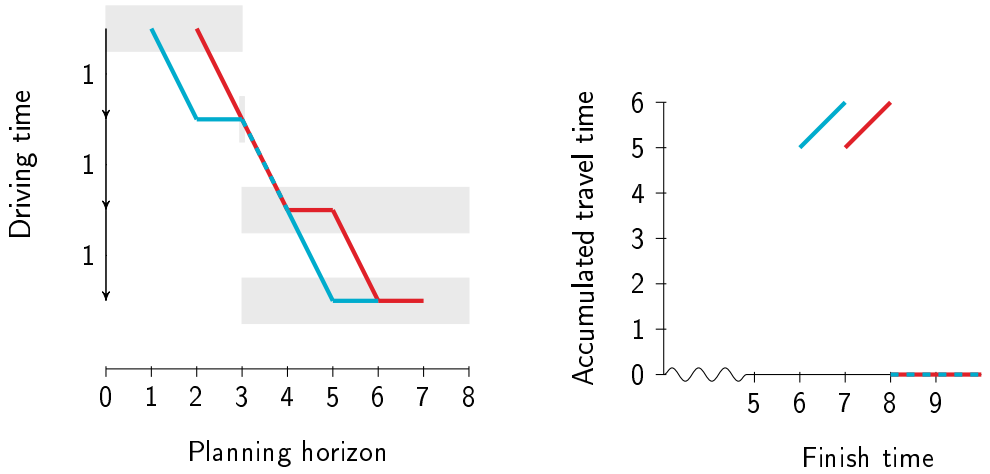
size is even shorter and that these 0.1 time units are in turn a multiple of the time step size. Let us denote this step size by ϵ in the following. With this assumption, degenerate pieces of length 0 are only introduced in step *Wait*.

A general observation is that the functions $\dot{D}_{i,j}^{step}$, $\dot{T}_{i,j}^{step}$, and $\overline{D}_{i,j}^{step}$ only consist of horizontal pieces (gradient 0). The function $\overline{T}_{i,j}^{step}$ is the only one of the four functions that may contain also diagonal pieces (gradient 1) besides horizontal ones. Another important observation is that all four functions of the same quadruple are either defined or undefined over one and the same point in time.

Before we go into the details of the proof in section 5.4.5.2, let us motivate the basic idea of the proof by some examples.

5.4.5.1 Motivational Examples

The function $\overline{T}_{i,j}^{driven}$ maps the arrival time at customer $i+1$ to the minimum accumulated travel time since the end of the last long break. It contains only horizontal (gradient 0) and diagonal (gradient 1) pieces. On a horizontal piece, the later the arrival, the later is the corresponding end of the last long break, otherwise the function value could not remain the same over this piece. On a diagonal piece however, the later the arrival, the higher is the function value, and so it is the corresponding end of the last long break that remains the same. Let us have another look at our example in Figure 5.4 (schedule view) and Figure 5.9 (function view). When completing step *Drive* of the second iteration at a time t between 6 and 7 (horizontal piece), the function value is 5, and the corresponding end of the last long break is $t-5$. However, when completing this step at a time t between 7 and 7.5 (diagonal piece), the function value is $t-2$, and the corresponding end of the last long break is 2. This is also the time when the first time window of the first customer closes.



(A) Schedule view. (B) Function view.

FIGURE 5.10: Two different diagonal pieces in $\overline{T}^{setup}_{4,4}$ (right) correspond to two different schedules (left). Both pieces are assigned to the same end of a time window at the second customer. Parameter setting is $limitD_{short} = 2$, $break_{short} = 1$, $break_{long} = 3$.

This is not a coincidence. In general, to every diagonal piece, there is a corresponding end of a time window. It is the time window that would be missed if the last long break was further prolonged. Or, in other words, it is the time window that prevents us from shifting the corresponding partial schedule since the end of the last long break to the right. And just like for every diagonal piece, there is also a corresponding end of a time window for every horizontal piece, defined analogously. In our example, the horizontal piece from 6 to 7 is associated with the end of the first time window of customer 1 just like the subsequent diagonal piece, whereas the horizontal piece from 8 to 12 is associated with the end of the second time window of customer 2. Like that, we assign (an end of) a time window to every piece. Should it happen that more than one time window would be missed at the same time, we assign the last of these time windows to the piece.

For the time being, let us focus on the diagonal pieces. It may happen that multiple diagonal pieces are associated with the same time window. In Figure 5.10a, two schedules are depicted. Both schedules have the same time window assigned to them, it is the one of the second customer. Figure 5.10b shows the corresponding two pieces of the travel time function $\overline{T}^{setup}_{4,4}$. How many diagonal pieces can be assigned to the same time window at most?

Suppose the assigned time window is of customer k. Then we claim that there cannot be more than k diagonal pieces assigned to this time window. Figure 5.11a shows four different schedules that all go through the very same end of a time window, namely the degenerate time window of the fourth customer at time 6. The argument why there cannot be more than k diagonal pieces assigned to the time window is basically the same as the one that we used to justify why it is sufficient to store at most as many driver states per point in time as there are customers. Look again at Figure 5.2. There can be at most k non-dominated driver states at the end of the time window. While there can be a diagonal piece for every such driver state, no two diagonal pieces can be assigned to the same non-dominated driver state at customer k. Since two different diagonal pieces refer to different ends of the last long break, they must also belong to either two different driver states or two different

time windows.

For horizontal pieces, an analogous argument applies. We now continue with the proof and introduce some formalism.

5.4.5.2 Proof, Continued

As with other proofs before, we start by having a look at the space complexity, that is, we count the number of pieces in the piecewise linear functions. For the most part of the proof, we will only have a look at the function $\overline{T}^{setup}_{x+1,x+1}$ that is created in iteration $x+1$. Every piece in $\overline{T}^{setup}_{x+1,x+1}$ is at least ϵ (time step size) long, unless $t_2 = \omega(\mathcal{H})$. (In the following, t_1 and t_2 always refer to the points in time as they are defined in section 5.4.3.1.) That is, after step *Setup*, there are no degenerate pieces of length 0 anymore which may have been introduced in step *Wait*, and so there is no need for special treatment of such pieces. Furthermore, every piece of $\overline{T}^{setup}_{x+1,x+1}$ is either horizontal (gradient 0) or diagonal (gradient 1), and it has a beginning. A point in time t is the beginning of a horizontal piece if and only if

$$\overline{T}^{setup}_{x+1,x+1}(t-\epsilon) \neq \overline{T}^{setup}_{x+1,x+1}(t) = \overline{T}^{setup}_{x+1,x+1}(t+\epsilon)$$

holds. Analogously, a point in time t is the beginning of a diagonal piece if and only if

$$\overline{T}^{setup}_{x+1,x+1}(t-\epsilon)+\epsilon \neq \overline{T}^{setup}_{x+1,x+1}(t) = \overline{T}^{setup}_{x+1,x+1}(t+\epsilon)-\epsilon$$

holds.

For the remainder of the proof, we need another definition: Let $\omega^*(T,t)$ be a mapping that assigns a point in time t to the earliest point in time $t' \geq t$ with

$$T(t) = T(t') \neq T(t'+\epsilon).$$

For instance, if t is on a horizontal piece of T, this function maps t to the end of that piece. A point in time on a diagonal piece is mapped to itself.

The proof consists of four parts. At first, we define how to assign every beginning of a piece to an index triple (section 5.4.5.3). Then, we show that every beginning of a diagonal piece is assigned to a different index triple (section 5.4.5.4). After that, we demonstrate that the number of horizontal pieces cannot be significantly larger that the number of diagonal pieces (section 5.4.5.5). Finally, we argue why these observations are sufficient to prove a polynomial-time bound of the algorithm (section 5.4.5.6).

5.4.5.3 How to Assign an Index Triple to the Beginning of a Piece

The first step of the proof is to assign an index triple (i,j,k) to every beginning of a piece in $\overline{T}^{setup}_{x+1,x+1}$ that is between t_1 and t_2. We call $i \leq x$ the customer index, $j \leq i$ the node index, and $k \leq w_i$ the time window index where the time window is one of customer i. The node index j denotes the customer at which the last short break is taken after the end of the last long break and before the end of the assigned time window. Note that there are no more than $O(nw)$ index triples. In the following, let z be the beginning of a piece, and let $t_1 \leq z < t_2$. How we assign an index triple (i,j,k) to z is shown in Algorithm 5 (here, with $z = t^{setup}_{x+1}$). Output of the algorithm is not only this triple but also a sequence of points in time that constitutes a partial schedule between customers i and x. In the main loop of the algorithm, the points in

time t_l^{driven}, t_l^{served}, t_l^{waited}, and t_l^{setup} are set for every $i \leq l \leq x$ and in this order since we go in reverse.

First, we consider step *Setup*. If the current node index j equals $i+1$, then we have to look for a suitable predecessor node. There must be a $j \leq i$ such that $H_{i,j}^{sp}(t_{i+1}^{setup})$ is defined and $\overline{T}_{i+1,i+1}^{setup}(t_{i+1}^{setup}) = \overline{T}_{i,j}^{driven}(H_{i,j}^{sp}(t_{i+1}^{setup})) + (t_{i+1}^{setup} - H_{i,j}^{sp}(t))$ holds, provided that t_{i+1}^{setup} does not lie on the constant-zero piece of $\overline{T}_{i+1,i+1}^{setup}$. Such a node index may not be unique. We choose to always set j to the highest such index. In addition, we set $t_i^{driven} := H_{i,j}^{sp}(t_{i+1}^{setup})$. However, in case $j \leq i$, we set $t_i^{driven} := t_{i+1}^{setup}$. In this case, node index j remains unchanged. In both cases, it is assured that $\overline{T}_{i,j}^{driven}(t_i^{driven})$ is defined.

To take account of steps *Drive* and *Serve*, we subtract both the driving and the service time from t_i^{driven}. Again, it is assured that $\overline{T}_{i,j}^{waited}(t_i^{waited})$ is defined. This also means that t_i^{waited} lies within a time window of customer i. Regarding step *Wait*, we set $t_i^{setup} := H_{i,j}^{shift}(t_i^{waited})$. If then $t_i^{setup} = t_i^{waited}$ holds, this means it is not necessary to wait for a time window at customer i on arrival at time t_i^{setup}. If in addition $\omega^*(\overline{T}_{i,j}^{waited}, t_i^{waited})$ equals the end of a time window of customer i, then the algorithm terminates and returns (i, j, k), where k is the index of that time window.

An important observation is that t_i^{setup} never lies on the constant-zero piece that is introduced in step *Setup*. This is because the condition in line 11 would have been fulfilled in the iteration before. Here, we use the prerequisite that every time window lies within the planning horizon. By the same argument, we can conclude that the algorithm returns an index triple for $i = 1$ at the latest.

Algorithm 5: Index triple assignment

Input : A customer index x and a point in time t_{x+1}^{setup}
Output: A triple (i, j, k) consisting of a customer index i, a node index j, and a time window index k, as well as a sequence of points in time $\left((t_l^{setup}, t_l^{waited}, t_l^{served}, t_l^{driven})\right)_{l=i,\ldots,x}$

1 $j := x + 1$;
2 **for** $i = x \ldots 1$ **do**
3 **if** $j = i + 1$ **then**
4 set j to the highest index such that $j \leq i$, $H_{i,j}^{sp}(t_{i+1}^{setup})$ is defined, and $\overline{T}_{i+1,i+1}^{setup}(t_{i+1}^{setup}) = \overline{T}_{i,j}^{driven}(H_{i,j}^{sp}(t_{i+1}^{setup})) + (t_{i+1}^{setup} - H_{i,j}^{sp}(t_{i+1}^{setup}))$ holds;
5 $t_i^{driven} := H_{i,j}^{sp}(t_{i+1}^{setup})$;
6 **else**
7 $t_i^{driven} := t_{i+1}^{setup}$;
8 $t_i^{served} := t_i^{driven} - drive_i$;
9 $t_i^{waited} := t_i^{served} - service_i$;
10 $t_i^{setup} := H_{i,j}^{shift}(t_i^{waited})$;
11 **if** $t_i^{setup} = t_i^{waited}$ *and* $\omega^*(\overline{T}_{i,j}^{waited}, t_i^{waited}) = \omega(\mathcal{W}_i^k)$ *for some* k **then**
12 **return** (i, j, k);

Let us have a look at some more examples. In Figure 5.11, there are four diagonal pieces corresponding to four schedules. All pieces are assigned to the same time

window, namely the first (and only) time window of customer 4. They are assigned to different index triples though. From left to right, the diagonal pieces are assigned to $(4,4,1), (4,3,1), (4,2,1)$, and $(4,1,1)$. Figure 5.12 shows two horizontal pieces that correspond to two different schedules. The characteristic of this example is that the "blue driver" takes a break less that the "red driver", and that the red piece is truncated by the blue piece. Both pieces are assigned to different time windows. The red piece is assigned to $(4,2,1)$, the blue piece to $(4,3,2)$. In the third example (Figure 5.13), $break_{short}$ is 0, so all breaks we see in the schedule view are long breaks. The three horizontal pieces are assigned to the same time window but not the same node index. From left to right, they are assigned to $(5,3,1)$, $(5,4,1)$, and $(5,5,1)$.

5.4.5.4 On the Number of Diagonal Pieces of $\overline{T}^{setup}_{x+1,x+1}$

In the second part of the proof, we show that two different diagonal pieces of $\overline{T}^{setup}_{x+1,x+1}$ must be assigned to two different index triples. So now, let z be the beginning of a diagonal piece of $\overline{T}^{setup}_{x+1,x+1}$, and let again $t_1 \leq z < t_2$. Suppose z is assigned to the triple (i,j,k). Besides the triple, the algorithm also returns the point in time t_i^{waited}, for which $\overline{T}^{waited}_{i,j}$ is defined. For z, we have

$$\overline{T}^{setup}_{x+1,x+1}(z) = \overline{T}^{waited}_{i,j}(t_i^{waited}) + (z - t_i^{waited}). \tag{5.14}$$

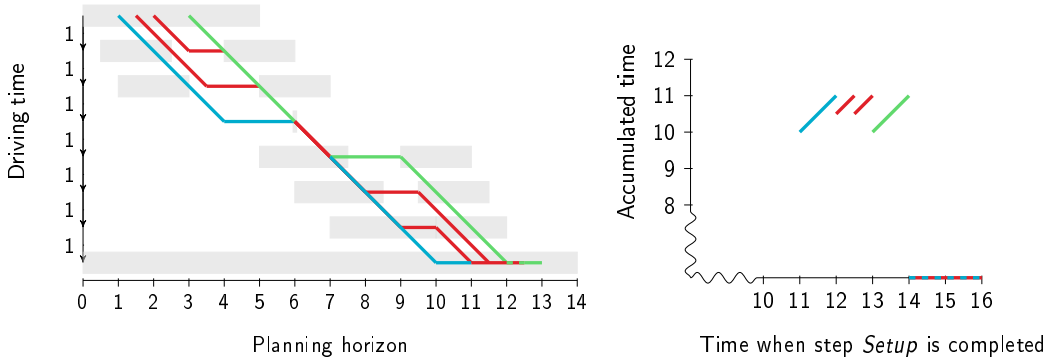
In the following, we use the short notation $e := \omega^*(\overline{T}^{waited}_{i,j}, t_i^{waited})$ to denote the end of the assigned time window. When the algorithm terminates, e must equal t_i^{waited}. This means the partial schedule that is created by the algorithm begins at the end of a time window. If that was not the case, the driver could depart from customer i later and likewise arrive at customer $x+1$ later, though with the same accumulated travel time since the end of the last long break. But then, z could not be the beginning of a diagonal piece.

Let $z' > z$ be the beginning of another diagonal piece. Let us assume that z' is assigned to the same index triple (i,j,k). Then also e must be the same. So analogously to equation 5.14, we have $\overline{T}^{setup}_{x+1,x+1}(z') = \overline{T}^{waited}_{i,j}(e) + (z' - e)$, which in turn equals $\overline{T}^{setup}_{x+1,x+1}(z) + (z' - z)$. But since $\overline{T}^{setup}_{x+1,x+1}$ only comprises horizontal or diagonal pieces, z' would have to be on the same diagonal piece that starts in z. This is a contradiction to the assumption that z' is the beginning of another diagonal piece. Hence, z' must be assigned to a different index triple. And since every beginning of a diagonal piece is assigned to a different index triple, there cannot be more than $O(nw)$ diagonal pieces.

5.4.5.5 On the Number of Horizontal Pieces of $\overline{T}^{setup}_{x+1,x+1}$

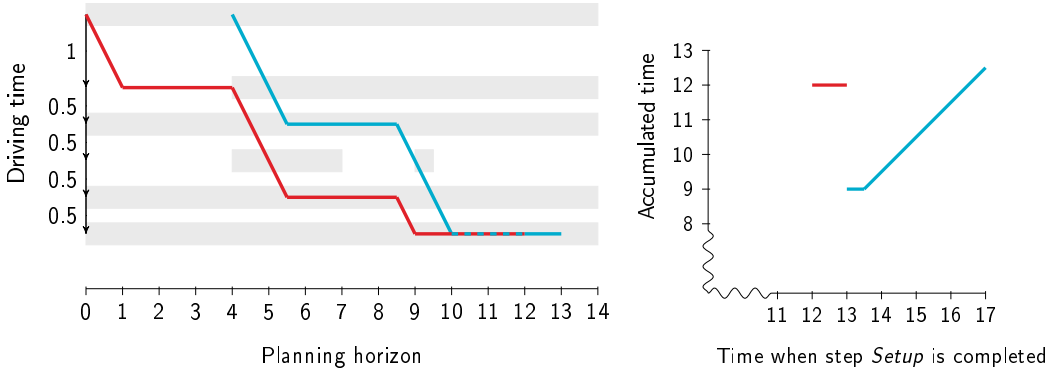
In the third part, we regard the number of horizontal pieces. So now, let z be the beginning of a horizontal piece of $\overline{T}^{setup}_{x+1,x+1}$. Again, let $t_1 \leq z < t_2$ and z be assigned to the triple (i,j,k). Besides the short notation $e := \omega^*(\overline{T}^{waited}_{i,j}, t_i^{waited})$, we shortly write $\ell := e - t_i^{waited}$.

By definition of the mapping ω^* above, $\overline{T}^{waited}_{i,j}(t)$ is the same for all t between t_i^{waited} and e. The driver could depart from customer i up to ℓ later than t_i^{waited} and arrive at customer $x+1$ by just as much later than z, with the same accumulated travel time since the end of the last long break as at time z.



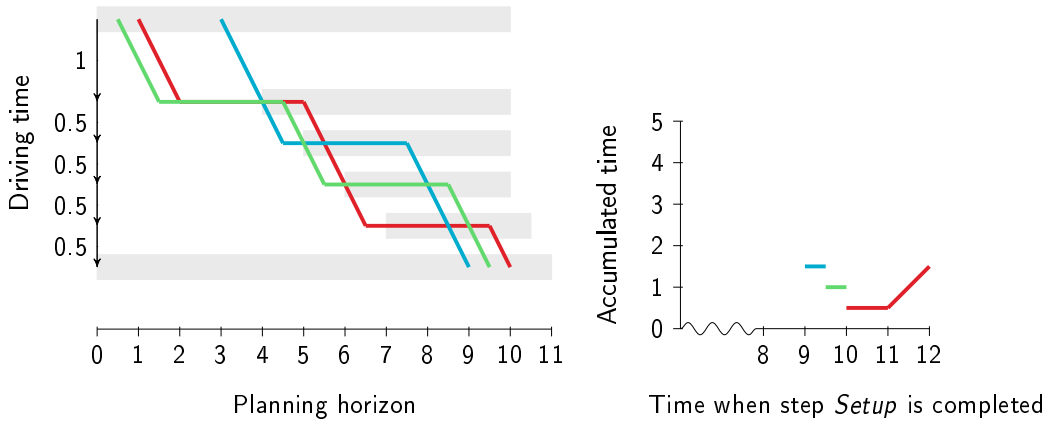
(A) Schedule view. (B) Function view.

FIGURE 5.11: Four different diagonal pieces in $\overline{T}^{setup}_{8,8}$ (right) correspond to four different schedules (left). All four pieces are assigned to the same end of a time window at customer 4. Parameter setting is $limitD_{short} = 4$, $break_{short} = 1$, $break_{long} = 4$.



(A) Schedule view. (B) Function view.

FIGURE 5.12: Two different horizontal pieces in $\overline{T}^{setup}_{6,6}$ (right) correspond to two different schedules (left). Both pieces are assigned to different time windows and different nodes. Parameter setting is $limitD_{short} = 1.5$, $break_{short} = 3$, $break_{long} > 8$.



(A) Schedule view. (B) Function view.

FIGURE 5.13: Three different horizontal pieces in $\overline{T}^{setup}_{6,6}$ (right) correspond to three different schedules (left). Parameter setting is $limitD_{short} = 1.5$, $break_{short} = 0$, $limitD_{long} = 1.5$, $break_{long} = 3$.

As a consequence, the functional value of $\overline{T}^{setup}_{i+1,i+1}$ for any time between z and $z + \ell$ can be less than or equal to the functional value at time z, but not greater. Suppose $z' > z$ is the beginning of another horizontal piece. If the functional value at any time t with $z < t \leq z'$ is higher than at time z, then z' must be greater than $z + \ell$. Both if $z' > z + \ell$ holds or if $z' \leq z + \ell$ and $\overline{T}^{setup}_{x+1,x+1}(z') < \overline{T}^{setup}_{x+1,x+1}(z)$ holds, then z' must be assigned to a different index triple.

After these observations, let us count the number of horizontal pieces. There are only two possibilities: Either a horizontal piece is contained in $\overline{T}^{setup}_{x+1,x+1}$ in its full length (that is, it is ℓ long), or a horizontal piece is truncated by another piece. From the above observations, it follows that it can only happen at most $O(nw)$ times that a piece is contained in full length. If the horizontal piece is truncated by another piece with a smaller functional value, the other piece must be assigned to a different triple. This may also happen only $O(nw)$ times. We conclude that there cannot be more than $O(nw)$ horizontal pieces.

5.4.5.6 On the Time Complexity

We have shown that the number of pieces of $\overline{T}^{step}_{x+1,j}$ with $j = x + 1$ and $step = setup$ is in $O(nw)$. We observe that the same bound holds for any other $j < x + 1$ and any other step. Furthermore, it is easy to see that among the four functions of a quadruple $\mathcal{F}^{step}_{x+1,j}$, the function $\overline{T}^{step}_{x+1,j}$ contains the most pieces. Since there are up to n quadruples per label and up to n labels, it follows that the space complexity is in $O(n^3w)$.

Most of the single operations of our solution approach can be implemented to run in linear time (linear in the number of pieces). The only super-linear operation is the minimum operation in step *Setup*. However, when we claim that every iteration can be implemented to run in $O(n^2w)$ time, this run-time is already included. We conclude that the time complexity is the same as the space complexity.

Theorem 5.4.1. *The complexity of EF-TDSP-2B with no-break-en-route policy is in* $O(n^3w)$.

5.5 Discussion of a Non-restrictive Break Policy

So far, the driver is assumed to be bound by the no-break-en-route policy. What if we allowed the driver to take breaks en route between customers? Let us consider the following problem instance: There are only two customers to be visited, and these are always open. But now suppose they are very far apart, so multiple short and long breaks en route are necessary. One of the factors that make the scheduling problem with two types of breaks challenging is the fact that it may be advantageous to prolong a due short break en route to an early (that is, not yet due) long break en route. How many non-dominated driver states can there be on arrival at the second and final customer, that is, after step *Drive*?

To give an example, let $break_{short} = 1$, $break_{long} = 50$, $limitD_{short} = 250$, $limitD_{long} = 255$, let both customers be a drive of $drive_1 = 12500$ apart, and let all time units be minutes for a better illustration. Here, a driver has to decide after 250 minutes of driving whether to take a short break of only 1 minute in order to be allowed to drive for another 5 minutes, or to take an early long break of 50 minutes immediately. The driver cannot choose an early long break more often than $\lceil drive_1 / limitD_{short} \rceil - 1 = 49$ times, provided that he always exhausts the maximum allowed driving time of $limitD_{short}$ between two long breaks.

TABLE 5.2: With breaks en route, there can be many non-dominated driver states on arrival at the next customer.

Number of early long breaks	Total break time	Accumulated driving time since last (short, long) break
49	2450	(250, 250)
48	2451	(245, 245)
47	2452	(240, 240)
...	...	...
2	2497	(15, 15)
1	2498	(10, 10)
0	2499	(5, 5)

Table 5.2 shows that there are as many as 50 different driver states on arrival at the second customer. We read from the first row of the table that, given that the driver takes the maximum number of 49 early long breaks (first column), the total time of all breaks en route is $49 \cdot break_{long} = 2450$ minutes (second column). After the 49th early long break, there are still another $drive_1 - 49 \cdot limitD_{short} = 250$ minutes to go to reach the second customer. This time matches with the accumulated driving time since the end of the last long break as well as the last short break on arrival there (third column). With every early long break less (subsequent rows), the driver reaches the second customer a minute later (due to an additional short break) but with 5 minutes less accumulated driving time (due to the difference $limitD_{long} - limitD_{short}$). So all these driver states do not dominate each other.

After step *Setup*, the number even doubles because of early breaks before service at the second customer. For instance, at time $drive_1 + 2451$, the driver may have an accumulated driving time since the last (short, long) break of either (245, 245) minutes or of (0, 250) minutes. At time $drive_1 + 2500$, the driver can be completely rested with respect to both types of breaks.

We learn from this example that the complexity of a driver states label in the case of two types of breaks does not solely depend on the number of customers and the number of their time windows, as long as we do not make any assumptions on the length of the planning horizon or any of the break rule parameters. However, in this thesis, we focus on statements on complexity that are independent of these. Hence, we do not further investigate the general case.

5.6 Conclusion and Outlook

We have investigated the truck driver scheduling problem with two types of breaks, in which, roughly speaking, a truck driver is obliged to take a short break after a short period of time and a long break after a long period of time, while also taking into account the time windows of the customers to be visited. Precisely, we have allowed for multiple time windows per customer and studied two variants of the problem that differ in their rulesets. In one of these variants, the considered ruleset for the short break is $\{drive\ until\ driven, drive\ until\ traveled\}_{short}$ and the one for the long break is $\{drive\ until\ driven, drive\ until\ traveled\}_{long}$.

We have presented an exact algorithm for the problem variant in which breaks are only allowed to be taken at customers and not en route between them (no-break-en-route policy). It is shown to run in $O(n^3w)$ time, dependent on the number of customers n and the number of their time windows w. This makes it the first polynomial-time algorithm for this problem. For the general case with a non-restrictive break policy, we have demonstrated that the number of non-dominated driver states is not only dependent on n and w but also the setting of the break rule parameters and the driving times. We conjecture that a strongly polynomial-time algorithm does not exist in this case.

The general case is beyond the scope of this thesis, and therefore, it remains an interesting subject for further research. We would like to learn more about the correlation between the break rule parameter settings and the exact number of non-dominated driver states on arrival at the next customer. Particularly, there are important special cases. In the standard setting of the rule parameters in the EU, the rule parameter $limitD_{long}$ is set to twice as much as $limitD_{short}$ (see Table 5.1). In this scenario, it is never advantageous to schedule an early long break more than once, no matter how long the driving time between two customers is. If in addition $\lceil drive_1 / limitD_{short} \rceil$ is odd, it is never beneficial at all, i.e., only due long breaks need to be scheduled. So with the standard rules of the EU, there are at most two non-dominated driver states on arrival at the next customer. It would be interesting to investigate if this can be exploited. However, these are only the standard rules. In the EU, the driver is allowed to drive for 10 hours instead of 9 hours twice a week. With this rule considered, things change again, and there can be more than two non-dominated driver states on arrival at the next customer.

As a compromise between the no-break-en-route policy and a non-restrictive break policy, it may be worthwhile to examine the *no-long-break-en-route policy*, in which short breaks are still allowed to be taken en route. But even if we stick to the restrictive no-break-en-route policy, there is still an open research field. One direction for research is to investigate in how far our polynomial-time approach could be enhanced to solve the truck driver scheduling problem with three or even more types of breaks. Another direction is to study in how far the ideas of chapter 4 regarding the minimum duration objective could be transferred in order to solve the minimum duration truck driver scheduling problem with (at least) two types of breaks. An exact dynamic programming based algorithm for the minimum duration truck driver scheduling problem with multiple types of breaks already exists (Goel, 2012c) but a polynomial-time algorithm is not yet known.

Chapter 6

Vehicle Routing and Truck Driver Scheduling with Multiple Time Windows

6.1 Introduction

In this chapter, we deal with the vehicle routing and truck driver scheduling problem (VRTDSP). As the name suggests, it combines the vehicle routing problem (here: with multiple time windows) with the truck driver scheduling problem. That is, a set of geographically dispersed customers needs to be visited within given time windows, and a fleet of trucks is given in order to fulfill their demand. The objective is to assign every customer to a truck and determine the sequence of customer visits for every truck such that a feasible truck driver schedule exists for every truck. As before, such a schedule is deemed feasible if all customers are served in time and the provisions on breaks are respected. Here, we restrict ourselves to the rules that are applicable in the EU for a planning horizon of one day. Other constraints, such as capacity constraints, do not play a role in our setting. Among all feasible solutions, we look for one with the minimum number of vehicles that have customers assigned to them, and – as secondary optimization goal – with the least total duration of all the vehicle routes.

As we have seen in previous chapters, checking the feasibility of a vehicle route is a non-trivial task if break rules have to be regarded. In this chapter, we shed light on the question how the feasibility of the routes can be checked efficiently within local search based heuristics. That is, we show how redundant computations can be avoided when certain neighborhoods of a solution are searched, particularly those neighborhoods that build on the concatenation of two partial routes. To this end, a pre-processing step is introduced in which some feasibility information regarding partial routes is collected within a forward propagation and a backward propagation phase. This pre-computed information is stored and used whenever the concatenation of two partial routes is checked for feasibility.

As an example of such a neighborhood, we illustrate the so-called *2-opt* neighborhood* in Figure 6.1. It was first proposed by Potvin and Rousseau (1995) as a variant of the 2-opt neighborhood introduced by Lin (1965). The transformation from one solution to a neighboring solution is called a *move*. A 2-opt* move (also known as a *crossover* move) means to first remove two links from two different vehicle routes, thus splitting the routes in two parts each. Then, two new links are inserted, each recombining a first part with the other second part. In the example, the links (c_{13}, c_{14}) and (c_{23}, c_{24}) are removed and replaced by (c_{13}, c_{24}) and (c_{23}, c_{14}).

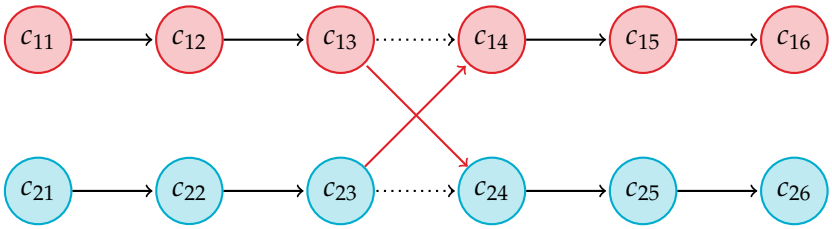


FIGURE 6.1: Example of a 2-opt* (also known as crossover) move. Two links (dotted black) are removed and replaced by two others (solid red).

Algorithm 6 outlines one iteration of a local search with regard to the 2-opt* neighborhood. Input to the algorithm is a feasible solution. For every pair of vehicle routes $\mathcal{R}'$ and $\mathcal{R}''$ in the solution, $|\mathcal{R}'+1| \cdot |\mathcal{R}''+1|$ recombinations of two partial routes are checked for feasibility, where $|\mathcal{R}|$ denotes the number of customers in route $\mathcal{R}$. Without loss of generality, we consider a problem variant in which the initial and final locations of the vehicles – usually called depots – are disregarded. In our setting, a vehicle route begins and ends with the visit of a customer, and it is assumed that the driver has not accumulated any driving time when the service begins at the first customer of the route. Only if the recombination of two partial routes is feasible, the move is evaluated, that is, the value of the objective function of the neighboring solution is calculated. Since one of the moves is the identity map, Algorithm 6 always returns a feasible move.

Algorithm 6: 2-opt* neighborhood search

Input : A feasible solution

1 **for** *every pair of vehicle routes* $\mathcal{R}'$ *and* $\mathcal{R}''$ *in the solution* **do**
2 **for** *every number* x' *between 0 and* $|\mathcal{R}'|$ **do**
3 **for** *every number* x'' *between 0 and* $|\mathcal{R}''|$ **do**
4 check if the concatenation of the first x' customers of $\mathcal{R}'$ and the last $|\mathcal{R}''| - x''$ customers of $\mathcal{R}''$ is a feasible vehicle route;
5 check if the concatenation of the first x'' customers of $\mathcal{R}''$ and the last $|\mathcal{R}'| - x'$ customers of $\mathcal{R}'$ is a feasible vehicle route;
6 if both are feasible, evaluate the gain of this move;
7 **return** the move that is evaluated best;

Algorithm 7 outlines a general scheme of a local search based algorithm. The first step is to create an initial solution. There is a trivial but feasible solution in which every route is a singleton, that is, every customer is assigned to its own route. The focus of this chapter is on the following pre-processing part of the algorithm. It is explained in more detail in section 6.3. In the main loop, the neighborhood (possibly even several neighborhoods) of the incumbent solution is sought for an improving move. In this step, that pre-processed information is exploited to accelerate the feasibility check and the evaluation of the moves.

If an improving move is found, it is implemented, that is, the corresponding neighboring solution becomes the incumbent solution. Since some pieces of the pre-processed information are invalidated by this move, they have to be updated accordingly. If such a move is not found, the main loop is left. We take this as a stopping

criterion and content ourselves with the first found local optimum. This is because the overall solution approach to the VRTDSP is not in the focus of this chapter. The purpose of Algorithms 6 and 7 is to put the presented speed-up technique into a broader context.

Algorithm 7: Basic local search based algorithm

```
1 create an initial feasible solution;
2 pre-compute travel time information for every route;
3 repeat
4 |   search for an improving move in the neighborhood(s) (e.g., call
  |     Algorithm 6);
5 |   if such an improving move exists then
6 |   |   update the incumbent solution accordingly;
7 |   |   update the pre-computed information accordingly;
8 until some stopping criterion is met;
9 return the incumbent solution;
```

Solution Framework Certainly, more sophisticated algorithms than Algorithm 7 exist. As mentioned in section 2.2, the algorithm by Goel and Vidal (2014) is arguably the currently best heuristic approach. But even this approach could be accelerated as the described implementation is based solely on forward propagation. However, in contrast to their work, we restrict ourselves to the break rules of the EU that are applicable within a planning horizon of one day.

Vidal et al. (2014) propose a unified solution framework for multi-attribute vehicle routing problems (see Vidal et al. (2013) for a survey on heuristics for such problems). Our problem at hand can be thought of as such a multi-attribute problem and thus their solution framework could be used to solve our problem. In their framework, the key components of the feasibility and gain evaluation of a route are five functions: three refer to phases of the pre-processing step (initialization, forward propagation, backward propagation) and two refer to the actual route evaluations. The authors call them `Eval2` and `EvalN`. The function `Eval2` evaluates the concatenation of exactly two partial routes whereas `EvalN` takes an arbitrary number of partial routes as input. Note that `EvalN` $(\mathcal{R}', \mathcal{R}^x, \mathcal{R}'')$ for three partial routes may be implemented by as many successive forward propagation calls as there are customers in the middle route $\mathcal{R}^x$ and finally one call to `Eval2`. According to this model, we show in this chapter how to efficiently implement an `Eval2` function for the concatenation of two partial routes, evaluating gain and feasibility with respect to break rules in the EU for a planning horizon of one day. This function could be plugged into the solution framework described by Vidal et al. (2014).

Other Related Work Savelsbergh (1985), Savelsbergh (1992), and Campbell and Savelsbergh (2004) describe efficient implementations of feasibility checks in case of the vehicle routing problem with time windows (that is, allowing for a single time window per customer). Ibaraki et al. (2005) and Hashimoto, Yagiura, and Ibaraki (2008) present such efficient evaluations in case of the vehicle routing problem with *general* time windows, in which – instead of time windows – a piecewise linear service cost function is given per customer that expresses the inappropriateness of service over time. Like we do, they all store meaningful information about partial routes to accelerate the feasibility and gain evaluations of new routes.

In the context of the VRTDSP, we are only aware of three papers (Ceselli, Righini, and Salani, 2009; Tilk, 2016; Schiffer et al., 2017) in which bidirectional labeling is performed to speed up the checks. In all three cases, only the single time window case is considered. Ceselli, Righini, and Salani (2009) develop a column generation algorithm for a rich vehicle routing problem that includes a constraint on the maximum driving time without break. Tilk (2016) presents an exact branch-and-price-and-cut approach to solve the VRTDSP. Based on the branch-and-price approach of Goel and Irnich (2017), the author develops a bidirectional labeling for the pricing problem. Schiffer et al. (2017) assess the impact of break rules in case of electric vehicles that may need to be recharged. They present a bidirectional feasibility check within an adaptive large neighborhood search for the VRTDSP with electric vehicles.

Contribution With multiple time windows per customer, the scheduling part of the VRTDSP becomes significantly harder. As a basis of an efficient implementation, we present a bidirectional labeling for the VRTDSP with multiple time windows per customer, considering the *drive until driven* rule as well as limits on the total driving time and the total travel time of each route. The described technique could be integrated into the framework of Vidal et al. (2014). And it has already been integrated into the vehicle routing service of PTV xServer.

Outline A problem definition is given in section 6.2. The main part of this chapter is the presentation of the integrated approach for the scheduling subproblem in section 6.3. A conclusion and an outlook follow in section 6.4.

6.2 Problem Definition

We are given a set of n customers $c_1, \dots, c_n$. Every customer c_i requests a service of duration $service_{c_i}$ that shall begin within one of the w_i time windows in the set $\{\mathcal{W}_{c_i}^1, \dots, \mathcal{W}_{c_i}^{w_i}\}$. The driving time between every two customers c_i and c_j is supposed to be determined in advance. It is denoted by $drive_{c_i,c_j}$. Also given is a planning horizon $\mathcal{H}$.

Let us call a sequence of customers that contains every customer at most once a *(vehicle) route*. A vehicle route is feasible if a schedule exists such that not only the customers' time windows are respected but also the drivers' working hours. In this chapter, we focus on the break rules of the European Union that are applicable for a planning horizon of one day. In this setting, we need to consider one rule regarding a short break: *drive until driven*. That is, after $limitD_{short}$ of driving, the driver is prohibited from driving unless he takes a short break of duration *break*. The duration of a long break does not play a role here as we do not want to be forced to schedule such a long break. Instead, we take $limitD_{long}$ and $limitT_{long}$ as maximum constraints, that is, we consider a route as infeasible if the total driving time on that route exceeds $limitD_{long}$ or if the total travel time exceeds $limitT_{long}$. Without loss of generality, we expect $service_{c_i} \leq limitT_{long}$ to hold for all customers, and that all drivers are completely rested with respect to the long break when they commence their work at each first customer of a route. It should be noted that, with *drive until driven* as the only break rule, a break that is longer than *break* has no advantage over a break that has only the minimum length.

Let us call a collection of vehicle routes a *solution* (of the VRTDSP) if every customer is contained in exactly one route. It is feasible if all its routes are feasible. The trivial solution $\{(c_1), \dots, (c_n)\}$ in which all routes are singletons is feasible. Starting

from an initial feasible solution, the task is to find a local optimum with respect to the 2-opt* neighborhood. The optimization goal is to minimize the number of routes in the solution as first criterion, and the total duration of all the routes as second criterion.

6.3 Integrated Approach

Let an initial feasible solution be given. That is, we are given a set of $m \leq n$ feasible vehicle routes $\mathcal{R}_1, \ldots, \mathcal{R}_m$ such that every given customer is contained in exactly one vehicle route. The integrated approach relies on a pre-processing step (line 2 of Algorithm 7). It is described in sections 6.3.1 through 6.3.4. First, we define the contents of a driver states label (section 6.3.1). Initialization and the general scheme is presented in section 6.3.2. Driver states labels are pre-computed for every customer of a vehicle route in a forward (section 6.3.3) and a backward (section 6.3.4) propagation phase. The actual feasibility check within the neighborhood search (line 4 of Algorithm 7) is described in section 6.3.5.

6.3.1 Driver States Label

Let $\mathcal{R}$ be one of the given vehicle routes, and let $\mathcal{R}[i]$ be the i-th customer in the sequence. Analogously to the algorithms in previous chapters, labels are computed iteratively. There are as many iterations as there are customers in the sequence, and every iteration is subdivided into the steps *Wait*, *Serve*, *Drive*, and *Setup*. In this chapter, let a *forward* driver states label $\overrightarrow{\mathcal{L}}^{step}_{\mathcal{R}[i]}$ after step *step* of iteration i refer to the i-th customer of the route $\mathcal{R}$. It comprises a scalar value $\overrightarrow{dl}^{step}_{\mathcal{R}[i]}$ that denotes the accumulated driving time since the last *long* break on one hand, and a sequence $(\overrightarrow{\mathcal{F}}^{step}_{\mathcal{R}[i],j})_{j \leq i}$ of i tuples on the other hand (compare chapter 5). In turn, for some $j \leq i$, such a tuple $\overrightarrow{\mathcal{F}}^{step}_{\mathcal{R}[i],j} = (\overrightarrow{ds}^{step}_{\mathcal{R}[i],j}, \overrightarrow{T}^{step}_{\mathcal{R}[i],j})$ comprises a scalar value $\overrightarrow{ds}^{step}_{\mathcal{R}[i],j}$ that denotes the accumulated driving time since the last *short* break. And, like in previous chapters, $\overrightarrow{T}^{step}_{\mathcal{R}[i],j}$ is a time-dependent travel time function. But unlike before, $\overrightarrow{T}^{step}_{\mathcal{R}[i],j}(t)$ denotes the minimum travel time at time t that the driver has accumulated since the last *long* break and thus since the beginning of the route.

We have to take early short breaks at customers into account. The second index $j \leq i$ indicates at which customer the last early short break was taken. As in previous chapters, we will see that $\overrightarrow{T}^{step}_{\mathcal{R}[i],j}$ is piecewise constant, and that $\overrightarrow{T}^{step}_{\mathcal{R}[i],j} - \mathrm{id}$ is monotonously decreasing (aside from gaps), that is, $\overrightarrow{T}^{step}_{\mathcal{R}[i],j}(t_1) + t_2 - t_1 \geq \overrightarrow{T}^{step}_{\mathcal{R}[i],j}(t_2)$ holds for any two points in time $t_1 < t_2$ for which $\overrightarrow{T}^{step}_{\mathcal{R}[i],j}$ is defined.

Since we label bidirectionally, there is also a *backward* driver states label $\overleftarrow{\mathcal{L}}^{step}_{\mathcal{R}[i]}$. Analogously to the forward label, it is a pair of a scalar value $\overleftarrow{dl}^{step}_{\mathcal{R}[i]}$ and a sequence $(\overleftarrow{\mathcal{F}}^{step}_{\mathcal{R}[i],j})_{j \leq |\mathcal{R}|-i+1}$ of $|\mathcal{R}| - i + 1$ tuples, where $|\mathcal{R}|$ denotes the number of customers in the sequence $\mathcal{R}$. Table 6.1 summarizes the contents of the two labels per customer. Figure 6.2 depicts (a part of) the dependency graph (see chapter 5) in the bidirectional case. Every node in the graph represents a tuple. Here, a vehicle route with 7 customers is assumed. Depicted are only the nodes that correspond to the first four forward labels and the last three backward labels.

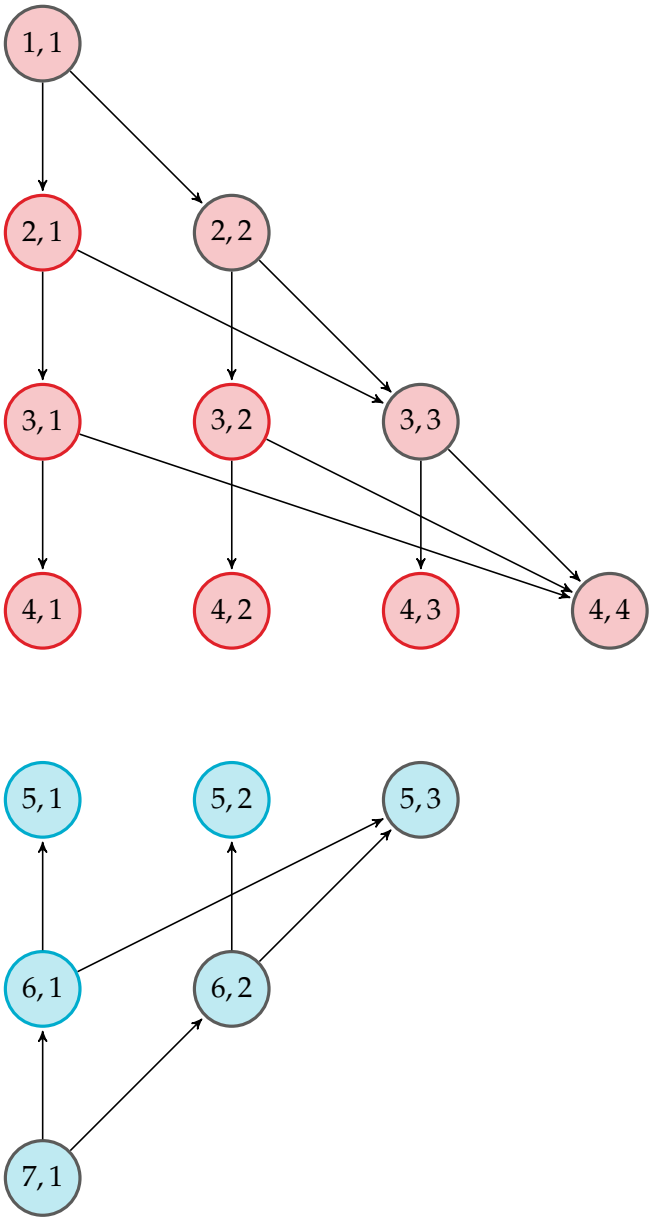


FIGURE 6.2: Dependency graph, bidirectional propagation.

TABLE 6.1: Driver states label summary (EF-TDSP-2B).

$step$	one of {*Setup*, *Wait*, *Serve*, *Drive* }
$\overrightarrow{dl}^{step}_{\mathcal{R}[i]}$, $\overleftarrow{dl}^{step}_{\mathcal{R}[i]}$	driving time to the i-th customer of route $\mathcal{R}$, accumulated since the last *long* break (in forward or backward direction)
$\overrightarrow{ds}^{step}_{\mathcal{R}[i],j}$, $\overleftarrow{ds}^{step}_{\mathcal{R}[i],j}$	driving time to the i-th customer of route $\mathcal{R}$, accumulated since last *short* break, with the last early short break taken at the j-th customer (in forward direction) or at the $\lvert\mathcal{R}\rvert - j + 1$-th customer (in backward direction) of the same route
$\overrightarrow{T}^{step}_{\mathcal{R}[i],j}$, $\overleftarrow{T}^{step}_{\mathcal{R}[i],j}$	time-dependent function, mapping points in time at the end of step $step$ at the i-th customer to the minimum accumulated *travel times* since the end of the last *long* break, with the last early short break taken at the j-th customer (in forward direction) or at the $\lvert\mathcal{R}\rvert - j + 1$-th customer (in backward direction) of the same route
$\overrightarrow{\mathcal{F}}^{step}_{\mathcal{R}[i],j}$, $\overleftarrow{\mathcal{F}}^{step}_{\mathcal{R}[i],j}$	the pairs $\left(\overrightarrow{ds}^{step}_{\mathcal{R}[i],j}, \overrightarrow{T}^{step}_{\mathcal{R}[i],j}\right)$ and $\left(\overleftarrow{ds}^{step}_{\mathcal{R}[i],j}, \overleftarrow{T}^{step}_{\mathcal{R}[i],j}\right)$
$\overrightarrow{\mathcal{L}}^{step}_{\mathcal{R}[i]}$, $\overleftarrow{\mathcal{L}}^{step}_{\mathcal{R}[i]}$	the pairs $\left(\overrightarrow{dl}^{step}_{\mathcal{R}[i]}, (\overrightarrow{\mathcal{F}}^{step}_{\mathcal{R}[i],j})_{j\leq i}\right)$ and $\left(\overleftarrow{dl}^{step}_{\mathcal{R}[i]}, (\overleftarrow{\mathcal{F}}^{step}_{\mathcal{R}[i],j})_{j\leq \lvert\mathcal{R}\rvert-i+1}\right)$

6.3.2 Outline of Propagation Scheme and Initialization

Algorithm 8 outlines the forward and the backward propagation phase of the pre-processing step (called in line 2 of Algorithm 7). It expects $\overrightarrow{\mathcal{L}}^{setup}_{\mathcal{R}[1]}$ and $\overleftarrow{\mathcal{L}}^{setup}_{\mathcal{R}[|\mathcal{R}|]}$ as input. We assume that the driver is completely rested with respect to the long break when the planning horizon begins, and also that there is enough time for another long break after the end of the planning horizon before the driver is engaged in work again. We initialize $\overrightarrow{\mathcal{L}}^{setup}_{\mathcal{R}[1]}$ by setting $\overrightarrow{dl}^{setup}_{\mathcal{R}[1]} := 0$, $\overrightarrow{ds}^{setup}_{\mathcal{R}[1],1} := 0$, and $\overrightarrow{T}^{setup}_{\mathcal{R}[1],1}(t) := 0$ for all $t \in \mathcal{H}$ (undefined for all other t), and analogously we initialize $\overleftarrow{\mathcal{L}}^{setup}_{\mathcal{R}[|\mathcal{R}|]}$ by setting $\overleftarrow{dl}^{setup}_{\mathcal{R}[|\mathcal{R}|]} := 0$, $\overleftarrow{ds}^{setup}_{\mathcal{R}[|\mathcal{R}|],1} := 0$, and $\overleftarrow{T}^{setup}_{\mathcal{R}[|\mathcal{R}|],1}(t) := 0$ for all $t \in \mathcal{H}$ (undefined for all other t).

When a move is applied, two links are changed, and the pre-processed information must be updated accordingly (line 7 of Algorithm 7). Suppose there is a new link that connects customer c_i with c_j. Then we have to propagate the label $\overrightarrow{\mathcal{L}}^{served}_{\mathcal{R}[c_i]}$ forward until the end of the route. Analogously, we have to propagate the label $\overleftarrow{\mathcal{L}}^{served}_{\mathcal{R}[c_j]}$ backward until the beginning of the route.

Preliminaries For the description of our approach, we introduce two functions for each customer c_i, the *forward waiting time function* $\overrightarrow{W}_{c_i}$ and the *backward waiting time function* $\overleftarrow{W}_{c_i}$. For a given point in time t, the forward waiting time function denotes the time the driver has to wait for the next time window to open. It is infinite after the last time window. The backward waiting time function denotes the time by how much the driver has missed the previous time window when we assume that all time windows of customer c_i were shifted to the right by the service time of customer c_i. We need to take account of the service time here because in this work (just like in related works), it is only required that the service must start within a time window,

Algorithm 8: Bidirectional label propagation

Input : $\overrightarrow{\mathcal{L}}^{setup}_{\mathcal{R}[1]}$, $\overleftarrow{\mathcal{L}}^{setup}_{\mathcal{R}[|\mathcal{R}|]}$

forall $i = 1 \dots |\mathcal{R}|$ **do**
- $\overrightarrow{\mathcal{L}}^{waited}_{\mathcal{R}[i]} :=$ `Wait` $(\overrightarrow{\mathcal{L}}^{setup}_{\mathcal{R}[i]})$;
- $\overrightarrow{\mathcal{L}}^{served}_{\mathcal{R}[i]} :=$ `Serve` $(\overrightarrow{\mathcal{L}}^{waited}_{\mathcal{R}[i]})$;
- **if** $i = |\mathcal{R}|$ **then**
  - break;
- $\overrightarrow{\mathcal{L}}^{driven}_{\mathcal{R}[i]} :=$ `Drive` $(\overrightarrow{\mathcal{L}}^{served}_{\mathcal{R}[i]})$;
- $\overrightarrow{\mathcal{L}}^{setup}_{\mathcal{R}[i+1]} :=$ `Setup` $(\overrightarrow{\mathcal{L}}^{driven}_{\mathcal{R}[i]})$;

forall $i = |\mathcal{R}| \dots 1$ **do**
- $\overleftarrow{\mathcal{L}}^{waited}_{\mathcal{R}[i]} :=$ `Wait` $(\overleftarrow{\mathcal{L}}^{setup}_{\mathcal{R}[i]})$;
- $\overleftarrow{\mathcal{L}}^{served}_{\mathcal{R}[i]} :=$ `Serve` $(\overleftarrow{\mathcal{L}}^{waited}_{\mathcal{R}[i]})$;
- **if** $i = 1$ **then**
  - break;
- $\overleftarrow{\mathcal{L}}^{driven}_{\mathcal{R}[i]} :=$ `Drive` $(\overleftarrow{\mathcal{L}}^{served}_{\mathcal{R}[i]})$;
- $\overleftarrow{\mathcal{L}}^{setup}_{\mathcal{R}[i-1]} :=$ `Setup` $(\overleftarrow{\mathcal{L}}^{driven}_{\mathcal{R}[i]})$;

that is, the time windows are not "symmetric". The backward waiting time function is infinite before the first time window. An example is given in Figure 6.3.

In section 2.4.3, we have introduced the notion of waiting intervals. Here we distinguish *forward and backward waiting intervals*. For customer c_i, the forward waiting intervals are defined to be the time periods before (or between) the time windows of customer c_i, i.e., $\overrightarrow{W}^1_{c_i} := [\alpha(\mathcal{H}), \alpha(\mathcal{W}^1_{c_i}))$ and $\overrightarrow{W}^j_{c_i} := (\omega(\mathcal{W}^{j-1}_{c_i}), \alpha(\mathcal{W}^j_{c_i}))$ for all $2 \leq j \leq w_i$. $\overrightarrow{W}_{c_i}$ denotes the union of the forward waiting intervals. Analogously, the backward waiting intervals are defined as follows: $\overleftarrow{W}^{w_i}_{c_i} := (\omega(\mathcal{W}^{w_i}_{c_i}) + service_{c_i}, \omega(\mathcal{H})]$ and $\overleftarrow{W}^j_{c_i} := (\omega(\mathcal{W}^j_{c_i}) + service_{c_i}, \alpha(\mathcal{W}^{j+1}_{c_i}) + service_{c_i})$ for all $1 \leq j \leq w_i - 1$. $\overleftarrow{W}_{c_i}$ denotes their union.

6.3.3 Forward Propagation

In this section we describe the four steps *Wait*, *Serve*, *Drive*, and *Setup* of each iteration. Both $\overrightarrow{dl}^{step}_{\mathcal{R}[i]}$ and $\overrightarrow{ds}^{step}_{\mathcal{R}[i],j}$ only change in step *Drive*. After the presentation of step *Setup*, we revise step *Wait* slightly and show how to remove redundant driver states.

6.3.3.1 Step *Wait*

In this step of an iteration $i \leq |\mathcal{R}|$, we take account of the time windows of customer c_i. The scalar values remain unchanged in this step, that is, $\overrightarrow{dl}^{waited}_{\mathcal{R}[i]} := \overrightarrow{dl}^{setup}_{\mathcal{R}[i]}$ and $\overrightarrow{ds}^{waited}_{\mathcal{R}[i],j} := \overrightarrow{ds}^{setup}_{\mathcal{R}[i],j}$ for every $j \leq i$. So in the following, it is all about determining $\overrightarrow{T}^{waited}_{\mathcal{R}[i],j}$ for every $j \leq i$.

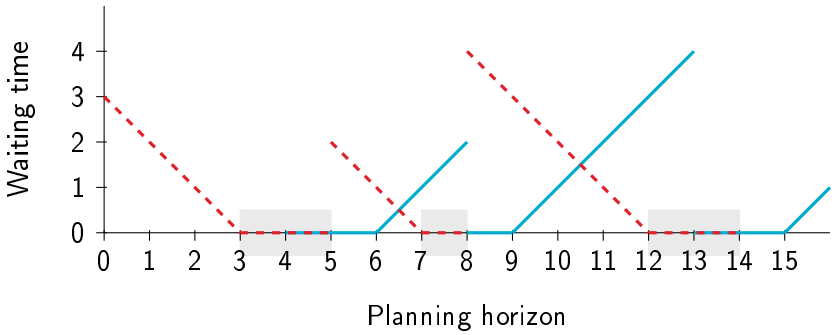


FIGURE 6.3: Forward waiting time function $\overrightarrow{W}$ in red and dashed. Backward waiting time function $\overleftarrow{W}$ in blue. These functions correspond to the time windows in the background (gray). Here, a service time of 1 is assumed.

We proceed as in step *Wait* of previous chapters and first define an auxiliary function for every $j \leq i$:

$$H_j^{shift}(t) := \max\{t' \mid t = t' + \overrightarrow{W}_{\mathcal{R}[i]}(t') \wedge \overrightarrow{T}^{setup}_{\mathcal{R}[i],j}(t') \neq \perp\}$$

where the maximum over the empty set is supposed to be $\perp$. Next, we treat the cases $j = i$ and $j < i$ separately and begin with the former. By construction of the algorithm, we know that $ds^{setup}_{\mathcal{R}[i],i} = 0$, that is, the driver is completely rested. In this case, waiting coincides with prolonging the break. So with the help of $H_i^{shift}(t)$, we set $\overrightarrow{T}^{waited}_{\mathcal{R}[i],i}(t)$ as follows:

$$\overrightarrow{T}^{waited}_{\mathcal{R}[i],i}(t) := \overrightarrow{T}^{setup}_{\mathcal{R}[i],i}(H_i^{shift}(t)) + (t - H_i^{shift}(t))$$

for all t for which $H_i^{shift}(t)$ is defined, and $\overrightarrow{T}^{waited}_{\mathcal{R}[i],i}(t) := \perp$ for all other t.

But what about $\overrightarrow{T}^{waited}_{\mathcal{R}[i],j}$ for a $j < i$? In this case, we have to be careful if we do not want to create dominated driver states. For example, we should only allow to wait for less than the minimum break duration *break* in this case because otherwise the driver state after waiting would be dominated by a state contained in $\overrightarrow{\mathcal{F}}^{waited}_{\mathcal{R}[i],i}$ already. But there is more, as Figure 6.4 suggests. Here, two schedules are depicted. By construction of the example, both schedules cannot be shifted to the right, and so a driver that follows either of these schedules has to wait until the time window of the fourth customer opens. In the red schedule, there is an early break planned at the second customer, whereas in the blue schedule, there is an early break planned at the third customer. So the driver state at the end of the red schedule is contained in $\overrightarrow{\mathcal{F}}^{setup}_{\mathcal{R}[4],2}$, the other in $\overrightarrow{\mathcal{F}}^{setup}_{\mathcal{R}[4],3}$. In this example, the state of a driver that follows the red schedule would be dominated by the state of a driver following the blue schedule if both drivers waited for the time window to open.

The rough idea how to further improve this step and not create dominated states is the following: Suppose the tuples are sorted by $ds^{setup}_{\mathcal{R}[i],j}$ in increasing order. Whenever we consider waiting for the beginning of a time window, we store the (best known) travel time for that point in time. We only allow to create a driver state for this point in time if the travel time is less than the previously stored value (initially infinite).

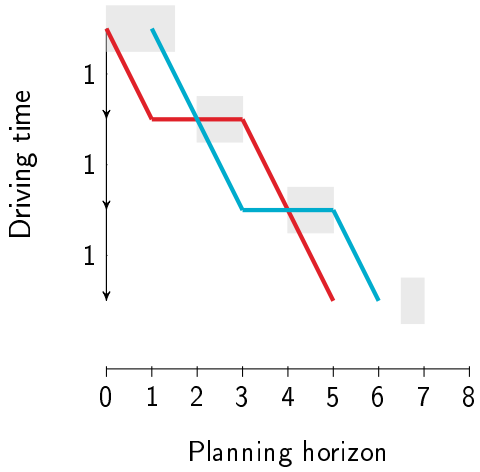


FIGURE 6.4: Both drivers must wait before the service at the fourth customer. But the driver state of the red driver would be dominated by the blue driver after waiting for the time window to open.

Now in more detail: Let $\mathcal{Q}$ be a sequence of w_i periods of time, that is, one for every time window of customer c_i. For every k from 1 to w_i and thus for every beginning $t := \alpha(\mathcal{W}_i^k)$ of a time window, we initially set $\mathcal{Q}[k]$ to $\overrightarrow{T}^{waited}_{\mathcal{R}[i],i}(t)$ if this value is defined and to ∞ otherwise. For every $j < i$, in increasing order of $ds^{setup}_{\mathcal{R}[i],j}$, and again every k from 1 to w_i (in no particular order), we only allow to wait for the beginning $t := \alpha(\mathcal{W}_i^k)$ of the k-th time window if the expected travel time $\overrightarrow{T}^{setup}_{\mathcal{R}[i],j}(H_j^{shift}(t)) + (t - H_j^{shift}(t))$ is less than $\mathcal{Q}[k]$. If it is, then we set $\mathcal{Q}[k]$ to this lower value. Precisely, we set

$$\overrightarrow{T}^{waited}_{\mathcal{R}[i],j}(t) := \overrightarrow{T}^{setup}_{\mathcal{R}[i],j}(H_j^{shift}(t)) + (t - H_j^{shift}(t))$$

for all t for which $H_j^{shift}(t)$ is defined and, should t equal the beginning of the k-th time window, for which $\overrightarrow{T}^{setup}_{\mathcal{R}[i],j}(H_j^{shift}(t)) + (t - H_j^{shift}(t))$ is less than $\mathcal{Q}[k]$. Otherwise we set $\overrightarrow{T}^{waited}_{\mathcal{R}[i],j}(t) := \bot$.

6.3.3.2 Step *Serve*

This step is again the simplest. All we have to do is to shift the pieces to the right and raise them by the service time, that is, for all $j \leq i$:

$$\overrightarrow{T}^{served}_{\mathcal{R}[i],j}(t) := \overrightarrow{T}^{waited}_{\mathcal{R}[i],j}(t - service_{\mathcal{R}[i]}) + service_{\mathcal{R}[i]}$$

for all t within the planning horizon, and $\overrightarrow{T}^{served}_{\mathcal{R}[i],j}(t) := \bot$ for all other points in time. Again, the scalar values remain unchanged in this step.

6.3.3.3 Step *Drive*

In this step, we consider the driving time $drive_{\mathcal{R}[i],\mathcal{R}[i+1]}$ from customer $\mathcal{R}[i]$ to customer $\mathcal{R}[i+1]$ (this step is only called if $i < |\mathcal{R}|$). In this section, we use the short notation $drive_i$ for $drive_{\mathcal{R}[i],\mathcal{R}[i+1]}$. The driving time rule may enforce a break (or even multiple breaks) en route. To decide whether this is the case or not, we introduce an

auxiliary variable h_j^{exc} for every $j \leq i$ that states by how much the driving time limit would be exceeded if the driver did not take a break en route:

$$h_j^{exc} := \overrightarrow{ds}^{served}_{\mathcal{R}[i],j} + drive_i - limitD_{short}$$

If $h_j^{exc} \leq 0$, then the driver is able to arrive at the next customer without having to take a break. We increase the scalar value concerning the accumulated driving time since last break by $drive_i$, and we shift (to the right) and raise the pieces of the travel time function by the same value:

$$\overrightarrow{ds}^{driven}_{\mathcal{R}[i],j} := \overrightarrow{ds}^{served}_{\mathcal{R}[i],j} + drive_i$$
$$\overrightarrow{T}^{driven}_{\mathcal{R}[i],j}(t) := \overrightarrow{T}^{served}_{\mathcal{R}[i],j}(t - drive_i) + drive_i$$

However, if $h_j^{exc} > 0$, then the driver must take a break en route after a driving time of $limitD_{short} - \overrightarrow{ds}^{served}_{\mathcal{R}[i],j}$, otherwise the driving time rule would be violated. Should even $h_j^{exc} > limitD_{short}$ hold, then additional breaks become due. In total, as many as

$$h_j^{\#} := \left\lceil h_j^{exc} / limitD_{short} \right\rceil$$

breaks must be taken en route. With this variable, we have everything we need to set $\overrightarrow{\mathcal{F}}^{driven}_{\mathcal{R}[i],j}$ in case h_j^{exc} is greater than zero. The accumulated driving time $\overrightarrow{ds}^{driven}_{\mathcal{R}[i],j}$ is set to the driving time after the last break en route, which is $h_j^{exc} - (h_j^{\#} - 1) \cdot limitD_{short}$. The travel time function $\overrightarrow{T}^{driven}_{\mathcal{R}[i],j}$ is shifted (to the right) and raised by the same time as time passes between departure from customer c_i and arrival at customer c_{i+1}, which is $drive_i + h_j^{\#} \cdot break$. To sum it up, we set:

$$\overrightarrow{ds}^{driven}_{\mathcal{R}[i],j} := h_j^{exc} - (h_j^{\#} - 1) \cdot limitD_{short}$$
$$\overrightarrow{T}^{driven}_{\mathcal{R}[i],j}(t) := \overrightarrow{T}^{served}_{\mathcal{R}[i],j}\Big(t - (drive_i + h_j^{\#} \cdot break)\Big) + (drive_i + h_j^{\#} \cdot break)$$

To conclude this step, we set $\overrightarrow{dl}^{driven}_{\mathcal{R}[i]} := \overrightarrow{dl}^{served}_{\mathcal{R}[i]} + drive_i$. Now, $\overrightarrow{\mathcal{L}}^{driven}_{\mathcal{R}[i]}$ contains all significant states of a driver who has just arrived at the next customer.

6.3.3.4 Step *Setup*

Like step *Drive*, this step is only called if $i < |\mathcal{R}|$. For all $j \leq i$ we set

$$\overrightarrow{\mathcal{F}}^{setup}_{\mathcal{R}[i+1],j} := \overrightarrow{\mathcal{F}}^{driven}_{\mathcal{R}[i],j}$$

The pair $\overrightarrow{\mathcal{F}}^{setup}_{\mathcal{R}[i+1],i+1}$ represents the case that an early short break is taken at customer $\mathcal{R}[i+1]$. Hence, we set $\overrightarrow{ds}^{setup}_{\mathcal{R}[i+1],i+1} := 0$. To set $\overrightarrow{T}^{setup}_{\mathcal{R}[i+1],i+1}$, we need an intermediate step. First, we introduce a function $\overrightarrow{T}'_{\mathcal{R}[i+1],i+1}$. It is defined to be the minimum of all the $\overrightarrow{T}^{driven}_{\mathcal{R}[i],j}$ ($j \leq i$), shifted (to the right) and raised by the minimum break duration. That is, we set it as follows:

$$\overrightarrow{T}'_{\mathcal{R}[i+1],i+1}(t) := \min_{j \leq i} \overrightarrow{T}^{driven}_{\mathcal{R}[i],j}(t - break) + break$$

for all $t \in \mathcal{H}$ and $\overrightarrow{T}'_{\mathcal{R}[i+1],i+1}(t) := \perp$ otherwise, where we treat $\perp$ like ∞.

In order to guarantee that $\overrightarrow{T}^{setup}_{\mathcal{R}[i+1],i+1} - \mathrm{id}$ is monotonously decreasing (apart from the gaps where it is undefined), we do the following (treat $\perp$ like ∞):

$$\overrightarrow{T}^{setup}_{\mathcal{R}[i+1],i+1}(t) := \begin{cases} \perp, & \exists t' < t : \overrightarrow{T}'_{\mathcal{R}[i+1],i+1}(t') + t - t' \leq \overrightarrow{T}'_{\mathcal{R}[i+1],i+1}(t) \\ \overrightarrow{T}'_{\mathcal{R}[i+1],i+1}(t), & \text{otherwise} \end{cases}$$

As an example where this is necessary, let us have a look at Figure 4.4 on page 63 again. We see a red schedule with an early short break at the second customer, and a blue schedule with an (implicit) early break at the first customer. Hence, the red driver state is contained in $\overrightarrow{\mathcal{F}}^{driven}_{\mathcal{R}[3],2}$, whereas the blue driver state is contained in $\overrightarrow{\mathcal{F}}^{driven}_{\mathcal{R}[3],1}$. The latest time t for which $\overrightarrow{T}^{driven}_{\mathcal{R}[3],1}(t)$ is defined is $t = 10.5$. $\overrightarrow{T}^{setup}_{\mathcal{R}[4],4}(t)$ needs to be set to $\perp$ for every $t > 10.5 + \mathit{break}$ in this example because for every such t it is better to follow the blue schedule and wait than to take the red schedule.

This concludes the last of the four steps. But still there are potentially dominated driver states contained in $\overrightarrow{T}^{setup}_{\mathcal{R}[i+1],i+1}$. To remove them, we need the knowledge of the time windows of the next customer. So this better fits into step *Wait*. Therefore, we revise this step.

6.3.3.5 Step *Wait*, Revised

Beginning with the second iteration ($i > 1$), it is advisable to apply a dominance rule when we create $\overrightarrow{T}^{waited}_{\mathcal{R}[i],i}$ from $\overrightarrow{T}^{setup}_{\mathcal{R}[i],i}$. This is due to the fact that an early break can only be beneficial if it starts outside of a time window. Let t be a time for which $H_i^{shift}(t)$ is defined. By construction, $H_i^{shift}(t)$ is only defined within the time windows of customer $\mathcal{R}[i]$. If $H_i^{shift}(t) - \mathit{break}$ is inside a time window, there is no reason for an early break because that break could be taken later without any downside. Precisely, we set

$$\overrightarrow{T}^{waited}_{\mathcal{R}[i],i}(t) := \overrightarrow{T}^{setup}_{\mathcal{R}[i],i}(H_i^{shift}(t)) + (t - H_i^{shift}(t))$$

only for those t for which $H_i^{shift}(t)$ is defined and additionally $H_i^{shift}(t) - \mathit{break} \in \overrightarrow{\mathcal{W}}_{\mathcal{R}[i]}$ holds, where $\overrightarrow{\mathcal{W}}_{\mathcal{R}[i]}$ denotes the union of the forward waiting intervals of customer $\mathcal{R}[i]$. We set $\overrightarrow{T}^{waited}_{\mathcal{R}[i],i}(t) := \perp$ for all other t.

Finally, we conjecture (but do not prove) that no driver state dominates another driver state that is contained in the same label.

6.3.4 Backward Propagation

In the backward propagation phase, we proceed analogously to the forward case. We do not go into every detail of this phase but focus on the main differences. One of these differences is that we iterate backwards from $i = |\mathcal{R}|$ down to 1. However, the steps remain in the same order as before. Another difference is that whenever we shift pieces, we now shift them to the left and not to the right. Step *Wait* could be revised again but we omit the details here.

6.3.4.1 Step *Wait*

The major part of this step is the definition of the auxiliary function H_j^{shift}. For every $j \leq |\mathcal{R}| - i + 1$, we set

$$H_j^{shift}(t) := \min\{t' \mid t = t' - \overleftarrow{W}_{\mathcal{R}[i]}(t') \wedge \overleftarrow{T}^{setup}_{\mathcal{R}[i],j}(t') \neq \perp\}$$

where the minimum over the empty set is supposed to be $\perp$. $\overleftarrow{W}_{\mathcal{R}[i]}$ is the backward waiting time function of customer $\mathcal{R}[i]$ as introduced in section 6.3.2. Note that $H_j^{shift}(t)$ is undefined for any t with $t - service_{\mathcal{R}[i]}$ outside of the customer's time windows.

As in the forward case, let $\mathcal{Q}$ be a sequence of w_i periods of time. This time, $\mathcal{Q}[k]$ stores the best known travel time for the shifted end $\omega(\mathcal{W}_i^k) + service_{\mathcal{R}[i]}$ of the k-th time window for every k from 1 to w_i.

We set

$$\overleftarrow{T}^{waited}_{\mathcal{R}[i],j}(t) := \overleftarrow{T}^{setup}_{\mathcal{R}[i],j}(H_j^{shift}(t)) + (H_j^{shift}(t) - t)$$

for all t for which $H_j^{shift}(t)$ is defined, and $\overleftarrow{T}^{waited}_{\mathcal{R}[i],j}(t) := \perp$ for all other t. If $j < |\mathcal{R}| - i + 1$, $t = \omega(\mathcal{W}_i^k) + service_{\mathcal{R}[i]}$ for some k, and $\overleftarrow{T}^{waited}_{\mathcal{R}[i],j}(t)$ is not less than $\mathcal{Q}[k]$, then we reset $\overleftarrow{T}^{waited}_{\mathcal{R}[i],j}(t)$ to $\perp$.

6.3.4.2 Step *Serve*

For every $j \leq |\mathcal{R}| - i + 1$, we set

$$\overleftarrow{T}^{served}_{\mathcal{R}[i],j}(t) := \overleftarrow{T}^{waited}_{\mathcal{R}[i],j}(t + service_{\mathcal{R}[i]}) + service_{\mathcal{R}[i]}$$

for all t within the planning horizon, and $\overleftarrow{T}^{served}_{\mathcal{R}[i],j}(t) := \perp$ for all other points in time.

6.3.4.3 Step *Drive*

In this step, the driver drives from customer $\mathcal{R}[i]$ to $\mathcal{R}[i-1]$. With $drive_{i-1}$ as a short notation for $drive_{\mathcal{R}[i-1],\mathcal{R}[i]}$, we set for every $j \leq |\mathcal{R}| - i + 1$

$$h_j^{exc} := \overleftarrow{ds}^{served}_{\mathcal{R}[i],j} + drive_{i-1} - limitD_{short}$$

If $h_j^{exc} \leq 0$:

$$\overleftarrow{ds}^{driven}_{\mathcal{R}[i],j} := \overleftarrow{ds}^{served}_{\mathcal{R}[i],j} + drive_{i-1}$$
$$\overleftarrow{T}^{driven}_{\mathcal{R}[i],j}(t) := \overleftarrow{T}^{served}_{\mathcal{R}[i],j}(t + drive_{i-1}) + drive_{i-1}$$

However, if $h_j^{exc} > 0$, then we set (with $h_j^{\#} := \left\lceil h_j^{exc} / limitD_{short} \right\rceil$ defined as before):

$$\overleftarrow{ds}^{driven}_{\mathcal{R}[i],j} := h_j^{exc} - (h_j^{\#} - 1) \cdot limitD_{short}$$
$$\overleftarrow{T}^{driven}_{\mathcal{R}[i],j}(t) := \overleftarrow{T}^{served}_{\mathcal{R}[i],j}(t + (drive_{i-1} + h_j^{\#} \cdot break)) + (drive_{i-1} + h_j^{\#} \cdot break)$$

Finally, we set $\overleftarrow{dl}^{driven}_{\mathcal{R}[i]} := \overleftarrow{dl}^{served}_{\mathcal{R}[i]} + drive_{i-1}$. The driver has now arrived at the customer $\mathcal{R}[i-1]$.

6.3.4.4 Step *Setup*

For all $j \leq |\mathcal{R}| - i + 1$, we set

$$\overleftarrow{\mathcal{F}}^{setup}_{\mathcal{R}[i-1],j} := \overleftarrow{\mathcal{F}}^{driven}_{\mathcal{R}[i],j}$$

In the following, let k be $|\mathcal{R}| - (i-1) + 1$. To set $\overleftarrow{\mathcal{F}}^{setup}_{\mathcal{R}[i-1],k}$, we need an intermediate step. First, we introduce a function $\overleftarrow{T}'_{\mathcal{R}[i],j}$ and set it as follows:

$$\overleftarrow{T}'_{\mathcal{R}[i-1],k}(t) := \min_{j \leq |\mathcal{R}|-i+1} \overleftarrow{T}^{driven}_{\mathcal{R}[i],j}(t + break) + break$$

for all $t \in \mathcal{H}$ and $\overleftarrow{T}'_{\mathcal{R}[i-1],k}(t) := \perp$ otherwise, where we treat $\perp$ like ∞.

In order to guarantee that $\overleftarrow{T}^{setup}_{\mathcal{R}[i-1],k} + \mathrm{id}$ is monotonously increasing (apart from the gaps where it is undefined), we proceed analogously to before (treat $\perp$ like ∞):

$$\overleftarrow{T}^{setup}_{\mathcal{R}[i-1],k}(t) := \begin{cases} \perp, & \exists t' > t : \overleftarrow{T}'_{\mathcal{R}[i-1],k}(t') + t' - t \leq \overleftarrow{T}'_{\mathcal{R}[i-1],k}(t) \\ \overleftarrow{T}'_{\mathcal{R}[i-1],k}(t), & \text{otherwise} \end{cases}$$

Finally, we set $\overleftarrow{ds}^{setup}_{\mathcal{R}[i-1],k} := 0$.

6.3.5 Feasibility Check of a Neighboring Solution

We now turn towards Algorithm 6 again. The task is to efficiently check the feasibility of all neighbors of the incumbent solution with respect to the 2-opt* neighborhood. In the end, we are interested in a feasible neighboring solution that is evaluated best. In the following, we describe the feasibility and gain evaluation of a single move.

Let $\mathcal{R}'$ and $\mathcal{R}''$ be two different vehicle routes. Suppose we take a first part from $\mathcal{R}'$ and a second part from $\mathcal{R}''$, precisely we take the first x customers from $\mathcal{R}'$ and the last y customers from $\mathcal{R}''$ in order to create a new vehicle route $\mathcal{R}'''$. Without loss of generality, we assume that both x and y are not 0. We use the short notation $c' := \mathcal{R}'[x]$ and $c'' := \mathcal{R}''[|\mathcal{R}''| - y + 1]$ to denote the customers that we want to connect. Let $drive_{c',c''}$ be the driving time between them.

To find out whether $\mathcal{R}'''$ is feasible or not, we compare the labels $\overrightarrow{\mathcal{L}}^{served}_{c'}$ and $\overleftarrow{\mathcal{L}}^{served}_{c''}$. If $\overrightarrow{dl}^{served}_{c'} + drive_{c',c''} + \overleftarrow{dl}^{served}_{c''}$ exceeds the total driving time limit $limitD_{long}$, $\mathcal{R}'''$ is infeasible. So we only continue if this is not the case. Then, the next step is to do the following for every index pair (j,k), where j is an index between 1 and x and k is an index between 1 and y. For a given index pair (j,k), we first compute the number $h^{\#}_{j,k}$ of due breaks en route:

$$h^{\#}_{j,k} := \left\lceil (\overrightarrow{ds}^{served}_{c',j} + drive_{c',c''} + \overleftarrow{ds}^{served}_{c'',k}) / limitD_{short} \right\rceil - 1$$

For instance, if $\overrightarrow{ds}^{served}_{c',j} + drive_{c',c''} + \overleftarrow{ds}^{served}_{c'',k} \leq limitD_{short}$, then no break en route is necessary. With this number, we can then calculate the minimum travel time of $\mathcal{R}'''$

with respect to the index pair (j,k) as the minimum of $\overrightarrow{T}^{served}_{c',j}(t) + \overleftarrow{T}^{served}_{c'',k}(t') + (t' - t)$ over all t and t' with $t + drive_{c',c''} + h^{\#}_{j,k} \cdot break \leq t'$. The travel time functions may be undefined for some t but we treat $\perp$ just like ∞ here. For a given t, the point in time t' is quickly determined as the earliest point in time that is not earlier than $t + drive_{c',c''} + h^{\#}_{j,k} \cdot break$ and for which $\overleftarrow{T}^{served}_{c'',k}(t')$ is defined.

After we have calculated the minimum travel time of $\mathcal{R}'''$ with respect to every index pair (j,k), the (overall) minimum travel time of $\mathcal{R}'''$ is the minimum over all j from 1 to x and all k from 1 to y. If it is undefined, then $\mathcal{R}'''$ is infeasible. If not, we finally check whether the minimum travel time of $\mathcal{R}'''$ exceeds the maximum allowed travel time $limitT_{long}$ or not.

Let the input of a checking routine be two labels $\overrightarrow{\mathcal{L}}^{served}_{c'}$ and $\overleftarrow{\mathcal{L}}^{served}_{c''}$, let $size(\mathcal{L})$ denote the size of the label $\mathcal{L}$, that is, the number of pieces of information contained therein, and let $nodes(\mathcal{L})$ denote the length of the contained sequence of tuples (corresponds to x or y here). Then the time it takes to evaluate a move is in $O(size(\overrightarrow{\mathcal{L}}^{served}_{c'}) \cdot nodes(\overleftarrow{\mathcal{L}}^{served}_{c''}) + size(\overleftarrow{\mathcal{L}}^{served}_{c''}) \cdot nodes(\overrightarrow{\mathcal{L}}^{served}_{c'}))$. We conjecture – but do not prove – that the size of a label is polynomial in the number of customers in the route and the number of their time windows. Note that the scheduling problem at hand is a subproblem of the problem treated in chapter 4, which was proven to be solvable in polynomial time.

Checking a neighboring solution means to do the above for two new links. That is, we do so not only for (c', c'') but also for $(\mathcal{R}'[x+1], \mathcal{R}''[|\mathcal{R}''| - y])$, unless $x = |\mathcal{R}'|$ or $y = |\mathcal{R}''|$. In order to evaluate the gain of the move, we sum up the minimum travel times of the two output routes and subtract the minimum travel times of the input routes. In the special case $x = |\mathcal{R}'|$ and $y = |\mathcal{R}''|$, the number of routes in the solution is decreased by one. Recall that minimizing the number of routes in the solution is the primary optimization goal.

6.4 Conclusion and Outlook

As the VRTDSP with multiple time windows is a highly complex problem, we have focused on local search based algorithms to solve it. We have shown how to propagate travel time information forward and backward. When such information is pre-computed, the concatenation of two partial routes can be evaluated efficiently. 2-opt* is an example of a neighborhood where this technique is particularly useful. However, bidirectional labeling can also be advantageous for other neighborhoods, such as the *insert* or the *swap* neighborhood in which one customer is moved from one route to another or two customers are exchanged between their routes, respectively. Our approach works for any setting of the break parameters (for instance, we could allow 10 hours of driving between two long breaks instead of only 9 hours) and with the general break policy that allows breaks en route at any time, unlike the approach of Schiffer et al. (2017). We conjecture that it takes polynomial time (in the number of customers and the number of time windows) to find the best move in the 2-opt* neighborhood.

For the sake of completeness, we have presented a very simple local search based algorithm to solve the VRTDSP. But our approach for the route evaluation could as well be integrated into the far more sophisticated solution framework of Vidal et al. (2014) by implementing the `Eval2` function. At PTV, we have incorporated the presented ideas into the move evaluation of the vehicle routing service of PTV

xServer. Here, we have even implemented an enhanced variant that works with two types of breaks under the no-long-break-en-route policy (see section 5.6).

In the European Union, there is a relaxing rule that allows to take a break in two parts. In this chapter, we have ignored this rule, even though an extension of the presented approach to the ruleset {*drive until driven*, *first-second-split*} is not overly difficult (see section 3.4).

We have proposed an exact scheduling method that always finds a feasible schedule if one exists and that returns the minimum travel time. However, the overall problem, the VRTDSP, is typically not solved to proven optimality in practice. Therefore, one direction for future research is to assess the pros and cons of a heuristic scheduling approach that no longer guarantees optimality of the subproblem. As long as the returned solution is feasible, it would suffice to have a method that does not always find a feasible solution if one exists and does not necessarily output the minimum travel time if it finds one. In particular, if the second optimization criterion was not the minimum total travel time but the minimum total distance, then the adverse effects of the non-optimality may be acceptable and outweighed by a shorter run-time.

Minimum Service Cost Objective We have considered a constraint on the maximum travel time of each vehicle route and the total travel time of all routes as secondary minimization goal. This is common both in theory and in practice. But sometimes, there are other cost drivers that should also be taken into account. To this end, let us suppose that the length of the planning horizon is at most $limitT_{long}$. In this scenario, we do not need to keep track of the minimum travel time accumulated since the last long break. Instead, we could focus a little more on actual costs.

So far, we have only considered hard time window constraints. Ibaraki et al. (2005) propose *general* time windows as a generalization of hard and even soft time windows. This means we are given a time-dependent service cost function that maps a time t to the cost that a service entails when it starts at time t. Hard time windows are a special case in which the costs are either 0 or infinite. With such a service cost function, we can assign a cost value between 0 and ∞ to a service that begins rather late or early. Even priorities of time windows can be expressed. In practice, this is very useful when perishable groceries need to be delivered. Let us take a delivery to a supermarket or a (fast food) restaurant as an example. The groceries need to be accepted and handled by the staff of the supermarket or the restaurant. Hence, the service cost of the delivery may be higher the more crowded the store or restaurant is and the busier the staff is with other things at that time.

In our algorithm, we could store the accumulated service cost since the end of the last long break instead of the accumulated travel time since then. With such a function, we could optimize the service level over all customers. Hashimoto, Yagiura, and Ibaraki (2008) even regard a time-dependent driving time function and a time-dependent driving cost function for every pair of customers in addition to the time-dependent service costs. Time-dependent driving costs could also be integrated into our approach – but only as long as the driving times remain time-independent.

Chapter 7

Truck Driver Routing on Time-Dependent Road Networks

We study the problem of computing time-dependent shortest routes for truck drivers. In contrast to conventional route planning, truck drivers have to obey government regulations that impose limits on non-stop driving times. Therefore, route planners must plan *break periods* in advance and select suitable parking lots. To ensure that maximum driving times are not exceeded, predictable congestion due to, e.g., peak hours should also be taken into account. Therefore, we introduce the truck driver routing problem in *time-dependent* road networks. It turns out that the combination of time-dependent driving times with constraints imposed by drivers' working hours requires computation of multiple time-dependent *profiles* for optimal solutions. Although conceptually simple, profile search is expensive. We greatly reduce (empirical) running times by calculating bounds on arrival and departure times during additional search phases to only query partial profiles and only to a fraction of the parking lots. Carefully integrating this approach with a *one-to-many* extension of time-dependent contraction hierarchies makes our approach practical. For even faster queries, we also propose a heuristic variant that works very well in practice. Excellent performance of our algorithms is demonstrated on a recent real-world instance of Germany that is much harder than time-dependent instances considered in previous works.

Note This chapter has been published before. Apart from a few minor changes, it is an exact quote of the paper by Kleff et al. (2017). As such, it is self-contained and does not require the knowledge of previous chapters.

7.1 Introduction

In many countries of the world, truck drivers are legally obligated to take breaks on a regular basis to obviate drivers' fatigue and hence increase road safety. For instance, Regulation (EC) No. 561/2006 of the European Union (European Parliament and Council of the European Union, 2006) demands a break of at least 45 minutes after at most 4.5 hours of driving. And according to the hours-of-service regulation in the United States (Federal Motor Carrier Safety Administration, 2011), a 30-minutes-break is mandatory after at most eight hours have elapsed. Truck drivers must park their vehicle at a suitable location before taking such a "lunch break". Due to the size of their trucks, the drivers are severely limited compared to car drivers when in search of a parking space. For assistance in finding appropriate and available parking

lots, truck drivers use mobile apps like Truck Parking Europe[1] that maintain databases of parking lots and display nearby lots to users. In this work, we investigate the following optimization problem: En route from one customer to another, when and where should the driver take a break (if at all) to conform to the provisions on breaks and arrive at the destination earliest possible?

We only consider one drive from a source to a destination. In general, a truck driver may visit multiple customers per day. In this case, the customers' time windows also have to be regarded. Moreover, if a trip takes more than one day, not only lunch breaks have to be scheduled but also longer rest breaks for the driver to sleep. The problem of scheduling breaks in order to comply with regulations while also taking customer time windows into account is known as the *truck driver scheduling problem* (Goel, 2010). However, the locations of the parking lots remain disregarded in this setting. In this work, we take a major first step towards combining *time-dependent route planning* and *truck driver scheduling*. We determine not only *when* but also *where* to take a break.

We consider time-dependent driving times to model predictable congestion. In this scenario, it might be beneficial to not depart from source right away, or to prolong a break, or to wait at a parking lot for a time that is too short to count as break. As an example of *short-term waiting*, imagine the following: At the time of arrival at a parking lot, the driving time to the destination would be two minutes longer than the remaining allowed driving time. Luckily, the driver just has to wait ten minutes for the congestion to disperse and for the driving time to drop by these two minutes. In contrast to the European Union, short-term waiting does not pay off in the United States because the lunch break becomes mandatory after eight hours have elapsed, and not after a certain accumulated driving time. In the following, we focus on the EU regulation.

Time-dependency makes the problem particularly challenging, and the question arises whether it can be solved efficiently in practice. We are interested both in optimal and in heuristic approaches. There are a couple of parameters to reduce the run-time, and we seek to shed light on their impact on the solution quality. For our experimental analysis, it is sufficient to assume that the driver stops at only one parking lot (if at all). For a planning horizon of one day, this is no substantial limitation in practice as a daily driving time of 9h (US: 11h) should not be exceeded, even though it may be extended to 10h twice a week. For the sake of completeness, we discuss the implications regarding multiple stops.

Related Work Route planning algorithms have received a large amount of attention in recent years, resulting in a multitude of *speedup techniques* (Bast et al., 2016; Sommer, 2014). In the *time-dependent* scenario, there are *driving time functions* associated with the edges. These map the time of the day to a driving time (Cooke and Halsey, 1966). Dijkstra's algorithm (Dijkstra, 1959; Dantzig, 1963) can be generalized (Dreyfus, 1969) to answer *earliest arrival* (EA) queries. However, *profile* queries asking for the driving time function between two vertices are not feasible for large road networks (Delling and Wagner, 2009). The reason is that profiles may have superpolynomial complexity (Foschini, Hershberger, and Suri, 2014), and maintaining them for all vertices makes Dijkstra's algorithm impractical.

Several classic speedup techniques have been generalized to the time-dependent scenario (Baum et al., 2016; Delling, 2011; Delling and Nannicini, 2012), typically focusing on fast EA queries. Efficient EA and profile queries at continental scale

[1] `https://truckparkingeurope.com/`

are provided by TCH (Batz et al., 2013), a generalization of Contraction Hierarchies (CH) (Geisberger et al., 2012). Batched shorted paths in the time-dependent scenario are studied by Geisberger and Sanders (2010). Recently, Strasser (2016) introduced a simple heuristic for time-dependent routing that is cheap in time and space, but drops optimality and provides no approximation guarantees.

As far as the truck driver scheduling problem is concerned, the interested reader can find descriptions of optimal algorithms for the EU variant of this problem in the papers of Goel (2010), Drexl and Prescott-Gagnon (2010), and Kok, Hans, and Schutten (2011) and for the US variant in those of Goel (2014), Koç et al. (2016), and Koç, Jabali, and Laporte (2017). Of these authors, Kok, Hans, and Schutten (2011) and Koç et al. (2016) propose a *mixed integer linear programming* formulation. The former even takes time-dependent driving times into account, the latter is the only one to include real-world data of parking lots (here: interstate rest areas) into their experimental analysis. However, in both cases not only the sequence of customers is fixed but also the path in the road graph. So in the former case the path cannot change over time, and in the latter case truck stops aside the path are disregarded. In the master thesis of Shah (2008), time-dependent routes for truck drivers subject to government regulations and time windows are solved heuristically. Finally, other lines of research have considered problems that resemble our setting but are *NP*-hard, such as crew scheduling (Smith, Boland, and Waterer, 2012), routing of electric vehicles (Baum et al., 2015), or time-dependent pollution-routing (Franceschetti et al., 2013).

Contribution and Outline We introduce the truck driver routing problem that asks for the fastest route between two customers that complies with legal provisions for truck drivers (section 7.2). To the best of our knowledge, we are the first to integrate the choice of routes, breaks and parking lots in one query – unlike previous works that first fix the route and then schedule breaks, possibly missing the optimal solution. Since rush hours severely affect driving times, we consider the time-dependent scenario. We propose a naive approach (section 7.3) that would be far too expensive in time and space without at least one of two described acceleration techniques (sections 7.3.1 and 7.3.2): An implementation based on TCHs achieves query times in the order of minutes on the German road network. Sophisticated bounds computations on top of that speed queries up by a factor of 25, yielding running times well below 10 seconds. Finally, a heuristic approach (section 7.3.3) enables queries below a second and less. Most of our experiments (section 7.4) are performed on a new instance of the German road network, currently used by PTV in production systems. It turns out to be much harder than the ten-year-old instance used in most publications so far. Before we conclude (section 7.6), we discuss the implications of allowing multiple stops (section 7.5).

7.2 Problem Statement and Preliminaries

The basic input for every variant of the *truck driver routing problem* is the following: Let a road network be given, modeled as a directed *graph* $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where *vertices* $v \in \mathcal{V}$ typically correspond to intersections and *edges* $(u, v) \in \mathcal{E}$ to road segments. The subset $\mathcal{P} \subset \mathcal{V}$ of the vertices contains exactly the *parking locations* that represent the parking lots (or even parking spaces) where the driver may take a break. The *minimum break period* and the *maximum driving time* until such a break is mandatory are denoted by *break* and *limit* respectively. These two parameters are sufficient to

handle the Regulation (EC) No. 561/2006 of the European Union (European Parliament and Council of the European Union, 2006) for a planning horizon of one day.

We are also given a sequence of exactly two customers to be visited, *source* $s \in \mathcal{V} \setminus \mathcal{P}$ and *destination* $d \in \mathcal{V} \setminus \mathcal{P}$. An *s–d-path* $Path_{s,d}$ (in $\mathcal{G}$) is a sequence $[v_1 = s, v_2, \ldots, v_k = d]$ of vertices such that $(v_i, v_{i+1}) \in \mathcal{E}$ and $v_i \neq v_j$ for all $1 \leq i < j \leq k$. A *(truck driver) route* $\mathcal{R}_{s,d}$ from s to d in turn is a sequence $[Path_{u_i,v_i}]_{1 \leq i \leq k}$ of paths such that $u_1 = s$ and $u_i \in \mathcal{P}$ for $1 < i \leq k$, $v_k = d$ and $v_i \in \mathcal{P}$ for $1 \leq i < k$, and $v_{i-1} = u_i$ for $1 < i \leq k$. In this chapter, we will focus on routes with a sequence length $|\mathcal{R}_{s,d}| := k$ of at most two. The general case is only discussed in section 7.5.

In the time-independent case, the weights on the edges are constants and indicate the driving time along the edge. A path is feasible iff the accumulated driving time along the path is no longer than *limit*, and a route is feasible iff all its paths are. The duration of a truck driver route $\mathcal{R}_{s,d}$ is the sum of the accumulated driving times of its paths plus $(|\mathcal{R}_{s,d}| - 1) \cdot break$. In time-independent truck driver routing, we are interested in a shortest feasible route from s to d if such a feasible route exists.

In time-dependent truck driver routing, we are given *time-dependent driving time functions* for every edge instead of constant driving times. That is, for every edge (u, v) there is a function $\Psi_{u,v} \colon \mathbb{R} \to \mathbb{R}^+$ that maps the time of departure from u to the driving time to v. In this work, all functions are supposed to be piecewise-linear. The driver is not allowed to wait at any vertex other than the parking locations or s. In this scenario, it is common to demand that the functions fulfill the *FIFO property* because the shortest-path problem would become *NP*-hard if it was not satisfied for all edges (Sherali, Ozbay, and Subramanian, 1998; Dean, 2004). We even presume that functions are continuous and fulfill the *strict FIFO property*, i.e., for arbitrary $t < t' \in \mathbb{R}$, the condition $t + \Psi(t) < t' + \Psi(t')$ holds for every edge (later departure leads to later arrival). This way, the *arrival time function* $\mathrm{id} + \Psi$ is bijective (id being the identity function) and we can build the inverse $(\mathrm{id} + \Psi)^{-1}$ that maps an arrival time to the appropriate departure time.

To check feasibility of a route $\mathcal{R}_{s,d} = [Path_{u_i,v_i}]_{1 \leq i \leq k}$, we also ask for *departure and arrival times* $dep(u_i)$ and $arr(v_i)$ for all i. This way, the duration of a path $Path_{u,v}$ can easily be computed by $arr(v) - dep(u)$ (must be positive) and the waiting time at a parking location by $dep(u_i) - arr(v_{i-1})$ (must be non-negative). To be feasible, no single path is allowed to be longer than *limit*. In addition, a route $\left[Path_{s,p}, Path_{p,d}\right]$ is feasible only if either the sum of the paths' durations does not exceed *limit* or there is a waiting time that counts as break at the parking location p in between. Among all feasible truck driver routes we look for one with the earliest arrival at d. To this end, we are also given a *lower bound* on the *earliest departure* $lbED(s)$ from s, i.e., we demand $dep(s) \geq lbED(s)$. It is only a lower bound because a feasible route with $dep(s) = lbED(s)$ may not exist. In this chapter, we call a vertex v *reachable* from u at time t if there is a feasible route $\mathcal{R}_{u,v}$ with $dep(u) = t$.

A *(driving time) profile* between u and v is a time-dependent function $\psi_{u,v} \colon \mathbb{R} \to \mathbb{R}^+$ that maps every departure time at u to the *shortest* driving time to v. If $(u, v) \in \mathcal{E}$, the profile is identical to the given driving time function $\Psi_{u,v}$. If not, we can compute the profile $\psi_{u,v}$ recursively either forward or backward using the *link operation* $\odot$ and the *merge operation* $\oplus$:

$$\psi_{u,v} := \bigoplus_{w \colon (w,v) \in \mathcal{E}} \psi_{u,w} \odot \Psi_{w,v} \qquad \text{or} \qquad \psi_{u,v} := \bigoplus_{w \colon (u,w) \in \mathcal{E}} \Psi_{u,w} \odot \psi_{w,v} \tag{7.1}$$

where $\psi \odot \varphi$ is defined to be $\psi + \varphi \circ (\mathrm{id} + \psi)$ and $\psi \oplus \varphi$ defined to be $\min(\psi, \varphi)$. A profile search can be implemented as described by Delling and Wagner (2009).

7.3 Solution Approach

We first describe a basic and rather naive approach to compute the earliest arrival at destination d. There are three ways in which d may be reachable from s: Either without passing a parking location at all, or by taking a break at a parking location, or by short-term waiting at a parking location. Accordingly, we will now compute three values opt_{none}, opt_{break}, and opt_{short}. The minimum of these is then the overall optimal solution.

At first, we investigate whether d can be reached from s without passing a parking location. To do this, we compute the driving time profile $\psi_{s,d}$ from s to d and then look up the earliest feasible departure time $dep_{s,d}$ from s in this respect: $dep_{s,d} := \min\{t : \psi_{s,d}(t) \leq limit \wedge t \geq lbED(s)\}$. With this, we can conclude: $opt_{none} := dep_{s,d} + \psi_{s,d}(dep_{s,d})$.

To consider the parking locations, we have to search forward and backward in order to compute the driving time profiles $\psi_{s,p}$ and $\psi_{p,d}$ for all $p \in \mathcal{P}$. In the case with a break at a parking location, the next step is, similarly as before and for every parking location $p \in \mathcal{P}$, to determine the earliest feasible departure time $dep_{s,p}$ from s when going to p as $dep_{s,p} := \min\{t : \psi_{s,p}(t) \leq limit \wedge t \geq lbED(s)\}$, and then to look up the earliest feasible departure time $dep_{p,d}$ from p when going to d after a break as $dep_{p,d} := \min\{t : \psi_{p,d}(t) \leq limit \wedge t \geq dep_{s,p} + \psi_{s,p}(dep_{s,p}) + break\}$. In turn, we can conclude: $opt_{break} := \min_{p \in \mathcal{P}}\{dep_{p,d} + \psi_{p,d}(dep_{p,d})\}$.

But maybe the optimal solution consists in just waiting at a parking location for a short time that does not necessarily count as break. To take this case into account, we determine the earliest feasible departure time $dep_{s,p,d}$ from p when going from s to d for every parking location $p \in \mathcal{P}$ as follows: $dep_{s,p,d} := \min\{t : \exists t' : \psi_{s,p}(t') + \psi_{p,d}(t) \leq limit \wedge lbED(s) \leq t' \leq t - \psi_{s,p}(t')\}$. Again, we conclude: $opt_{short} := \min_{p \in \mathcal{P}}\{dep_{s,p,d} + \psi_{p,d}(dep_{s,p,d})\}$.

This description is only a sketch. It is meant to give an overview. A naive implementation would certainly be far too slow for any practical use. This motivates the following three acceleration approaches: by narrowing down profile searches, by time-dependent contraction hierarchies, and heuristically.

7.3.1 Acceleration by Narrowing Down Searches

Some computations can be performed faster than others. The idea is to spend little extra time on quick computations in order to gain bounds that help us speed up the expensive calculations such as the profile search.

We define $ubMax(\psi)$ as an upper bound on the maximum value of the profile ψ, i.e., $ubMax(\psi) \geq \max_{t \in \mathbb{R}} \psi(t)$. Analogously, $lbMin(\psi)$ is a lower bound on the minimum value of ψ. A query for these bounds, called a *profile bounds query* here, can be answered by applying Dijkstra's algorithm (Dreyfus, 1969) on a graph where the constant edge weights are the minimum (maximum) values of the respective driving time functions. Given a departure time t in addition, an earliest arrival (EA) query asks for the earliest arrival at d when departing at time t. Both queries can be processed rapidly and are described in greater detail by Batz et al. (2013). In our context, a usual EA query only gives a lower bound $lbEA(d)$ on the earliest arrival if $lbEA(d) > t + limit$. To highlight this, we call it an *lbEA query*.

Computing Partial Profiles. One of the key acceleration techniques in this chapter is to only compute a *partial profile*. A partial profile maps a departure time $t \in \mathbb{R}$ to a driving time in $\mathbb{R}^+ \cup \{\perp\}$, where $\perp$ can be read as *undefined*. We have to distinguish a *partial forward profile* from a *partial backward profile*. More precisely, the following holds for a partial forward profile ψ^f given a *departure time range* $[t^{begin}, t^{end}] \subset \mathbb{R}$: $\psi^f(t) \in \mathbb{R}^+$ for $t^{begin} \leq t \leq t^{end}$ and $\psi^f(t) = \perp$ otherwise. An analog statement holds for a partial backward profile ψ^b given an *arrival time range* $[t^{begin}, t^{end}] \subset \mathbb{R}$: $\psi^b(t) \in \mathbb{R}^+$ for $(\mathrm{id} + \psi^b)^{-1}(t^{begin}) \leq t \leq (\mathrm{id} + \psi^b)^{-1}(t^{end})$ and $\psi^b(t) = \perp$ otherwise.

A partial (forward or backward) profile for a given (departure or arrival time) range can be computed similar to before. If $(u, v) \in \mathcal{E}$, we set

$$\psi^f_{u,v}(t) := \begin{cases} \Psi_{u,v}(t), & \text{if } t \text{ in range} \\ \perp, & \text{otherwise} \end{cases} \qquad \psi^b_{u,v}(t) := \begin{cases} \Psi_{u,v}(t), & \text{if } t + \Psi_{u,v}(t) \text{ in range} \\ \perp, & \text{otherwise} \end{cases}$$

If not, we use the same (forward or backward) recursion formula as before in (7.1). But we have to adjust the definitions of the link and merge operations and distinguish the forward from the backward case. The forward and backward link operations for a partial profile and a driving-time function of some edge are now defined as follows:

$$(\psi^f_{u,v} \odot^f \Psi_{v,w})(t) := \begin{cases} \psi^f_{u,v}(t) + \Psi_{v,w}(t + \psi^f_{u,v}(t)), & \text{if } \psi^f_{u,v}(t) \neq \perp \\ \perp, & \text{otherwise} \end{cases}$$

$$(\Psi_{u,v} \odot^b \psi^b_{v,w})(t) := \begin{cases} \Psi_{u,v}(t) + \psi^b_{v,w}(t + \Psi_{u,v}(t)), & \text{if } \psi^b_{v,w}(t + \Psi_{u,v}(t)) \neq \perp \\ \perp, & \text{otherwise} \end{cases}$$

The forward and backward merge operations for two partial profiles for the same vertex pair and range are now defined as follows:

$$(\psi^f \oplus^f \varphi^f)(t) := \begin{cases} \min\{\psi^f(t), \varphi^f(t)\}, & \text{if } \psi^f(t) \neq \perp \wedge \varphi^f(t) \neq \perp \\ \perp, & \text{otherwise} \end{cases}$$

$$(\psi^b \oplus^b \varphi^b)(t) := \begin{cases} \psi^b(t), & \text{if } \psi^b(t) \neq \perp \wedge (\mathrm{id} + \varphi^b)^{-1}(t + \psi^b(t)) \leq t \\ \varphi^b(t), & \text{if } \varphi^b(t) \neq \perp \wedge (\mathrm{id} + \psi^b)^{-1}(t + \varphi^b(t)) < t \\ \perp, & \text{otherwise} \end{cases}$$

Given a source, a destination, and a range, we call a query for a partial profile a *profile range query*.

One-to-one queries. At first, we perform a *one-to-one* lbEA query for s and d and departure time $lbED(s)$, that is, we compute the earliest arrival at d as if there was no break to take when leaving s earliest possible. If this lower bound $lbEA(d)$ on the earliest arrival is no later than $lbED(s) + limit$, it is tight, and we have found the requested earliest arrival at d.

The second step is to compute a lower bound $lbMin(\psi_{s,d})$ on the driving time from s to d. If this bound is already greater than $2 \cdot limit$, we stop here because d is considered to be not reachable from s as we only take one break into account.

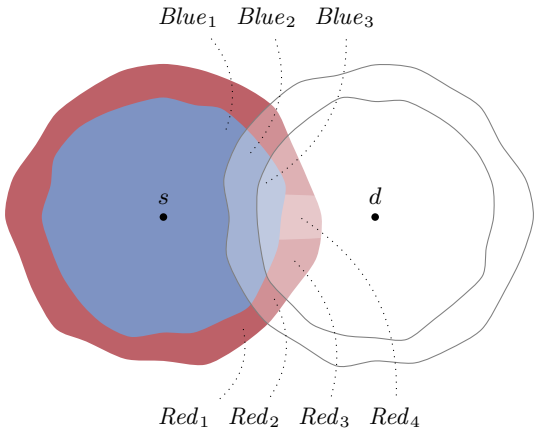


FIGURE 7.1: The set sequences $Blue_1 \supset Blue_2 \supset Blue_3$ and $Red_1 \supset Red_2 \supset Red_3 \supset Red_4$. The two sets $Blue_1$ and Red_1 are disjoint.

If $lbMin(\psi_{s,d}) \leq limit$, then an optimal solution may incorporate short-term waiting at a parking location. We store this information by setting $lbWaiting := 0$. Otherwise we set $lbWaiting := break$ because it is certain that the driver will have to take a break at one of the parking locations.

One-to-many-to-one queries. We perform both an lbEA search and a profile bounds search from s to all potentially reachable parking locations, that is, we compute $lbMin(\psi_{s,p})$, $ubMax(\psi_{s,p})$ and $lbEA(p)$ for all $p \in \mathcal{P}$ with $lbMin(\psi_{s,p}) \leq limit$. We insert all those parking locations p with $lbEA(p) \leq lbED(s) + limit$ into a set $Blue_1$. So for all $p \in Blue_1$, the lower bound $lbEA(p)$ is tight and equals the earliest arrival $EA(p)$ at p. We add the other potentially reachable parking locations p, i.e. with $lbEA(p) > lbED(s) + limit$ (and also $lbMin(\psi_{s,p}) \leq limit$ by construction), to a set Red_1. These are the ones for which the lower bound is known to be not tight. The set Red_1 may remain empty, especially if waiting at s was not allowed. An empty set Red_1 helps to speed up computation as we can omit the forward profile range query later. If both sets are empty, there is no feasible solution. $Blue_1$ and Red_1 are each the first element of a sequence of subsets of $\mathcal{P}$ that we will construct in the following. An illustration is shown in Figure 7.1.

The next step is to conduct a profile bounds search from d backwards to all potentially reachable parking locations in $Blue_1 \cup Red_1$, i.e., we compute $lbMin(\psi_{p,d})$ and $ubMax(\psi_{p,d})$ for all $p \in Blue_1 \cup Red_1$ with $lbMin(\psi_{p,d}) \leq limit$. Let $Blue_2$ (resp. Red_2) be the subset of parking locations in $Blue_1$ (resp. Red_1) that are also potentially reachable backwards. Again, if $Blue_2 \cup Red_2$ is empty, there is no feasible solution. With the bounds on the driving time we get (better) bounds on the earliest arrival at d. We can set the upper bound $ubEA(d)$ to $\min\{EA(p) + break + ubMax(\psi_{p,d}) : p \in Blue_2 \wedge ubMax(\psi_{p,d}) \leq limit\}$, where the minimum over the empty set is considered to be infinite. If $lbWaiting = break$ and improving, we can update the lower bound $lbEA(d)$ to $\min\{lbEA(p) + break + lbMin(\psi_{p,d}) : p \in Blue_2 \cup Red_2\}$.

A profile range search backwards from d in the range $[lbEA(d), ubEA(d)]$ to all $p \in Blue_2 \cup Red_2$ yields a partial profile $\psi_{p,d}$ for all these p. It is defined for exactly those departure times t from p for which $t + \psi_{p,d}(t) \in [lbEA(d), ubEA(d)]$ holds. For all $p \in Blue_2$ we can now determine an upper bound $ubED(p)$ on the earliest departure from p as the earliest point in time t such that $t \geq EA(p) + break$ and $\psi_{p,d}(t) \leq limit$. In case $lbWaiting = break$, this bound is tight. In the other case, we may be able to improve it by the earliest point in time t for which $t \geq EA(p)$ and

$\psi_{p,d}(t) \leq \mathit{limit} - (EA(p) - \mathit{lbED}(s))$ holds. However, we might not be able to find such an upper bound because neither of the conditions are met. So let $\mathit{Blue}_3 \subset \mathit{Blue}_2$ be the set of parking locations for which $\mathit{ubED}(p)$ can be determined. Then, we may improve the upper bound $\mathit{ubEA}(d)$ on the earliest arrival at d by $\min\{\mathit{ubED}(p) + \psi_{p,d}(\mathit{ubED}(p)) : p \in \mathit{Blue}_3\}$.

On the other hand, we calculate a lower bound $\mathit{lbED}(p)$ on the earliest departure from p for all $p \in \mathit{Red}_2$ as the earliest point in time t with $t \geq \mathit{lbEA}(p) + \mathit{break}$ and $\psi_{p,d}(t) \leq \mathit{limit}$. If $\mathit{lbWaiting} = 0$, we may have to lower this bound to the earliest point in time t with $t \geq \mathit{lbEA}(p)$ and $\psi_{p,d}(t) \leq \mathit{limit} - \mathit{lbMin}(\psi_{s,p})$. And, again, let $\mathit{Red}_3 \subset \mathit{Red}_2$ be the set of parking locations for which $\mathit{lbED}(p)$ can be determined.

Let $\mathit{Red}_4 \subset \mathit{Red}_3$ be the set of parking locations p for which the inequality $\mathit{lbED}(p) + \psi_{p,d}(\mathit{lbED}(p)) < \mathit{ubEA}(d)$ holds. So Red_4 contains those parking locations for which a forward profile range search is inevitable. If this set is empty and $\mathit{lbWaiting} = \mathit{break}$, then $\mathit{ubEA}(d)$ is tight, so we are done. If not, we need to compute an upper bound $\mathit{ubED}(p)$ on the departure time from p for all $p \in \mathit{Red}_4$ (and $p \in \mathit{Blue}_2$ if $\mathit{lbWaiting} = 0$): It is the point in time t with $t + \psi_{p,d}(t) = \mathit{ubEA}(d)$. With the upper bound for all p, we can obtain an upper bound $\mathit{ubED}(s)$ on the departure from s: It is $\max\{\mathit{ubED}(p) - \mathit{lbWaiting} - \mathit{lbMin}(\psi_{s,p})\}$ over all $p \in \mathit{Red}_4$ (and $p \in \mathit{Blue}_2$ if $\mathit{lbWaiting} = 0$).

Finally, we conduct a forward profile range search from s to all $p \in \mathit{Red}_4$ (and $p \in \mathit{Blue}_2$ if $\mathit{lbWaiting} = 0$) for the departure time range $[\mathit{lbED}(s), \mathit{ubED}(s)]$. Now we have everything we need together: In case $\mathit{lbWaiting} = \mathit{break}$, we compute $\mathit{opt}_{\mathit{break}}$ similar to before, except that the earliest arrival at d via the parking locations in Blue_3 is already known and has to be determined only for Red_4. In case $\mathit{lbWaiting} = 0$, we have to compute $\mathit{opt}_{\mathit{short}}$ in addition, but only for $\mathit{Blue}_2 \cup \mathit{Red}_4$, and also $\mathit{opt}_{\mathit{none}}$ (provided that waiting at s is allowed). To speed up the computation of $\mathit{opt}_{\mathit{none}}$, we only perform a forward profile range search from s to d for the range $[\mathit{lbED}(s), \mathit{ubEA}(d) - \mathit{lbMin}(\psi_{s,d})]$.

7.3.2 Acceleration by Contraction Hierarchies

In the previous section, we proposed techniques to reduce the number of profile searches and restrict the remaining profile searches to smaller ranges. We accelerate our approach even further by speeding up the profile searches (and EA queries) themselves using *time-dependent contraction hierarchies* (Batz et al., 2013). (T)CHs were originally proposed for point-to-point queries, whereas we also need to compute a variant of one-to-many queries (from a source vertex to all parking lots). In this section we recap the (time-dependent) contraction hierarchies algorithm and describe our modifications of it.

A contraction hierarchy (CH) (Geisberger et al., 2012) is built by *contracting* the vertices of a graph in increasing order of importance. Intuitively, vertices that lie on many shortest paths (such as vertices on highways) are considered important. To contract a vertex v, it is (temporarily) removed from the graph, and *shortcuts* are added between its neighbors in order to preserve distances in the remaining graph. *Witness searches* are performed to determine whether a shortcut is necessary or can be discarded. For each pair of neighbors u, w with $(u, v) \in \mathcal{E}, (v, w) \in \mathcal{E}$, we run a Dijkstra search from u to w. Only if the path via v is the *unique* shortest u–w-path, we add the shortcut between u and w. In the time-dependent case, we need to run a profile search from u to w. A shortcut can only be omitted if it is not needed *at any point in time*.

CH queries are a modified variant of bidirectional Dijkstra, where both forward and reverse search relax only upward edges, i.e., edges going from less to more important vertices. In the time-dependent scenario, the reverse search is particularly difficult, because the time of arrival at the target is unknown. In a basic query variant, the reverse search only marks all edges in the reverse search space from d, and the forward search is allowed to additionally relax all marked arcs. More sophisticated query variants compute bounds during the reverse search that guide the forward search into the direction of d.

The obvious approach to compute EA queries or profiles from a source to all parking lots $\mathcal{P}$ runs $|\mathcal{P}|$ point-to-point TCH queries. However, we can do better with the following modification. During the contraction process, we *block* all vertices representing parking lots, i.e., we disallow to contract them. After contraction, there remains a *core graph* at the top of the hierarchy, consisting of all parking lots and (shortcut) arcs between them. Queries from a source s to all parking lots now boil down to a forward search from s that relaxes no edges to less important vertices. As long as the query has not yet reached the core, it behaves like a normal forward CH search. On the core graph, it behaves like a standard Dijkstra search. We can accelerate the search using the *stall-on-demand* optimization (Geisberger et al., 2012) and stop it as soon as all parking lot vertices are settled, or a certain time limit is reached. Since blocking arbitrary vertices can lead to suboptimal contraction orders, we do not contract all vertices but the parking lots, but rather stop contraction as soon as the remaining graph becomes too dense.

7.3.3 Heuristic Acceleration

In our study, we schedule waiting times on the assumption that the time-dependent driving times are deterministic. This is not the case in real-life. So it is questionable whether a route with, for instance, scheduled short-term waiting would be acceptable in practice. This is the motivation for the *restricted waiting policy* that disallows waiting at s, short-term waiting at any parking location, and the prolongation of a break. To conform to this policy, the driver must depart immediately at time $lbED(s)$ and may take a break of exactly 45 minutes if inevitable. In this scenario, it is not necessary to query any profiles, even if d cannot be reached directly without break. Then, the *Red* sets are ignored, and instead of computing partial profiles backwards from d to $Blue_2$, we conduct multiple *lbEA* searches forward from the parking locations in $Blue_2$, getting a better and better upper bound on the earliest arrival at d.

7.4 Experiments

In this section, we first describe the data and the test setup and then analyze runtime and solution quality of the described approaches. Our experiments are based on two versions of the road network of Germany with time-dependent driving time functions, see Table 7.1. The older network from the year 2006 has been used by several other studies related to time-dependent routing (see section 7.1) and contains car driving times based on a traffic model. The very recent data from 2017 is quite different: The new data is more detailed with respect to time dependency, there are more edges with driving time functions that are not constant, and the total number of breakpoints representing the functions is larger. The driving times are based on

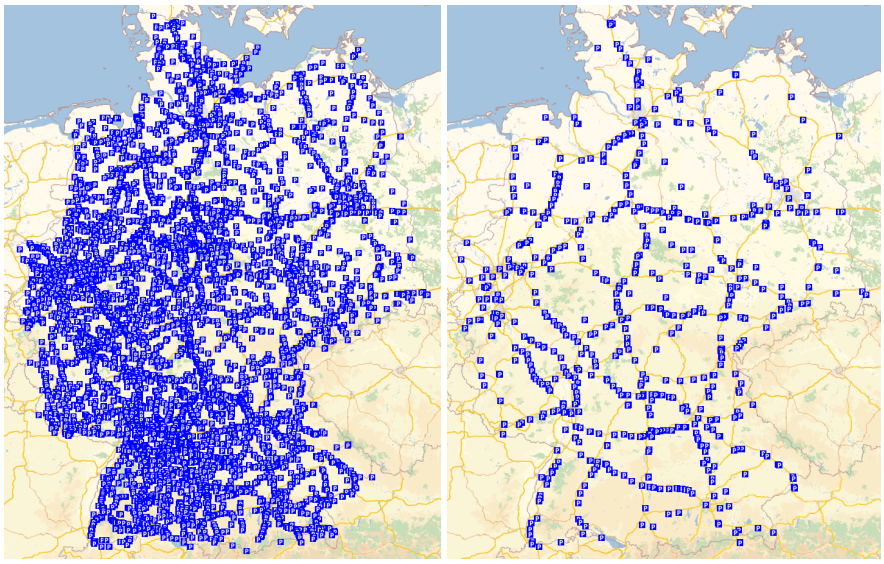


FIGURE 7.2: The left image shows all available parking lots in Germany, the right image shows the reduced set with only big parking lots.

historic data provided by TomTom[2] which is post-processed by PTV such that it models truck driving times.

TABLE 7.1: Key figures of the input data used for the experiments. TD Edges denotes the relative number of edges with a time-dependent and not constant driving time function.

Road network	Vertices	Edges	TD Edges	Breakpoints	Parking set / subset
Germany 2017	7.2 M	15.7 M	28.6 %	136.9 M	6 596 / 759
Germany 2006	5.1 M	12.6 M	3.7 %	20.9 M	6 447 / 731

We use the database of PTV Group's Truck Parking Europe app[3]. It contains currently more than 25 000 parking lots all over Europe. Some parking lots cannot be linked to the old road network of 2006. Therefore, the number of parking locations is a bit lower than in the road network of 2017. The database does not only contain rest areas with fuel stations, restrooms, and restaurants but also parking areas without any facilities. It is not clear if or under what circumstances the choice of a parking area without facilities would be acceptable in practice. We will take this into account by also testing our algorithm with a smaller subset of parking lots that offer 30 parking bays or more each. Figure 7.2 shows these two sets of parking lots.

Test Setup. We run our experiments on a VMware ESX cluster. Our machine has four cores of a 2.2 GHz Intel Xeon E5-2698 v4, 64 GB main memory, and runs Ubuntu 16.04. Besides the construction of the contraction hierarchies the algorithms use only one core. Our code is written in C++ and compiled with gcc 5.4, optimization level -O3. Our CH implementation is based on the code by Batz (Batz et al., 2013; *KaTCH*) and has been extended as described in section 7.3. We set the size of the CH cores to 0.2 % of the vertices in case of the whole parking set and 0.02% in case of the subset.

[2] `https://www.tomtom.com/`

[3] `https://truckparkingeurope.com/`

TABLE 7.2: Number of truck driver route queries per category.

	C1	C2	C3	C4	C5	Over all
Query set 2017	4278	210	4943	165	404	10000
Query subset 2017	877	36	980	31	76	2000
Query set 2006	7109	126	2754	1	10	10000

TABLE 7.3: Mean run-time per category in seconds for different scenarios.

Scenario	C1	C2	C3	C4	C5	Over all
Default scenario	0.0038	18.1756	5.9549	121.9516	0.0053	5.3392
Restricted waiting	0.0033	0.2925	0.2187	0.1163	0.0910	0.1212
Parking subset	0.0041	5.8109	1.0646	7.8424	0.0057	0.7796
Naive approach	2.8018	287.1991	227.5335	228.4150	195.4254	128.8562
Query subset 2017	0.0039	18.5811	5.8160	121.5858	0.0056	5.0708
Germany 2006	0.0013	0.9829	0.3932	23.8170	0.0021	0.1239

This results in a CH search graph size of 38.90 GB in the former and 37.28 GB in the latter case (and 2.03 GB in the case of the 2006 road network).

Since our test data is the road network of Germany, we consider the EU regulation, i.e., *break*=45 min and *limit*=4.5 h. We generate 10 000 truck driver route queries for both versions of the road network. To this end, we randomly select vertices s and d and (a lower bound on) the earliest departure from s between 6 am and 9 am. Since the run-time of these queries can differ a lot, we assign each of them to one of five categories: Category C1 comprises the queries for which the *lbEA* query suffices, i.e., $lbEA(d) \leq lbED(s) + limit$. Category C2 contains the ones with $lbEA(d) > lbED(s) + limit$ and $lbMin(\psi_{s,d}) \leq limit$, category C3 the ones with $lbMin(\psi_{s,d}) > limit$ and $ubMax(\psi_{s,d}) \leq 2 \cdot limit$, and category C4 the ones with $ubMax(\psi_{s,d}) > 2 \cdot limit$ and $lbMin(\psi_{s,d}) \leq 2 \cdot limit$. Finally, category C5 holds the instances with $lbMin(\psi_{s,d}) > 2 \cdot limit$ that cannot be solved.

Table 7.2 lists how these queries are distributed among the five categories. In case of the *query set 2006*, there are far more queries in C1 because with car driving times the vehicle's range is larger. Also in this list is the *query subset 2017*. We need this smaller subset of queries to measure the run-time of the long running naive approach.

Results on Run-Time. Table 7.3 shows the mean run-time for different scenarios, broken down into the five categories. The categories themselves are not part of the input of the algorithm. For the *default scenario*, we use the 2017 road graph, all described acceleration techniques, all parking locations, and allow waiting of any duration. The other scenarios deviate from this in one aspect each. In the default scenario, the run-time varies a lot with the category. A query from category C4 takes more than 30 000 times longer than one from C1. Since there are far more queries in C1 than in C4, the mean run-time over all 10 000 queries is still less than 6 seconds. Queries from C4 take so long because in 106 cases no upper bound on the earliest arrival at d can be determined, so a full backward profile search is necessary. Figure 7.3 illustrates the run-time distribution among the 10 000 queries.

In case of the *restricted waiting* policy, waiting at s is not allowed and waiting at any parking location is only allowed if the waiting time equals exactly the time for

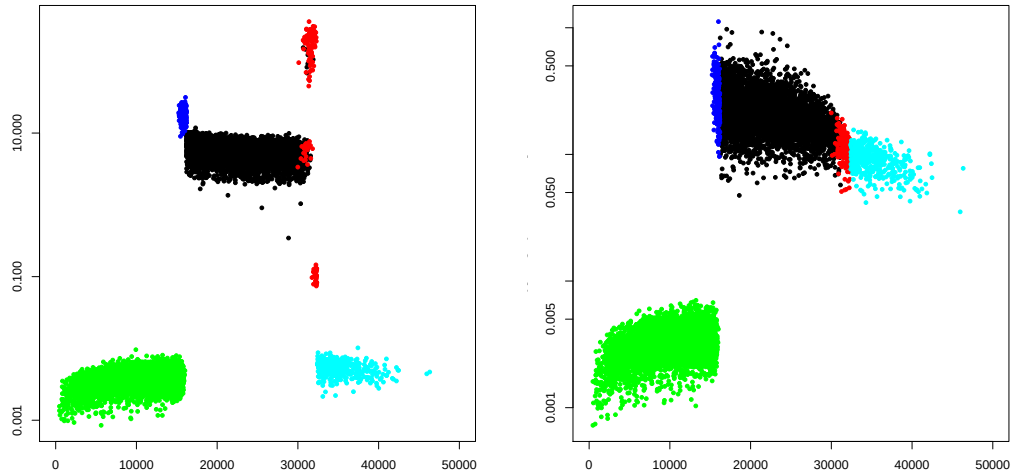


FIGURE 7.3: Run-time of each *s*-*d*-query in the default scenario (left) and according to restricted waiting policy (right), $lbMin(\psi_{s,d})$ on abscissa and run-time in seconds on ordinate (on a logarithmic scale). Points are colored by category. Scales differ.

a break. This speeds the calculation up by a factor of 40 over all queries. In the *parking subset* scenario, we allow waiting only at larger parking lots. We run the algorithm on the Germany 2017 network but the search graphs differ. Compared to the default setting, the smaller size of the core graph leads to faster one-to-many profile range queries (approx. by a factor of 7.5) but slower one-to-one profile range queries (approx. by a factor of 1.2). In both of these scenarios, not all queries can be solved. Solution quality is discussed later.

The *naive approach* does not make use of the acceleration based on partial profiles as described in section 7.3.1 but still CH as in section 7.3.2. Because of the long run-time of the naive approach, the run-times are based on the reduced query subset 2017 (see Table 7.2). For better comparability, we also give the run-times of the default scenario for the reduced query subset. An achieved speed-up of 25 over all queries proves the effectiveness of our described acceleration in general. The main aspect of it is the computation of only partial profiles that concerns category C3 primarily. Here, we even achieve a speed-up of almost 40.

In case of the *Germany 2006*, we run the accelerated approach on the 2006 road graph that was used in the original TCH publication (Batz et al., 2013). The run-time is smaller by an order of magnitude compared to our recent data.

Some more numbers are of interest. A crucial issue of our bounds-based acceleration is to find a (good) upper bound $ubEA(d)$. In the default scenario, there are 113 cases in which such a bound cannot be determined and so a complete profile needs to be searched for backwards. A complete profile search backwards takes 138.7 s on average. In contrast, a profile range search is performed in 5168 cases and takes 5.8 s on average. The mean length of these ranges, i.e. $ubEA(d) - lbEA(d)$, is 604 s. A second important aspect of the acceleration is to avoid the profile (range) search forward if the set Red_4 is empty. This set contains elements only in 50 cases and then only a few, most often just one. Figure 7.4 shows a sample query with empty set Red_4.

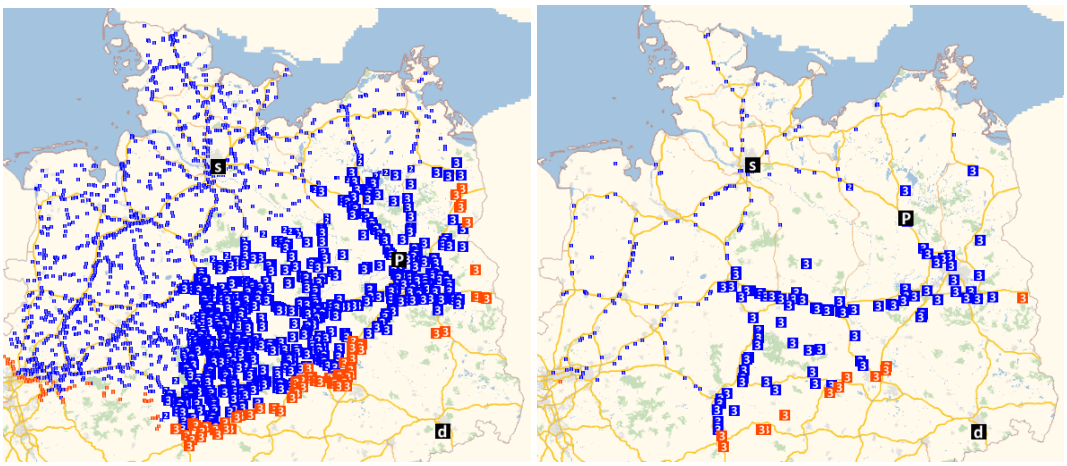


FIGURE 7.4: Sample query from Hamburg to Dresden in the default scenario (left) and in the parking subset scenario (right). Different parking lots (P) are selected. The largest squares represent the sets $Blue_3$ and Red_3.

Results on Quality. Table 7.4 compares the solution quality of the default scenario to the *restricted waiting* and the *parking subset* scenario. The results of the *naive approach* are identical to the default, and the results of *Germany 2006* are hardly comparable, particularly since the driving times in this setting are based on a car model.

In the default setting, 9558 of 10 000 queries can be solved. We observe that the travel time, i.e., the driving time plus all waiting time (at s and at parking), exceeds 15 hours in some cases, presumably to exploit the short driving times during the night. Such a solution is feasible according to our problem statement but most likely it would neither be acceptable in practice nor legal as truck drivers have to take a sleep rest daily. In the following, we call a solved query legal if the travel time does not exceed 15 hours. In case of the restricted waiting policy, a solved query is always legal.

We also state how many queries are solved (legally and) optimally, i.e., how often is the calculated earliest arrival at d identical to the default scenario. In the parking subset scenario, this happens in 58% of the cases, even though there are less than 12% of the parking lots in the subset. Parking lots with more than 30 parking bays are most often located right next to a freeway (Autobahn in Germany), whereas many of the small parking lots are further away from it. In the restricted waiting scenario, only 21 of the solved queries are not solved optimally. So in the vast majority of the cases, the computational effort spent on taking waiting of any duration into account does not pay off. For instance, short-term waiting is scheduled only 11 times in the default scenario.

TABLE 7.4: Comparison of solution quality for different scenarios. Mean and maximum deviation is in seconds over all queries that are legal but not optimal.

Scenario	solved	legal	optimal & legal	mean dev	max dev
Default scenario	9558	9512	9512	0	0
Restricted waiting	9474	9474	9453	1211	2265
Parking subset	9518	9470	5518	127	17559

7.5 Enhancement to Multiple Stops

Our algorithm is tailored to the one-stop case. What are the implications if we allow more than one stop? For instance, if there were two drivers on board, they could take turns and stop three times for a change before they must take a rest and sleep. From a conceptual perspective, the multi-stop case is not too difficult. Let $\mathcal{P}_s$ be the parking locations that are reachable from s at some point in time without taking a break along the path, and let $\mathcal{P}_d$ be the parking locations that are potentially reachable backwards from d, i.e., $lbMin(\psi_{p_d,d}) \leq limit$ for all $p_d \in \mathcal{P}_d$. Moreover, suppose we had precomputed a $|\mathcal{P}| \times |\mathcal{P}|$ matrix M of travel time profiles such that for two parking locations p_s and p_d, $M[p_s, p_d]$ maps the departure time from p_s (where the driver is expected to have taken a break) to the shortest travel time to p_d, including as many breaks as needed and also one at p_d (unless $p_s = p_d$). With this, a truck driver route query boils down to three steps: First, we compute the earliest arrival at every $p_s \in \mathcal{P}_s$. Then, we determine the earliest departure from every $p_d \in \mathcal{P}_d$ with the help of M as follows:

$$ED(p_d) = \min_{p_s \in \mathcal{P}_s} EA(p_s) + break + M[p_s, p_d](EA(p_s) + break)$$

Having done that, we can finally calculate the earliest arrival at d, also checking if d could be reached without any break.

We could easily adapt the restricted waiting policy heuristic to this general case. It is short-term waiting that makes the computation of the earliest arrival at every $p_s \in \mathcal{P}_s$ challenging. In order to do so, we could propagate a time-dependent function forward (here: mapping an arrival time to the minimum accumulated driving time). But as we have seen, propagating a time-dependent function is expensive. So from a practical point of view, it would be important to again find ways of narrowing down the search, like finding good bounds and only propagating partial functions as we have demonstrated before. In addition to this challenge, our assumption that we have a matrix M in memory is not realistic. Due to the superpolynomial complexity of the travel time profiles, we would most likely need hundreds of GB of main memory for the parking lots in Germany. So the question is raised what a good trade-off would be between memory consumption and computational effort (and solution quality).

7.6 Conclusion and Outlook

We have introduced the truck driver routing problem and described an exact algorithm for it. While a naive approach would be far too costly in time and space, it can be made feasible using our two proposed acceleration methods. One is a modification of TCH. Additionally narrowing down TCH searches by several fast bounds computations and queries of only partial profiles results in an extra speed-up of 25 and practical run-times. We have also suggested a heuristic based on the policy of restricted waiting and analyzed its effect. In this setting, truck driver route queries take well below one second without losing too much solution quality. Similarly effective is the restriction of the parking set to the more relevant parking locations.

In this chapter, we have left out our experiments with approximated driving time functions. Using the algorithm of Imai and Iri (1987) to approximate the functions of both original and shortcut edges further reduces the run-time, especially of profile (range) queries. In doing so, we only sacrifice a precision that is not justified in

practice. Future work includes a solution to the combined truck driver routing and scheduling problem for a given sequence of customers by using the results of this chapter as a building block. Moreover, it would be interesting to reevaluate the existing work on algorithms for time-dependent route planning on the new benchmark instance. We conjecture that other shortcut-based methods such as TD-CRP (Baum et al., 2016) also suffer significantly from the new instance. It could be promising to further investigate shortcut-free approaches like the ALT algorithm (Delling and Nannicini, 2012).

Our algorithm will also be evaluated in the EU research projects AEOLIX and Clusters 2.0.

Chapter 8

Truck Driver Scheduling and Routing on Time-Dependent Road Networks

In the *truck driver scheduling problem*, a truck driver needs to visit some customers in a given order and tries to find a schedule such that not only every customer is visited within one of the customer's time windows but also the regulations on breaks are respected. For instance, the European Union stipulates a break of (at least) 45 minutes after (at most) 4.5 hours of driving. In the *truck driver scheduling and routing problem*, also the underlying road network and dedicated parking locations within the network are considered. Here, one is not only interested in a schedule but also in a route in the network from the first to the last customer via the other customers and (if need be) some parking locations. In our study, we regard the time-dependent scenario in which the driving time along a segment of the network depends on the time of day. So the optimal route (and schedule) may vary over time. In this setting, the objective is to find a route and a corresponding schedule such that the customers are visited in time, the regulations on breaks are respected, and the finish time of the route is earliest possible within a given planning horizon.

We present the first exact algorithm for the truck driver scheduling and routing problem. Especially in the time-dependent scenario, an efficient implementation is crucial due to the complexity of driving time profiles. Hence, we describe acceleration techniques. These are based on the ideas already presented in chapter 7. In addition, we introduce a heuristic that is at least two orders of magnitude faster than the exact approach (despite our acceleration efforts) and still does not sacrifice much of the quality of the exact approach. We evaluate our algorithms by letting both randomly generated and real-world queries run on a recent network instance of Germany.

8.1 Introduction

After investigating the *truck driver scheduling problem* in chapters 3 through 5 and the *truck driver routing problem* in chapter 7, we now turn towards the combined *truck driver scheduling and routing problem*. In this problem, we are given a road graph. Some of the vertices in the graph represent customers that have to be served by a truck driver in a given order and within certain time windows. Some other vertices in the graph represent parking locations where the driver is able to park the truck in order to take a break. A break may become due according to some rules that are specific for a certain variant of the truck driver scheduling and routing problem. The objective is to find a route in the graph that connects the customers in the given

order and a schedule within a given planning horizon that respects the customers' time windows as well as the rules on breaks. When a break is scheduled, the driver must be at a customer or at a parking location. At any other location, the driver cannot stop. Among all feasible schedules, we look for one for which the finish time, that is, the time when the final customer is served, is earliest possible.

A simple example of a road graph with parking locations is given in Figure 8.1. Suppose the driving time between consecutive customers is 1 time unit each, the customers' time windows are non-restrictive, and a break becomes due after slightly less than 2 time units of driving. One feasible solution is to take exactly one break at the parking location p_5. It is better than taking a break at p_2 because the detour via p_5 is shorter than the detour via p_2. Another feasible solution is to take a break both at customers c_2 and c_3. In case an additional break takes less time than the detour via p_5, this is the best solution.

In general, several rules may apply to the same type of break. In this chapter, we consider only one break rule. In the European Union, a truck driver may no longer drive after accumulating 4.5 hours of driving since the last break unless he takes another break of at least 45 minutes (European Parliament and Council of the European Union, 2006). We call this the *drive until driven* rule or simply the EU rule. For the sake of completeness, we mention that a driver is allowed to take the break in two parts, a first split break of at least 15 minutes and a second split break of at least 30 minutes. However, this remains disregarded in the following.

In the United States, a similar rule exists. Here, a truck driver may no longer drive after 8 hours have elapsed since the end of the last break unless he takes another break of at least 30 minutes (Federal Motor Carrier Safety Administration, 2011). We call this the *drive until traveled* rule or the US rule. The focus of this chapter is on the EU rule but we shortly discuss the differences between the two problem variants and also necessary changes to our algorithm later (section 8.3.2.6). Both in the EU and the US, also the total driving time per day is restricted (EU: 9 hours, US: 11 hours, see section 2.1). This is ignored in the main part of this chapter. Just like with the US rule, we discuss the implications of this additional constraint towards the end of this chapter (section 8.7).

Besides regulations on breaks, we also take predictable congestion in the road network into account. Typically, the driving times are highest during the rush hours in the morning and the early evening and lowest during the night. To model this, the weight of a graph's edge is supposed to be a function over time that maps the departure time from the tail of the edge to the time it takes to traverse it.

With time-dependent driving times, one of the differences between the EU and the US rule becomes apparent. In the EU, *short-term waiting*, that is, waiting for less than the minimum break duration, may pay off. For instance, it may be favorable to stop at a parking location for only a short period of time when the remaining driving time to the next customer decreases over time. In the US, a break becomes due after a certain time has elapsed and thus voluntary waiting has no advantage. This is at least true if, from two drivers who take the same road(s), the one who departs later also arrives later. This property is called the strict FIFO property (first-in-first-out).

In this chapter, we assume that the strict FIFO property holds for the driving time functions of all edges. If the FIFO property was not satisfied for all such functions, the time-dependent truck driver scheduling and routing problem would become *NP*-hard (Sherali, Ozbay, and Subramanian, 1998; Dean, 2004).

In the following, short-term waiting at parking locations is not taken account of. If a driver stops at such a location, we require him to stay at least for the minimum break duration. However, we do consider a prolongation of that break. We disallow

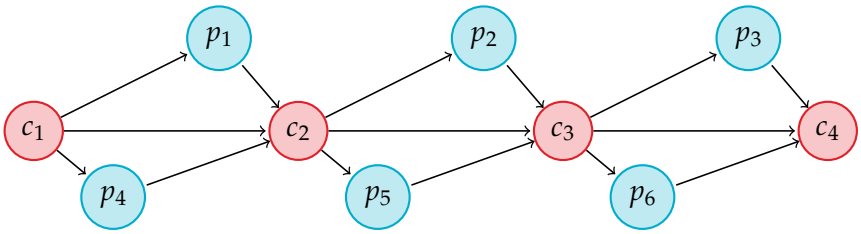


FIGURE 8.1: Example road graph with parking locations.

short-term waiting en route for two reasons: One is that, considering the uncertainty of traffic forecasts, a scheduled short-term waiting period at a parking location is most likely not wanted in practice. Another is that it is shown in chapter 7 that it is of little benefit.

As in chapter 7, we solely focus on at most one parking location en route between consecutive customers. This is not a limitation in practice for a planning horizon of less than a day. The implications of an enhancement to multiple stops are discussed in the previous chapter (section 7.5).

In this chapter, we present both an exact approach and a heuristic for the described variant of the truck driver scheduling and routing problem on time-dependent road networks. Analogously to section 7.3.1 from the previous chapter, we describe how to accelerate the exact approach by computing bounds first. With the heuristic, we aim to find a solution with a good trade-off between run-time and quality. Especially, we want to know what quality we can expect when we only allow a run-time of less than a second on average. Both algorithms are evaluated on the same road graph of Germany as in the previous chapter 7 and with regard to both random and real-world test queries.

Related work To the best of our knowledge, the truck driver scheduling and routing problem has not been studied before, except from the master theses of Shah (2008) and Kinz (2016), and the bachelor thesis of Bräuer (2016). Shah (2008) presents a heuristic for a slightly different variant of the time-dependent truck driver scheduling and routing problem in which breaks can be scheduled at every vertex in the graph. Kinz (2016) studies a time-independent variant of the truck driver scheduling and routing problem without the limitation to a single parking location between consecutive customers. Bräuer (2016) describes an exact approach for the same truck driver scheduling and routing problem as in this chapter and also considers the time-dependent scenario. This chapter builds upon that work and enhances it in different directions.

Related to our problem at hand are the problems studied by Kok, Hans, and Schutten (2011) and Koç et al. (2016). In both papers, a mixed integer linear programming formulation for the truck driver scheduling problem is proposed. Kok, Hans, and Schutten (2011) even regard the time-dependent variant of the problem. Koç et al. (2016) include real-world data of interstate rest areas into their experimental analysis. However, in both papers, the route in the road graph between two customers or between a customer and a rest area is fixed. Likewise related are the electric vehicle routing problem with truck driver scheduling (Schiffer et al., 2017) and the traveling salesman problem with multiple time windows and hotel selection (Baltz et al., 2015). In these two problems, one seeks to find an optimal order in which the customers are to be visited. This makes these problems *NP*-hard in contrast to

(at least the time-independent variant of) our problem at hand (see section 8.3.3 on complexity).

Tuin, Weerdt, and Batz (2018) investigate the truck driver routing problem with truck driving bans. In their scenario, the driving time on a road segment does not change over time whereas the travel time does. This is due to the driving bans. In Germany, for instance, driving a truck is forbidden on every Sunday between midnight and 10 pm. The authors do not consider explicit parking locations. Instead, it is assumed that waiting is possible everywhere and at any time.

Outline First, we give a formal definition of the present variant of the truck driver scheduling and routing problem in section 8.2. Then, we show how to solve it exactly and how to implement the algorithm efficiently. The presentation of the exact approach is spread over two sections. While the focus is on the inherent scheduling problem in section 8.3, the focus of section 8.4 is on the inherent routing problem. Subsequently, the heuristic is described in section 8.5. In section 8.6, we show the results of the experiments on random test queries and on some real-world test queries, also comparing the exact approach with the heuristic. A problem enhancement is shortly discussed in section 8.7. Finally, in section 8.8, a conclusion is presented and an outlook on future work is given.

8.2 Problem Definition

Given is a sequence $\mathcal{C} = [c_1, \ldots, c_n]$ of n different *customers*. Everyone of these requests a certain service that takes $service_i$ time and has to begin within one of the w_i *time windows* in the sequence $[\mathcal{W}_i^1, \ldots, \mathcal{W}_i^{w_i}]$ for any i between 1 and n. The service is performed by a truck driver who has to comply with provisions on breaks when visiting the customers in the given order. In addition, the driver only operates within a time interval $\mathcal{H}$, called *planning horizon*. W.l.o.g. the driver is assumed to be located at the first customer when the planning horizon begins.

Also given is a *road graph* $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V}$ denotes the set of *vertices* and $\mathcal{E}$ the set of *edges*. All customers in $\mathcal{C}$ are contained in $\mathcal{V}$. Some of the vertices in $\mathcal{V} \setminus \mathcal{C}$ constitute the set $\mathcal{P}$ of *parking locations* that correspond to those areas where the truck driver may park the vehicle in order to take a break. For every edge (u, v), there is a corresponding driving time that may vary over the time of day. In general, we call a function that maps the time of departure from a vertex u to the shortest driving time in $\mathcal{G}$ to some other vertex v the *(driving time) profile* of (u, v). For every edge $(u, v) \in \mathcal{E}$, a profile $P_{u,v} \colon \mathbb{R} \to \mathbb{R}^+$ belongs to the input. It is supposed to be given as a piecewise linear and continuous function that respects the strict FIFO property. For a vertex pair $(u, v) \notin \mathcal{E}$, we can derive the profile $P_{u,v}$ recursively either forward or backward using the *link operation* $\odot$ and the *merge operation* $\oplus$ (unless v is not reachable from u in $\mathcal{G}$):

$$P_{u,v} := \bigoplus_{w\colon (w,v)\in\mathcal{E}} P_{u,w} \odot P_{w,v} \qquad \text{or} \qquad P_{u,v} := \bigoplus_{w\colon (u,w)\in\mathcal{E}} P_{u,w} \odot P_{w,v} \tag{8.1}$$

where $P_{u,w} \odot P_{w,v}$ is defined to be $P_{u,w} + P_{w,v} \circ (\mathrm{id} + P_{u,w})$ and $P_{u,w} \oplus P_{w,v}$ defined to be the minimum of the two functions. And id denotes the identity function with $\mathrm{id}(t) = t$ for all t.

A *truck driver route* $\mathcal{R}$ for a sequence $\mathcal{C} = [c_1, \ldots, c_n]$ of n customers is a sequence $[(u_k, v_k)]_{1 \le k \le \ell}$ of $n - 1 \le \ell \le 2(n-1)$ vertex pairs, called *route segments*,

such that every v_k is reachable from u_k in $\mathcal{G}$ and the following conditions hold:

$$u_1 = c_1 \tag{8.2}$$
$$u_k = c_i \Rightarrow v_k = c_{i+1} \vee (v_k \in \mathcal{P} \wedge v_{k+1} = c_{i+1}) \quad \text{for all } k < \ell, i < n \tag{8.3}$$
$$u_{k+1} = v_k \quad \text{for all } k < \ell \tag{8.4}$$
$$u_\ell = c_i \Rightarrow v_\ell = c_{i+1} \quad \text{for all } i < n \tag{8.5}$$
$$v_\ell = c_n \tag{8.6}$$

Here, constraint (8.3) makes use of the restriction to at most one stop en route between two customers. Note that while we demand all customers to be different, the same parking location may be visited multiple times.

A *truck driver schedule* for a truck driver route $\mathcal{R} = [(u_k, v_k)]_{1 \le k \le \ell}$ is a triple $(\mathcal{D}, \mathcal{A}, \mathcal{B})$ of three sequences of points in time: For all $k \le \ell$, let $\mathcal{D}[k]$ be the scheduled *departure time* from vertex u_k, and $\mathcal{A}[k]$ be the scheduled *arrival time* at vertex v_k. For all $i \le n$, $\mathcal{B}[i]$ denotes the scheduled *start time* of the service at customer i. A truck driver schedule is *feasible* only if the following basic conditions hold:

$$\alpha(\mathcal{H}) \le \mathcal{B}[1] \le \mathcal{D}[1] - \mathit{service}_1 \tag{8.7}$$
$$\mathcal{B}[i] \in \mathcal{W}_i \quad \text{for all } i \le n \tag{8.8}$$
$$\mathcal{D}[k] + P_{u_k, v_k}(\mathcal{D}[k]) \le \mathcal{A}[k] \quad \text{for all } k \le \ell \tag{8.9}$$
$$v_k = c_i \Rightarrow \mathcal{A}[k] \le \mathcal{B}[i] \le \mathcal{D}[k+1] - \mathit{service}_i \quad \text{for all } k < \ell, 1 < i < n \tag{8.10}$$
$$v_k \in \mathcal{P} \Rightarrow \mathcal{A}[k] \le \mathcal{D}[k+1] - \mathit{break} \quad \text{for all } k < \ell \tag{8.11}$$
$$\mathcal{A}[n] \le \mathcal{B}[n] \le \omega(\mathcal{H}) - \mathit{service}_n \tag{8.12}$$

where *break* denotes the *minimum break period*, $\alpha(\mathcal{H})$ the beginning of the planning horizon, $\omega(\mathcal{H})$ its end, and $\mathcal{W}_i$ the multi-interval of the time windows of customer i (recall sections 2.4.1 and 2.4.3). According to (8.11), we demand that the driver stays at a parking location at least as long as *break*.

In addition, we need to specify when a break becomes due. In this chapter, we focus on the rule *drive until driven*, i.e., after the driver has *driven* for a while, he is no longer allowed to *drive*, unless he takes a break of length *break*. Let *limit* be the *maximum allowed accumulated driving time without break*. After a break of at least *break* time, he is considered as completely rested and may drive again until he has *driven* for another *limit* time at most.

Before we give a complete definition of a feasible truck driver schedule, we introduce the notion of a *break index*. Let a break index be an index $k < \ell$ such that for all $i \le n$

$$v_k = c_i \Rightarrow \mathcal{A}[k] + \mathit{break} \le \mathcal{B}[i] \vee \mathcal{B}[i] \le \mathcal{D}[k+1] - \mathit{service}_i - \mathit{break} \tag{8.13}$$

holds (compare (8.10)). That is, either v_k is a parking location, or there is enough buffer for a break at the customer at which the driver arrives at the end of the k-th route segment $\mathcal{R}[k]$.

So a truck driver schedule is feasible only if the driving time of every route segment does not exceed *limit*. If the total driving time of several consecutive route segments does exceed *limit*, then there must be a break scheduled in between, that is, there must be a parking location in between or a customer that has enough buffer for a break. In other words, a truck driver schedule is feasible if and only if the basic

conditions and the following two additional constraints hold:

$$\mathcal{A}[k] - \mathcal{D}[k] \leq \mathit{limit} \qquad \text{for all } k \leq \ell \tag{8.14}$$

$$\sum_{k=j'}^{j} \mathcal{A}[k] - \mathcal{D}[k] > \mathit{limit} \Rightarrow \exists k : j' \leq k < j \wedge k \text{ is a break index} \qquad \text{for all } j' < j, j \leq \ell \tag{8.15}$$

Now we define the *truck driver scheduling and routing problem* to be the problem of finding both a feasible truck driver schedule and a corresponding truck driver route for a given customer sequence - or the information that there is no feasible schedule. And among all feasible truck driver schedules, we prefer one with the earliest finish time, that is, the earliest time when the service at the final customer can be completed.

Problem variant Even though we concentrate on the rule *drive until driven* in this chapter, we also define the problem variant with regard to the rule *drive until traveled* for the sake of completeness. The only thing we need to do is to replace constraint (8.15) with the following:

$$\mathcal{A}[j] - \mathcal{D}[j'] > \mathit{limit} \Rightarrow \exists k : j' \leq k < j \wedge k \text{ is a break index} \qquad \text{for all } j' < j, j \leq \ell \tag{8.16}$$

8.3 Scheduling Part of the Exact Approach

In this section, we focus on the inherent scheduling problem and do not want routing issues to distract us. To this end, we disregard the underlying road graph and instead suppose that we are given an oracle that is able to tell us the driving time profile for any pair of vertices. This assumption will then be dropped in the next section 8.4.

The scheduling approach resembles the ones presented in previous chapters. Again, there are n iteration cycles, so as many iterations as there are customers. And every iteration is partitioned into steps. This time, we distinguish five of them: *SetupBeforeService*, *Wait*, *Serve*, *SetupAfterService*, and *Drive*. A generic truck driver scheduling and routing algorithm is outlined in Algorithm 9 (compare Algorithm 1 in chapter 3). Both in step *SetupBeforeService* and in step *SetupAfterService*, the same function `Setup` of the algorithm is called.

As before, $\mathcal{L}_i^{step}$ denotes the driver states label at the end of step *step* in iteration i. In this chapter, a label comprises nothing but one time-dependent function, which we call $D_i^{step}(t)$ in the following. For a point in time t at the end of step *step* in iteration i, $D_i^{step}(t)$ denotes the minimum accumulated driving time since the end of the last (i.e. most recent) break. Since waiting does not increase the accumulated driving time and is allowed anytime, the function is monotonically decreasing. For some points in time t, for instance outside of the planning horizon, the function value may be $\perp$, which is to be read as "undefined".

Algorithm 9 returns the first point in time t at which $D_n^{served}(t)$ is defined, or returns $\perp$ if such a t does not exist. We use the same notation as in previous chapters: For an interval $\mathcal{I}$, let $\alpha(\mathcal{I})$ be the beginning and $\omega(\mathcal{I})$ the end of that interval. And for a time-dependent function D, let $\alpha(D)$ be $\min\{t \mid D(t) \neq \perp\}$ and $\omega(D)$ be $\max\{t \mid D(t) \neq \perp\}$, or $\perp$ if $D(t) = \perp$ for every t.

Algorithm 9: Generic truck driver scheduling and routing algorithm

Input : $\mathcal{L}_0^{driven}$
Output: Earliest finish time or $\perp$ if no feasible schedule exists

forall $i = 1 \dots n$ **do**
  $\mathcal{L}_i^{setup1} :=$ `Setup` $(\mathcal{L}_{i-1}^{driven})$;
  $\mathcal{L}_i^{waited} :=$ `Wait` $(\mathcal{L}_i^{setup1})$;
  $\mathcal{L}_i^{served} :=$ `Serve` $(\mathcal{L}_i^{waited})$;
  if $i = n$ **then**
    return $\alpha(\mathcal{L}_n^{served})$;
  $\mathcal{L}_i^{setup2} :=$ `Setup` $(\mathcal{L}_i^{served})$;
  $\mathcal{L}_i^{driven} :=$ `Drive` $(\mathcal{L}_i^{setup2})$;

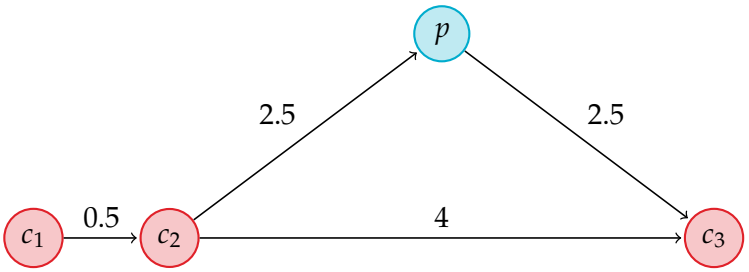


FIGURE 8.2: Example graph with three customers and one parking location. Free-flow driving times are written on edges.

8.3.1 Initialization

We focus on the offline use case, that is, the trip is planned some time before the planning horizon. The planning horizon usually begins in the morning and ends in the evening. And the driver is completely rested when he starts the trip in the morning.

$$D_0^{driven}(t) := \begin{cases} 0, & \text{for } t \in \mathcal{H} \\ \perp, & \text{otherwise} \end{cases}$$

8.3.2 Steps of Algorithm in Detail

Figure 8.2 shows an example graph with three customers and one parking location. Customer c_1 is always open, whereas customer c_2 has one time window from 1.5 to 3.5 and c_3 one time window from 10.5 to 15. All three customers demand a service of 0.5. The driving time from c_1 to c_2 is always 0.5, independent of the time of day. The driving time functions on the other edges are time-dependent and presented later in section 8.3.2.5. Here, we only state the *free-flow* driving times. For a single edge, the free-flow driving time equals the shortest driving time over time, and for a path, it is the sum of the free-flow driving times of the path's edges. For an arbitrary pair of vertices (u, v), it is the smallest free-flow driving time over all paths from u to v. From c_2 to c_3, the free-flow driving time is 4, and both from c_2 to p and from p to c_3, it is 2.5. Let the planning horizon be from 0 to 15, and $break = 3$, and $limit = 4$.

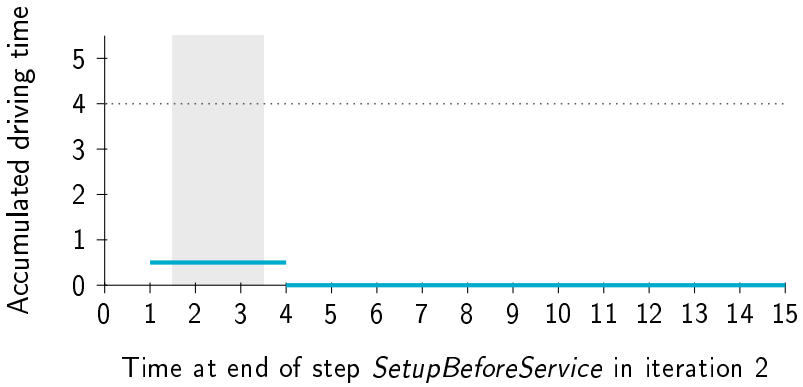


FIGURE 8.3: Function of accumulated driving times D_2^{setup}.

8.3.2.1 Step *SetupBeforeService*

The driver may take a break immediately after arrival and before service. This may be beneficial if the driver would otherwise have to wait. To take that into account, we simply reset the accumulated driving time to 0 for every point in time that is *break* later than the earliest possible arrival time, which is simply the first defined point of D_{i-1}^{driven}.

$$D_i^{setup1}(t) := \begin{cases} D_{i-1}^{driven}(t), & t < \alpha(D_{i-1}^{driven}) + break \\ 0, & \alpha(D_{i-1}^{driven}) + break \leq t \leq \omega(\mathcal{H}) \\ \bot, & \text{otherwise} \end{cases}$$

Figure 8.3 depicts D_2^{setup} as in the example. The earliest possible arrival $\alpha(D_1^{driven})$ at customer c_2 is at time 1, after 0.5 of service and 0.5 of driving. In step *SetupBeforeService*, all function values starting from $1 + break = 4$ are reset to 0.

8.3.2.2 Step *Wait*

Before service, it may be necessary to wait for a time window to open. However with time-dependent driving times, it may also be beneficial to wait *after* service. This is because traffic may clear over time, and thus a favorable parking location or the next customer may become reachable only at a later departure time. Or suppose the next customer opens rather late so there is waiting time when the driver departs right away after service. If the driver wants to be at the next customer when the next time window opens, he rather waits for the shortest driving time that still takes him to the next customer in time. In step *Wait*, we take account of waiting both before and after service.

To this end, we introduce the following notation for this step: For the multi-interval $\mathcal{W}_i$ of the w_i time windows of customer i, let $\overline{\overline{\mathcal{W}}}_i$ denote the multi-interval of the time periods between or after the time windows of customer i, i.e.,

$$\overline{\overline{\mathcal{W}}}_i^j := (\omega(\mathcal{W}_i^j), \alpha(\mathcal{W}_i^{j+1})) \text{ for all } 1 \leq j < w_i, \text{ and } \overline{\overline{\mathcal{W}}}_i^{w_i} := (\omega(\mathcal{W}_i^{w_i}), \omega(\mathcal{H})].$$

(Note that we expect all time windows to lie within the planning horizon $\mathcal{H}$, as already stated in section 2.4.1.) We call these *favorable departure waiting intervals*. Their definition resembles the one of the waiting intervals (compare section 2.4.3).

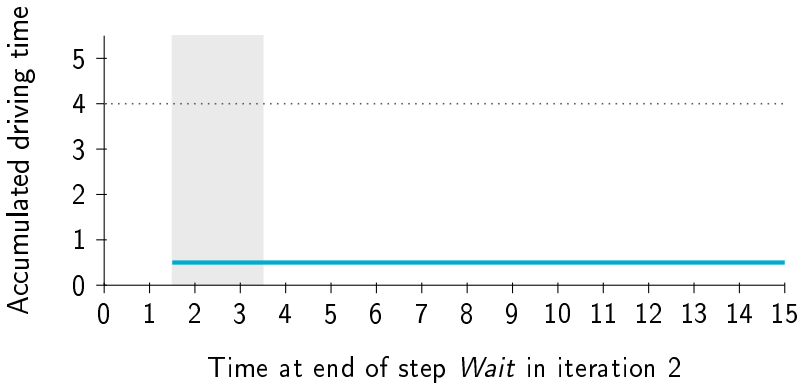


FIGURE 8.4: Function of accumulated driving times D_2^{waited}.

In the following, let us suppose the service time $service_i$ at the current customer is 0. When the driver arrives inside a time window, there is no reason to wait, so we set $D_i^{waited}(t) := D_i^{setup1}(t)$. But what about a time t inside a favorable departure waiting interval? Service is not allowed to begin at time t but it is at the end of the previous time window. Let $t' < t$ be the end of that time window. Then we set $D_i^{waited}(t) := D_i^{setup1}(t')$. We summarize:

$$D_i^{waited}(t) := \begin{cases} D_i^{setup1}(t), & t \in \mathcal{W}_i \\ D_i^{setup1}(\max\{t' \mid t' \in \mathcal{W}_i \wedge t' \leq t\}), & t \in \overline{\overline{\mathcal{W}}}_i \\ \perp, & \text{otherwise} \end{cases}$$

But what if $service_i > 0$ holds? The service time itself is taken into account in the next step. Nothing changes in this step because for the accumulated driving times, it is irrelevant whether the driver first performs service and then waits or the other way round.

This step can be thought of as a 2-in-1 step. We could have first set the function D_i^{waited} to undefined outside of the time windows, then allow for the service time, and finally consider the waiting time for a favorable departure after service. The advantage of regarding waiting both before and after service in the same step is that this way, we do not introduce any gaps in the function of accumulated times.

Function D_2^{waited} in our example is pictured in Figure 8.4. This function is no longer defined over the interval from 1 to 1.5 as it is neither within a time window nor within a departure interval. All function values after the end of the time window at time 3.5 are set to $D_2^{setup}(3.5) = 1$ in order to take account of the possibility to wait after service.

8.3.2.3 Step *Serve*

Step *Serve* is trivial. All we do is shifting the pieces of the function to the right by the service time $service_i$ and cutting off pieces that go beyond the end of the planning horizon.

$$D_i^{served}(t) := \begin{cases} D_i^{waited}(t - service_i), & t \in \mathcal{H} \\ \perp, & \text{otherwise} \end{cases}$$

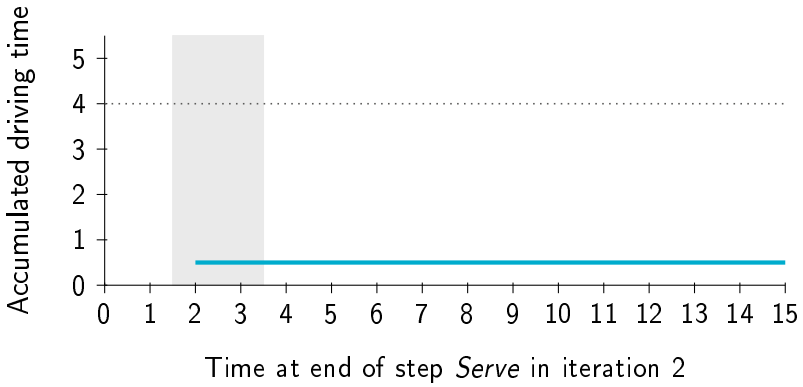


FIGURE 8.5: Function of accumulated driving times D_2^{served}.

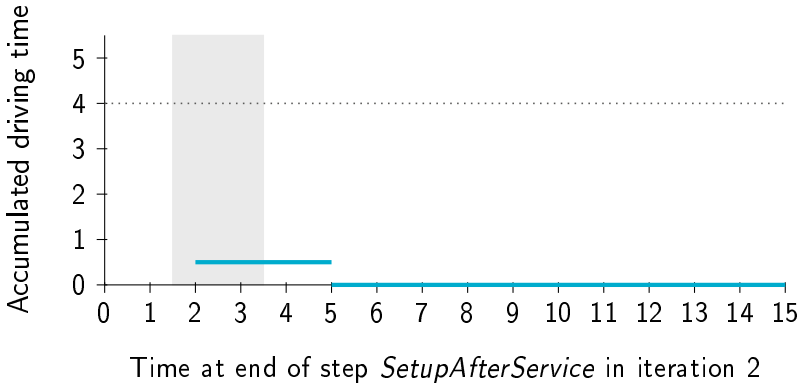


FIGURE 8.6: Function of accumulated driving times D_2^{setup2}.

Figure 8.5 presents function D_2^{served}. The one piece of D_2^{setup} is simply shifted to the right by the service time of 0.5. The fact that waiting after service could be advantageous is already taken care of in step *Wait*.

8.3.2.4 Step *SetupAfterService*

The driver may take a break after service and before departure. To take that into account, we reset the function values to 0 for every point in time that is *break* later than the earliest possible time $\alpha(D_i^{served})$ at which service may be completed. This is analogously to step *SetupBeforeService*.

$$D_i^{setup2}(t) := \begin{cases} D_i^{served}(t), & t < \alpha(D_i^{served}) + break \\ 0, & \alpha(D_i^{served}) + break \leq t \leq \omega(\mathcal{H}) \\ \bot, & \text{otherwise} \end{cases}$$

In our example, function D_2^{setup2} looks as depicted in Figure 8.6.

8.3.2.5 Step *Drive*

Since we request all driving time functions on the road segments to respect the strict FIFO property, so do the computed driving time profiles such as $P_{u,v}$ for the driving time from u to v. It follows that the function id $+P_{u,v}$ that maps a departure time

to an arrival time is bijective. So we can define $\overline{P}_{u,v} := \mathrm{id} - (\mathrm{id} + P_{u,v})^{-1}$ to be the function that maps an arrival time at v to the corresponding driving time from u to v. Even though strictly speaking not correct, we do not see a danger of confusion when we call $\overline{P}_{u,v}$ the *inverse* driving time function between u and v in the following. In the remainder of this section, we assume that the inverse driving time functions are known between every two customers and between every customer and every parking location (or vice versa). Details on an efficient implementation of this step are explained in section 8.4. In this section, we focus on the concept.

Given the inverse driving time function $\overline{P}_{c_i,c_{i+1}}$ between the current and the next customer, we can compute the earliest arrival at customer $i+1$ if not taking a break en route, which we denote by $a_{c_i,c_{i+1}}$ in the following. We need to consider that the driver cannot depart earlier than $\alpha(D_i^{setup2})$, and on arrival, the minimum accumulated driving time since last break must not exceed the driving time limit. So the earliest arrival is at time $a_{c_i,c_{i+1}} := \min\{t \mid t - \overline{P}_{c_i,c_{i+1}}(t) \geq \alpha(D_i^{setup2}) \wedge D_i^{setup2}(t - \overline{P}_{c_i,c_{i+1}}(t)) + \overline{P}_{c_i,c_{i+1}}(t) \leq limit\}$. With this, we set an auxiliary function $D'_{c_i,c_{i+1}}$ that maps an arrival time to the minimum accumulated driving time since last break in case we ignore all parking locations:

$$D'_{c_i,c_{i+1}}(t) := \begin{cases} \min\limits_{a_{c_i,c_{i+1}} \leq t' \leq t} \{D_i^{setup2}(t' - \overline{P}_{c_i,c_{i+1}}(t')) + \overline{P}_{c_i,c_{i+1}}(t')\}, & t \geq a_{c_i,c_{i+1}} \wedge t \in \mathcal{H} \\ \bot, & \text{otherwise} \end{cases}$$

Now we turn towards the case that the driver takes a break at a parking location $p \in \mathcal{P}$ en route between the two customers. Let in this case the earliest arrival at p be denoted by $a_{c_i,p}$ and the earliest arrival at c_{i+1} by $a_{p,c_{i+1}}$. Analogously to the definition of $a_{c_i,c_{i+1}}$, we set $a_{c_i,p} := \min\{t \mid t - \overline{P}_{c_i,p}(t) \geq \alpha(D_i^{setup2}) \wedge D_i^{setup2}(t - \overline{P}_{c_i,p}(t)) + \overline{P}_{c_i,p}(t) \leq limit\}$. For the earliest arrival at customer $i+1$, we exploit that we excluded the possibility of short-term waiting, that is, the driver always takes a break at a parking location. We set $a_{p,c_{i+1}} := \min\{t \mid t - \overline{P}_{p,c_{i+1}}(t) \geq a_{c_i,p} + break \wedge \overline{P}_{p,c_{i+1}}(t) \leq limit\}$. Again, we use this to set an auxiliary function $D'_{p,c_{i+1}}$. It maps an arrival time to the minimum accumulated driving time since last break in case the driver stops at parking location p:

$$D'_{p,c_{i+1}}(t) := \begin{cases} \min\limits_{a_{p,c_{i+1}} \leq t' \leq t} \{\overline{P}_{p,c_{i+1}}(t')\}, & t \geq a_{p,c_{i+1}} \wedge t \in \mathcal{H} \\ \bot, & \text{otherwise} \end{cases}$$

Finally we set $D_i^{driven}(t)$ to the minimum over the auxiliary functions, that is,

$$D_i^{driven}(t) := \min_{v \in \mathcal{P} \cup \{c_i\}, D'_{v,c_{i+1}}(t) \neq \bot} D'_{v,c_{i+1}}(t)$$

for every t inside the planning horizon.

Let us get back to our example. The free-flow driving time from c_2 to c_3 is 4, and the free-flow driving time both from c_2 to p and from p to c_3 is 2.5. However, as we can tell by the Figures 8.7 and 8.8, there is hardly a time when the traffic is flowing freely. Most of the time, the inverse driving time functions depicted therein lie above the free-flow driving times.

The driver may either reach the next customer c_3 directly or via parking location p. By construction of the example, c_3 is not easy to reach directly as the maximum allowed driving time is only 4. And so the driver may arrive at c_3 not before time

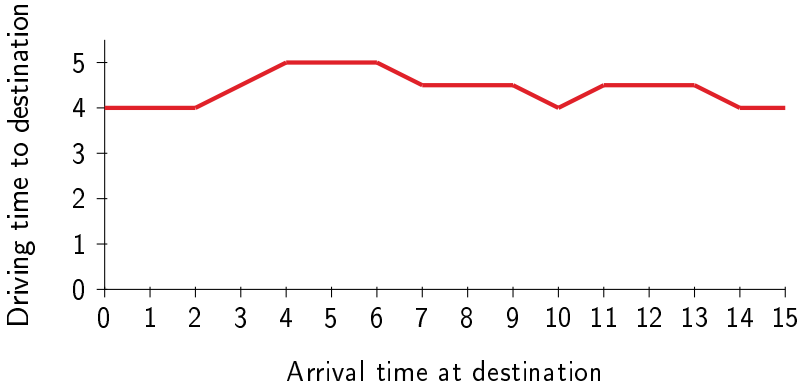


FIGURE 8.7: Inverse driving time function $\overline{P}_{c_2,c_3}$ for a drive from customer c_2 to customer c_3.

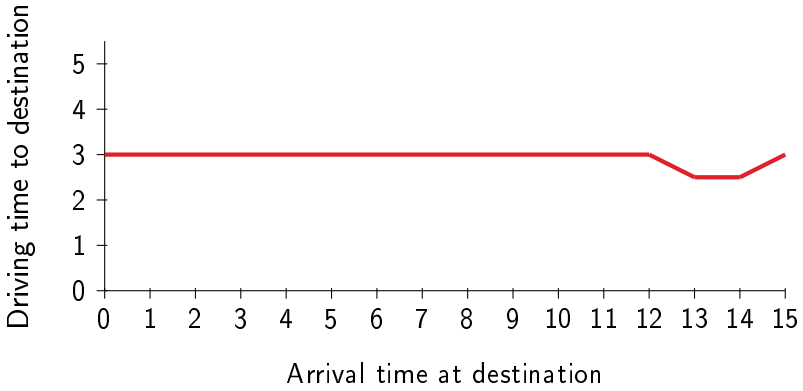


FIGURE 8.8: Inverse driving time function valid both for the drive from customer c_2 to parking location p and from there to customer c_3.

10. In order to arrive at that time, the driver has to depart at time 6 from c_2. Since $D_2^{setup2}(6) = 0$, that is, the driver is completely rested at time 6, the arrival at time 10 is indeed feasible.

Going via the parking location takes longer. The driver may arrive at p not before time 5 and, as he takes a break there, depart not before time 8. Thus, the driver reaches c_3 not before time 11.

Function D_2^{driven} is shown in Figure 8.9. From 10 until 11, the minimum accumulated driving time since the end of the last break is 4, which can be achieved when taking the direct way. From 11 until the end of the planning horizon, it is in line with the inverse driving time function $\overline{P}_{p,c_3}$ (the minimum from the left, to be precise) as depicted in Figure 8.8.

Acceleration by Preprocessing We could pre-compute some bounds to help accelerate our scheduling algorithm. To this end, let $drive_i$ be a lower bound on the driving time from customer i to customer $i+1$. With this, we estimate the *latest possible departure time* from each customer by first setting $lpd_n = \omega(\mathcal{H})$ and then recursively for all i from $n-1$ to 1:

$$lpd_i = \max\{t \mid t \in \mathcal{W}_{i+1} \wedge t \leq lpd_{i+1} - service_{i+1}\} - drive_i$$

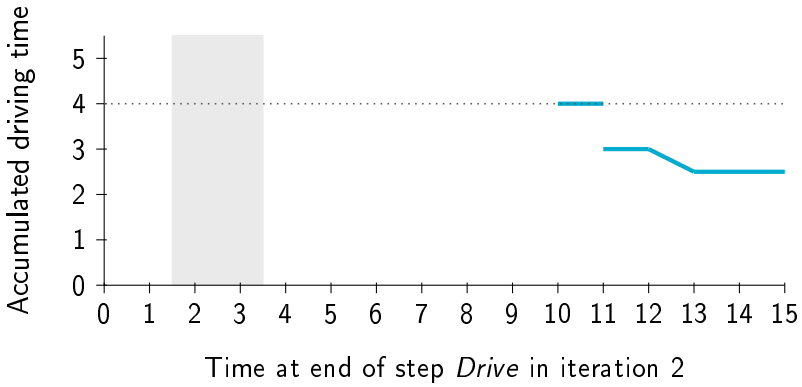


FIGURE 8.9: Function of accumulated driving times D_2^{driven}.

We could use this estimate by setting $D_i^{setup2}(t)$ to $\perp$ for every $t > lpd_i$. This may help identify infeasible instances earlier.

8.3.2.6 Necessary Changes with Regard To Rule *drive until traveled*

If we regard rule *drive until traveled* instead of rule *drive until driven*, the algorithm needs to be adjusted. Instead of a function D_i^{step} that keeps the minimum accumulated driving time for every point in time, a label now holds a function T_i^{step}. For a time t, $T_i^{step}(t)$ denotes the minimum accumulated *travel time* since the end of the last break with respect to step *step* in iteration i. The main difference is that the function T_i^{step} may contain "gaps" in which it is not defined.

The steps *SetupBeforeService* and *SetupAfterService* do not need to be altered. In step *Wait*, the waiting time must be added to the accumulated travel time. Short-term waiting is not beneficial. So after service, the driver should either take a break or depart immediately. There is no reason to wait a little. With the help of the auxiliary function $H^{shift}(t)$ (see section 3.3.3.2), we set:

$$T_i^{waited}(t) := \begin{cases} T_i^{setup1}\left(H^{shift}(t)\right) + (t - H^{shift}(t)), & H^{shift}(t) \neq \perp \\ \perp, & \text{otherwise} \end{cases}$$

In step *Serve*, the service time is added to the accumulated travel time:

$$T_i^{served}(t) := \begin{cases} T_i^{waited}(t - service_i) + service_i, & t \in \mathcal{H} \\ \perp, & \text{otherwise} \end{cases}$$

In step *Drive*, it may now happen that $T_i^{setup2}(t)$ is undefined for some time t. For the sake of simplicity, we omit to re-write step *Drive* and introduce special handling of cases in which the minimum accumulated travel time since last break is undefined. From an arithmetic point of view, $\perp$ can be treated just like ∞.

8.3.3 Complexity Analysis

Given pre-computed profiles For the time being, let us suppose that all inverse driving time functions $\overline{P}_{u,v}$ between every two consecutive customers as well as between every customer and every parking location (in both directions) have been pre-computed. In addition, let us suppose that all driving time functions are constant functions for the sake of simplicity. This means $\overline{P}_{c_i,c_{i+1}}$ as well as, for every $p \in \mathcal{P}$, $\overline{P}_{c_i,p}$ and $\overline{P}_{p,c_{i+1}}$ only comprise one piece. Then how many pieces can D_i^{step} comprise at most?

We observe that in every iteration i

1. D_i^{setup1} can have at most one piece more than D_{i-1}^{driven},
2. D_i^{waited} can have at most as many pieces as D_i^{setup1},
3. D_i^{served} can have at most as many pieces as D_i^{waited},
4. D_i^{setup2} can have at most one piece more than D_i^{served}, and
5. D_i^{driven} can have at most $|\mathcal{P}|$ pieces more than D_i^{setup2} because
   - $D'_{c_i,c_{i+1}}$ can have at most as many pieces as D_i^{setup2} and
   - $D'_{p,c_{i+1}}$ has at most one piece for each $p \in \mathcal{P}$.

We conclude that D_n^{served} comprises at most $n(|\mathcal{P}|+2)+1$ pieces, provided that the inverse driving time functions are all constant. Over all n iterations, the total number of created pieces is in $n^2(|\mathcal{P}|+1)$.

As far as the time complexity of the scheduling algorithm is concerned, we observe that it does not suffice to count the number of created pieces but that the number of time windows has an impact on the run-time, too. In step *Wait* of iteration i, we iterate over the w_i time windows of customer i. Hence, the total run-time is in $O(n^2(|\mathcal{P}|+1)+w)$ (where w is the sum of w_i), disregarding the time it takes to pre-compute the inverse driving time functions. In particular, the time complexity is in $O(n^2+w)$ if there are no parking locations (which then coincides with the no-break-en-route policy as introduced in chapter 5).

Figure 8.10 shows a graph with two customers and seven parking locations between them. By construction of this example, p_k is closer to c_2 but the detour via this parking location is larger when compared to p_j for any $1 \leq j < k \leq 7$. Suppose c_2 is not reachable from c_1 without a break en route. Then, D_1^{driven} consists of $|\mathcal{P}| = 7$ pieces. Figure 8.11 depicts the corresponding Pareto front of the driving time between the two customers via a parking location on one hand and the driving time from a parking location to the next customer on the other hand.

Complexity of computing profiles In case of constant driving time functions on every edge, the shortest paths from a single source to some other vertices like from a customer to all parking locations can be computed in $O(|\mathcal{E}|+|\mathcal{V}|\log|\mathcal{V}|)$ time (Fredman and Tarjan, 1987). The same complexity holds for the computation of shortest paths to a single destination like from all parking locations to the next customer. It takes $(n-1)$ single-source and $(n-1)$ single-destination (distance restricted) shortest path computations to have pre-computed all necessary profiles. So the time-independent truck driver scheduling and routing problem can be solved in polynomial time.

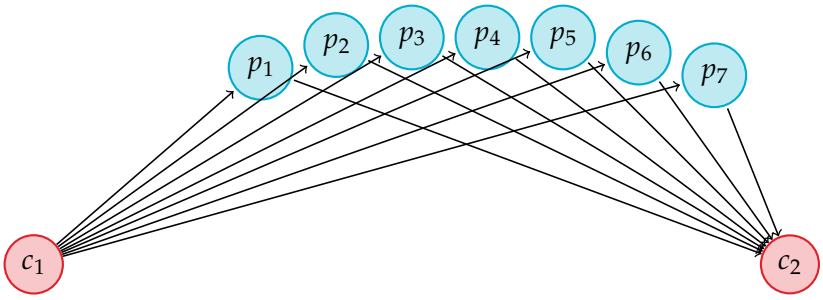


FIGURE 8.10: "Pareto front" of parking locations. The closer a parking location is to c_1, the shorter is the detour but the longer is the remaining drive to c_2.

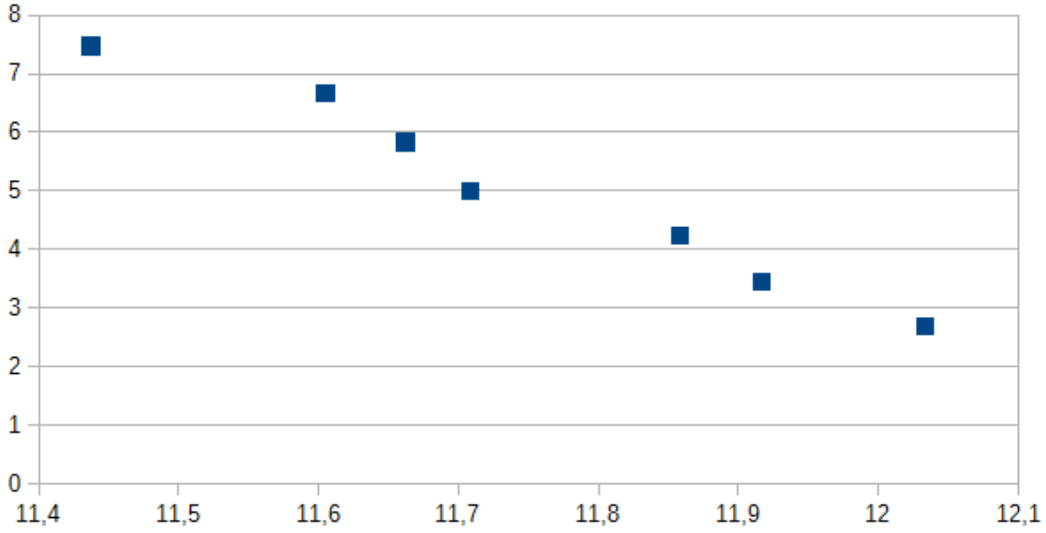


FIGURE 8.11: Pareto front corresponding to Figure 8.10. Driving time between customers via a parking location on x-axis and driving time from parking location to next customer on y-axis. The left-most data point corresponds to the parking location p_1, the right-most to p_7.

With piecewise linear but not necessarily constant driving time functions, the shortest path between two vertices in the graph can change $|\mathcal{V}|^{\Theta(\log|\mathcal{V}|)}$ times, and so the computed profiles can have super-polynomial complexity (Foschini, Hershberger, and Suri, 2014). This means that in this case, the computation of profiles dominates the run-time of our exact truck driver scheduling and routing algorithm. This complexity is the motivation for the acceleration techniques that are described in section 8.4.

8.3.4 Schedule and Route Deduction

Algorithm 1 only returns the earliest finish time of a route. But we are also interested in a corresponding route itself and a feasible schedule. We can deduce both from the computed functions, and Algorithm 10 shows how.

Algorithm 10: Schedule and route deduction

1 $\mathcal{B} := \mathcal{A} := \mathcal{D} := \mathcal{R} := \emptyset$;
2 $\mathcal{B} \mathrel{+}= \alpha(D_n^{waited})$;
3 **forall** $i = n-1 \dots 1$ **do**
4 **if** $\mathcal{B}[0] \geq \alpha(D_i^{driven}) + break$ **then**
5 $\mathcal{A} \mathrel{+}= \min\{t \mid D_i^{driven}(t) = D_i^{driven}(\mathcal{B}[0] - break)\}$;
6 **else**
7 $\mathcal{A} \mathrel{+}= \min\{t \mid D_i^{driven}(t) = D_i^{driven}(\mathcal{B}[0])\}$;
8 pick a vertex v from $\arg\min_{u \in \mathcal{P} \cup \{c_i\}} D'_{u,c_{i+1}}(\mathcal{A}[0])$;
9 $\mathcal{R} \mathrel{+}= (v, c_{i+1})$;
10 $\mathcal{D} \mathrel{+}= \mathcal{A}[0] - \overline{P}_{v,c_{i+1}}(\mathcal{A}[0])$;
11 **if** $v \neq c_i$ **then**
12 $\mathcal{A} \mathrel{+}= a_{c_i,v}$;
13 $\mathcal{R} \mathrel{+}= (c_i, v)$;
14 $\mathcal{D} \mathrel{+}= \mathcal{A}[0] - \overline{P}_{c_i,v}(\mathcal{A}[0])$;
15 **if** $\mathcal{D}[0] \geq \alpha(D_i^{served}) + break$ **then**
16 $\mathcal{B} \mathrel{+}= \max\{t \mid t \in \mathcal{W}_i \wedge t \leq \mathcal{D}[0] - break - service_i\}$;
17 **else**
18 $\mathcal{B} \mathrel{+}= \max\{t \mid t \in \mathcal{W}_i \wedge t \leq \mathcal{D}[0] - service_i\}$;

The sequence $\mathcal{R}$ of route segments as well as the sequences $\mathcal{B}, \mathcal{A}$, and $\mathcal{D}$ that constitute a truck driver schedule are initially empty. The algorithm works from back to front. The operator "+=" inserts the operand on the right-hand side at the beginning of the sequence on the left-hand side, so that $\mathcal{B}[0], \mathcal{A}[0]$, and $\mathcal{D}[0]$ refer to the most recently added points in time. All three sequences remain monotonously increasing.

We explain the algorithm step by step by the example used in section 8.3.2 (see Figure 8.2). Customer 3 has one time window that begins at time 10.5. As we can see in Figure 8.9, the driver is able to be at the final customer right when that time window opens. So in line 2, the time 10.5 is added to the front of the sequence $\mathcal{B}$ that contains those points in time at which the service at a customer commences.

Next, we want to determine the arrival time at customer 3. As we have already discussed, c_3 cannot be reached before 11 when going via the parking location. We

notice that, according to $D_2^{driven}(10.5)$ in Figure 8.9, the minimum accumulated driving time since last break is 4. But we learn from the inverse driving time function in Figure 8.7 that the driving time from c_2 to c_3 is 4.25 when arriving at time 10.5. This does not match. The accumulated driving time of 4 at time 10.5 can only be achieved when the driver arrives at 10 and then waits until 10.5. The waiting time before service is taken into account in line 5 or line 7. In our example, the condition in line 4 is not fulfilled, so the arrival time is determined in line 7. We observe that $D_2^{driven}(t)$ equals $D_2^{driven}(10.5)$ for no t earlier than 10. So indeed, it is time 10 that is added to the front of the sequence $\mathcal{A}$ of arrival times.

In general, $D'_{u,c_{i+1}}(\mathcal{A}[0]) = D_i^{driven}(\mathcal{A}[0])$ may hold for several vertices from $\mathcal{P} \cup \{c_i\}$. In line 8, we pick an arbitrary vertex from this list. For our algorithm and our problem definition, all vertices from the list are equally suitable. However, there may be additional optimization criteria such as preferences where to take a break, in which case we may refine this line. In line 9, (c_2, c_3) is added to $\mathcal{R}$; and in line 10, time 6 is added to the sequence $\mathcal{D}$ of departure times. In lines 16 or 18, depending on whether there is enough buffer for a break, the waiting time after service is considered. It is assured that every point in time that is added to $\mathcal{B}$ is inside a time window.

After the second iteration of the for-loop, the truck driver route that is finally returned is $\mathcal{R} = [(c_1, c_2), (c_2, c_3)]$. We present the three sequences $\mathcal{B}, \mathcal{A}$, and $\mathcal{D}$ as output by the algorithm in Table 8.1.

TABLE 8.1: Truck driver schedule as returned by Algorithm 10 in case of the example graph as input. The truck driver route is $\mathcal{R} = [(c_1, c_2), (c_2, c_3)]$.

	c_1	c_2	c_3
$\mathcal{B}$	1.5	2.5	10.5
$\mathcal{D}$	2	6	
$\mathcal{A}$		2.5	10

In general, we try to schedule the arrival, departure, and start times as late as possible. But Algorithm 10 is a heuristic in the sense that there may be a feasible truck driver schedule with the same finish time but a later start time that is not found by the algorithm.

8.4 Routing Part of the Exact Approach

We now turn towards the routing part of the exact solution approach. In the description of step *Drive* in section 8.3.2.5, we assume to know the inverse driving time function $\overline{P}_{u,v}$ for every $u \in \mathcal{P} \cup \{c_i\}$ and $v \in \mathcal{P} \cup \{c_{i+1}\}$, where at least one of them is a customer vertex. In the following, we deal with the efficient computation of these profiles.

Conceptually, computing these profiles is not too involved. A profile search can be implemented as described by Delling and Wagner (2009). However, the runtime for computing such a profile depends on the number of its breakpoints, that is, on the number of points in time at which the slope of the profile changes. As we have already mentioned in section 8.3.3, that number may grow super-polynomially in the number of vertices (Foschini, Hershberger, and Suri, 2014). This makes the profile search by far the most time-consuming part of the exact approach. Hence,

the focus of this section is on speed-up techniques. We have already introduced such techniques in section 7.3 of the previous chapter. In the following, we proceed similar to the approaches described in that section.

One of the acceleration measures is to use time-dependent contraction hierarchies (TCH) (Batz et al., 2013). Here, we distinguish a pre-processing phase and a query phase, and the data collected during the first phase then speeds up the second phase. It is described in more detail in section 7.3.2. It is re-used in this chapter without any changes.

Another acceleration method is to narrow down the profile searches and only compute *partial profiles*. Partial profiles and *profile range queries* have been introduced and defined in section 7.3.1. Here, the basic idea is to first spend little extra time on finding bounds on the earliest arrival at the next customer. Then, with the aid of these bounds, we know to which range of a profile we can restrict the profile queries in step *Drive*. In comparison with the approach in section 7.3.1, three things are different:

1. In the previous chapter, we expect the driver to be completely rested when he departs from the source s. Now, when the driver departs from some customer i at time t, we have to consider the driver state $D_i^{setup2}(t)$, which is not necessarily 0 as before. So we have to do more.

2. In the following, we do not consider short-term waiting. This in turn makes it a little simpler.

3. In the previous chapter, we are only interested in the earliest arrival time at the destination d. Now, we need to know the driver state on arrival at the next customer for a certain time span that begins with the earliest arrival time. It ends already *break* later because the accumulated driving time would be reset to 0 for this and any later point in time in the succeeding step *SetupBeforeService*.

In section 8.4.1, we describe how to find both a lower and an upper bound on the earliest arrival time at the next customer. Then, in section 8.4.2, we explain how we use these bounds for the profile range queries.

8.4.1 Computing Bounds on Earliest Arrival Time at Next Customer

In step *Drive* of iteration i, we are given D_i^{setup2} from the previous step. But in the following, we are interested in this function at only two points in time: Let $t^\alpha := \alpha(D_i^{setup2})$ be the earliest departure time from customer c_i and $t^0 := \min\{t \mid D_i^{setup2}(t) = 0\}$ be the first point in time at which the driver is completely rested. In the first iteration, these are the same points in time. In general, $t^0 \leq t^\alpha + break$ holds.

In the following, a *single departure query* (in contrast to a profile query) asks for the driving time between customer i and $i+1$ such that the driving time is minimum with respect to a given departure time but disregarding all break rules. Adding this minimum driving time to the given departure time yields the earliest arrival at customer $i+1$ with respect to that departure time, and so it can be seen as a variant of an earliest arrival query. There are two reasons why we omit the term earliest arrival query here: One is that "earliest" arrival may suggest that the value is absolute whereas it is with regard to a departure time. Since we consider two significant departure times, this may cause confusion (calling it lbEA does not help here). The other - more important one - is that we want earliest arrival to mean earliest arrival with regard to break rules (absolute, only relative to planning horizon).

To find an upper bound on the earliest arrival at the next customer requires some effort. We describe the procedure in four steps.

1. At first, we conduct a one-to-one single departure search for the departure from customer i at time t^α, which gives us $P_{c_i,c_{i+1}}(t^\alpha)$. With this, we get a lower bound on the earliest arrival $lbEA(c_{i+1}) := t^\alpha + P_{c_i,c_{i+1}}(t^\alpha)$ at the next customer. If the driver may reach the next customer without due break when departing at time t^α, so if $D_i^{setup2}(t^\alpha) + P_{c_i,c_{i+1}}(t^\alpha) \leq limit$, then the lower bound is tight and we have found the earliest arrival $EA(c_{i+1})$ at the next customer. In this case, we can skip the other steps.

2. We compute a lower bound $lbMin(P_{c_i,c_{i+1}})$ on the driving time between the two customers, i.e., we conduct a one-to-one profile bounds query. Should $lbMin(P_{c_i,c_{i+1}})$ even be longer than $2 \cdot limit$, then we consider the next customer as not reachable and stop here. If $lbMin(P_{c_i,c_{i+1}})$ is at least longer than $limit$, a break at a parking location is inevitable, so we continue with the next step. Otherwise we proceed similar to the first step and conduct a second one-to-one single departure search for the departure time t^0. In case $P_{c_i,c_{i+1}}(t^0) \leq limit$ holds, it is sufficient to take a break at the current customer before departure. Then, $ubEA(c_{i+1}) := t^0 + P_{c_i,c_{i+1}}(t^0)$ is an upper bound on the earliest arrival at the next customer, and we skip the other steps again. Otherwise we still look for a valid upper bound.

3. We perform a one-to-many single departure search to all potentially reachable parking locations for the departure time t^α, together with a profile bounds search. The latter helps us limit the search space because we can stop searching if the lower bound on the driving time stored in the label at the top of the heap exceeds $limit$. Let $\mathcal{P}' \subset \mathcal{P}$ be the subset of parking locations such that every $p \in \mathcal{P}'$ may be reachable from customer c_i at some point in time, more precisely, those with $lbMin(P_{c_i,p}) \leq limit$. For some of these, even $D_i^{setup2}(t^\alpha) + P_{c_i,p}(t^\alpha) \leq limit$ holds, that is, parking location p is known to be reachable from customer c_i at time t^α. Let $\mathcal{P}'_\alpha \subset \mathcal{P}'$ be the set of these parking locations.

   The next step is to conduct a profile bounds search from the next customer backwards to all potentially reachable parking locations in $\mathcal{P}$, i.e., we compute $lbMin(P_{p,c_{i+1}})$ and $ubMax(P_{p,c_{i+1}})$ for all $p \in \mathcal{P}$ with $lbMin(P_{p,c_{i+1}}) \leq limit$. In turn, let $\mathcal{P}'' \subset \mathcal{P}$ be the subset of parking locations such that customer c_{i+1} may be reachable from every $p \in \mathcal{P}''$ at some point in time, i.e., those with $lbMin(P_{p,c_{i+1}}) \leq limit$. If $\mathcal{P}'' \cap \mathcal{P}'$ is empty but at the same time $lbMin(P_{c_i,c_{i+1}}) > limit$, there is no feasible solution. Figure 8.12 illustrates the interrelation between the sets of parking locations.

   For all $p \in \mathcal{P}'_\alpha \cap \mathcal{P}'' =: \mathcal{P}''_\alpha$, the earliest arrival there is known. It is at $t^\alpha + P_{c_i,p}(t^\alpha)$, and thus the earliest departure is at $t_p := t^\alpha + P_{c_i,p}(t^\alpha) + break$. For every $p \in \mathcal{P}''_\alpha$ with $ubMax(P_{p,c_{i+1}}) \leq limit$, an upper bound on the earliest arrival at customer c_{i+1} is $t_p + ubMax(P_{p,c_{i+1}})$. So if such parking locations exist, the minimum of these bounds is the upper bound we are looking for, and we skip the last step. (If $lbMin(P_{c_i,c_{i+1}}) > limit$, a break has to be taken at a parking location, and so we could also improve our lower bound on the earliest arrival at the next customer. We omit the details here.)

4. However, should such a parking location not exist, then we proceed analogously to the previous step but for the departure time t^0. That is, we determine

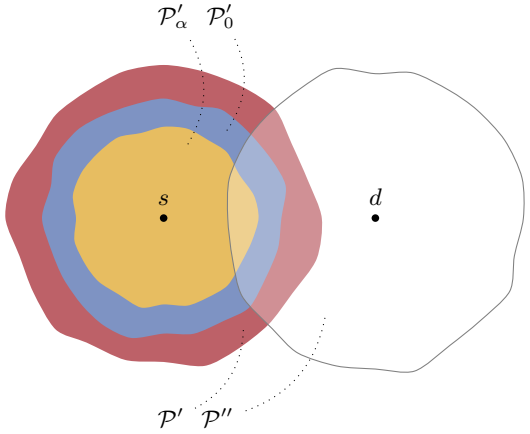


FIGURE 8.12: The parking locations sets $\mathcal{P}' \supset \mathcal{P}'_0 \supset \mathcal{P}'_\alpha$ and $\mathcal{P}''$.

sets $\mathcal{P}''_0 \subset \mathcal{P}'_0 \subset \mathcal{P}'$ in an analogue way. In case there is a parking location in $\mathcal{P}''_0$ with $ubMax(P_{p,c_{i+1}}) \leq limit$, an upper bound at the next customer can be found. Otherwise the only upper bound on the earliest arrival at the next customer we know for sure is the end of the planning horizon.

After these steps, we know a lower and an upper bound on the earliest arrival at the next customer. If the computed lower bound on earliest arrival time at customer c_{i+1} is outside of the planning horizon, there is no feasible solution.

8.4.2 Profile Range Queries

We propagate the profile range $[lbEA(c_{i+1}), ubEA(c_{i+1}) + break]$ backwards to the current customer c_i and all parking locations with both $lbMin(P_{c_i,p}) \leq limit$ and $lbMin(P_{p,c_{i+1}}) \leq limit$. That is, we get to know $\overline{P}_{v,c_{i+1}}(t)$ for all $v \in \mathcal{P}''' \cup \{c_i\}$ with $\mathcal{P}''' := \mathcal{P}' \cap \mathcal{P}''$ but only for $t \in [lbEA(c_{i+1}), ubEA(c_{i+1}) + break]$ because we do not need to know the driving time profile for other points in time.

The next step is to compute $\overline{P}_{c_i,p}(t)$ for all $p \in \mathcal{P}'''$ over a certain time interval. We calculate an upper bound $ubED(p)$ on earliest departure from a parking location $p \in \mathcal{P}'''$. It is $ubED(p) := ubEA(c_{i+1}) + break - \overline{P}_{p,c_{i+1}}(ubEA(c_{i+1}) + break)$. And $ubED(c_i) := \max_{p \in \mathcal{P}'''} ubED(p) - break - lbMin(P_{c_i,p})$ gives an upper bound on the earliest departure from customer c_i. We perform another profile range search. This time we propagate forward from customer c_i in the range $[t^\alpha, ubED(c_i)]$. This way, we obtain $\overline{P}_{c_i,p}(t)$ for all $p \in \mathcal{P}'''$ but only for those t such that $t - \overline{P}_{c_i,p}(t) \in [t^\alpha, ubED(c_i)]$.

Finally, we have computed the inverse driving time functions for all relevant relations and over all relevant points in time. This concludes the routing part of the exact approach.

8.5 Heuristic

We split the presentation of the heuristic in two parts. We start with describing a basic version of the heuristic (section 8.5.1) before we then turn to a worthwhile enhancement of it (section 8.5.2).

8.5.1 Basic Heuristic

The basic heuristic is divided into the same steps as the exact approach. The only step of the heuristic that deviates from the exact approach is step *Drive* as this is the most time consuming step. The idea is to rather process multiple single departure queries than to invoke a single profile search. The heuristic presented in section 7.3.3 of the previous chapter is based on the same idea. As we have seen before (recall Tables 7.3 and 7.4 of the previous chapter), a profile search is very expensive and, at least in our setting, hardly worth the computational effort. In the following, we describe the adjusted version of step *Drive* in iteration i.

Just like before in section 8.4.1, there are two significant departure times at customer i. Let again $t^\alpha := \alpha(D_i^{setup2})$ be the earliest departure time from customer c_i and $t^0 := \min\{t \mid D_i^{setup2}(t) = 0\}$ be the first point in time at which the driver is completely rested. Our heuristic sets $D_i^{driven}(t) := \perp$ for all t but one, so $D_i^{driven}(t)$ is only defined for a single point in time t. This point in time is determined in one of the following four steps:

1. If $D_i^{setup2}(t^\alpha) + P_{c_i,c_{i+1}}(t^\alpha) \leq \mathit{limit}$, the driver may arrive at the next customer without having to take a break. For $t := t^\alpha + P_{c_i,c_{i+1}}(t^\alpha)$, we set
$$D_i^{driven}(t) := D_i^{setup2}(t^\alpha) + P_{c_i,c_{i+1}}(t^\alpha)$$

2. However, if a break is inevitable, we check whether it is sufficient to take a break at the customer before departure. This is the case if $P_{c_i,c_{i+1}}(t^0) \leq \mathit{limit}$ holds. Analogously to step 1, we set for $t := t^0 + P_{c_i,c_{i+1}}(t^0)$
$$D_i^{driven}(t) := P_{c_i,c_{i+1}}(t^0)$$

3. It may happen that the next customer is too far away and not reachable without a break en route. Let $\mathcal{P}'_\alpha \subset \mathcal{P}$ be the subset of parking locations such that every $p \in \mathcal{P}'_\alpha$ is reachable from customer c_i at time t^α, i.e., $D_i^{setup2}(t^\alpha) + P_{c_i,p}(t^\alpha) \leq \mathit{limit}$. For all $p \in \mathcal{P}'_\alpha$, the earliest arrival at p is at $t^\alpha + P_{c_i,p}(t^\alpha)$ and the earliest departure is at $t_p := t^\alpha + P_{c_i,p}(t^\alpha) + \mathit{break}$. In turn, let $\mathcal{P}''_\alpha \subset \mathcal{P}'_\alpha$ be the subset of parking locations such that customer c_{i+1} is reachable from every $p \in \mathcal{P}''_\alpha$ at the earliest departure time, i.e., $P_{p,c_{i+1}}(t_p) \leq \mathit{limit}$. For all $p \in \mathcal{P}''_\alpha$, the earliest arrival at customer c_{i+1} is $t_p + P_{p,c_{i+1}}(t_p)$. Provided that $\mathcal{P}''_\alpha$ is not empty, let $p^* \in \mathcal{P}''_\alpha$ be a parking location for which $t_{p^*} + P_{p^*,c_{i+1}}(t_{p^*})$ is minimum (primary criterion) and, in case the minimum is attained at more than one parking location, $P_{p^*,c_{i+1}}(t_{p^*})$ is minimum (secondary criterion). For $t := t_{p^*} + P_{p^*,c_{i+1}}(t_{p^*})$, we set
$$D_i^{driven}(t) := P_{p^*,c_{i+1}}(t_{p^*})$$

4. Should $\mathcal{P}''_\alpha$ be empty, we proceed analogously to the previous step for the departure time t^0. This time, we seek for the reachable parking locations $\mathcal{P}'_0$ at time t^0, and in turn for the subset $\mathcal{P}''_0$ of these from which customer c_{i+1} is reachable after a break at a parking location. Again, we first calculate the earliest departure time t_p for all $p \in \mathcal{P}'_0$ and the earliest arrival time at the next customer for all $p \in \mathcal{P}''_0$. Then we choose an optimal parking location $p^* \in \mathcal{P}''_0$ regarding the same criteria as in the previous step. Just like before, we set for $t := t_{p^*} + P_{p^*,c_{i+1}}(t_{p^*})$
$$D_i^{driven}(t) := P_{p^*,c_{i+1}}(t_{p^*})$$

If even $\mathcal{P}_0''$ is empty or if the computed earliest arrival time at customer c_{i+1} is outside of the planning horizon, we cannot find a feasible schedule and terminate the algorithm prematurely, returning $\perp$.

8.5.2 Enhancement of the Heuristic

The basic heuristic prefers an early break at a customer over a break at a parking location, so it does not fully exploit the driving time potential. When the total driving time of the route is close to $2 \cdot \mathit{limit}$ for example, this behavior may cause the basic heuristic to schedule two breaks while the exact approach is able to find a better solution with only one. The goal of the enhancement is to reduce the number of scheduled breaks by one if possible.

The enhancement consists in a *backtracking* component. It is based on the following idea: Suppose the basic heuristic schedules two breaks, the first of which is an early break at a customer. In order to exploit the allowed driving time better, the backtracking heuristic schedules the first break at a parking location en route to the next customer instead. If we are lucky, the second break can be omitted this way. If not, we go back again and pick another parking location, one that is still reachable from the customer but even closer to the next customer than the previously picked parking location. This is repeated until either the second break can be omitted or another parking location with that property does not exist. In the latter case, a solution with two breaks is the best that the heuristic can find.

The overall heuristic proceeds as follows: The first thing we do is to call the basic heuristic in order to gain an initial solution. Here, we may allow the planning horizon to be exceeded by the minimum break duration as it is our aim to save a break and find an earlier finish time. The backtracking heuristic is only called if the basic heuristic schedules more than one break. Otherwise the solution found by the basic heuristic is returned provided that it remains within the planning horizon.

Pseudo code of the backtracking heuristic is presented by Function BTHeuristic. It is based on a recursive reformulation of Algorithm 1. Input to `BTHeuristic` is not only the label $\mathcal{L}_{i-1}^{\mathit{driven}}$ of the previous iteration, but also the current iteration i itself and a limit z on the number of breaks that the function is allowed to schedule. So if the basic heuristic schedules two breaks for instance, we call `BTHeuristic` with $(\mathcal{L}_0^{\mathit{driven}}, 1, 1)$ as input.

In line 3 of `BTHeuristic`, it is checked whether there is an *inevitable early break* before the service at customer c_i. This is the case if the waiting time for the opening of the next time window is so long that it may as well count as break, i.e., if both $i > 1$ and $D_i^{\mathit{waited}}(\alpha(D_i^{\mathit{waited}})) = 0$ holds. Here, the first customer is special because the driver rather starts later than waits. If there is an inevitable early break indeed, but the parameter z forbids to take a break, the backtracking heuristic does not find a feasible solution. In line 12, we use the ternary operator ?: that evaluates to $z - 1$ if there is an inevitable early break, and to z otherwise.

Lines 13 through 19 treat the case that one break is necessary. At first, we compute a set of vertices $\mathcal{P}_1^*$ where it is feasible to take a break. If there is an inevitable early break before service, one break does not suffice, because we already know that the next customer cannot be reached directly, despite the early break. In this case, we set $\mathcal{P}_1^* := \emptyset$ and thus skip this if-clause. Otherwise, let $\mathcal{P}_\alpha'$ and $\mathcal{P}_\alpha''$ be defined just like in section 8.5.1, that is, let $\mathcal{P}_\alpha'' \subset \mathcal{P}_\alpha'$ be the subset of parking locations such that customer c_{i+1} is reachable from every $p \in \mathcal{P}_\alpha''$ at the earliest departure time t_p, i.e., $P_{p,c_{i+1}}(t_p) \leq \mathit{limit}$. If in addition $P_{c_i,c_{i+1}}(t^0) \leq \mathit{limit}$ holds, we also set $t_{c_i} := t^0$ and

Function BTHeuristic

Input : Label $\mathcal{L}_{i-1}^{driven}$, customer index i, maximum number z of breaks to schedule

Output: A finish time of a feasible schedule or $\perp$ if no feasible schedule is found

$D_i^{setup1} :=$ `Setup` (D_{i-1}^{driven});
$D_i^{waited} :=$ `Wait` (D_i^{setup1});
if $i > 1 \wedge D_i^{waited}(\alpha(D_i^{waited})) = 0 \wedge z = 0$ **then**
  return $\perp$;
$D_i^{served} :=$ `Serve` (D_i^{waited});
if $i = n$ **then**
  return $\alpha(D_n^{served})$;
$D_i^{setup2} :=$ `Setup` (D_i^{served});
$t^\alpha := \alpha(D_i^{setup2})$;
if $D_i^{setup2}(t^\alpha) + P_{c_i,c_{i+1}}(t^\alpha) \leq limit$ **then**
  $D_i^{driven}(t) := \begin{cases} D_i^{setup2}(t^\alpha) + P_{c_i,c_{i+1}}(t^\alpha), & t = t^\alpha + P_{c_i,c_{i+1}}(t^\alpha) \\ \perp, & \text{otherwise} \end{cases}$;
  return `BTHeuristic`$(D_i^{driven}, i+1, (i > 1 \wedge D_i^{setup2}(t^\alpha) = 0)\,?\,z-1 : z)$;
if $z \geq 1$ **then**
  compute $\mathcal{P}_1^*$ and departure times t_v for all $v \in \mathcal{P}_1^*$;
  forall *v in $\mathcal{P}_1^*$, ascending in $t_v + P_{v,c_{i+1}}(t_v)$* **do**
    $D_i^{driven}(t) := \begin{cases} P_{v,c_{i+1}}(t_v), & t = t_v + P_{v,c_{i+1}}(t_v) \\ \perp, & \text{otherwise} \end{cases}$;
    $obj :=$ `BTHeuristic`$(D_i^{driven}, i+1, z-1)$;
    if $obj \neq \perp$ **then**
      return *obj*;
if $z \geq 2$ *and* $\mathcal{P}_1^* = \emptyset$ **then**
  compute $\mathcal{P}_2^*$ and departure times t_p for all $p \in \mathcal{P}_2^*$;
  forall *p in $\mathcal{P}_2^*$, ascending in $t_p + P_{p,c_{i+1}}(t_p)$* **do**
    $D_i^{driven}(t) := \begin{cases} P_{p,c_{i+1}}(t_p), & t = t_p + P_{p,c_{i+1}}(t_p) \\ \perp, & \text{otherwise} \end{cases}$;
    $obj :=$ `BTHeuristic`$(D_i^{driven}, i+1, z-2)$;
    if $obj \neq \perp$ **then**
      return *obj*;
return $\perp$;

add c_i to some otherwise empty set $\mathcal{Q}$. So for all $v \in \mathcal{P}''_\alpha \cup \mathcal{Q}$, the earliest arrival at customer c_{i+1} is $t_v + P_{v,c_{i+1}}(t_v)$ with an accumulated driving time of $P_{v,c_{i+1}}(t_v)$.

Now let $\mathcal{P}^*_1 \subset \mathcal{P}''_\alpha \cup \mathcal{Q}$ be the maximum subset such that for any two $u, v \in \mathcal{P}^*_1$ either the arrival time is earlier or the accumulated driving time since last break is lower, i.e., either $t_u + P_{u,c_{i+1}}(t_u) < t_v + P_{v,c_{i+1}}(t_v)$ or $P_{u,c_{i+1}}(t_u) < P_{v,c_{i+1}}(t_v)$ holds. So every $v \in \mathcal{P}^*_1$ leads to a *Pareto optimum*. The backtracking heuristic iterates over the set $\mathcal{P}^*_1$ that is supposed to be sorted by $t_v + P_{v,c_{i+1}}(t_v)$ in ascending order. As soon as a feasible solution is found by the recursive call of `BTHeuristic`, it is returned. On the other hand, $\perp$ is returned if no feasible solution can be found, provided that $\mathcal{P}^*_1$ is not empty.

If $\mathcal{P}^*_1$ is empty, two breaks are necessary. This case is treated in the next if-clause from line 20 to 26. We proceed similar to the previous if-clause. Analogously to before, we determine a set $\mathcal{P}^*_2 \subset \mathcal{P}''_0$ of parking locations such that either the arrival time at the next customer or the accumulated driving time on arrival is better. Since two breaks are scheduled, we reduce z accordingly when we call `BTHeuristic` recursively.

When the initial call of `BTHeuristic` finally returns a solution, it is guaranteed to contain at most z breaks, so by construction a break less than the initial solution. However, on very odd and artificial instances, the new solution may still be worse, in which case we stick to the initial solution. At last, we also have to check that the planning horizon is respected if we allowed its exceedance before.

8.6 Experiments

The basic test setup is the same as in the previous chapter. We conduct all our experiments with respect to the road network of Germany. The corresponding road graph is provided by TomTom[1]. Also from TomTom is the original data regarding time-dependent changes in speed on the graph's edges. This data is modified by PTV in order to model speeds (and driving times derived from these) of trucks instead of cars as in the original data. The data on parking locations stems from the Truck Parking Europe app[2]. This app provides a service to truck drivers by displaying nearby parking lots, their occupancy status, and available facilities. All our experiments are conducted on the same road graph, referred to as *Germany 2017* in section 7.4 of the previous chapter, and the same set of parking locations. Key figures of the input data are given in Table 8.2. The break rule parameters are set according to the EU regulation, that is, *break*=45 min and *limit*=4.5 h.

TABLE 8.2: Key figures of the road network *Germany 2017* and the set of parking locations, where % TD denotes the percentage of edges with a time-dependent (i.e. not constant) driving time function.

$\lvert\mathcal{V}\rvert$	$\lvert\mathcal{E}\rvert$	% TD	# Breakpoints	$\lvert\mathcal{P}\rvert$
7.2 M	15.7 M	28.6 %	136.9 M	6 596

Just like the input data, the used technical configuration remains the same as in section 7.4. We still run our experiments on a VMware ESX cluster. Our machine uses four cores of a 2.2 GHz Intel Xeon E5-2698 v4, 64 GB main memory, and runs Ubuntu 16.04. Our code is written in C++ and compiled with gcc 5.4, optimization

[1] `https://www.tomtom.com/`

[2] `https://app.truckparkingeurope.com/`

level -O3. Besides the construction of the contraction hierarchies the algorithms use only one core. Our CH implementation is based on the code by Batz (Batz et al., 2013; *KaTCH*) and has been extended as described in section 7.3.2. We set the size of the CH cores to 0.2 % of the vertices, which results in a search graph size of 38.90 GB.

Let us quickly recall a few of the findings presented in chapter 7 that are valid for the truck driver scheduling and routing problem just like they are for the truck driver routing problem. We do this because we do not repeat all our previous experiments in the new context.

- The acceleration by computing bounds as described in section 8.4 is effective. In chapter 7, a speed-up of 25 over 2000 randomly generated queries with two customers is reported (recall Table 7.3). Among those instances where one break is known to be both necessary and sufficient, that speed-up factor even reaches 40. So the effectiveness is proven for the special case of two customers. We claim that this result can be transferred to the general case. In Christian Bräuer's bachelor thesis (Bräuer, 2016), the reported mean run-time of an unaccelerated exact approach on instances with 6 customers is more than half an hour. Pre-empting the results of our experiments a little, this is an indicator that we can expect a similar speed-up factor with 6 customers as with 2 customers. Due to the long run-time of the unaccelerated exact approach, we forgo further investigating the exact speed-up factor in the general case.

- Our algorithms, especially the exact approach, can be further sped up by only taking a subset of parking locations into account, e.g., those with better amenities and/or those with a higher chance of being available on arrival. On one hand, this may decrease the solution quality and even the number of solvable queries. On the other hand, besides the speed-up, this may raise the user's satisfaction with a solution, provided that a feasible solution can still be found. In chapter 7, we examine the subset of parking locations with at least 30 parking bays each. This subset consists of less than 12% of the locations in the original set. We find that this filter has a good trade-off between run-time and quality. Again, we claim that this result is also applicable to the general case with more than two customers. However, we only consider the complete parking set in the following. It is an interesting line of research to investigate the impact of different filters on quality and run-time. But this is not in the focus of this chapter.

- In chapter 7, we also compare the run-time of the exact approach (for the special case of two customers) on two different road graphs representing the road network of Germany. Predominantly, these two road graphs differ in the number of edges with time-dependent information as well as the number of breakpoints of the driving time functions on these edges. Even though the road graph *Germany 2017* has less than 7 times more breakpoints than the other road graph, we find that the exact approach takes even more than 40 times longer on this instance. This increase may be explained by the superpolynomial (worst-case) complexity of driving time profiles. We expect that this result can be transferred to the general case. We restrict ourselves to the computationally harder road graph *Germany 2017* in the following.

The focus of our experiments is on comparing the run-time and the solution quality of the exact approach against the heuristic (section 8.6.2). But before we can

do that, we need to have test queries. Two types of test queries are introduced in section 8.6.1.

8.6.1 Test Setup

We test our algorithms on two different types of queries: Randomly generated queries and real-world queries.

Random Queries Each of the random queries contains a sequence of six customers. These are chosen as follows: The first customer is picked at random. Next, with the help of a (limited) one-to-many *lower profile bound query*, we randomly pick the next customer among those vertices that are between 95% and 100% of 8/5h away (with respect to the free-flow driving time) from the current customer. We repeat this step four more times until six customers are chosen in total.

By construction, consecutive customers are at least a drive of 8/5h (minus 5%) apart, so it is not feasible to visit four in a row without taking a break in between. The total driving time is at least 8h (minus 5%) but it can be expected to be higher due to congested road segments. Since we are interested in examining the general behavior of the algorithms, we disregard the service times and time windows, i.e., all time windows are always open and every service time is 0. The beginning of the planning horizon is randomly selected between 5 and 7 in the morning of a Tuesday, and the end is set such that it is non-restrictive. We generate 1000 queries this way.

Real-World Queries With real-world queries, we want to investigate the impact of time windows and other aspects that would otherwise be hard to set realistically. Our real-world queries are taken from PTV Drive & Arrive[3]. This is a cloud-based service to distribute information about the current estimated time(s) of arrival to all stakeholders in the supply chain, and it is integrated into the route optimization suite PTV Route Optimiser ST[4].

A user of this software suite may transfer planned routes to PTV Drive & Arrive in order to activate tracking of the vehicle. However, tracking information is not of interest here. For our test queries, we are only interested in the sequence of locations as well as the time window and the service time per location. It is unfortunate for our purposes that PTV Drive & Arrive only supports a single time window because we cannot investigate the effects that multiple time windows per customer may create. In a PTV Drive & Arrive request that originates from PTV Route Optimiser ST, we may also be given *planned* times of arrival and a *planned* start time of the route. These times refer to the (time-independent) scheduling component of the route optimization suite and are ignored here.

We picked three different users of PTV Drive & Arrive (and PTV Route Optimiser ST). At some day in March 2018, we selected the most recent queries of these users with the additional properties that all customers are within Germany and the designated vehicle is a heavy goods vehicle (gross vehicle mass of at least 7.5t). We refer to these test query sets as RW-1, RW-2, and RW-3 in the following, and to the union of these three as RW-Total.

Tables 8.3, 8.4, and 8.5 summarize the main properties of all test query sets: Table 8.3 lists the mean number of customers per query and the total number of queries in each test query set. Table 8.4 reports key figures regarding the distribution of the

[3] `https://driveandarrive.ptvgroup.com/`

[4] `https://www.ptvgroup.com/en/solutions/products/ptv-route-optimiser/`

service time in each (real-world) query set. And Table 8.5 provides the same key figures regarding the distribution of the time window length in each of these sets.

In all our queries the beginning of the planning horizon coincides with the beginning of the time window of the first customer, so the planning horizon is not restrictive. In the set RW-1, the first customer mostly opens at 5 a.m., but in 12 cases it is an hour later. In the set RW-2, it is always at 4 a.m., and in the set RW-3, it is always at 5 a.m.. We also notice that in case of the sets RW-1 and RW-3, the planned routes are round trips from the same depot, i.e., the first and the last customer correspond to the very same site in each of the two sets. Such a pattern does not exist in RW-2.

TABLE 8.3: Properties of test query sets: Mean number of customers and number of queries per test query set.

Query set	$\varnothing \lvert\mathcal{C}\rvert$	# Queries
Random	6	1000
RW-1	17.9	120
RW-2	10.8	490
RW-3	7.6	200
RW-Total	11.1	810

TABLE 8.4: Properties of test query sets: Service time (minimum, median, (arithmetic) mean, maximum) in seconds (rounded).

Query set	Min	Median	Mean	Max
RW-1	0	555	670	6434
RW-2	0	620	534	2805
RW-3	0	900	678	3420
RW-Total	0	624	591	6434

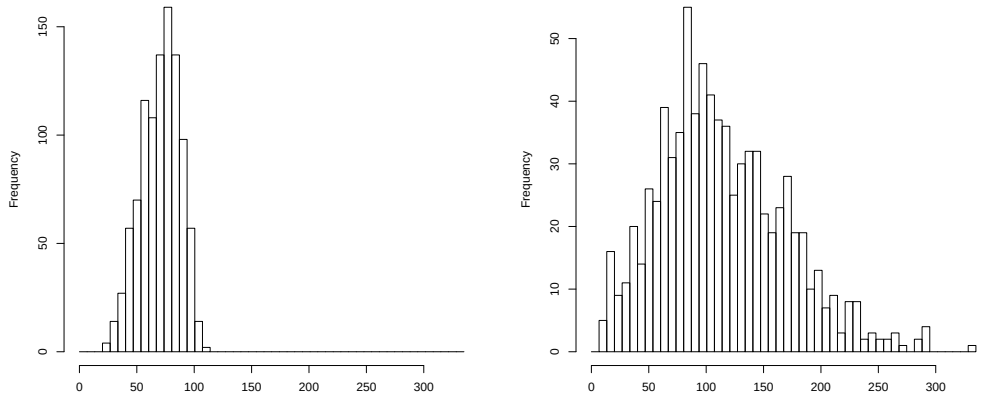
TABLE 8.5: Properties of test query sets: Time window length (minimum, median, (arithmetic) mean, maximum) in hh:mm (rounded).

Query set	Min	Median	Mean	Max
RW-1	0:44	5:21	8:15	24:00
RW-2	2:49	11:50	13:07	24:00
RW-3	1:45	8:45	10:16	24:00
RW-Total	0:44	11:49	11:28	24:00

8.6.2 Experimental Analysis

We examine the exact approach as well as the heuristic on the random test queries and the three real-world test query sets. The focus of the analysis is on the run-time of the exact approach and on the trade-off between run-time and solution quality of the heuristic.

Exact approach on random queries The random queries are constructed in a way such that at least one break is necessary. And should one break also be sufficient, then it has to be a break at a parking location between customers 3 and 4. To reach



(A) Random queries. (B) Real-world queries.

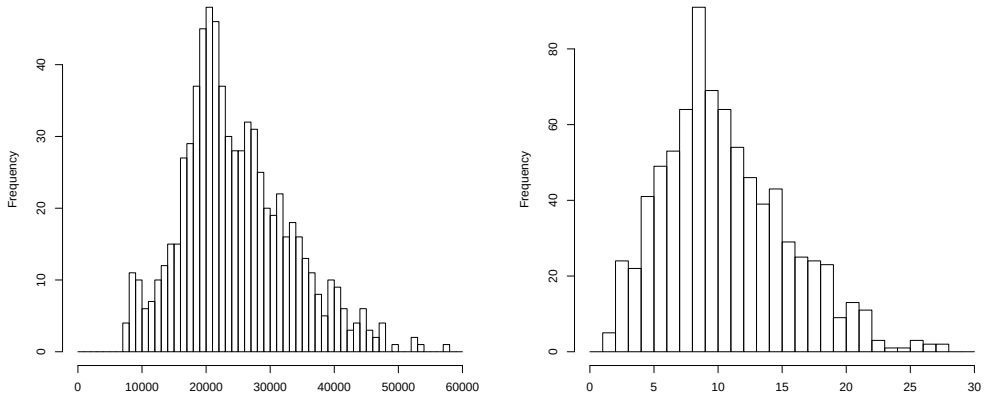
FIGURE 8.13: Run-time histograms for exact approach on random (left) and real-world (right) queries. Run-time in seconds.

that parking location, a detour may be unavoidable. Whether one break is sufficient or not depends on how strong the effect of the congestion is on one hand, and how far such a detour is on the other. That is, how much does the total driving time of a route (considering congestion and possibly including detours) deviate from the total direct free-flow driving time?

As it turns out, the effects of congestion and detours are not too strong, at least not with regard to our instance of the German road network. While the mean total direct free-flow driving time of the random queries is 97.5% of 8h by construction, we find that the mean total (actual) driving time is 8h 17 and thus 6.2% longer. The maximum total (actual) driving time encountered is 8h 47. And so in 999 cases, it suffices to schedule one break at a suitable parking location between customers 3 and 4. Only in one single case a second break is inevitable because such a suitable parking location does not exist.

The average run-time of the exact approach on a random query is 70.5 seconds. Even though the random test queries are rather homogeneous, the run-time varies between 23.1 and 109.4 seconds. A run-time histogram is given in Figure 8.13a. Let us contrast the average run-time of 70.5 seconds with the run-time on instances with exactly two customers as stated in the previous chapter 7. There, the reported mean run-time on instances, for which one break is known to be both necessary and sufficient, is approximately 6 seconds (see Table 7.3). So even if we had to do it 5 times for 5 drives between customers, that would only make about 30 seconds. The increased run-time can be explained by the fact that the profile range of the sub-queries is larger by at least the break length, so the acceleration technique described in section 8.4 is less effective. Even though consecutive customers are closer together than those in chapter 7 so that a significantly smaller part of the graph needs to be investigated, this does not make up for the increased run-time due to the higher number of breakpoints in the profiles.

To make the impact of time-dependency on run-time even clearer, let us assume free-flow driving times throughout the day. If we replace the time-dependent driving time profile on each edge with a constant function and set the free-flow driving time as its value, then the average run-time on a random query drops to only 0.30 seconds.



(A) Total travel time in seconds. (B) Number of customers.

FIGURE 8.14: Histograms on total travel time and on number of customers with respect to the real-world queries.

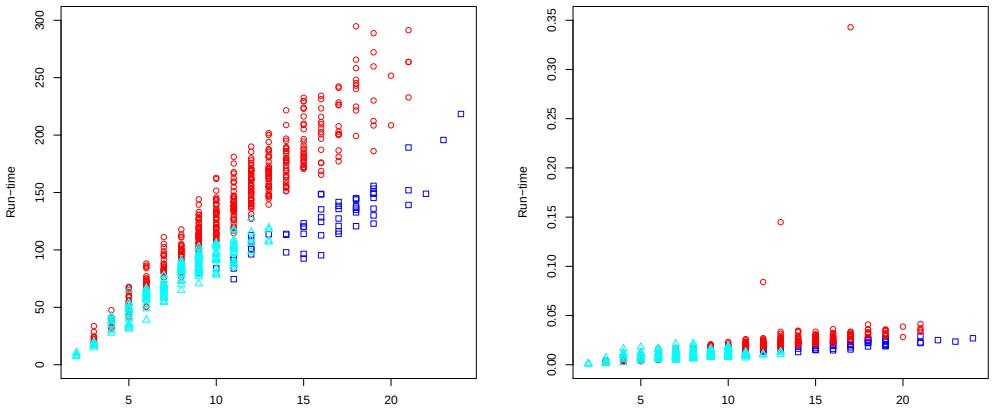
Exact approach on real-world queries Approximately 10% of all real-world queries are not feasible (77 out of 810), simply because one of the time windows is missed (recall that we set the planning horizon to be non-restrictive). Most of the infeasible queries are from RW-1. Of this query set alone, about 50% of the queries are not feasible. In most cases, this is not due to effects caused by congestion as 65 of all 77 infeasible queries remain so even if we ignore time-dependency and assume free-flow driving times. The reason why so many (RW-1) instances are infeasible even under free-flow conditions is not known to us. Most likely, the routes were generated with respect to a profile of a significantly faster vehicle. But it may also be that some time window restrictions were overruled by the user of PTV Route Optimiser ST.

We observe that in many cases, a break is scheduled before the service at the second customer. This is due to the very early beginning of the planning horizon between 4 a.m. and 6 a.m., which is when the first customer's time window opens. Even after service at the first customer and driving to the next, this often still leaves enough time for a break, simply because the second customer is not open yet. As for RW-3, the second customer does not open before 7 a.m. in 182 cases (out of 200), which leaves enough time for a break in 126 cases. Such a break is scheduled even though it may later turn out to be unnecessary.

There are two important limits to observe: One is on the total driving time and the other is on the total travel time. We find 9 instances in which the total travel time exceeds 13 hours, in one case the total travel time is even longer than 15 hours. It should be noted that these instances may not be in accordance with the applicable regulation in the EU because it stipulates that a driver takes a so-called daily rest of 11 hours (in certain cases 9 hours) every 24 hours (see section 2.1.1). If we set the planning horizon restrictive, these instances would also be considered as infeasible. A travel time histogram is given in Figure 8.14a.

The total driving time limit of 9 hours is exceeded in three cases. In two of these, also the total travel time is too high. Only in one case, the found schedule would be feasible with respect to the total travel time but infeasible with respect to the total driving time. The restriction of the total driving time is subject of section 8.7.

The number of customers per query varies a lot, namely between 2 and 28. A



(A) Exact approach. (B) Heuristic.

FIGURE 8.15: Run-time in seconds of exact approach (left) and of heuristic (right) on solvable real-world queries by number of customers in a route. Blue squares for RW-1 queries, red circles for RW-1 queries, and cyan triangles for RW-3 queries.

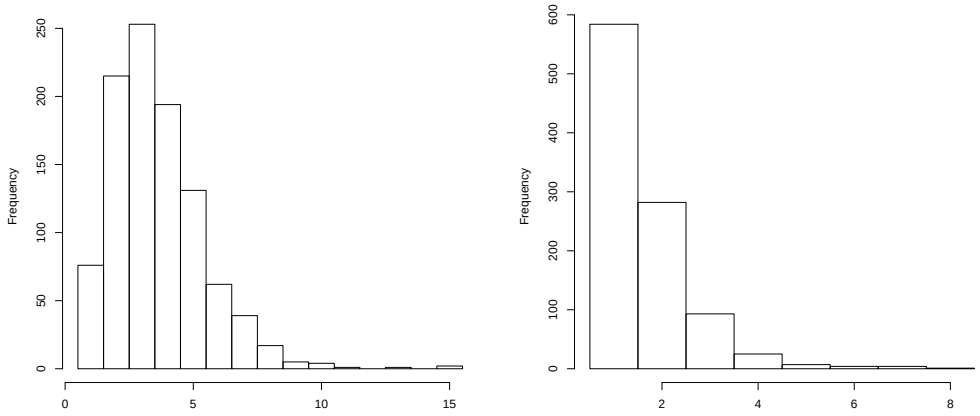
histogram is shown in Figure 8.14b. Accordingly, the run-times are far more spread than those for the random queries (see Figure 8.13a and 8.13b). On average, the run-time is 114.4 seconds. It is higher than the average run-time of a random query because here, the average number of customers is also higher. The mean run-time over the 53 instances with 6 customers alone is only 65.2 seconds and thus within the scope of the run-time on random queries.

In the scatter plot of Figure 8.15a, the run-time of each real-world query is plotted against the number of customers in that query. For queries from RW-1 we use blue squares, for those from RW-2 red circles, and those from RW-3 cyan triangles. As we can deduce from this figure, the run-time grows more or less linearly in the number of customers per query. However, unsolvable queries may run significantly shorter and thus distort the picture. Hence, unsolvable queries are exempted here.

On some instances, the driving time is so short that a break never becomes due. In our implementation of the exact approach, this is not exploited in order to further speed the computation up. In contrast, the run-time of the heuristic benefits from this.

Heuristic on random queries As already described, the random queries are constructed in a way such that a break has to be taken on a parking location if we want to schedule only one break whenever possible. And so we use the enhanced version of the heuristic. The solution quality of this enhanced heuristic is amazingly good on the random query set. In fact, in 998 cases, the travel time matches with the solution computed by the exact approach. Only in two of the 1000 cases, there is a difference in travel time of approximately 45 seconds.

The mean run-time, however, is only 0.54 seconds. It is so much lower than the run-time of the exact approach not only because we omit the expensive linking and merging of driving time profiles, but also because most parking locations can simply be ignored due to dominance rules. The histogram in Figure 8.16a illustrates how many parking lots are contained in the set $\mathcal{P}_1^*$ of "Pareto-optimal" parking locations. As we can see, there are never more than 15, and the cardinality of the set is only



(A) Cardinality of $\mathcal{P}_1^*$. (B) Number of considered parking locations.

FIGURE 8.16: Histograms on the number of parking locations with respect to the heuristic on the random query set.

3.6 on average. The histogram on the right (Figure 8.16b) shows how many of these parking locations are evaluated, until the recursive call to the heuristic in line 17 of the pseudo-code is successful. As we can conclude from this histogram, in many cases it suffices to evaluate only one parking location of the "Pareto set" $\mathcal{P}_1^*$. On average, only 1.6 parking locations are enough, and at most 8.

Heuristic on real-world queries As we already know, only 733 of the 810 real-world queries are solvable. Of these, the heuristic manages to find the optimal solution in 600 cases. In most of the other cases, the solution of the heuristic is only slightly worse. But in three cases, the deviation is close to the break duration of 45 minutes. While the median deviation is only 42.87 seconds, the mean deviation is 379 seconds and the maximum deviation is 2695 seconds.

While the exact approach is slower on the real-world instances than on the random instances, it is the other way round for the heuristic. The mean run-time is only 0.016 seconds and thus more than three orders of magnitude faster. This is because in the vast majority of cases, there is no need to search for a parking location, which is the most time-consuming step of the heuristic. In fact, only in three cases, the backtracking component is used that tries to reduce the number of breaks by scheduling one break at a parking location instead of two at customers. In two of the three cases, this is successful. In one case, two breaks remain scheduled, each at a customer. Figure 8.15b displays the run-time per number of customers. Three outliers are clearly visible. They correspond to exactly those three cases in which the backtracking component is activated.

Concluding remarks Table 8.6 summarizes the run-times again. On the random queries, the heuristic is 130 times faster on average than the exact approach. On the real-world queries, it is even more than 7000 times faster. In case of the random queries, the heuristic finds the optimal solution in 998 out of 1000 cases. In case of the real-world queries, this is true in 600 out of 733 cases.

We can conclude that the heuristic offers an excellent trade-off between run-time and quality. The exact approach takes into account that it may be advantageous to

prolong a break or to not depart from a customer immediately after service (short-term waiting at customers). However, the experiments reveal that the benefit is only marginal and the computational effort – of computing partial profiles even – does not pay off.

It should be noted that we have tested our algorithms on a single road graph and that we rely on the accuracy of the TomTom-based driving time profiles on the graph's edges. This raises the question in how far the conclusions that we draw from the experimental results remain valid when this data changes. In general, the steeper the descent is between two breakpoints of a profile, the shorter the driver has to wait for a significant drop of the driving time and the more beneficial could be short-term waiting. This means that as long as the number of the profiles' pieces with a steep descent does not increase substantially and/or their descent becomes even steeper, we claim that our conclusions still hold. In other words, as long as the driving times decrease due to dispersing congestion and not due to lifted road closures for instance, we expect our conclusions to remain valid in practice.

TABLE 8.6: Mean run-time of different algorithms on different query sets (in seconds).

	Random	RW-Total
Exact	70.5	114.4
Heuristic	0.54	0.016
Exact (free-flow)	0.30	0.58

8.7 Discussion of an Additional Constraint on the Total Driving Time

As pointed out in the introduction, there is an extension of the problem that we also want to shed light on. Both in the EU and the US, there may also be a given *maximum total driving time*. For instance, the EU stipulates a limit on the total daily driving time of 9 hours (even though it allows a total driving time of up to 10 hours twice a week (European Parliament and Council of the European Union, 2006)). This means a solution may be feasible according to our problem statement but it may still not be *legal*. Let *limitTotal* denote such a limit on total driving time. With this, a feasible schedule is only considered as legal if the sum $\sum_{k=1}^{\ell} \mathcal{A}[k] - \mathcal{D}[k]$ of driving times along the ℓ route segments does not breach that limit, i.e., we set as an additional constraint

$$\sum_{k=1}^{\ell} \mathcal{A}[k] - \mathcal{D}[k] \leq \mathit{limitTotal} \tag{8.17}$$

Finding a legal schedule is a more complex problem. And this is already true in the scenario with time-independent driving times. In Figure 8.17, an example road graph with two parking locations is given. In this example, the driving times are time-independent constants and written on the edges. All depicted edges are shorter than *limit*. Let $c + d > \mathit{limit}$ and $a + b + c > \mathit{limit}$, so a break must be scheduled at c_2 and either p_1 or p_2. Now suppose there is only one time window at c_2 that opens not before $a + b + c + w + \mathit{break}$. If the break is taken at p_1, the driver has an accumulated driving time since last break of c on arrival at c_2, which is b less than if the break is taken at p_2. But on the other hand, if a break (of length $\mathit{break} + w$) is taken at p_2, a

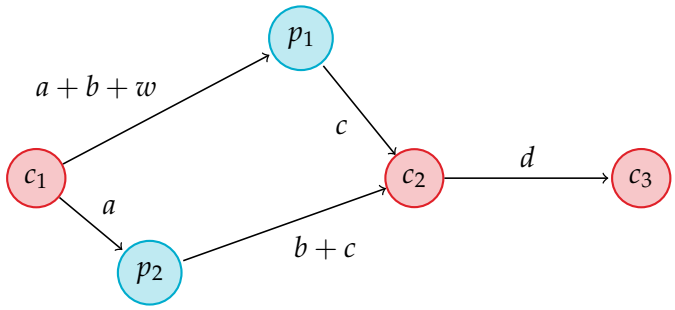


FIGURE 8.17: Example road graph with three customers and two parking locations. Time-independent (and positive) driving times written on edges.

shorter total driving time of only $a+b+c+d$ is possible, w less than when taking the break at the other parking location.

This example is constructed in a way that our exact approach (for the non-extended problem) does not find the solution with a break at p_2. It discards it due to the worse accumulated driving time since last break when the time window of c_2 opens. So should $a+b+c+d+w$ be longer than the maximum total driving time and $a+b+c+d$ be shorter, then this approach does not find a legal solution even though one exists. However, we observe that the heuristic finds that solution in this example. This is because the heuristic prefers the parking location via which the earliest arrival at the next customer can be achieved, which is p_2 in the example.

In practice, it almost always turns out that the returned solution is also legal. This is because both criteria, the minimum accumulated driving time since last break and the minimum accumulated total driving time, are not independent, they both benefit from short driving times in general. And it may not be necessary to include the restriction on the total driving time if the total service time is considerable. With the break rule parameters of the EU, we observe that if all service times sum up to more than 2h 30 (or 1h 30), then the total driving time of a feasible solution cannot exceed 9 hours (or 10 hours) because otherwise at least two breaks of 45 minutes are scheduled and the sum of all driving times, service times, and break times breaches the length of the planning horizon of 13 hours. So in such cases, we could as well ignore the restriction.

With the RW-1 query set, the mean total service time per query is clearly over 2h 30. With the RW-2 query set, the mean total service time is over 1h 30, and with the RW-3 query set, it is only slightly below that. And so from all 810 real-world queries, there is only one case in which the returned solution has a driving time of slightly more than 9 hours and a travel time of less than 13 hours. In this particular case, the returned solution is feasible but is not legal unless the "10-hour-exception" is used.

8.8 Conclusion and Outlook

We have introduced the *truck driver scheduling and routing problem* on road networks, which is the problem of finding both a schedule and a route in the road network such that the route connects the customers to be visited and (if need be) some parking locations, the schedule complies with the applicable legislation, the customers are visited within one of their time windows, and breaks are only taken at customers or parking locations. For this problem, both an exact method and a heuristic have been

presented. Our focus has been on the time-dependent scenario in which the driving times depend on the time of day.

For the exact method, we have described how it can be accelerated by computing bounds in order to only conduct profile range searches. We evaluated both methods on a road network of Germany. We found out that the heuristic is at least two orders of magnitude faster and still finds the optimal solution in the majority of the cases. In fact, the mean run-time of the heuristic is well below one second and thus certainly fast enough to be applied in practice.

Outlook For future research, we see several interesting directions.

In this chapter, we have concentrated on predictable congestion. For instance, a higher driving time can be expected on roads in and around cities during the rush hours in the morning of every workday. But what about a traffic jam in consequence of an accident? Such events remain disregarded in our setting. Of course, we could re-calculate the CH search graph every time new information about the current traffic situation comes in. However, the computation of the search graph takes too long to be used in this fashion. A solution could be *time-dependent customizable contraction hierarchies*, which is an on-going research subject. In general, customizable contraction hierarchies (Dibbelt, Strasser, and Wagner, 2016) is an extension of contraction hierarchies, where the pre-processing step is divided into two phases. In the first phase, only the unweighted topology of the graph is exploited. In the second phase, the auxiliary data is adapted to a specific weight. In a real-time scenario, only the second phase has to be called again.

Another open question for research is how to take the features of parking areas (including those at customers) into account. At some of them, there may be restaurants and convenience stores, whereas some others may not offer any public facilities at all. Depending on the amenities they provide, some parking areas may be more popular among drivers than others. But it is not only about the facilities, also safety could be an issue. In general, we see it as an interesting line of research to include the fact that some parking areas are "better" than others into the objective function.

In practice, parking areas may be occupied. The easiest way to take that into account would be to remove those parking areas from consideration that are presumably occupied. But this neglects the temporal dimension. Suppose we had an oracle that tells us for every point in time whether a parking space is available. Then it would be another interesting research subject to include that oracle into our algorithm. It should be noted that an additional complexity comes in as the driver cannot wait for a parking space to become available when the parking area is occupied. In practice, we are far from having such an oracle, unfortunately. One exception are parking spaces that can be reserved via a booking system such as the one integrated into Truck Parking Europe.

Chapter 9

Conclusion and Outlook

In this thesis, we have dealt with various scheduling and routing problems that arise in the context of regulations on drivers' working hours. Towards the end of it, we summarize the main findings again (section 9.1) and review the directions for further research that we find the most interesting (section 9.2).

9.1 Conclusion

In this thesis, we have focused on the regulations that are effective in the European Union and the United States. First, we have classified the most important break rules and their parameters. Then, we have investigated three classes of optimization problems: the truck driver scheduling problem, the vehicle routing and truck driver scheduling problem, and the truck driver scheduling and routing problem. In the following, let us recapitulate the research questions.

9.1.1 Truck Driver Scheduling

In the truck driver scheduling problem, we are given some customers and a sequence in which these customers are to be visited. On one hand, all customers must be visited within their time windows, and on the other hand, some break rules need to be respected. As far as this problem is concerned, the main research question was:

Which problem variants of the truck driver scheduling problem can (still) be solved in polynomial time - and how?

To this end, we have presented polynomial-time algorithms for several problem variants. All currently known results regarding a polynomial-time bound only hold in the single time window case. Hence, we have focused solely on the multiple time window case, where the number of time windows per customer is not limited. We make no assumptions on the lengths of the time windows or the time between consecutive time windows. It is a common feature of our polynomial-time algorithms to regard time-dependent and piecewise linear functions in order to store non-dominated driver states.

Table 9.1 summarizes the main results. We have observed several rulesets as they are relevant both in the European Union and the United States. Precisely, we have considered the rulesets *RulesetUS* = {*drive until driven*, *drive until traveled* } and *RulesetEU* = {*drive until driven*, *work until traveled* } as well as *RulesetEU+* = {*drive until driven*, *work until traveled*, *first-second-split* }, according to our classification of break rules presented in section 2.3. Thus, our algorithms belong to those few that can be used both in the EU and the US without major changes.

In case two types of breaks need to be distinguished, two rulesets must be specified, one for the short break and one for the long break. In case of a single type of

TABLE 9.1: Main results of this thesis regarding polynomial-time bounds of truck driver scheduling problems with multiple time windows per customer (compare Table 3.1).

Short break	Long break	Policy	Goal	Complexity	Chapter
RulesetUS	-	non-restrictive	EF	$O(nw)$	3
RulesetEU	-	non-restrictive	EF	$O(nw)$	3
RulesetEU+	-	non-restrictive	EF	$O(nw)$	3
RulesetUS	-	non-restrictive	MD	$O(n^3w^2)$	4
RulesetEU	-	non-restrictive	MD	$O(n^3w^2)$	4
RulesetEU	*RulesetEU*	no-break-en-route	EF	$O(n^3w)$	5
RulesetUS	*RulesetUS*	no-break-en-route	EF	$O(n^3w)$	5

break, we always regard a non-restrictive break policy, that is, breaks can be taken at any time, particularly, while en route between two customers. In case of two types of breaks, the polynomial-time bound requires the no-break-en-route policy. This policy only allows drivers to take breaks while at a customer, that is, before or after service at that customer.

We regard two different optimization goals. One is to find the earliest finish time (EF) of a schedule, that is, the earliest completion time of the service at the last customer. The other goal is to find the minimum duration (MD) of a schedule, that is, the shortest time between the start time of the service at the first customer and the completion time of the service at the final customer. The EF objective is asymptotically not more complex that the decision problem that only asks whether a feasible schedule exists. However, the MD objective is more complex.

In Table 9.1, the time complexity is given with respect to the number of customers n and the total number of time windows w. If every customer had only a single time window, we could replace w by n. With the polynomial-time bound for the minimum duration variants, we falsify the *NP*-hardness conjecture of Xu et al. (2003) for an important special case.

The secondary research question

Which variants can no longer be solved in strongly polynomial time?

is not definitively answered. But one variant that is a promising candidate is the problem with two types of breaks together with a non-restrictive break policy. At least we have shown by way of example that the number of non-dominated driver states does not solely depend on the number of customers and time windows but also on the setting of the break rule parameters and the driving times. While this is not a proof, it is still an indication that a strongly polynomial algorithm may not exist for this problem variant.

9.1.2 Vehicle Routing and Truck Driver Scheduling

In the vehicle routing and truck driver scheduling problem, the truck driver scheduling problem appears as a subproblem. It needs to be solved as part of the feasibility check inside an algorithm for the vehicle routing problem. Since the literature on the family of vehicle routing problems already contains a plethora of solution approaches, we had focused on the aspect of how to integrate the feasibility check regarding the subproblem into a solution framework for the vehicle routing problem. The research question was:

How can the feasibility in respect of drivers' working hours be checked efficiently within local search based heuristics for the vehicle routing and truck driver scheduling problem?

The efficient feasibility check that we have described in this thesis consists of two ingredients. One ingredient is storing information on partial routes in order to prevent that the same information is computed over and over again while evaluating the neighborhood of the incumbent solution. The second ingredient is the bidirectional propagation of this information. While both ingredients are not new in general, they have not been described in the context of our problem variant at hand. This variant is characterized by multiple time windows per customer, the necessary break rules for day trips in the EU, and a general break policy that allows breaks en route at any time.

9.1.3 Truck Driver Scheduling and Routing

In the truck driver scheduling and routing problem, we are given a road graph. Some of the vertices correspond to customers and some others to parking areas. When a break is scheduled, it must be scheduled either at a customer or at a parking area. For a given order in which the customers are to be visited, we need to find a route in the road graph and a corresponding schedule such that the finish time is earliest possible. For an even more realistic setting, we also regard predictable congestion. The research question was formulated as follows:

Given real-world data like a road graph representing the German road network and information on predictable congestion, how quickly can optimal truck driver routes and schedules be computed, and what is a good trade-off between run-time and solution quality (and memory consumption)?

The short answer is that it takes several seconds to respond to an exact query, and this holds despite our acceleration efforts. In the worst case, a respond may take up to a few minutes even. The main reason is that the calculation of driving time profiles is computationally very expensive. Our mentioned acceleration efforts refer to two applied techniques. One technique is to use time-dependent contraction hierarchies. Here, we also exploit the limitation that we allow to stop for a break only once between consecutive customers. Another technique is to compute upper and lower bounds on arrival times. With these bounds, we know over which ranges we do not need to calculate the driving time profiles. Both techniques preserve optimality.

We have conducted all our experiments on the basis of the road network of Germany and congestion data provided by TomTom (and post-processed by PTV). Besides exact approaches, we have presented heuristics. These heuristics do not consider short-term waiting or the prolongation of a break. They are very close to what the driver would do in practice. And they are significantly faster. On real-world queries with several customers, the respective heuristic performed three order of magnitude faster on average. Moreover, the optimal solution was found in 600 out of 733 cases. To sum up, we find that the heuristics perform extremely well and offer an excellent trade-off between run-time and quality.

As far as the trade-off between run-time and memory consumption is concerned, some questions remain open and are discussed in the following outlook.

9.2 Outlook

Truck Driver Scheduling A rather obvious direction for future research is to combine some of our presented ideas and show that even more variants of the TDSP can be solved in polynomial time. For instance, a polynomial-time approach for the TDSP-2B with no-break-en-route policy and with minimum duration objective would certainly be valuable. On the other hand, the conjecture of Drexl and Prescott-Gagnon (2010) is still open, just like our conjecture that a strongly polynomial-time algorithm for the TDSP-2B with an unrestricted break policy does not exist.

Vehicle Routing and Truck Driver Scheduling At PTV, we have already extended the presented approach in order to also consider a second type of break as it is necessary for a planning horizon of several days. It is another important variant to not consider the rule *drive until driven* but the rule *drive until traveled* as it is needed for the lunch break rule in the United States.

Truck Driver Scheduling and Routing We have used time-dependent contraction hierarchies as a speed-up technique in our experiments, both for our exact and our heuristic approaches. However, the created search graph has become quite large, approximately 40 GB for a recent road network of Germany. For a world map, the size would be prohibitive. So one line of research is to find ways to reduce the memory consumption of the search graph.

Another direction for future research is it to consider temporary road closures and/or driving bans. Recently, Tuin, Weerdt, and Batz (2018) investigated the case with truck driving bans where waiting is allowed everywhere and at any time. Bräuer (2018) studies the case with temporary road closures where waiting is only allowed at dedicated parking locations, that is, the FIFO property does not hold. However, break rules are disregarded here. In both works, the driving time along an edge is supposed to be time-independent. The consideration of temporary road closures and driving bans is both challenging and rewarding since it is certainly relevant in practice.

Bibliography

ADAC (2018). *ADAC Staubilanz 2017*. URL: https://www.adac.de/der-adac/verein/aktuelles/staubilanz-2017/.

Archetti, Claudia and Martin Savelsbergh (2009). "The Trip Scheduling Problem". In: *Transportation Science* 43.4, pp. 417–431. DOI: 10.1287/trsc.1090.0278. eprint: http://dx.doi.org/10.1287/trsc.1090.0278. URL: http://dx.doi.org/10.1287/trsc.1090.0278.

Baltz, Andreas, Mourad El Ouali, Gerold Jäger, Volkmar Sauerland, and Anand Srivastav (2015). "Exact and heuristic algorithms for the Travelling Salesman Problem with Multiple Time Windows and Hotel Selection". In: *Journal of the Operational Research Society* 66.4, pp. 615–626. ISSN: 1476-9360. DOI: 10.1057/jors.2014.17. URL: https://doi.org/10.1057/jors.2014.17.

Bartodziej, P., U. Derigs, D. Malcherek, and U. Vogel (Apr. 2009). "Models and algorithms for solving combined vehicle and crew scheduling problems with rest constraints : an application to road feeder service planning in air cargo transportation". In: *OR Spectrum* 31.2, pp. 405–429. ISSN: 1436-6304. DOI: 10.1007/s00291-007-0110-7. URL: https://doi.org/10.1007/s00291-007-0110-7.

Bast, Hannah, Daniel Delling, Andrew V. Goldberg, Matthias Müller–Hannemann, Thomas Pajor, Peter Sanders, Dorothea Wagner, and Renato F. Werneck (2016). "Route Planning in Transportation Networks". In: *Algorithm Engineering - Selected Results and Surveys*. Vol. 9220. Lecture Notes in Computer Science. Springer, pp. 19–80.

Batz, Gernot V., Robert Geisberger, Peter Sanders, and Christian Vetter (2013). "Minimum Time-Dependent Travel Times with Contraction Hierarchies". In: *ACM Journal of Experimental Algorithmics* 18, 1.4:1–1.4:43.

Batz, Gernot Veit. *KaTCH*. URL: https://github.com/GVeitBatz/KaTCH/.

Baum, Moritz, Julian Dibbelt, Andreas Gemsa, Dorothea Wagner, and Tobias Zündorf (2015). "Shortest Feasible Paths with Charging Stops for Battery Electric Vehicles". In: *Proceedings of the 23rd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (GIS'15)*. ACM, 44:1–44:10.

Baum, Moritz, Julian Dibbelt, Thomas Pajor, and Dorothea Wagner (2016). "Dynamic Time-Dependent Route Planning in Road Networks with User Preferences". In: *Proceedings of the 15th International Symposium on Experimental Algorithms (SEA'16)*. Vol. 9685. Lecture Notes in Computer Science. Springer, pp. 33–49.

Beaudry, Alexandre, Gilbert Laporte, Teresa Melo, and Stefan Nickel (2010). "Dynamic transportation of patients in hospitals". In: *OR Spectrum* 32.1, pp. 77–107. ISSN: 1436-6304. DOI: 10.1007/s00291-008-0135-6. URL: https://doi.org/10.1007/s00291-008-0135-6.

Bernhardt, Alexandra, Teresa Melo, Thomas Bousonville, and Herbert Kopfer (2016). *Scheduling of driver activities with multiple soft time windows considering European regulations on rest periods and breaks*. Tech. rep. 12. Hochschule für Technik und Wirtschaft des Saarlandes. DOI: 10.13140/RG.2.2.27918.77122.

— (2017). *Truck driver scheduling with combined planning of rest periods, breaks and vehicle refueling*. Tech. rep. 14. Hochschule für Technik und Wirtschaft des Saarlandes.

DOI: 10.13140/RG.2.2.13781.73448. URL: http://hdl.handle.net/10419/175088.

Bowden, Zachary E. and Cliff T. Ragsdale (2018). "The truck driver scheduling problem with fatigue monitoring". In: *Decision Support Systems* 110, pp. 20 –31. ISSN: 0167-9236. DOI: 10.1016/j.dss.2018.03.002. URL: http://www.sciencedirect.com/science/article/pii/S0167923618300484.

Braekers, Kris, Katrien Ramaekers, and Inneke Van Nieuwenhuyse (2016). "The vehicle routing problem: State of the art classification and review". In: *Computers & Industrial Engineering* 99, pp. 300 –313. ISSN: 0360-8352. DOI: 10.1016/j.cie.2015.12.007. URL: http://www.sciencedirect.com/science/article/pii/S0360835215004775.

Bräuer, Christian (2016). "Optimale zeitabhängige Pausenplanung für LKW-Fahrer mit integrierter Parkplatzwahl". Bachelor thesis. Karlsruhe Institute of Technology.

— (2018). "Route Planning with Temporary Road Closures". Master thesis. Karlsruhe Institute of Technology.

Campbell, Ann Melissa and Martin Savelsbergh (2004). "Efficient Insertion Heuristics for Vehicle Routing and Scheduling Problems". In: *Transportation Science* 38.3, pp. 369–378. DOI: 10.1287/trsc.1030.0046. eprint: https://pubsonline.informs.org/doi/pdf/10.1287/trsc.1030.0046. URL: https://pubsonline.informs.org/doi/abs/10.1287/trsc.1030.0046.

Ceselli, Alberto, Giovanni Righini, and Matteo Salani (2009). "A Column Generation Algorithm for a Rich Vehicle-Routing Problem". In: *Transportation Science* 43.1, pp. 56–69. DOI: 10.1287/trsc.1080.0256. eprint: https://doi.org/10.1287/trsc.1080.0256. URL: https://doi.org/10.1287/trsc.1080.0256.

Coelho, Leandro C, Jean-Philippe Gagliardi, Jacques Renaud, and Angel Ruiz (May 2016). "Solving the vehicle routing problem with lunch break arising in the furniture delivery industry". In: *Journal of the Operational Research Society* 67.5, pp. 743–751. ISSN: 1476-9360. DOI: 10.1057/jors.2015.90. URL: https://doi.org/10.1057/jors.2015.90.

Cooke, Kenneth L. and Eric Halsey (1966). "The Shortest Route Through a Network with Time-Dependent Internodal Transit Times". In: *Journal of Mathematical Analysis and Applications* 14.3, pp. 493–498.

Dantzig, George B. (1963). *Linear Programming and Extensions*. Princeton University Press.

Dean, Brian C. (2004). "Algorithms for Minimum-Cost Paths in Time-Dependent Networks with Waiting Policies". In: *Networks* 44.1, pp. 41–46.

Delling, Daniel (2011). "Time-Dependent SHARC-Routing". In: *Algorithmica* 60.1, pp. 60–94.

Delling, Daniel and Giacomo Nannicini (2012). "Core Routing on Dynamic Time-Dependent Road Networks". In: *Informs Journal on Computing* 24.2, pp. 187–201.

Delling, Daniel and Dorothea Wagner (2009). "Time-Dependent Route Planning". In: *Robust and Online Large-Scale Optimization*. Vol. 5868. Lecture Notes in Computer Science. Springer, pp. 207–230.

Derigs, Ulrich, René Kurowsky, and Ulrich Vogel (2011). "Solving a real-world vehicle routing problem with multiple use of tractors and trailers and EU-regulations for drivers arising in air cargo road feeder services". In: *European Journal of Operational Research* 213.1, pp. 309 –319. ISSN: 0377-2217. DOI: 10.1016/j.ejor.2011.03.032. URL: http://www.sciencedirect.com/science/article/pii/S0377221711002694.

Dibbelt, Julian, Ben Strasser, and Dorothea Wagner (2016). "Customizable Contraction Hierarchies". In: *ACM Journal of Experimental Algorithmics* 21, 1.5:1–1.5:49.

Dijkstra, Edsger W. (1959). "A Note on Two Problems in Connexion with Graphs". In: *Numerische Mathematik* 1.1, pp. 269–271.

Drexl, Michael and Eric Prescott-Gagnon (2010). "Labelling algorithms for the elementary shortest path problem with resource constraints considering EU drivers' rules". In: *Logistics Research* 2.2, pp. 79–96. ISSN: 1865-0368. DOI: 10.1007/s12159-010-0022-9. URL: http://dx.doi.org/10.1007/s12159-010-0022-9.

Drexl, Michael, Julia Rieck, Thomas Sigl, and Bettina Press (Nov. 2013). "Simultaneous Vehicle and Crew Routing and Scheduling for Partial- and Full-Load Long-Distance Road Transport". In: *Business Research* 6.2, pp. 242–264. ISSN: 2198-2627. DOI: 10.1007/BF03342751. URL: https://doi.org/10.1007/BF03342751.

Dreyfus, Stuart E. (1969). "An Appraisal of Some Shortest-Path Algorithms". In: *Operations Research* 17.3, pp. 395–412.

European Parliament and Council of the European Union (Mar. 2002). "Directive 2002/15/EC of the European Parliament and of the Council of 11 March 2002 on the organisation of the working time of persons performing mobile road transport activities". In: *Official Journal of the European Union* L 80, pp. 35–39.

— (Nov. 2003). "Directive 2003/88/EC of the European Parliament and of the Council of 4 November 2003 concerning certain aspects of the organisation of working time". In: *Official Journal of the European Union* L 299, pp. 9–19.

— (Apr. 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." In: *Official Journal of the European Union* L 102.1, pp. 1–13.

European Transport Safety Council (2001). *The Role of Driver Fatigue in Commercial Road Transport Crashes*. https://etsc.eu/wp-content/uploads/The-role-of-driver-fatigue-in-commercial-road-transport-crashes.pdf.

Federal Motor Carrier Safety Administration (May 2000). "Hours of Service of Drivers; Driver Rest and Sleep for Safe Operations; Proposed Rule". In: *Federal Register* 65.85, pp. 25539–25611. URL: https://federalregister.gov/a/00-10703.

— (2008). "Hours of Service of Drivers". In: *Federal Register* 73.224, pp. 69569–69586. URL: https://federalregister.gov/a/E8-27437.

— (2011). "Hours of Service of Drivers". In: *Federal Register* 76.248, pp. 81133–81188. URL: https://federalregister.gov/a/2011-32696.

Foschini, Luca, John Hershberger, and Subhash Suri (2014). "On the Complexity of Time-Dependent Shortest Paths". In: *Algorithmica* 68.4, pp. 1075–1097.

Franceschetti, Anna, Doroth'ee Honhon, Tom Van Woensel, Tolga Bektaş, and Gilbert Laporte (2013). "The Time-Dependent Pollution-Routing Problem". In: *Transportation Research Part B: Methodological* 56, pp. 265–293.

Fredman, Michael L. and Robert E. Tarjan (1987). "Fibonacci Heaps and Their Uses in Improved Network Optimization Algorithms". In: *Journal of the ACM* 34.3, pp. 596–615.

Geisberger, Robert and Peter Sanders (2010). "Engineering Time-Dependent Many-to-Many Shortest Paths Computation". In: *Proceedings of the 10th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS'10)*. Vol. 14. OpenAccess Series in Informatics (OASIcs).

Geisberger, Robert, Peter Sanders, Dominik Schultes, and Christian Vetter (2012). "Exact Routing in Large Road Networks Using Contraction Hierarchies". In: *Transportation Science* 46.3, pp. 388–404.

Goel, A., T. Vidal, and A.L. Kok (2019). *To team up or not – Single versus team driving in European road freight transport*. Tech. rep. Rio de Janeiro, Brasil: PUC–Rio. URL: https://w1.cirrelt.ca/~vidalt/papers/Team-vs-Single.pdf.

Goel, Asvin (2009). "Vehicle Scheduling and Routing with Drivers' Working Hours". In: *Transportation Science* 43.1, pp. 17–26. DOI: 10.1287/trsc.1070.0226. eprint: https://doi.org/10.1287/trsc.1070.0226. URL: https://doi.org/10.1287/trsc.1070.0226.

— (2010). "Truck driver scheduling in the European Union". In: *Transportation Science* 44.4, pp. 429–441.

— (Dec. 2012a). "A mixed integer programming formulation and effective cuts for minimising schedule durations of Australian truck drivers". In: *Journal of Scheduling* 15.6, pp. 733–741. ISSN: 1099-1425. DOI: 10.1007/s10951-012-0282-0. URL: https://doi.org/10.1007/s10951-012-0282-0.

— (2012b). "The Canadian minimum duration truck driver scheduling problem". In: *Computers & Operations Research* 39.10, pp. 2359 –2367. ISSN: 0305-0548. DOI: 10.1016/j.cor.2011.12.016. URL: http://www.sciencedirect.com/science/article/pii/S0305054811003728.

— (2012c). "The minimum duration truck driver scheduling problem". In: *EURO Journal on Transportation and Logistics* 1.4, pp. 285–306.

— (2014). "Hours of Service Regulations in the United States and the 2013 Rule Change". In: *Transport Policy* 33, pp. 48–55. ISSN: 0967-070X. DOI: 10.1016/j.tranpol.2014.02.005.

— (2018). "Legal aspects in road transport optimization in Europe". In: *Transportation Research Part E: Logistics and Transportation Review* 114, pp. 144 –162. ISSN: 1366-5545. DOI: 10.1016/j.tre.2018.02.011. URL: http://www.sciencedirect.com/science/article/pii/S1366554517309936.

Goel, Asvin, Claudia Archetti, and Martin Savelsbergh (2012). "Truck driver scheduling in Australia". In: *Computers & Operations Research* 39.5, pp. 1122 –1132. ISSN: 0305-0548. DOI: 10.1016/j.cor.2011.05.021. URL: http://www.sciencedirect.com/science/article/pii/S0305054811001559.

Goel, Asvin and Stefan Irnich (2017). "An Exact Method for Vehicle Routing and Truck Driver Scheduling Problems". In: *Transportation Science* 51.2, pp. 737–754. DOI: 10.1287/trsc.2016.0678. eprint: https://doi.org/10.1287/trsc.2016.0678. URL: https://doi.org/10.1287/trsc.2016.0678.

Goel, Asvin and Leendert Kok (2012a). "Efficient Scheduling of Team Truck Drivers in the European Union". In: *Flexible Services and Manufacturing Journal* 24.1, pp. 81–96. ISSN: 1936-6582. DOI: 10.1007/s10696-011-9086-3.

— (2012b). "Truck Driver Scheduling in the United States". In: *Transportation Science* 46.3, pp. 317–326. DOI: 10.1287/trsc.1110.0382. eprint: http://dx.doi.org/10.1287/trsc.1110.0382. URL: http://dx.doi.org/10.1287/trsc.1110.0382.

Goel, Asvin and Louis-Martin Rousseau (Dec. 2012). "Truck driver scheduling in Canada". In: *Journal of Scheduling* 15.6, pp. 783–799. ISSN: 1099-1425. DOI: 10.1007/s10951-011-0249-6. URL: https://doi.org/10.1007/s10951-011-0249-6.

Goel, Asvin and Thibaut Vidal (2014). "Hours of Service Regulations in Road Freight Transport: An Optimization-Based International Assessment". In: *Transportation Science* 48 (3), pp. 391–412.

Hashimoto, Hideki, Mutsunori Yagiura, and Toshihide Ibaraki (May 2008). "An iterated local search algorithm for the time-dependent vehicle routing problem with time windows". In: *Discrete Optimization* 5.2, pp. 434–456. DOI: doi:10.1016/j.disopt.2007.05.004.

Ibaraki, T., S. Imahori, M. Kubo, T. Masuda, T. Uno, and M. Yagiura (May 2005). "Effective Local Search Algorithms for Routing and Scheduling Problems with General Time-Window Constraints". In: *Transportation Science* 39.2, pp. 206–232. ISSN: 1526-5447. DOI: 10.1287/trsc.1030.0085. URL: http://dx.doi.org/10.1287/trsc.1030.0085.

Imai, Hiroshi and Masao Iri (1987). "An Optimal Algorithm for Approximating a Piecewise Linear Function". In: *Journal of Information Processing* 9.3, pp. 159–162.

Jong, Cor de, Goos Kant, and André van Vliet (1996). *On Finding Minimal Route Duration in the Vehicle Routing Problem with Multiple Time Windows*. Tech. rep. Department of Computer Science, Utrecht University. URL: http://www.cs.uu.nl/research/projects/alcom/wp4.3.html.

Kinz, Monika (2016). "Optimale Pausenplanung von LKW-Fahrern mit integrierter Parkplatzwahl". MA thesis. Technische Universität Darmstadt.

Kleff, Alexander, Christian Bräuer, Frank Schulz, Valentin Buchhold, Moritz Baum, and Dorothea Wagner (2017). "Time-Dependent Route Planning for Truck Drivers". In: *Computational Logistics: 8th International Conference, ICCL 2017, Southampton, UK, October 18-20, 2017, Proceedings*. Ed. by Tolga Bektaş, Stefano Coniglio, Antonio Martinez-Sykora, and Stefan Voß. Cham: Springer International Publishing, pp. 110–126. DOI: 10.1007/978-3-319-68496-3_8. URL: https://doi.org/10.1007/978-3-319-68496-3_8.

Koç, Çagri, Tolga Bektas, Ola Jabali, and Gilbert Laporte (2016). "A comparison of three idling options in long-haul truck scheduling". In: *Transportation Research Part B: Methodological* 93, Part A, pp. 631 –647. ISSN: 0191-2615. DOI: 10.1016/j.trb.2016.08.006. URL: http://www.sciencedirect.com/science/article/pii/S019126151630145X.

Koç, Çağrı, Ola Jabali, and Gilbert Laporte (2017). "Long-Haul Vehicle Routing and Scheduling with Idling Options". In: *Journal of the Operational Research Society* 69.2, pp. 235–246. ISSN: 1476-9360. DOI: 10.1057/s41274-017-0202-y. URL: https://doi.org/10.1057/s41274-017-0202-y.

Kok, A. L., C. M. Meyer, H. Kopfer, and J. M. J. Schutten (2010). "A Dynamic Programming Heuristic for the Vehicle Routing Problem with Time Windows and the European Community Social Legislation". In: *Transportation Science* 44(4), pp. 442–454.

Kok, A.L., E.W. Hans, and J.M.J. Schutten (2011). "Optimizing departure times in vehicle routes". In: *European Journal of Operational Research* 210.3, pp. 579 –587. ISSN: 0377-2217. DOI: http://dx.doi.org/10.1016/j.ejor.2010.10.017. URL: http://www.sciencedirect.com/science/article/pii/S0377221710006582.

Lahyani, Rahma, Mahdi Khemakhem, and Frédéric Semet (2015). "Rich vehicle routing problems: From a taxonomy to a definition". In: *European Journal of Operational Research* 241.1, pp. 1 –14. ISSN: 0377-2217. DOI: 10.1016/j.ejor.2014.07.048. URL: http://www.sciencedirect.com/science/article/pii/S0377221714006146.

Lin, Shen (Dec. 1965). "Computer Solutions of the Traveling Salesman Problem". In: *Bell System Technical Journal* 44.10, pp. 2245–2269. DOI: 10.1002/j.1538-7305.1965.tb04146.x.

Meyer, Christoph Manuel (2011). *Vehicle Routing under Consideration of Driving and Working Hours: A Distributed Decision Making Perspective*. ISBN 978-3-8349-2942-6. Gabler Verlag, Wiesbaden. DOI: 10.1007/978-3-8349-6732-9.

National Academies of Sciences, Engineering, and Medicine (2016). *Commercial Motor Vehicle Driver Fatigue, Long-Term Health, and Highway Safety: Research Needs*. Washington, DC: The National Academies Press. DOI: 10.17226/21921.

Potvin, Jean-Yves and Jean-Marc Rousseau (Dec. 1995). "An Exchange Heuristic for Routeing Problems with Time Windows". In: *Journal of the Operational Research Society* 46.12, pp. 1433–1446. ISSN: 1476-9360. DOI: 10.1057/jors.1995.204. URL: https://doi.org/10.1057/jors.1995.204.

Prescott-Gagnon, Eric, Guy Desaulniers, Michael Drexl, and Louis-Martin Rousseau (2010). "European Driver Rules in Vehicle Routing with Time Windows". In: *Transportation Science* 44.4, pp. 455–473. DOI: 10.1287/trsc.1100.0328. eprint: http://dx.doi.org/10.1287/trsc.1100.0328. URL: http://dx.doi.org/10.1287/trsc.1100.0328.

Rancourt, Marie-Eve, Jean-Francois Cordeau, and Gilbert Laporte (2013). "Long-Haul Vehicle Routing and Scheduling with Working Hour Rules". In: *Transportation Science* 47.1, pp. 81–107. DOI: 10.1287/trsc.1120.0417. eprint: http://pubsonline.informs.org/doi/pdf/10.1287/trsc.1120.0417. URL: http://pubsonline.informs.org/doi/abs/10.1287/trsc.1120.0417.

Sahoo, Surya, Seongbae Kim, Byung-In Kim, Bob Kraas, and Alexander Popov Jr. (Jan. 2005). "Routing Optimization for Waste Management". In: *Interfaces* 35.1, pp. 24–36. ISSN: 0092-2102. DOI: 10.1287/inte.1040.0109. URL: http://dx.doi.org/10.1287/inte.1040.0109.

Savelsbergh, M. W. P. (Dec. 1985). "Local search in routing problems with time windows". In: *Annals of Operations Research* 4.1, pp. 285–305. ISSN: 1572-9338. DOI: 10.1007/BF02022044. URL: https://doi.org/10.1007/BF02022044.

Savelsbergh, Martin and Marc Sol (1998). "Drive: Dynamic Routing of Independent Vehicles". In: *Operations Research* 46.4, pp. 474–490. DOI: 10.1287/opre.46.4.474. eprint: https://pubsonline.informs.org/doi/pdf/10.1287/opre.46.4.474. URL: https://pubsonline.informs.org/doi/abs/10.1287/opre.46.4.474.

Savelsbergh, Martin W. P. (1992). "The Vehicle Routing Problem with Time Windows: Minimizing Route Duration". In: *ORSA Journal on Computing* 4.2, pp. 146–154. DOI: 10.1287/ijoc.4.2.146. eprint: https://doi.org/10.1287/ijoc.4.2.146. URL: https://doi.org/10.1287/ijoc.4.2.146.

Schiffer, Maximilian, Gilbert Laporte, Michael Schneider, and Grit Walther (2017). *The impact of synchronizing driver breaks and recharging operations for electric vehicles*. Tech. rep. GERAD. URL: https://www.gerad.ca/en/papers/G-2017-46.

Shah, Vidit Divyang (2008). "Time dependent truck routing and driver scheduling problem with hours of service regulations". MA thesis. Northeastern University, Massachusetts. URL: http://hdl.handle.net/2047/d10016995.

Sherali, Hanif D., Kaan Ozbay, and Shivaram Subramanian (1998). "The Time-Dependent Shortest Pair of Disjoint Paths Problem: Complexity, Models, and Algorithms". In: *Networks* 31.4, pp. 259–272.

Smith, Olivia J., Natashia Boland, and Hamish Waterer (2012). "Solving Shortest Path Problems with a Weight Constraint and Replenishment Arcs". In: *Computers & Operations Research* 39.5, pp. 964–984.

Sommer, Christian (2014). "Shortest-Path Queries in Static Networks". In: *ACM Computing Surveys* 46.4, 45:1–45:31.

Statistisches Bundesamt (Destatis) (Nov. 2017). *Verkehrsunfälle: Unfälle von Güterkraftfahrzeugen im Straßenverkehr*. https://www.destatis.de/.

Strasser, Ben (2016). *Intriguingly Simple and Efficient Time-Dependent Routing in Road Networks*. Technical report abs/1606.06636. ArXiv e-prints.

Tilk, Christian (2016). *Branch-and-price-and-cut for the vehicle routing and truck driver scheduling problem*. Technical Report LM-2016-04. Mainz, Germany: Chair of Logistics Management, Gutenberg School of Management and Economics, Johannes Gutenberg University Mainz.

Tuin, Marieke van der, Mathijs de Weerdt, and Gernot Veit Batz (2018). "Route Planning with Breaks and Truck Driving Bans Using Time-Dependent Contraction Hierarchies". In: *Twenty-Eighth International Conference on Automated Planning and Scheduling (ICAPS 2018)*. URL: https://aaai.org/ocs/index.php/ICAPS/ICAPS18/paper/view/17745.

United States House Committee on Rules (Dec. 2014). *Consolidated and Further Continuing Appropriations, 2015*.

Vidal, T., T.G. Crainic, M. Gendreau, and C. Prins (2013). "Heuristics for multi-attribute vehicle routing problems: A survey and synthesis". In: *European Journal of Operational Research* 231.1, pp. 1–21. DOI: 10.1016/j.ejor.2013.02.053. URL: https://www.cirrelt.ca/DocumentsTravail/CIRRELT-2012-05.pdf.

— (2014). "A unified solution framework for multi-attribute vehicle routing problems". In: *European Journal of Operational Research* 234.3, pp. 658–673. DOI: 10.1016/j.ejor.2013.09.045. URL: https://www.cirrelt.ca/DocumentsTravail/CIRRELT-2013-22.pdf.

Vidal, Thibaut, Teodor Gabriel Crainic, Michel Gendreau, Nadia Lahrichi, and Walter Rei (2012). "A Hybrid Genetic Algorithm for Multidepot and Periodic Vehicle Routing Problems". In: *Operations Research* 60.3, pp. 611–624. DOI: 10.1287/opre.1120.1048. eprint: https://doi.org/10.1287/opre.1120.1048. URL: https://doi.org/10.1287/opre.1120.1048.

Xu, Hang, Zhi-Long Chen, Srinivas Rajagopal, and Sundar Arunapuram (2003). "Solving a Practical Pickup and Delivery Problem". In: *Transportation Science* 37.3, pp. 347–364. DOI: 10.1287/trsc.37.3.347.16044. eprint: http://pubsonline.informs.org/doi/pdf/10.1287/trsc.37.3.347.16044. URL: http://pubsonline.informs.org/doi/abs/10.1287/trsc.37.3.347.16044.

Zäpfel, Günther and Michael Bögl (2008). "Multi-period vehicle routing and crew scheduling with outsourcing options". In: *International Journal of Production Economics* 113.2. Special Section on Advanced Modeling and Innovative Design of Supply Chain, pp. 980 –996. ISSN: 0925-5273. DOI: 10.1016/j.ijpe.2007.11.011. URL: http://www.sciencedirect.com/science/article/pii/S0925527307003611.