Erasmus University Rotterdam

Erasmus School of Economics

Master thesis

Departure time optimization of a set of dependent routes

Daniëlle Hooijenga (366347)

Supervisor:
Dr. R. Spliet

Supervisor:
G.P. Groenendijk MSc

Co-reader:
Prof. dr. A.P.M. Wagelmans

Abstract

In this thesis, we look into departure time optimization in a network of dependent routes. The routes can be dependent, for example, because a pickup in one route needs to take place before the delivery in a different route. Currently, this problem is solved by ORTEC by optimizing the routes one by one. In this thesis, a labeling algorithm is proposed to solve the complete network simultaneously. Different approaches of this algorithm are given, both exact and heuristic. By means of numerical experiments the proposed algorithm is compared to the algorithm as currently used by ORTEC. From the experiments we find that small instances can be solved to optimality within acceptable computation times. However, when the instances grow bigger, the computation times of the exact approach grow exponentially. The heuristics show varying performances, both in computation time and solution quality, where the construction algorithms show the most promising results. When multiple time windows are present, the computation times of all versions of the algorithm show a large increase.

Keywords: Departure time optimization, precedence constraints, drivers' legislation, multiple time windows, labeling algorithm, cost functions, dominance

Contents

1 Introduction **4**

2 Literature review **9**

2.1 Departure time optimization 9

2.2 Vehicle routing problem 10

2.3 Other applications 11

3 Problem formulation **14**

3.1 Locations 14

3.2 Multiple time windows 14

3.3 Dependencies 15

3.4 Drivers' legislation 15

3.5 Objective 16

4 Current method **17**

4.1 Slack 17

4.2 Initialization 18

4.3 Start value 18

4.4 Finish value 18

4.5 Post steps 19

4.6 Example 20

5 Proposed method **22**

5.1 Representation 22

5.2 Labels 24

5.3 Cost functions . . . 25
5.3.1 Cost coefficients . . . 25
5.3.2 Determining the possible start times . . . 26
5.3.3 Determining the costs . . . 26
5.4 Drivers' legislation functions . . . 27
5.5 Dominance . . . 28
5.5.1 Exact approach . . . 29
5.5.2 Heuristic approach . . . 30
5.6 Construction algorithms . . . 32
5.6.1 Construction algorithm 1 . . . 33
5.6.2 Construction algorithm 2 . . . 34
5.7 Retrieve final solution . . . 35
5.8 Example . . . 35
5.9 Analysis . . . 38

6 Computational results **40**
6.1 Data description . . . 41
6.2 Exact Approach . . . 41
6.3 Heuristics . . . 46
6.3.1 Heuristic 1: One route . . . 46
6.3.2 Heuristic 2: One time . . . 50
6.3.3 Heuristic 3: Last break/rest . . . 52
6.4 Construction algorithms . . . 57
6.4.1 Construction algorithm 1 . . . 57
6.4.2 Construction algorithm 2 . . . 60

7 Conclusion **64**

8 Further research **66**
8.1 Time-dependent travel times . . . 66
8.2 Drivers' legislation . . . 67
8.3 Construction algorithms . . . 68
8.4 Linear number of labels . . . 68
8.5 Optimize route by route . . . 69

Bibliography **70**

A Mathematical Programming Formulation **72**

A.1 Sets 72

A.2 Parameters 72

A.3 Variables 73

A.4 Model 74

Chapter 1

Introduction

Transportation companies deal with the challenge of transporting a set of orders from customers using their available resources, such as truck drivers, trucks, and trailers. In order to stay competitive, transportation companies are interested in optimizing their vehicle route plans to lower their transportation costs. An efficient vehicle route plan can be achieved by solving a vehicle routing problem (VRP), of which the objective usually is to minimize total traveling time or total traveling distance. ORTEC provides, among others, advanced software related to vehicle routing.

An important problem related to the VRP is to create a schedule for the truck driver. This driver needs to perform a sequence of actions, involving driving, waiting, and (un)loading the vehicle. This sequence of actions will be called a route. As a truck driver is the most expensive resource of a company, it is of great importance to make efficient use of this resource. By shortening the duration of the routes the use of truck drivers decreases, resulting in a decrease in costs. The objective of this problem is thus to minimize the duty time of truck drivers. The decisions are restricted by constraints concerning drivers' legislation, (multiple) time windows at customers, and dependencies in and between routes.

In this paper the problem of optimizing the start times of the routes, while adhering to all applicable constraints, as are mentioned above, is considered. The decision on the start time of a route needs to be recalculated each time anything changes in the route, requiring the developed method to be fast. An

example of a route is given in Figure 1.1. If there are no further restrictions, this route can be optimized by omitting both wait actions and delaying the start time by the duration of both wait actions together.

Figure 1.1: Example of a route

As was mentioned before, one of the restrictions on the decisions to be made, are dependencies within and between routes. Several dependencies are distinguished. Firstly, the dependency can be within a route. The input to the problem is an already established route, in which the order of visiting the locations needs to remain the same. This implies that each task in the route is dependent on its predecessor. Another dependency is the resource dependency, indicating that certain tasks need to be executed by the same truck and driver and/or the same trailer, in a specified order, this dependency can take place within a route or between different routes. Finally, there can be a dependency of an order, where an order is a sequence of multiple tasks, where one task must be executed before the other can take place. An example is when something needs to be picked up before it can be delivered. Even though we distinguish between multiple types of dependencies, all of them can be seen as precedence constraints, that is, in each case some task needs to be finished before some other task can start. The dependent tasks can be either in the same route or in different routes.

In the current way of solving applied by ORTEC, all routes are optimized one by one. This implies that besides optimizing the routes, also a decision on the order of optimizing needs to be made. Because of the dependencies between routes, the order of optimization does matter, as is shown in Example 1.1. However, as in general there is no clear ordering, the main challenge is to find a method which can optimize all routes simultaneously rather than sequentially. In this way, no optimal order of beautifying needs to be constructed.

Example 1.1

An example of a dependency between two routes is shown in Figure 1.2. In this example, task 1 needs to be finished before task 3 can start. This can be the case if, for example, the two tasks need to be executed using the same trailer. When optimizing these routes the following rules should be adhered to: the end time of the route which is currently optimized is not allowed to change. However, the end times of all other routes are allowed to be delayed. Furthermore, actions in a route can only be delayed, that is, they cannot be advanced in time. This rule applies as in practice all actions are already scheduled as early as possible.

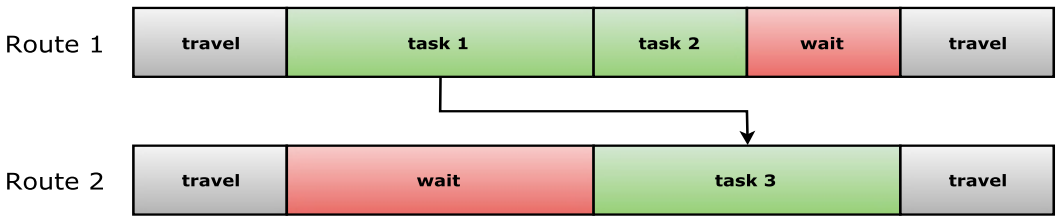

Figure 1.2: Example of a dependency between two routes. Task 1 must be finished before task 3 can start

In case route 1 would be optimized first, the wait action of one hour would be omitted, as there are no further restrictions on this route itself. As a result task 1 is finished at 4, and thus task 3 needs to be delayed for one hour, because of its dependence on task 1. This is done by introducing an extra hour of waiting before task 3 starts. Afterwards, route 2 is optimized by omitting the complete waiting action and delaying the start time by three hours.

In case route 2 would be optimized first, the only wait action is omitted, delaying the start time by two hours. Next, route 1 is optimized. Again, the wait action of this route is omitted. As task 1 now starts and finishes one hour later, task 3 needs to be delayed for one hour as well. This results in a wait action of one hour before task 3 is executed.

Clearly, in this example it is optimal to first optimize route 1 and then route 2, as in that order the optimal solution is found. ■

Another important restriction on the truck driver schedules is the drivers' legislation. Certain rules concerning the driving hours and break periods of the truck drivers are enforced by law. The set of rules which will be taken into account, are the daily rules of the European driving hours regulations (2006). These rules are as follows:

- A driving period may contain at most 4.5 hours of accumulated driving time.
- A working period may contain at most 6 hours of accumulated working time.
- The driving period and the working period end with a break of at least 45 minutes.
- The duration of a daily rest period is at least 11 hours.
- Within 24 hours after the end of a daily rest period, a new daily rest period must have been taken. This implies that a non-rest period, or the daily working time, is restricted to at most 13 hours.
- The total daily driving time may not exceed 9 hours. The daily driving time is the total driving time between the end of one daily rest period and the beginning of the following rest period.

Breaks and rests can take place at a customer location, both before and after servicing this customer. It needs to be decided for every location at what time service starts and what the departure time is. Moreover, it is also possible to have a break or rest while traveling from one customer to another.

The terminology throughout the remainder of this paper is as follows. A route is a sequence of actions, involving driving, waiting, and (un)loading the vehicle. A task is any action a driver needs to perform outside driving and waiting. A waiting action implies that the driver is neither traveling, nor servicing a customer.

The remainder of this paper is structured as follows. In Chapter 2 an overview of relevant literature is given, followed by a problem formulation in

Chapter 3. The current and proposed method are given in Chapters 4 and 5, respectively. Chapter 6 describes the results found by the proposed method, which are also compared to the results of the current method. These results are compared to the results of the current method. In Chapter 7 the conclusion of the research is given, followed by recommendations for further research on the topic in Chapter 8.

Chapter 2

Literature review

A lot of research has been conducted on the Vehicle Routing Problem (VRP), which is concerned with the routing of a fleet of vehicles along a set of locations. However, the problem considered in this paper concerns a problem related to the VRP, which is executed after the VRP has been solved. The problem of route duration minimization with synchronization constraints, like precedence constraints, is less intensively studied. In this chapter, several problems studied in literature are presented.

2.1 Departure time optimization

Kok et al. (2011) consider the vehicle departure time optimization (VDO) problem where multiple constraints are taken into account. Their model accounts for drivers' legislation, time-dependent travel times, and single time windows at customers. The formulation given is based on the European driving hours regulations. An ILP-formulation is given for the constructed model, which is tested on a selection of 100-customer problem instances developed by Solomon (1987). It is shown by computational experiments that the VDO can be solved to optimality with computation times of less than one second, achieving duty time reductions of 8%, compared to the duty times if the vehicles depart at time zero. An important difference between the model considered by Kok et al. (2011) and the one considered in this paper, is that Kok et al. (2011) do not

consider possible dependencies between routes.

A labeling algorithm is applied to a similar problem by Goel (2012). The author proposes a dynamic programming approach using labels to find a feasible truck driver schedule with minimum route duration, while taking into account drivers' legislation and multiple time windows. Similar to the problem considered in the paper by Kok et al. (2011), in this problem also no dependencies between routes are taken into account. It is shown that the labeling method requires significantly less computation time, up to 200 times less, than a commercial solver for all instances, while still solving the problem to optimality.

2.2 Vehicle routing problem

A survey on VRPs with multiple synchronization constraints was created by Drexl (2012). Several types of synchronization are distinguished, of which operation synchronization suits the problem, considered in this paper, best. Operation synchronization implies that one task should be finished before a different task can be started. The survey gives an overview of problems concerning synchronization constraints studied and the solution methods used by different authors. In the different problems considered in the survey, the objective is to construct a tour along a set of customers. This is in contrast to the goal of the problem considered in this paper, where the purpose is to determine the optimal starting time of a route, where the route itself has already been determined.

The paper by Labadie et al. (2014) also deals with a VRP with synchronization constraints. Two different types of synchronization constraints are specified: simultaneous synchronization, where a vertex requires at least two visits simultaneously, and precedence synchronization, where a vertex requires several successive visits. In the VRP considered by the authors, simultaneous synchronization is applied. The considered problem is solved using a construction heuristic followed by an iterated local search metaheuristic. Small instances can be solved to optimality, however, instances of more realistic size achieve a maximum improvement of 16% compared to the initial input solu-

tions.

Bredström and Rönnqvist (2007) consider the combined vehicle routing and scheduling problem with synchronization constraints, which imply that at least two customers need simultaneous service. The problem is solved using a branch-and-price framework based on column generation and time window branching. The developed algorithm is applied to a set of problems concerning homecare staff scheduling, where on average ten percent of the customers need simultaneous service from two staff members. It is shown by the authors that 44 out of 60 problems used for numerical experimentation are solved to optimality. The authors indicate that precedence constraints can be considered without any major modifications.

Dohn et al. (2011) use a column generation based solution approach to solve their formulation of the vehicle routing problem with time windows and temporal dependencies, where multiple vehicles need to visit a customer simultaneously or by some precedence relation. First, two formulations of the given problem are introduced, followed by the Dantzig-Wolfe decompositions of these formulations. The decompositions allow for a column generation based solution approach. The dependencies are modeled as precedence constraints. The solution method is tested on the instances developed by Solomon (1987), where temporal dependencies are introduced by the authors. It is shown that instances with 50 customers are too hard to solve. Smaller instances can be solved with widely varying computation times.

2.3 Other applications

A solution approach using Dantzig-Wolfe decomposition is applied by Stojković and Soumis (2001). The decomposition, in combination with a branch-and-bound method, is applied to the operational pilot scheduling problem, in which planned duties for a set of available pilots to cover a set of flights are modified. Constraints which must be respected include not exceeding the maximum duty duration and flight precedence constraints. The proposed method is shown to solve the test instances to optimality in a few seconds.

A slightly different problem, the synchronized vehicle dispatching problem,

is considered by Rousseau et al. (2003). In this problem there is a need to service customers using multiple resources which are subject to synchronization constraints. The authors remark that these synchronization constraints make this problem quite hard to solve using traditional local search methods. Instead, a constraint programming method is proposed.

In the book by Baptiste et al. (2012) a general-shop problem with precedence constraints is solved using constraint programming. The objective of the considered problem is to minimize the makespan. The first phase consists of finding a good but not necessarily optimal solution. In the second phase constraint programming is applied, with as an extra constraint that the makespan must be smaller than or equal to the one found in the first phase.

Bredström and Rönnqvist (2008) created a MIP model for the combined vehicle routing and scheduling problem with temporal precedence and synchronization constraints. This model can be solved to optimality for small instances. For larger instances a heuristic is applied.

A branch-and-cut approach is applied to the asymmetric traveling salesman problem with precedence constraints by Ascheuer et al. (2000). In this problem, the precedence constraints require that certain nodes precede certain other nodes in any feasible directed tour. The authors show that real world instances can be solved to optimality within a few minutes of CPU time. One of the differences with the problem considered in this paper is again the purpose of the problem. The goal of solving the ATSP is to find a shortest route, rather than shortest route duration. Another difference is, that in the ATSP with precedence constraints, these constraints only exist within a tour, whereas in the problem considered here, precedence constraints can also exist between different routes.

Kolisch and Hartmann (2006) provide an overview of good performing heuristics for resource-constrained project scheduling. In this problem, precedence constraints between tasks are encountered. The authors state that the best performing methods are the metaheuristics. The six dominating methods are all population-based metaheuristics. One of these good performing heuristics is presented by Valls et al. (2004). This approach consists of two phases. In the first phase the initial population of schedules is constructed, after which

these are evolved until high quality solutions have been obtained. In the second phase the vicinities near the high quality schedules are further explored. The algorithm is shown to produce high quality solutions in an average time of less than five seconds. However, the metaheuristics attempt to improve a given schedule by changing the structure of the provided schedule. Actions can, in that case, be assigned to different routes, or the order of actions within a route can change. This procedure of changing the structure of a schedule is unsuitable for the problem considered in this paper, as in this problem the structure of the routes must remain the same.

Ioachim et al. (1998) consider a shortest path problem with time windows and linear costs on the start time of service on the nodes. Two applications of this problem are given: job-shop scheduling and aircraft routing and scheduling. The authors present an optimal dynamic programming algorithm to solve the shortest path problem. The algorithm proposed is shown to clearly outperform the discretization approach in terms of CPU time.

Chapter 3

Problem formulation

In this chapter, a description of the considered problem is given, describing all constraints which will be taken into account, and the objective of the optimization problem. In Appendix A a mathematical programming formulation can be found.

3.1 Locations

Consider a set of routes $R = \{1, ..., T\}$. Each route consists of locations that need to be visited. Let each route $r \in R$ consist of a set of locations $L_r = \{0, ..., N_r + 1\}$. In this set 0 and $N_r + 1$ are the start and end location of the route, respectively, and $C_r = \{1, ..., N_r\}$ are the customer locations of this route. At each customer location, service with duration $k_{r,n}$ must be carried out. The driving time between two consecutive locations in one route is $C_{r,(n,n+1)}$.

3.2 Multiple time windows

A location can have multiple time windows, in which case the start time of service must fall within one of those time windows. The number of time windows at location n in route r will be denoted by $t_{r,n}$. The jth time window at location n in route r, $j \in \{1, ..., t_{r,n}\}$, will be denoted by $[l^j_{r,n}, u^j_{r,n}]$. Which time

window is used at location n in route r is indicated by the binary variable $I^j_{r,n}$, which equals one if the jth time window is used and zero otherwise.

3.3 Dependencies

The precedence constraints are contained in the set $P^{prec} = \{((r,n),(u,v))|r \in R, n \in L_r, u \in R, v \in L_u\}$. For each pair $((r,n),(u,v))$ in this set, it must hold that service at location n in route r must be finished before service at location v in route u can start.

3.4 Drivers' legislation

In order to adhere to the drivers' legislation, some break and rest periods may be needed throughout the routes. The drivers do not necessarily start their routes rested, as they may have performed a different route before the given route. The accumulated driving time, the accumulated working time, the daily driving time, and the daily working time of the driver at the start of the route are indicated by $adt_{r,0}$, $awt_{r,0}$, $ddt_{r,0}$, and $dwt_{r,0}$, respectively. A break has a minimum length of b^{break}_{min} and the minimum length of a rest period is b^{rest}_{min}.

A break must be taken at the latest when the accumulated driving time or the accumulated working time reaches a pre-specified limit, d^{acc}_{max} or w^{acc}_{max}, respectively. A rest must be taken at the latest when the maximum daily driving time (d^{daily}_{max}) or the maximum daily working time (w^{daily}_{max}) is reached. Note that waiting time which does not serve as a break or rest also counts as working time. The break and rest periods can take place at any location in the route, before or after service at the location. Furthermore, a break or rest can be taken during the traveling time between two locations. It is assumed that the break or rest can be taken whenever wanted during this traveling time, there is no need to travel to a special resting place. We allow for at most one break or rest during each travel action.

3.5 Objective

The decisions which need to be made for every route $r \in R$ are the start time per location ($X^s_{r,n}$), the departure time ($X^d_{r,n}$) from each location and when a break or rest is taken. For a waiting period ($W^p_{r,n}$) to function as a break or a rest, it must be at least as long as the minimum break duration (b^{break}_{min}) or the minimum rest duration (b^{rest}_{min}).

The objective of the considered problem is to minimize the total route duration over the complete network of all routes, that is, the sum over all route durations, the time between departure from the start location and arrival at the end location, should be minimized.

Chapter 4

Current method

In this chapter the slack algorithm is described. This algorithm is currently used by ORTEC to decide the start times of routes. The slack algorithm optimizes the network of routes not simultaneously, but rather route by route. The algorithm uses two values to determine the amount of time the start time can be delayed. These two values are recalculated at the start and finish of each action in a route.

The structure of this chapter is as follows: first, the two values used in the algorithm are explained. Next, the initialization and updating of these values are explained. Finally, an overview of the algorithm is given and an example is provided.

4.1 Slack

Slack is defined as the time by which the start of a route can at most be postponed without violating any restriction along the route and without delaying the end time of the route. That is, slack is the maximum amount the start of the initial route can be delayed such that each task starts within one of its time windows, while not delaying the end time. When delaying the start time, if possible, wait time is reduced. In case there is no more wait time, the start times of the succeeding actions are also delayed. The end time is not allowed to be delayed because the end time influences the start- and end times

of routes that take place later in time. As a consequence, in the worst case, the entire planning needs to be recalculated, which is not preferred in practice. The slack is thus the amount by which the start time of a route will eventually be delayed. In order to find this value, two other values are used. These values are updated at the beginning and at the end of each action in the route. First of all, the possible slack gives an upper bound on the slack which might be found. The slack will thus never exceed this amount. Furthermore, the used slack is the slack that is consumed over the already considered actions. That is, the used slack either stays equal in an iteration or increases, and the final value of the used slack found is the amount by which the start time of the route is delayed.

4.2 Initialization

The algorithm is initialized by setting the possible slack equal to the finish time minus the start time of the original route. This amount equals the duration of the original route, which is given as input, and is an upper bound on the amount by which the start time can eventually be delayed. The used slack is initialized by setting it to zero. These are the start values of the first action of the optimized route.

4.3 Start value

The value of both the possible slack and the used slack at the start of an action in a route is equal to the values at the finish of the previous action.

4.4 Finish value

If the action is a wait action, we check whether the duration of the wait is smaller than the possible slack. If this is the case, it is thus possible to completely take away the waiting time. The possible slack is reduced by the duration of the wait action and the used slack is increased by the duration of the

wait action. If the waiting time is longer than the possible slack, the possible slack is reduced to zero and the used slack is increased by the start value of the possible slack, as this is the amount by which the route can still be delayed.

If the action is not a wait action, we check if it has time-concerning restrictions, like a time window. When this is the case, the possible slack is reduced to the amount of time by which the route can still be delayed without violating this restriction. The used slack does not change. When there are no time-concerning restrictions, both values stay equal to their start value.

4.5 Post steps

For VRP instances with single time windows and a fixed end time, but without drivers' legislation, multiple time windows, or any type of dependency between routes, the slack algorithm finds the optimal outcome. However, for more complex instances of the VRP, including one or more of the above constraints, this is not necessarily the case. In this case feasibility is not guaranteed and a post-step is performed. In this step the reduced delay for which a feasible solution is created is determined. That is, the largest violation concerning a time restriction is determined and the start time of the route is advanced in time by this amount. Note that a delay of the end time is also considered to be a violation.

Algorithm 1 gives an overview of the method described above.

Algorithm 1 Slack algorithm

```
for route in routes do
    Initialize possibleSlack
    Initialize usedSlack
    for action in route do
        Update possibleSlack
        Update usedSlack
    end for
    Delay route
    Find largest violation
    Advance start time
end for
```

4.6 Example

To illustrate the use of the slack algorithm, an example is given in this section. In Figure 4.1, an example of a route is shown. The given numbers represent the *possibleSlack* and the *usedSlack*, respectively. Furthermore, the first task has a time window, meaning that service of the task needs to start between 3 and 4.

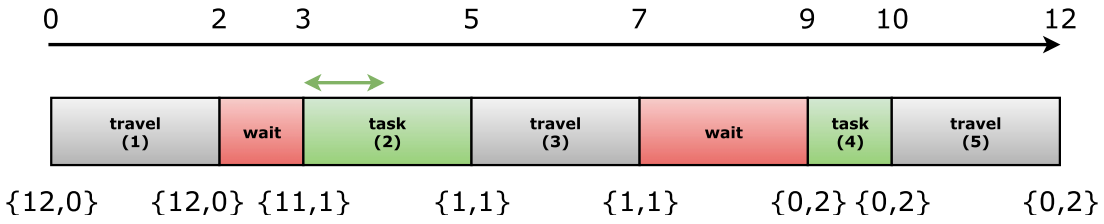

Figure 4.1: Example of the slack algorithm

The *possibleSlack* is initialized to the total duration of the initial route, as this is an upper bound on the slack we may find. In this case, this value is 12. Furthermore, the *usedSlack* is initialized to 0. At the end of the first travel action, nothing has changed yet, so the values stay the same. Next, we encounter a wait with a duration of one hour. There is sufficient *possibleSlack* to fully

cover the wait, so the *usedSlack* is increased by 1, whereas the *possibleSlack* is decreased by 1. Then, a task with a time window of length 1 is encountered. Because of the time window, the start time cannot be delayed by more than one hour. Therefore, the *possibleSlack* is decreased to 1. For the next travel action nothing changes, so the values stay equal. At the second wait action, with a duration of two hours, the *possibleSlack* is 1, so the start time can be delayed by one more unit. This implies that the *usedSlack* is increased by 1 and the *possibleSlack* is decreased by 1. The *possibleSlack* now equals zero, meaning there are no more options to delay the start time. The solution we find is to delay the start time by 2 units.

Chapter 5

Proposed method

In this chapter, we describe the labeling algorithm. This method uses labels consisting of cost functions to indicate all possible starting times for each action in a network of routes. By assigning a cost to each possible starting time, the optimal solution can be determined. Each label is extended considering all possible alternatives for the on- and off-duty periods of the driver. As this implies that many labels will be created, dominance rules will be applied to decrease the number of labels. This method is inspired by Goel (2012) in combination with the shortest path algorithm by Ioachim et al. (1998).

The description of the method is structured as follows. First, a slightly different representation of routes is presented. Next, the labels and the extension of these labels are discussed. Then, the dominance rules, for both an exact approach and some heuristic approaches, are described. After that, two construction algorithms are presented, followed by the procedure to retrieve the solution from the cost functions. Next, an example of the labeling algorithm is given. Finally, an overview of the complete algorithm is given together with an analysis of the algorithm.

5.1 Representation

In order to integrate the dependencies between routes in the method, we will look at all the routes as one complete network, represented by a directed graph

$G = (V, E)$, with V the set of nodes and E the set of directed edges. Here, a node in V is indicated by two indices, $(r, n) \in V$, where r represents the route and n is the index of the node in the route. Both the tasks and the traveling actions are represented by nodes. This implies that there is no (traveling) time defined on the edges in the graph. Instead, all nodes have a specified duration ($k_{r,n}$), which is the traveling time in case the node represents a traveling action and the service time in case the node represents a task.

In graph G there is a directed edge between any pair of nodes between which there is a dependency, both within a route or between different routes. An example of the representation as a graph is given in Figure 5.1. In this network, nodes (1,1) until (1,4) represent one route and nodes (2,1) until (2,4) represent another route. Furthermore, a dependency between nodes (1,2) and (2,3) is shown, implying that action (1,2) should be finished before action (2,3) can start.

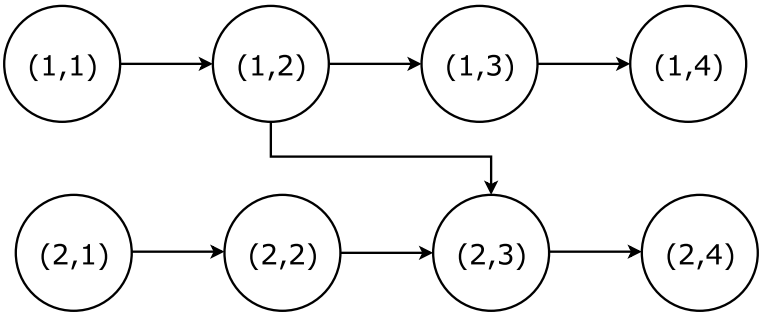

Figure 5.1: Example of the representation of a network of routes as a graph

For each node in the network, certain information needs to be known. This information consists of the following: the type of the action, the time windows of each action, the duration of this action, its direct predecessors in the network, the route it belongs to, and the driver carrying out the action.

The labeling algorithm is an iterative approach, starting by scheduling one node and continuing until all nodes have been scheduled. If a node is scheduled, this implies that this node has been given its labels. In each iteration, any node not having any unscheduled predecessors can be the next node to be scheduled. Note that graph G cannot contain any cycles as in this case some task needs to be finished before another task can start, while this task must also be finished

before the first considered task can start, leading to a contradiction. By the absence of cycles, the order as described above will always account for all nodes in the graph. As an example, consider Figure 5.1. A feasible order of scheduling in this case is $(1,1) \rightarrow (2,1) \rightarrow (1,2) \rightarrow (2,2) \rightarrow (1,3) \rightarrow (2,3) \rightarrow (1,4) \rightarrow (2,4)$.

5.2 Labels

A label is a collection of information about the considered node in one specific (partial) solution. The labels which are used are defined per node. A node can have multiple labels as multiple combinations of on- and off-duty periods of the driver are considered as well as every time window at every action. In this label it is noted which time window is used at the specific node and whether a break or a rest is taken before carrying out the action or during travel. Furthermore, the cost function and drivers' legislation functions of the specific node are contained in the label. The cost function and functions of drivers' legislation variables will be explained in more detail later. The notation used is as follows. The binary variable $Y_{r,n}^{p,l}$, $p \in \{s,b\}$, $l \in \{break, rest\}$, $(r,n) \in V$ equals one if a break or rest is taken before service (p = s) or during travel (p = b) at node (r,n), and zero otherwise. Note that now a break or a rest can only be scheduled before starting the task rather than before or after. As traveling actions are also represented as nodes, when a break is scheduled before traveling, this can also be seen as taking a break after servicing the previous customer. Moreover, a break or rest can be scheduled, during travel time. A break needs to be scheduled during a travel when during this time the accumulated driving time or the accumulated working time reaches its limit. Similarly, a rest needs to be scheduled during travel time if during this travel the limit for the daily driving time or the daily duty time is reached. Furthermore, $I_{r,n}^{j}$, $j \in \{1, ..., t_{r,n}\}$, equals one if the jth time window at node (r,n) is used and zero otherwise. In order to account for the drivers' legislation, the variable $dl_{r,n} = (adt_{r,n}, awt_{r,n}, ddt_{r,n}, dwt_{r,n})$ is used. Here $adt_{r,n}$ represents the accumulated driving time at the end of node (r,n), $awt_{r,n}$ is the accumulated working time at the end of node (r,n), $ddt_{r,n}$ represents the daily driving time

at the end of node (r, n), and $dwt_{r,n}$ represents the daily working time at the end of node (r, n). Finally, the duration of a break or rest, is represented by $b^l_{min}, l \in \{break, rest\}$ and the duration of action (r, n) is given by $k_{r,n}$.

5.3 Cost functions

Each label contains a cost function. Let the variable $X^s_{r,n}, (r, n) \in V$ represent the starting time of the action at node (r, n). The cost at a node represents the minimal costs of the path taken so far, if service starts at time $X^s_{r,n}$. That is, at the qth node of route $r \in R$, the cost at this node is the minimum value for $\sum_{n=1}^{q} \lambda_{r,n} X^s_{r,n}$ which can be reached, while adhering to the set constraints. Here, $\lambda_{r,n}$ is the cost coefficient of node (r, n). The cost coefficients are further explained in Section 5.3.1.

5.3.1 Cost coefficients

Before the cost function can be defined more detailed, several nodes need to be added to the network. For each route in the network, a source node and a sink node are added, at the beginning and the end of the route, respectively. The source node makes sure that a break or rest can also be planned before the first action in a route, which may be needed whenever the driver is not rested when starting the route. The start time of the sink node represents the end time of the route. The introduced source and sink nodes all have a duration of zero. Next, all nodes get a cost coefficient, denoted by $\lambda_{r,n}, (r, n) \in V$. These coefficients can be deduced from the objective we want to minimize. The objective to our problem is to minimize $\sum_{r \in R}(X^s_{r,d} - X^s_{r,s})$, that is the sum over all routes of the start time at the sink node minus the start time at the source node. This is equal to the sum of the durations of all routes in the network. From the objective we deduce that the source node gets cost coefficient $\lambda_{r,s} = -1, \forall r \in R$, the sink node gets cost coefficient $\lambda_{r,d} = +1, \forall r \in R$, and all other nodes get cost coefficient zero.

5.3.2 Determining the possible start times

The first step in constructing a cost function is to determine at which times service at the node can start. Let the earliest start time of node (r, n) be denoted by $t^e_{r,n}$. To determine the earliest possible start time, we need to take into account all direct predecessors. We take the maximum over all direct predecessors of the earliest start time plus the service time plus any break or rest time during service (only possible if the predecessor is a travel action). To this, we add the break or rest time at the node itself. Finally, we take the maximum of the total, with the lower bound of the used time window at the current node. That is, $\max\{\max_{(u,v):((u,v),(r,n))\in E}\{t^e_{u,v} + k_{u,v} + \sum_{l\in\{break,rest\}}(b^l_{min}Y^{b,l}_{u,v} + b^l_{min}Y^{s,l}_{r,n})\}, \sum_{j=1}^{t_{r,n}} l^j_{r,n}I^j_{r,n}\}$. The latest start time ($t^l_{r,n}$) is simply the upper bound of the time window.

5.3.3 Determining the costs

The costs at each node can be deduced from the cost function of its direct predecessor in the same route, together with the cost coefficient at the considered node. For each point in time, the cost is the cost of the previous node in the same route at the time instant at which service should have started at that node in order to start service at the specified time at the considered node, plus the costs of the node itself. Let $C^S_{r,n}(X^s_{r,n})$ denote the cost at node (r, n) in solution S, if service at that node starts at time $X^s_{r,n}$. Then, the costs at node (r, n) at time $X^s_{r,n}$ are defined by $C^S_{r,n}(X^s_{r,n}) = C^S_{r,n-1}(\min\{X^s_{r,n} - k_{r,n-1} - \sum_{l\in\{break,rest\}}(Y^{b,l}_{r,n-1}b^l_{min} + Y^{s,l}_{r,n}b^l_{min}), u_{r,n-1}\}) + \lambda_{r,n}X^s_{r,n}$. The costs need to be calculated at each breakpoint of the cost function. Between the breakpoints, the cost function is defined as the linear piece segments connecting the breakpoints. Besides the earliest start time and the latest start time, breakpoints occur when a node's predecessor has a breakpoint or when waiting time occurs for a specific start time.

5.4 Drivers' legislation functions

As we need to adhere to the drivers' legislation, we first need to specify how the drivers' legislation variables are calculated. The variables which will be looked at are the accumulated driving time (adt), accumulated working time (awt), daily driving time (ddt), and daily working time (dwt). The accumulated driving time and accumulated working time are reset to zero if a break or a rest is taken. The other two variables require a rest in order to be reset. The values of the variables differ with the starting time of service. For some starting times, waiting time might be present, which also counts as working time, and thus needs to be added to the corresponding variables. However, if the waiting time is long enough to serve as a break or a rest, then the variables are reset. If we determine the accumulated driving time, accumulated working time, daily driving time, and daily working time for each starting time on which the cost function of an action is defined, we obtain a function for each variable separately. In order to calculate the values of the variables at time t, we check the value of the corresponding variable at the previous node of the same route at the time at which service should have started in order for service at the current node to start at time t. To this value, we add the duration of the action and the waiting time, if present. If the waiting time exceeds the duration of a break or rest, we can set the variables to the duration of the action. Algorithm 2 gives an overview of the initialization and updating of the drivers' legislation variables. Note that the initial values given to these variables are not necessarily zero. That is, when a driver starts a route, it might be that he already worked a shift right before it, without having a break or rest period in between. In the algorithm, $s_{r,S}$ denotes the source node of route r in solution S, $u_{u,S}$ is the upper bound on the time window of node u which is used in solution S. Furthermore, $D_{r,n}$ is an indicator variable, which equals one if the action of node (r, n) is a traveling action and zero otherwise. Moreover, the binary variable $Y_{r,n}^{p,l}$, $l \in \{break, rest\}$, equals one if a break or rest is taken before (p = s) or during (p = b) an action, and zero otherwise.

Algorithm 2 Calculation of drivers' legislation variables

Initialization

$adt_{s_{r,S}} \leftarrow adt_{r,0} \quad \forall r, \quad \forall S$

$awt_{s_{r,S}} \leftarrow awt_{r,0} \quad \forall r, \quad \forall S$

$ddt_{s_{r,S}} \leftarrow ddt_{r,0} \quad \forall r, \quad \forall S$

$dwt_{s_{r,S}} \leftarrow dwt_{r,0} \quad \forall r, \quad \forall S$

Update

$shouldHaveStarted \leftarrow t - \sum_{l \in \{break, rest\}} (b^l_{min} Y^{s,l}_{r,n}) - k_{r,n}$

$X^s_{r,n} \leftarrow \min\{u_{u,S}, shouldHaveStarted\}$

$W^s_{r,n} \leftarrow shouldHaveStarted - X^s_{r,n}$

if $W^s_{r,n} \geq b^{break}_{min}$ **and** $W^s_{r,n} < b^{rest}_{min}$ **then**

$adt_{r,n} \leftarrow k_{r,n} D_{r,n}$

$awt_{r,n} \leftarrow k_{r,n}$

$ddt_{r,n} \leftarrow ddt_{r,n-1} + k_{r,n} D_{r,n}$

$dwt_{r,n} \leftarrow dwt_{r,n-1} + k_{r,n} + W^s_{r,n}$

else if $W^s_{r,n} \geq b^{rest}_{min}$ **then**

$adt_{r,n} \leftarrow k_{r,n} D_{r,n}$

$awt_{r,n} \leftarrow k_{r,n}$

$ddt_{r,n} \leftarrow k_{r,n} D_{r,n}$

$dwt_{r,n} \leftarrow k_{r,n}$

else

$adt_{r,n} \leftarrow adt_{r,n-1} + k_{r,n} D_{r,n}$

$awt_{r,n} \leftarrow awt_{r,n-1} + k_{r,n} + W^s_{r,n}$

$ddt_{r,n} \leftarrow ddt_{r,n-1} + k_{r,n} D_{r,n}$

$dwt_{r,n} \leftarrow dwt_{r,n-1} + k_{r,n} + W^s_{r,n}$

end if

5.5 Dominance

In order to avoid too many labels being created, in each iteration, after a nodes has been given all its labels, it will be checked whether some solutions can be deleted, while still reaching the optimal solution. In each iteration, every partial solution has the same node as the last scheduled node in each

route. The values which are compared in the dominance step thus belong to the same node.

5.5.1 Exact approach

In case we want to reach the optimal solution, a solution can only be deleted when we know that any extension of this solution can also be done to a different solution and yields a solution that is at least as good. In order to guarantee optimality, we require that the cost function of the dominated solution is entirely above the cost function of the dominating solution. The same holds for all functions of the drivers' legislation variables. These conditions as stated should hold for the last scheduled node of each route. Moreover, the feasible start times of the dominating solution should at least cover the feasible start times of the dominated solution. Let w_r be the last scheduled node of route r. The feasible start times are represented by the intervals $[t^e_{S,w_r}, t^l_{S,w_r}]$ and $[t^e_{\bar{S},w_r}, t^l_{\bar{S},w_r}]$, where t^e_{S,w_r} and $t^e_{\bar{S},w_r}$ are the earliest start times in solution S and $\bar{S}$, respectively. Similarly, t^l_{S,w_r} and $t^l_{\bar{S},w_r}$ are the latest start times in solution S and $\bar{S}$, respectively. In short, if the following conditions hold, solution $\bar{S}$ can be deleted, while still reaching the optimal solution.

- $C_{S,w_r}(t) \leq C_{\bar{S},w_r}(t) \qquad \forall r \in R, \forall t \in [t^e_{S,w_r}, t^l_{S,w_r}]$
- $adt_{S,w_r}(t) \leq adt_{\bar{S},w_r}(t) \qquad \forall r \in R, \forall t \in [t^e_{S,w_r}, t^l_{S,w_r}]$
- $awt_{S,w_r}(t) \leq awt_{\bar{S},w_r}(t) \qquad \forall r \in R, \forall t \in [t^e_{S,w_r}, t^l_{S,w_r}]$
- $ddt_{S,w_r}(t) \leq ddt_{\bar{S},w_r}(t) \qquad \forall r \in R, \forall t \in [t^e_{S,w_r}, t^l_{S,w_r}]$
- $dwt_{S,w_r}(t) \leq dwt_{\bar{S},w_r}(t) \qquad \forall r \in R, \forall t \in [t^e_{S,w_r}, t^l_{S,w_r}]$
- $t^e_{S,w_r} \leq t^e_{\bar{S},w_r} \qquad \forall r \in R$
- $t^l_{S,w_r} \geq t^l_{\bar{S},w_r} \qquad \forall r \in R$

To check the conditions mentioned above, we first need to determine the earliest and latest start time of both solutions. We can deduce these from the cost function. The earliest start time is the earliest point in time for which

the cost function is defined. Similarly, the latest start time is the latest point in time for which the cost function is defined. Next, to compare the functions of the different solutions, we check all breakpoints of the functions and their values at these breakpoints. The values at all breakpoints should be smaller or equal for the dominating solution for the conditions to hold.

5.5.2 Heuristic approach

Besides the dominance rules as described above, we can also relax the dominance rules a little bit, such that more solutions are deleted. As a result, the computation time is expected to be reduced, at the cost of losing the guarantee of optimality.

One route

Instead of checking the last scheduled node of each route, we can also only look at the route containing the last scheduled node. Let w_c be the last scheduled node of the entire network. The conditions for a solution to dominate a different solution remain the same.

- $C_{S,w_c}(t) \leq C_{\bar{S},w_c}(t) \qquad \forall t \in [t^e_{S,w_c}, t^l_{S,w_c}]$
- $adt_{S,w_c}(t) \leq adt_{\bar{S},w_c}(t) \qquad \forall t \in [t^e_{S,w_c}, t^l_{S,w_c}]$
- $awt_{S,w_c}(t) \leq awt_{\bar{S},w_c}(t) \qquad \forall t \in [t^e_{S,w_c}, t^l_{S,w_c}]$
- $ddt_{S,w_c}(t) \leq ddt_{\bar{S},w_c}(t) \qquad \forall t \in [t^e_{S,w_c}, t^l_{S,w_c}]$
- $dwt_{S,w_c}(t) \leq dwt_{\bar{S},w_c}(t) \qquad \forall t \in [t^e_{S,w_c}, t^l_{S,w_c}]$
- $t^e_{S,w_c} \leq t^e_{\bar{S},w_c}$
- $t^l_{S,w_c} \geq t^l_{\bar{S},w_c}$

The conditions can be checked in the same way as was done for the exact approach. The only difference is that we do not need to check the conditions for all routes, but rather for only one route.

Minimum cost

Instead of looking at the complete cost functions, we can also only consider the starting times at the last scheduled node of each route which would have been chosen if this partial solution would be the final solution. This is thus the point where costs are minimum. Let this point in time be represented by t_{min}. If the costs and the drivers' legislation variables are all larger than or equal than those of another solution, the solution is dominated and can be omitted.

- $C_{S,w_r}(t_{min}) \leq C_{\bar{S},w_r}(t_{min}) \qquad \forall r \in R$
- $adt_{S,w_r}(t_{min}) \leq adt_{\bar{S},w_r}(t_{min}) \qquad \forall r \in R$
- $awt_{S,w_r}(t_{min}) \leq awt_{\bar{S},w_r}(t_{min}) \qquad \forall r \in R$
- $ddt_{S,w_r}(t_{min}) \leq ddt_{\bar{S},w_r}(t_{min}) \qquad \forall r \in R$
- $dwt_{S,w_r}(t_{min}) \leq dwt_{\bar{S},w_r}(t_{min}) \qquad \forall r \in R$

In order to check these conditions, we now do not have to check the values of the functions at every breakpoint of the functions. Instead, we need to determine which point in time has the lowest cost. We can do this by taking the minimum of all costs of the different breakpoints of the cost function and check at which point in time this cost occurs. Next, the conditions can be checked for this time instant.

This heuristic may delete more solutions than the exact approach as one point on the graph being lower for one solution than for the other, does not guarantee the entire graph to be underneath the other. This may occur, for example, when one function is strictly decreasing, while the other starts lower but has a higher slope.

Last break and rest

We wish to schedule breaks and rests as late as possible in a route as in practice drivers will only take a break or a rest when actually needed. Moreover, by scheduling breaks and rests as late as possible, this might save needing an extra break or rest. In order to achieve breaks and rests to be scheduled as

late as possible in the final solution, we check for every route in each solution at which node the last break and the last rest have taken place. Denote these nodes by $x_{r,S}^{break}$ and $x_{r,S}^{rest}$, respectively. If in one solution the last break and the last rest of each route take place earlier than the last break and last rest in a different solution which is still feasible, then this solution can be deleted. When a solution does not contain a break or a rest, $x_{r,S}^{break}$ or $x_{r,S}^{rest}$ will be set to a sufficiently large value M, which should be larger than the number of nodes in the current route. In this way it will always be more beneficial to not have a break or rest.

- $x_{r,S}^{break} \geq x_{r,\bar{S}}^{break} \qquad \forall r \in R$
- $x_{r,S}^{rest} \geq x_{r,\bar{S}}^{rest} \qquad \forall r \in R$

5.6 Construction algorithms

In the three heuristics presented above, we changed the dominance rules such that more solutions are omitted in every iteration. In contrast to this, we can also create fewer solutions, by using a construction algorithm. The expectation is that a construction algorithm requires less computation time as fewer solutions are created, and as a result also fewer solutions need to be compared when applying the dominance rules.

In this section two slightly varying construction algorithms are given. The first plans a break or rest before an action when a certain bound is exceeded and only plans a break or rest in a travel when it is needed. By this approach, we aim to find solutions in which breaks and rests are not planned when the drivers' legislation variables are very low, as in practice this will also not occur. The second also only plans a break or rest before an action whenever a pre-set bound is exceeded, but does try every possibility in a travel. Compared to the first construction algorithm, this algorithm covers some more options and we are interested to see whether this indeed leads to a better solution quality.

5.6.1 Construction algorithm 1

The purpose of this construction algorithm is to construct fewer solutions than in the initial labeling algorithm, while still reaching a solution of good quality. This goal is achieved by not trying every possibility of scheduling a break, rest or neither before every action and in travels. Instead of scheduling breaks and rests at every action, we now only schedule a break or rest whenever a certain pre-set bound on the drivers' legislation variables has been exceeded. Let $d^{acc}_{limit} \leq d^{acc}_{max}$ and $d^{daily}_{limit} \leq d^{daily}_{max}$ be the pre-set limits for the accumulated driving time and daily driving time, respectively. Also, let $w^{acc}_{limit} \leq w^{acc}_{max}$ and $w^{daily}_{limit} \leq w^{daily}_{max}$ be the pre-set limits for the accumulated working time and daily working time, respectively. Note that when the limits are chosen to be exactly equal to the maximum allowed value, the likelihood of finding a feasible solution will be very low. However, when these values are chosen to be very small, the algorithm will resemble the original algorithm, where always all possibilities of having a break, rest or none are created. An overview of this procedure is given in Algorithm 3. Note that this does not replace the entire labeling algorithm as described before, but instead of creating all solutions this shown procedure is carried out to determine which labels are created. In this procedure, $isTravel$ indicates whether the type of the considered node is a travel.

Algorithm 3 Construction algorithm 1

if $ddt \geq d_{limit}^{daily}$ **or** $dwt \geq w_{limit}^{daily}$ **or** $adt \geq d_{limit}^{acc}$ **or** $awt \geq w_{limit}^{acc}$ **then**
    Create partial solution with rest
    Create partial solution with break
    Create partial solution without break/rest
else
    Create partial solution without break/rest
end if
if $isTravel$ **then**
    if $ddt + drivingTime \geq d_{limit}^{daily}$ **or** $dwt + drivingTime \geq w_{limit}^{daily}$ **then**
        Create partial solution with rest in travel
    else if $adt + drivingTime \geq d_{limit}^{acc}$ **or** $awt + drivingTime \geq w_{limit}^{acc}$ **then**
        Create partial solution with rest in travel
        Create partial solution with break in travel
    else
        Create partial solution without break/rest in travel
    end if
end if

5.6.2 Construction algorithm 2

Contrary to the first construction algorithm, in this algorithm we try to schedule a break, rest or neither in a travel regardless of the values of the drivers' legislation at this action. Whenever a break or rest is not actually needed during the travel time, it will be planned at the end of the travel, which is equal to planning it at the start of the next action. By allowing this, we create more possibilities than in the first construction algorithm, while keeping the (worst-case) complexity lower than in the initial labeling algorithm. The procedure to determine which labels are created in the second construction algorithm is shown in Algorithm 4.

Algorithm 4 Construction algorithm 2

if $ddt \geq d^{daily}_{limit}$ **or** $dwt \geq w^{daily}_{limit}$ **or** $adt \geq d^{acc}_{limit}$ **or** $awt \geq w^{acc}_{limit}$ **then**
    Create partial solution with rest
    Create partial solution with break
    Create partial solution without break/rest
else
    Create partial solution without break/rest
end if
if $isTravel$ **then**
    Create partial solution with rest in travel
    Create partial solution with break in travel
    Create partial solution without break/rest in travel
end if

5.7 Retrieve final solution

Once all cost functions have been constructed, we can determine the final solution. This solution can be determined from the found cost functions by backtracking. So, in contrast to the extension of the path, we now start at the last node of a route. The order in which the start times are determined is simply the reversed order in which we scheduled the nodes. The start time at the last node of a route is the earliest time for which the minimum cost is reached, as an earlier end time is preferred over a later end time. Next, for the other nodes, the latest time for which the solution is feasible is taken. In case of a dependency, the start time is the minimum of all found start times, as otherwise the solution would not be feasible. So for every node (r, n) the start time is $\min_{(u,v):((u,v),(r,n))\in E}\{X^s_{r,n} - k_{u,v} - \sum_{l\in\{break,rest\}}(b^l_{min}Y^{s,l}_{r,n} + b^l_{min}Y^{b,l}_{u,v})\}$.

5.8 Example

Consider the network of routes given in Figure 5.2. This network consists of two routes, with four actions each. There are two dependencies between the

routes. Action (1,2) needs to be finished before action (2,2) can start and similarly action (2,3) must be finished before action (1,4) can start. We will show how the labeling algorithm works for this network, without scheduling any breaks or rests. All actions have a duration of one hour and currently all actions are scheduled completely in the corresponding time windows, which are represented by double arrows. Note that only the start time of an action needs to fall inside its time window.

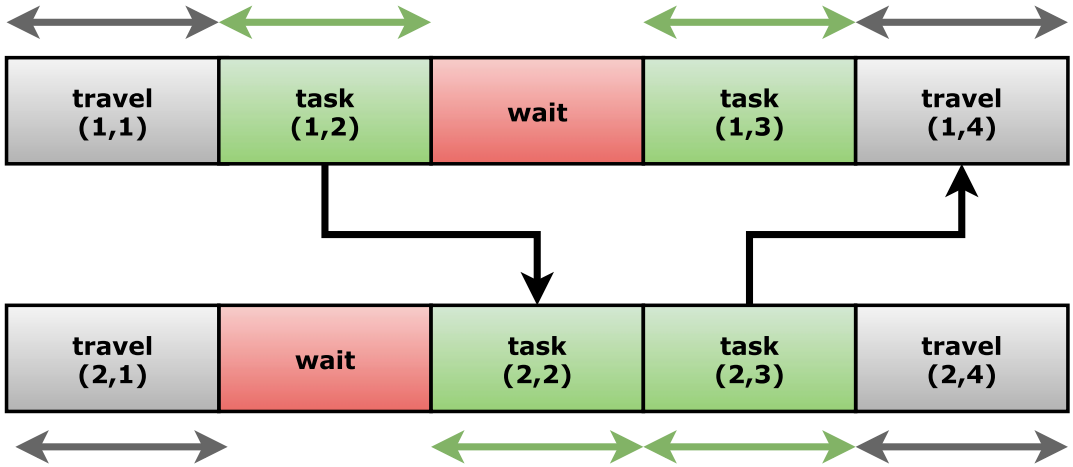

Figure 5.2: Example of a network consisting of two routes with dependencies between the routes

The first step is to represent the network as a graph and to include a source node and a sink node for both routes. The result of this step is shown in Figure 5.3.

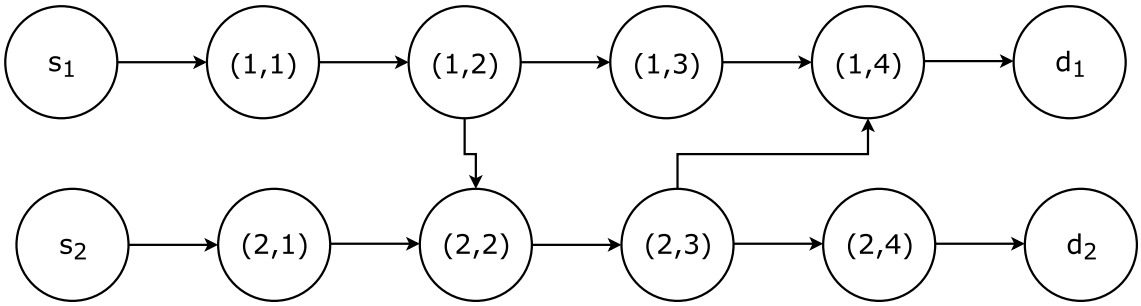

Figure 5.3: Graph representation with source and sink nodes added to the network

Next, the cost functions are constructed. A feasible order to do this is $s_1 \rightarrow s_2 \rightarrow (1,1) \rightarrow (2,1) \rightarrow (1,2) \rightarrow (2,2) \rightarrow (1,3) \rightarrow (2,3) \rightarrow (1,4) \rightarrow (2,4) \rightarrow d_1 \rightarrow d_2$. In this order none of the cost functions is constructed before all cost functions of its predecessors have been constructed. The resulting cost functions are shown in Figure 5.4. We start by creating the cost function of the source node of route 1. This node has no time windows, but as it precedes node 1 in route 1 and has no duration, we can take the time window of node (1,1) as the boundaries for the cost function. The source node has cost coefficient -1, so if this action is started at time 0, the cost is 0. Similarly, if the action is started at time 1, the cost is -1. For the source node of route 2 the same holds. As the source nodes have a duration of zero and no breaks or rests are scheduled, the cost functions for nodes (1,1) and (2,1) are the same as for the source nodes of the corresponding route. Next, for node 2 in route 1, the cost at time 1 is 0, as starting at time 1 at node 2 implies we should have started at time 0 at node 1 of the same route, against cost 0. Using the same reasoning, the cost at time 2 is -1. When constructing the cost function for node (2,2), we need to take into account the cost function of node (1,2) as well as that of node (2,1) to determine the feasible start times, as they are both direct predecessors of node (2,2). However, for the costs, by definition of the costs, we only look at the direct predecessor from the same route, which is node (2,1). So, if we want the start time of node (2,2) to be 2, the start time of node (2,1) should be 1, resulting in a cost of -1. In order for node (2,2) to start at time 3, the start time at node (2,1) should also be 1, as starting later at this node is not feasible. Waiting is allowed at no cost, resulting in cost -1 for node (2,2) at time 3. This procedure continues until the cost function of the sink node of route 2 has been constructed.

From the found cost functions, we can deduce the solution, by backtracking in the reversed order as in which the cost functions were constructed. So, we start at node d_2 with choosing the start time to be the earliest time at which the minimum costs are achieved. This is at time instant 5. For node d_1 we find the same value. Similarly, we find a start time of 4 for node (2,4), as well as for node (1,4). Then, for nodes (2,3) and (1,3) the start time becomes 3. Next, node (2,2) gets start time 2. When determining the start time for node (1,2),

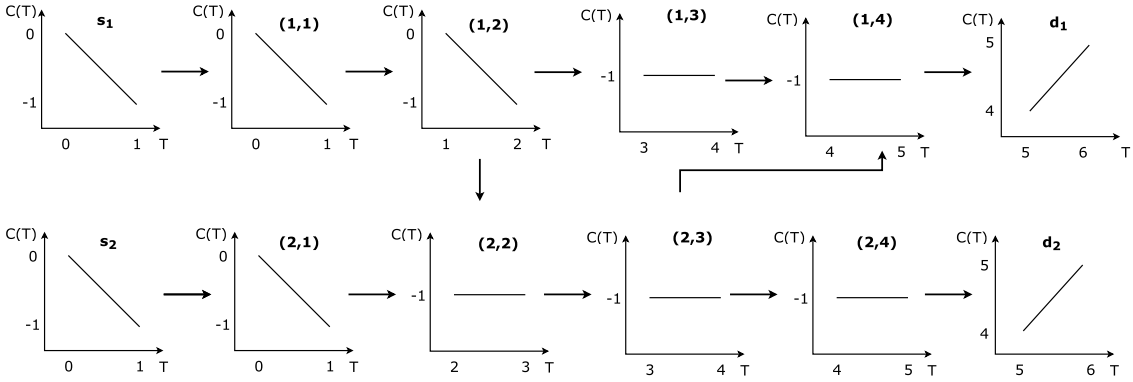

Figure 5.4: Cost functions of all actions

we need to check the start times of both node (2,2) and node (1,3). From node (1,3), we would choose start time 2, however, as seen from node (2,2), node (1,2) needs to start at time instant 1. In this case, the minimum is chosen, as otherwise the solution would not be feasible. The start time for node (1,2) thus becomes 1. Finally, node (2,1) gets start time 1 and node (1,1) is found to have start time 0. The source nodes of route 1 and route 2 get the same start times as nodes (1,1) and (2,1), respectively. So, in the optimal solution, route 1 starts at time 0 and route 2 starts at time 1. ■

5.9 Analysis

An overview of the algorithm as described above is given in Algorithm 5.

In the worst-case scenario, we need to create all solution possibilities and no solution is omitted by the dominance rules. The number of possibilities depends on the number of nodes in the entire network, as well as on the number of time windows each node has. First assume that a break or rest during a travel action is not possible. In this case, each node can either have a break, a rest, or neither. If present, the break or rest is taken before the action of the node starts. Combining this information we get a total of $\prod_{(r,n)\in V} 3t_{r,n}$ possibilities. Where $t_{r,n}$ is the number of time windows of node $(r,n) \in V$. An upper bound on this value is $(3t_{max})^{|V|}$, where $|V|$ is the number of nodes in the network and t_{max} is the maximum number of time windows a node in the network has. The

Algorithm 5 Labeling algorithm

```
Add source and sink nodes to every route in the network
Determine feasible order for scheduling all nodes in the network
for Every node do
    for Every partial solution do
        for Every time window do
            Create solution without break/rest
            Create solution with break
            Create solution with rest
        end for
    end for
    Apply dominance rules
end for
Find best solution
```

exponential growth of the number of created solutions immediately explains the need for dominance.

Now we extend this to the problem considered in this paper, where it is allowed to have a break or rest during travel. For every travel action there are three options: schedule a break, schedule a rest, or schedule neither. In the worst-case, all actions are travel actions. Then, for every action, combining the possibilities for breaks and rests before and during the action, we get nine possibilities. As a result, the upper bound on the number of solutions grows to $(9t_{max})^{|V|}$.

Chapter 6

Computational results

In this chapter the results of numerical experiments are shown. The data used for these experiments is described in Section 6.1. The results of the different experiments are shown in Section 6.2 to Section 6.4. First, experiments are done to find out until what number of nodes and what number of time windows in an instance, the exact approach can solve an instance within acceptable computation times. Next, we compare the heuristics with different sets of dominance rules to find out their performance. Finally, the performance of the construction algorithms is examined.

The labeling algorithm as described in this thesis is implemented in Java and ran on an Intel(R) Core(TM) i7-3740QM CPU at 2.70GHz with 8GB RAM. The slack algorithm as is currently used by ORTEC is implemented in C++ and ran on the same machine. To get reliable values for the computation times, all instances are ran five times. The given computation times are the average over these five runs. Note that, because the two algorithms are implemented in different languages, we cannot compare the computation times directly. However, the ratio between computation times still gives an indication of relative performance.

6.1 Data description

The data used to perform the numerical experiments consists of customer data. This ensures us that the data is representative for reality and gives a meaningful insight in the performance of the proposed algorithm compared to the current algorithm. The data set from which the instances are extracted, consists of 49 different routes, with the durations of these routes varying between 15 minutes and 384 hours. The average duration of a route is approximately 15 hours. All routes take place within Europe, more specifically, most of the routes take place in the Netherlands, Belgium, and Germany. As was shown before, the complexity depends on the number of nodes, the number of time windows per node, and the percentage of travels in a route. The average number of nodes per route is 9. On average 24 percent of all nodes are travels. The information given on the data consists of the order in which actions need to take place in which route, which driver performs which route, the duration of the actions, time windows on the actions, the type of the actions, and the dependencies between actions. All the actions have a single time window. In order to be able to test the algorithm with instances where the actions have multiple time windows, the single time windows are separated in equal pieces. In this way no opening time for the actions is lost, so this ensures us that a feasible solution can still be found.

6.2 Exact Approach

We are interested to see whether instances with certain characteristics can be solved by the exact approach within reasonable computation times. The two main factors having an influence on the number of partial solutions created are the total number of nodes in the network and the number of time windows per node. In the instances used, the number of nodes and the number of time windows are varied and we apply the exact approach of the labeling algorithm to these instances. The characteristics of the used instances are shown in Table 6.1.

#Nodes	#Dep/#R	Instance	#TW	Instance	#TW	Instance	#TW
10	0.5	1	1	11	2	21	3
11	0.5	2	1	12	2	22	3
16	0.5	3	1	13	2	23	3
19	1	4	1	14	2	24	3
20	2	5	1	15	2	25	3
25	0.5	6	1	16	2	26	3
28	1.67	7	1	17	2	27	3
31	0.6	8	1	18	2	28	3
35	0.6	9	1	19	2	29	3
40	1.5	10	1	20	2	30	3

Table 6.1: Characteristics of the instances used to test the algorithms

The first column gives the number of nodes in each instance. The second column shows the number of dependencies relative to the number of routes in an instance. The fourth column gives the maximum number of time windows in each instance, where the instance number is shown in the third column. Similarly, the sixth and the eight column give the number of time windows per action for the instance numbers mentioned in columns five and seven, respectively.

The results of the given instances, generated using the exact approach of the labeling algorithm, are shown in Tables 6.2, 6.3, and 6.4.

	Exact		**Slack**	
Instance	**CPU (s)**	**#Labels**	**CPU (s)**	**Solution / Exact**
1	0.01	3	0.3	1
2	0.02	12	0.3	1
3	0.03	3	0.6	1.02
4	0.5	120	1.1	1.07
5	0.02	12	0.8	1
6	4.0	3	1.0	1.05
7	140.7	9	0.7	1.06
8	191.0	432	1.5	1.01
9	845.4	3	1.5	1.04
10	929.5	336	1.4	1.04

Table 6.2: Results of the exact approach and the slack algorithm for the instances with one time window

The second and the third column of Table 6.2 show the results of the exact approach for instances with one time window. First, in the second column the CPU time is given in seconds. Next, in the third column, the number of labels at the end of the algorithm are shown. In the fourth column the computation time of the slack algorithm is shown in seconds. The fifth column gives the solution quality of the slack algorithm relative to the solution of the exact approach.

From this table we can see that small instances are solved by the labeling algorithm in a short amount of computation time, generally being faster than the slack algorithm. However, when the number of nodes in the network grows, the computation time rises quickly, whereas the computation times of the slack algorithm increase a lot slower. As a result, the computation times of the exact approach quickly outgrow those of the slack algorithm. Concerning the solution quality of the slack algorithm, we find that on average an improvement of 2.6 percent can be made by the labeling algorithm. The given results clearly show a trade-off between solution quality and computation time, as in general we will need more computation time to make an improvement in the solution quality.

The results for the exact approach for the instances with two time windows are shown in Table 6.3. Note that when no solution is listed for the exact approach, we can still compare the solution of the slack algorithm to the solution of the exact approach as, compared to the instances with single time windows, the optimal solution did not change.

	Exact		**Slack**	
Instance	**CPU (s)**	**#Labels**	**CPU (s)**	**Solution / Exact**
11	0.02	3	0.3	1
12	0.06	27	0.5	1
13	0.08	6	0.6	1.02
14	24.3	677	1.1	1.07
15	0.08	27	0.8	1
16	463.6	12	2.1	1.05
17	*	*	1.0	1.06
18	*	*	1.8	1.01
19	*	*	1.4	1.35
20	*	*	1.4	1.04

Table 6.3: Results of the exact approach and the slack algorithm for the instances with two time windows

As can be expected, the computation times have increased as compared to the instances with a single time window. Whenever there is a * mentioned in the CPU column, this implies that the instance was not solved within two hours. In general, the computation times increase with the number of nodes, as we expect. However, this trend does not hold for instance 15 which has a similar amount of nodes as instance 14, but requires a lot less computation time. This may be explained by the lower number of labels at the end of the algorithm for this instance as compared to instance 14. This may indicate that throughout the algorithm fewer solutions are created, which will result in a lower computation time.

For the slack algorithm, we can see a decrease in solution quality. While

with single time windows we found a deviation of 2.6 percent, this has now increased to 5.6 percent. This increase is clearly caused by one specific instance which deviates 35 percent from the optimal solution. When looking at this instance it seems that it does not have any special characteristics causing this large deviation. The bad solution quality is caused by more rest time being scheduled than strictly necessary.

	Exact		**Slack**	
Instance	**CPU (s)**	**#Labels**	**CPU (s)**	**Solution / Exact**
21	0.04	10	0.4	1
22	0.2	44	0.4	1
23	1.0	6	0.6	1.02
24	314.5	1456	1.4	1.07
25	2.0	132	0.6	1
26	*	*	2.3	1.05
27	*	*	1.0	1.06
28	*	*	2.2	1.01
29	*	*	1.5	1.35
30	*	*	1.5	1.04

Table 6.4: Results of the exact approach and the slack algorithm for the instances with three time windows

From the results of the instances with three time windows, we can see that, as expected, the computation times increase even more. This increase is so large that, even though among the five smallest instances the maximum computation time is approximately five minutes, the five largest instances cannot be solved within two hours. When looking at the computation times of the five instances which are solved within two hours, instance 24 clearly has the largest computation time. This may be explained by the large number of solutions which are left at the end of the algorithm, as this may indicate a large number of solutions being created throughout the algorithm.

The solution quality of the slack algorithm also becomes slightly worse as compared to instances with two time windows. The deviation from the optimal

solution is now 5.9 percent opposed to 5.6 percent for two time windows.

From the results of the exact approach we can conclude that, as we expected, the computation times generally grow with the number of nodes in the network, as well as with the number of time windows. However, when we look at the number of labels compared to the number of nodes in a network, it seems that there is not necessarily such a relationship. Sometimes the number of labels at the end of the algorithm are very high for a low number of nodes, whereas the number of labels is also sometimes very low for one of the larger instances.

6.3 Heuristics

In this section, the results are shown for the heuristics where different sets of dominance rules are applied. The same instances as before are used to examine the performance of these heuristics. The results of the three heuristics are compared to the exact approach and to the results of the slack algorithm.

6.3.1 Heuristic 1: One route

For the first heuristic, we only take into account one route in the dominance rules. The results of this heuristic are given in Table 6.5 (one time window), Table 6.6 (two time windows), and Table 6.7 (three time windows).

	Heuristic 1			**Slack**	
Instance	**CPU (s)**	**#Labels**	**H1 / Exact**	**CPU (s)**	**Solution / H1**
1	0.01	3	1	0.3	1
2	0.02	12	1	0.3	1
3	0.03	3	1	0.6	1.02
4	0.03	11	1.23	1.1	0.87
5	-	0	-	0.8	-
6	0.03	3	1.27	1.0	0.82
7	-	0	-	0.7	-
8	-	0	-	1.5	-
9	-	0	-	1.5	-
10	-	0	-	1.4	-

Table 6.5: Results of heuristic 1 - One route with one time window

Similar as in the results of the exact approach, the second and third column show the CPU time and the number of labels at the end of the algorithm. In addition, the fourth column shows the solution of the heuristic compared to the solution of the exact approach, when this heuristic solution is known. When the number of labels is shown to be zero, this implies that the heuristic did not find a feasible solution to the given instance. The fifth and sixth column show the results of the slack algorithm, where the solution is compared to that of the heuristic.

The most noticeable thing about the results of the first presented heuristic, is that often no feasible solution is found. Furthermore, whenever a feasible solution is found, the solution quality is rather variable. It may find the optimal solution occasionally, but most of the time, the solution is quite far from the optimum. The labeling algorithm finds a better solution than the slack algorithm only once. The computation times of the first heuristic are very low, making this heuristic faster than the slack algorithm. However, these computation times are reached at a high trade-off in solution quality.

	Heuristic 1			**Slack**	
Instance	**CPU (s)**	**#Labels**	**H1 / Exact**	**CPU (s)**	**Solution / H1**
11	0.02	3	1	0.3	1
12	0.06	27	1	0.5	1
13	0.07	6	1	0.6	1.02
14	0.1	24	1.25	1.1	0.87
15	-	0	-	0.8	-
16	-	0	-	2.1	-
17	-	0	-	1.0	-
18	0.05	12	1.07	1.8	0.94
19	-	0	-	1.4	-
20	-	0	-	1.4	-

Table 6.6: Results of heuristic 1 - One route with two time windows

From Table 6.6 we can see that the computation times are still low when the instances have two time windows. Similar as for the instances with one time window, only five out of ten of the instances are solved by this heuristic. However, it may be interesting to find that in this case the eighth instance is solved instead of the sixth. It may be the case that because of the multiple time windows, a label with one of the time windows dominates a label with the other time window, while in fact it would have been better to keep the first label. This can explain why no feasible solution is found for instance 16. On the other hand, more (and different) labels are created, which may cause instance 18 to now find a solution, where for instance 8 no solution was found.

	Heuristic 1			Slack	
Instance	**CPU (s)**	**#Labels**	**H1 / Exact**	**CPU (s)**	**Solution / H1**
21	0.07	10	1	0.4	1
22	0.2	44	1	0.4	1
23	1.2	6	1	0.6	1.02
24	0.8	37	1.23	1.4	0.87
25	-	0	-	0.6	-
26	-	0	-	2.3	-
27	-	0	-	1.0	-
28	0.2	12	1.07	2.2	0.94
29	-	0	-	1.5	-
30	-	0	-	1.5	-

Table 6.7: Results of heuristic 1 - One route with three time windows

When the first heuristic is applied to the instances with three time windows, the computation times increase somewhat, but stay in an acceptable range. When comparing the results of these instances to the instances with two time windows, it is remarkable that for the fourth instance the solution found is slightly better when three time windows are used instead of two. However, the solution is still 23 percent from the optimum.

From the results from heuristic 1 we can conclude that this heuristic performs rather badly, showing that it is important to take into account all different routes in the set of dominance rules. To understand why it is important to take into account all routes, consider two different partial solutions for an instance with two routes and let the last scheduled node be in route 1. Furthermore, let the first partial solution be solution 1, in this partial solution, the costs at the last scheduled node are lower than in the other partial solution, solution 2. On the other hand, let the drivers' legislation variables for route 1 be smaller in solution 2 as compared to solution 1. This implies that, for the last scheduled node of route 1, we will keep both partial solutions. Now assume that the next node is added to route 2 and let solution 2 dominate solution 1, when only looking at route 2. If we consider all routes, we would still keep both partial

solutions. However, in heuristic 1, solution 1 would now be omitted, while this solution may lead to a better solution than solution 2. This example also gives an indication as to why this heuristic does not find a feasible solution for a lot of instances, as we may omit a feasible solution at the cost of an infeasible solution.

6.3.2 Heuristic 2: One time

In the second heuristic, the dominance rules only take one start time into account instead of all start times. The results of the heuristic are compared to the exact approach and to the results of the slack algorithm. Tables 6.8, 6.9, and 6.10 show the results of heuristic 2.

	Heuristic 2			**Slack**	
Instance	**CPU (s)**	**#Labels**	**H2 / Exact**	**CPU (s)**	**Solution / H2**
1	0.01	3	1	0.3	1
2	0.01	2	1	0.3	1
3	0.03	3	1	0.6	1.02
4	0.3	2	1	1.1	1.07
5	0.02	2	1	0.8	1
6	0.1	3	1	1.0	1.05
7	7.2	9	1	0.7	1.06
8	117.8	1	1	1.5	1.01
9	32.8	3	1	1.5	1.04
10	156.3	2	1	1.4	1.04

Table 6.8: Results of heuristic 2 - One time with one time window

The results as were found by use of the second heuristic for instances with a single time window are shown in Table 6.8. From these results we can see that the performance of the second heuristic is a lot better than those of the first heuristic. For all the presented instances, this heuristic finds the optimal solution. However, just like in the exact approach the computation times grow rather quickly with the number of nodes. Still, the computation time for the

largest instance (instance 10) is almost six times lower than that of the exact approach, while obtaining the exact same solution.

	Heuristic 2			**Slack**	
Instance	**CPU (s)**	**#Labels**	**H2 / Exact**	**CPU (s)**	**Solution / H2**
11	0.02	3	1	0.3	1
12	0.03	4	1	0.5	1
13	0.1	3	1	0.6	1.02
14	5.0	4	1	1.1	1.07
15	0.05	4	1	0.8	1
16	-	0	-	2.1	-
17	1413.9	36	1	1.0	1.06
18	*	*	*	1.8	*
19	*	*	*	1.4	*
20	*	*	*	1.4	*

Table 6.9: Results of heuristic 2 - One time with two time windows

From Table 6.9 we can see that for one instance no feasible solution is found. Moreover, for the three largest instances, the instances do not terminate within two hours, while the slack algorithm takes at most approximately 2 seconds. For the instances for which the heuristic finds a feasible solution within two hours, it always finds the optimal solution.

	Heuristic 2			Slack	
Instance	**CPU (s)**	**#Labels**	**H2 / Exact**	**CPU (s)**	**Solution / H2**
21	0.03	9	1	0.4	1
22	0.07	6	1	0.4	1
23	0.5	3	1	0.6	1.02
24	46.5	6	1	1.4	1.07
25	1.1	6	1	0.6	1
26	23.6	9	1.01	2.3	1.04
27	6722.4	216	1	1.0	1.06
28	*	*	*	2.2	*
29	*	*	*	1.5	*
30	*	*	*	1.5	*

Table 6.10: Results of heuristic 2 - One time with three time windows

Surprisingly, the instance for which no solution was found when the instance had two time windows, now does find a solution. This solution is not optimal, but it is rather close, with a deviation of one percent.

6.3.3 Heuristic 3: Last break/rest

The dominance rules of the third heuristic only look at the last scheduled break or rest of the different routes in a network. The results of this heuristic are shown in Tables 6.11, 6.12, and 6.13.

	Heuristic 3			**Slack**	
Instance	**CPU (s)**	**#Labels**	**H3 / Exact**	**CPU (s)**	**Solution / H3**
1	0.02	2	1	0.3	1
2	0.01	8	1	0.3	1
3	0.04	13	1.22	0.6	0.84
4	0.06	32	1	1.1	1.07
5	0.03	16	1	0.8	1
6	0.3	84	1	1.0	1.05
7	0.2	64	1	0.7	1.06
8	0.6	94	1.03	1.5	1.01
9	1.5	168	1	1.5	1.04
10	18.0	512	1	1.4	1.04

Table 6.11: Results of heuristic 3 - Last break/rest with one time window

From the results of the third heuristic we can see that for all instances a feasible solution is found. In eight out of ten instances this is the optimal solution. For the other two instances, the deviations vary substantially. The solution of instance 8 is rather good, with a deviation of only three percent from the optimum it outperforms the solution found by the slack algorithm. On the other hand, the solution quality of instance 3 is really bad with a deviation of 22 percent from the optimum, which is clearly worse than the solution found by the slack algorithm.

	Heuristic 3			**Slack**	
Instance	**CPU (s)**	**#Labels**	**H3 / Exact**	**CPU (s)**	**Solution / H3**
11	0.01	2	1	0.3	1
12	0.02	8	1.04	0.5	0.96
13	0.04	13	1.22	0.6	0.84
14	0.07	32	1.31	1.1	0.82
15	0.04	16	1.03	0.8	0.97
16	0.3	234	1	2.1	0.94
17	0.3	192	1.26	1.0	0.84
18	0.8	132	1.08	1.8	0.94
19	-	0	-	1.4	-
20	20.5	1536	1.18	1.4	0.88

Table 6.12: Results of heuristic 3 - Last break/rest with two time windows

When the instances have two time windows instead of one, we find that for one instance no feasible solution is found anymore. Moreover, the solution quality decreased remarkably. Now, only for two out of ten instances the optimal solution is found and on average the solutions deviate 12 percent from the optimum. This deviation is considerably more than the deviation of the slack algorithm from the optimum (5.6 percent).

	Heuristic 3			Slack	
Instance	**CPU (s)**	**#Labels**	**H3 / Exact**	**CPU (s)**	**Solution / H3**
21	0.01	2	1	0.4	1
22	0.02	8	1.18	0.4	0.85
23	0.04	13	1.18	0.6	0.86
24	0.08	32	1.41	1.4	0.76
25	0.05	16	1.14	0.6	0.87
26	0.3	234	1.04	2.3	0.92
27	0.3	192	1.34	1.0	0.79
28	0.9	132	1.10	2.2	0.92
29	1.3	468	1.10	1.5	1.23
30	26.3	1536	1.28	1.5	0.81

Table 6.13: Results of heuristic 3 - Last break/rest with three time windows

Compared to the instances with two time windows, when a third time window is added, the solution quality decreases, even though it was bad already. The deviation from the optimum increases from 12 to 18 percent.

Overall, what we can see from the results of the third heuristic is that, when the instances have single time windows, the performance is reasonably good. However, when the instances have two or three time windows, the solution quality decreases dramatically. This can be explained by the fact that neither the cost functions nor the possible starting times are taken into account in the dominance rules. As a result, whenever we have two partial solutions, for example both without any breaks or rests, but actions scheduled in different time windows, the solutions are treated as being equally good. In this case, we assume that it does not matter which solution we keep and thus the solution we would actually want to keep may be omitted. In case of single time windows, this situation is not possible. An approach to achieve better results using this heuristic may be to add an extra dominance rule, stating that the start times of the dominating solution should at least cover the start times of the dominated solutions. This is the same dominance rule as is stated in the exact approach and in the first heuristic.

In Figure 6.1 a comparison is made between the number of labels created throughout the algorithm, when using the exact approach or one of the three heuristics. For this comparison, instance 4 is used as the number of nodes in this instance is close to the average number of nodes over the ten instances and this instance can be solved by each approach of the labeling algorithm.

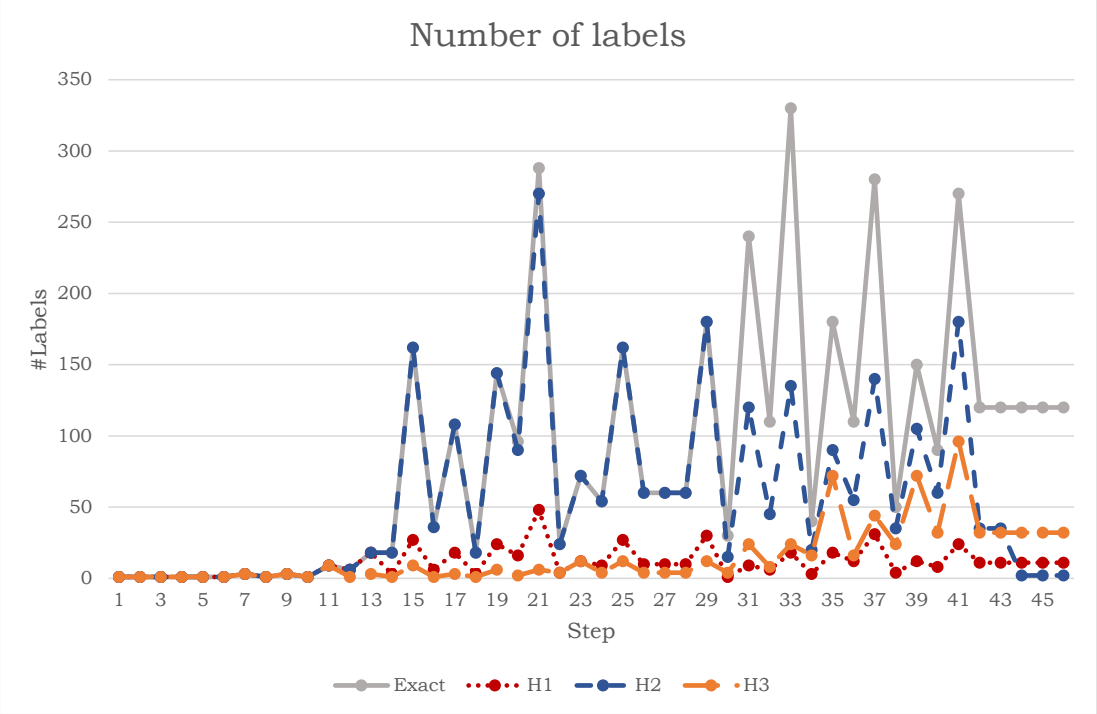

Figure 6.1: The number of labels throughout the labeling algorithm for the exact approach and the three heuristics

From the figure we can see that mainly heuristic 1 and heuristic 3 clearly create fewer labels than the exact approach. Even though the second heuristic has very few labels left at the end of the algorithm, throughout the algorithm, until approximately two third of all steps, this heuristic usually has a similar amount of labels as the exact approach. With help of the given figure, the relative computation times can easily be explained. The heuristic which generally has the lowest number of labels (heuristic 1) also has the lowest computation time (0.03 seconds), followed by heuristic 3 (0.06 seconds), heuristic 2 (0.3 seconds), and the exact approach (0.5 seconds).

6.4 Construction algorithms

In this section, the results of the presented construction algorithms are shown. For these algorithms, we make use of the set of dominance rules from the exact approach, in order to not have more than one heuristic element in the algorithms. The results are shown for different values as pre-set bounds on the drivers' legislation variables.

6.4.1 Construction algorithm 1

The results of the first construction algorithm for two different parameters are given in Table 6.14 (one time window), Table 6.15 (two time windows), and Table 6.16 (three time windows). Similar to the results presented before, the CPU time is given in seconds, the number of labels at the end of the algorithm are given, and the solution relative to the exact solution is shown.

	50%			**75%**			**Slack**	
#	**CPU(s)**	**#L**	**Sol/Exact**	**CPU(s)**	**#L**	**Sol/Exact**	**CPU(s)**	**Sol/Exact**
1	0.02	7	1	0.01	1	1	0.3	1
2	0.03	34	1	0.02	15	1	0.3	1
3	0.03	13	1.02	0.02	1	1.02	0.6	1.02
4	0.2	238	1	0.08	100	1	1.1	1.07
5	0.09	102	1	0.02	15	1	0.8	1
6	1.1	7	1	0.1	1	1	1.0	1.05
7	0.8	28	1	0.06	5	1	0.7	1.06
8	0.07	10	1	-	-	-	1.5	1.01
9	187.9	7	1	0.2	1	1	1.5	1.04
10	3.6	1020	1	0.1	100	1	1.4	1.04

Table 6.14: Results of the first construction algorithm with one time window

From Table 6.14, we can see that this algorithm always finds a feasible solution when the bounds for the drivers' legislation variables are set to 50 percent of the actual limits. In nine out of ten cases, the optimal solution is

found, and in the other case the deviation is limited to two percent. When the parameter is increased to 75 percent, the computation times decrease, as fewer solutions are created. The average computation time has decreased by 99.6 percent. The disadvantage of increasing the parameter is that no longer a feasible solution is found for every instance, even though this remains limited to one instance for which no feasible solution is found.

When we compare the results of the construction algorithm to the results of the slack algorithm, we find that when we use 50 percent as parameter, the construction algorithm always finds a solution with equal quality or better compared to the slack algorithm. However, the average computation time of the construction algorithm is 19.4 seconds, whereas this is 0.9 seconds for the slack algorithm. If we look at the construction algorithm with a parameter of 75 percent, we find that once no feasible solution is found, but other than that, we always find a solution which is at least as good as the solution found by the slack algorithm. In this case, the average computation time is 0.07 seconds, which is thus faster than the slack algorithm.

	50%			**75%**			**Slack**	
#	**CPU(s)**	**#L**	**Sol/Exact**	**CPU(s)**	**#L**	**Sol/Exact**	**CPU(s)**	**Sol/Exact**
11	0.01	7	1	0.01	1	1	0.3	1
12	0.06	77	1	0.03	30	1	0.5	1
13	0.09	25	1.02	0.04	2	1.02	0.6	1.02
14	2.4	837	1	1.1	486	1	1.1	1.07
15	0.4	231	1	0.04	30	1	0.8	1
16	77.1	20	1	4.9	2	1	2.1	1.05
17	75.6	84	1	0.5	15	1	1.0	1.06
18	0.8	10	1	-	-	-	1.8	1.01
19	-	-	-	-	-	-	1.4	1.35
20	141.2	4185	1	1.2	486	1	1.4	1.04

Table 6.15: Results of the first construction algorithm with two time windows

For the instances with two time windows, we find similar results as for the instances with single time windows. The computation times are substantially

smaller when a higher percentage is used as parameter. However, for one less instance a feasible solution is found when the parameter is increased from 50 percent to 75 percent. The average computation times of the construction algorithm (with 75 percent) and the slack algorithm are quite similar, with an average CPU of 1.0 seconds for the construction algorithm against 1.1 seconds for the slack algorithm. Like before, whenever the construction algorithm finds a feasible solution, it is always at least as good as the solution found by the slack algorithm.

	50%			**75%**			**Slack**	
#	**CPU(s)**	**#L**	**Sol/Exact**	**CPU(s)**	**#L**	**Sol/Exact**	**CPU(s)**	**Sol/Exact**
21	0.02	24	1	0.01	3	1	0.4	1
22	0.1	124	1	0.05	54	1	0.4	1
23	0.8	37	1.02	0.3	4	1.02	0.6	1.02
24	28.4	1976	1	1.8	276	1	1.4	1.07
25	29.8	1116	1	0.2	42	1	0.6	1
26	1271.5	20	1	4.7	10	1	2.3	1.05
27	*	*	*	27.7	75	1	1.0	1.06
28	*	*	*	-	-	-	2.2	1.01
29	*	*	*	35.0	30	1	1.5	1.35
30	*	*	*	7.1	132	1	1.5	1.04

Table 6.16: Results of the first construction algorithm with three time windows

When the instances have three time windows per action, the computation times increase so fast when the parameter is set to 50 percent that the largest four cases cannot be solved within two hours (indicated by a *). When the parameter is set to 75 percent, we also see a large increase in computation times. Compared to the instances with two time windows the average computation time increases from 1.0 seconds to 8.5 seconds, while for the slack algorithm this increase is a lot smaller, from 1.1 seconds to 1.3 seconds on average.

6.4.2 Construction algorithm 2

For the second version of the presented construction algorithm, the same parameter settings as for the first version are used. The results are given in Table 6.17 (one time window), Table 6.18 (two time windows), and Table 6.19 (three time windows).

	50%			**75%**			**Slack**	
#	**CPU(s)**	**#L**	**Sol/Exact**	**CPU(s)**	**#L**	**Sol/Exact**	**CPU(s)**	**Sol/Exact**
1	0.01	7	1	0.01	1	1	0.3	1
2	0.03	34	1	0.02	15	1	0.3	1
3	0.03	9	1	0.02	2	1	0.6	1.02
4	0.2	238	1	0.2	100	1	1.1	1.07
5	0.09	102	1	0.02	15	1	0.8	1
6	2.7	7	1	1.0	1	1	1.0	1.05
7	0.8	28	1	0.6	5	1	0.7	1.06
8	0.07	10	1	0.1	1	1	1.5	1.01
9	4018.6	7	1	1.1	1	1	1.5	1.04
10	155.6	1020	1	0.2	70	1	1.4	1.04

Table 6.17: Results of the second construction algorithm with one time window

From Table 6.17, we can see that we find the optimal solution for all instances. When we use 75 percent as parameter, the average computation time is decreased by 99.93 percent compared to using 50 percent as a parameter, while still always obtaining the optimal solution. When we compare the results of the construction algorithm (75 percent) to the results of the slack algorithm, we find that on average the computation times of the construction algorithm (0.3 seconds) are lower than the computation times of the slack algorithm (0.9 seconds). Also, the construction algorithm also finds the optimal solution, whereas for the slack algorithm this is not true.

	50%			**75%**			**Slack**	
#	**CPU(s)**	**#L**	**Sol/Exact**	**CPU(s)**	**#L**	**Sol/Exact**	**CPU(s)**	**Sol/Exact**
11	0.01	7	1	0.01	1	1	0.3	1
12	0.07	77	1	0.03	30	1	0.5	1
13	0.06	10	1	0.04	2	1	0.6	1.02
14	2.1	864	1	4.6	918	1	1.1	1.07
15	0.5	231	1	0.04	30	1	0.8	1
16	356.0	20	1	-	-	-	2.1	1.05
17	50.2	84	1	9.5	15	1	1.0	1.06
18	2.1	10	1	-	-	-	1.8	1.01
19	-	-	-	-	-	-	1.4	1.35
20	1798.4	4050	1	1.1	324	1	1.4	1.04

Table 6.18: Results of the second construction algorithm with two time windows

When the instances have multiple time windows, we do not longer find a feasible solution for every instance. However, when a solution is found, it is optimal. The computation times vary widely, with the largest instance taking almost half an hour.

	50%			**75%**			**Slack**	
#	**CPU(s)**	**#L**	**Sol/Exact**	**CPU(s)**	**#L**	**Sol/Exact**	**CPU(s)**	**Sol/Exact**
21	0.02	24	1	0.01	3	1	0.4	1
22	0.2	124	1	0.05	54	1	0.4	1
23	0.5	10	1	0.1	6	1	0.6	1.02
24	20.5	2054	1	1.9	276	1	1.4	1.07
25	30.8	1116	1	0.2	162	1	0.6	1
26	*	*	*	8.8	10	1	2.3	1.05
27	*	*	*	45.4	15	1	1.0	1.06
28	*	*	*	-	-	-	2.2	1.01
29	*	*	*	39.5	30	1	1.5	1.35
30	*	*	*	8.0	2592	1	1.5	1.04

Table 6.19: Results of the second construction algorithm with three time windows

For instances with three time windows, we see the same as for the first construction algorithm, the computation time becomes so large for the largest instances, that it does not finish within two hours. When we look at the increase of average computation time with the increase in time windows, we see that the computation times of the construction algorithm grow faster than that of the slack algorithm. Whereas the average computation time of the slack algorithm only increases from 0.9 seconds for single time windows to 1.3 seconds for three time windows, the computation times for the construction algorithm (75 percent) increase from 0.5 seconds to 11.6 seconds.

In Figure 6.2 a comparison is made between the labels throughout the algorithm for the exact approach and the two construction algorithms. The same instance as before was used (instance 4). As a parameter, 75 percent is used, as this parameter produced the best results overall.

As can be expected, we find that the first construction algorithm, fewer labels are created. In this algorithm, a break or rest is only scheduled in a travel whenever this is actually needed. As the second construction heuristic always creates labels with a break and a rest for a travel, more labels are pro-

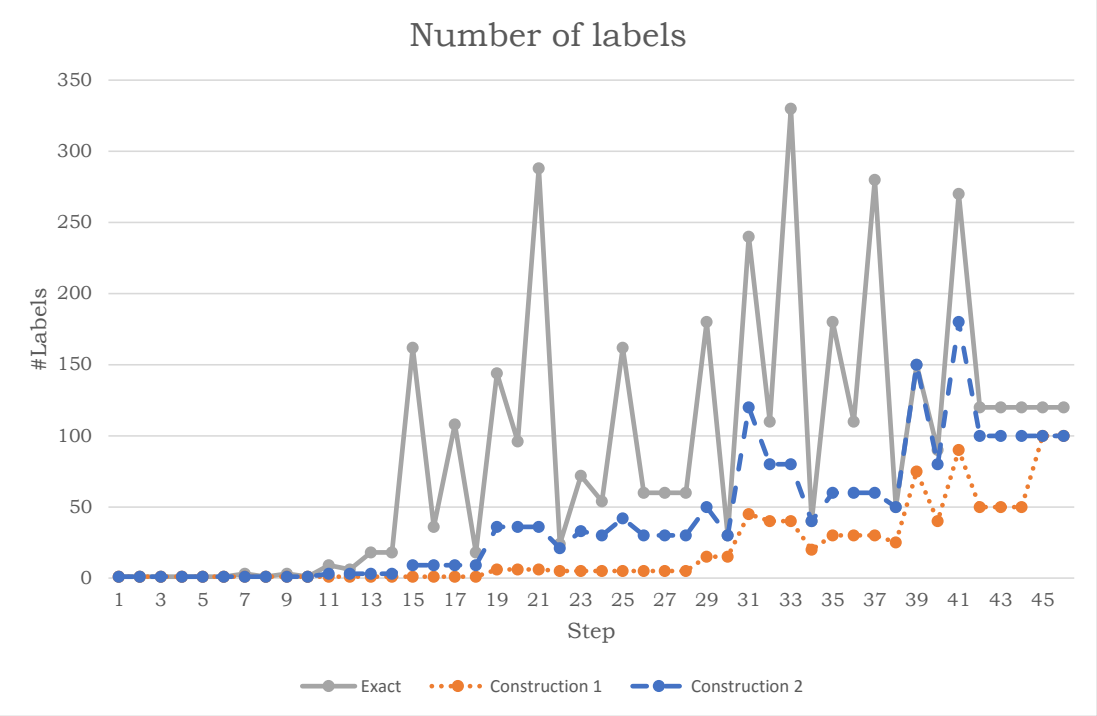

Figure 6.2: The number of labels throughout the labeling algorithm for the exact approach and the two construction algorithms

duced. From the figure, the computation times can easily be explained as the algorithm with the lowest number of labels (construction algorithm 1) also has the lowest computation time (0.08 seconds). The second lowest computation time is reached by the second construction heuristic (0.2 seconds) which also has the second lowest overall number of labels. Finally, the exact approach, with most labels, also has the highest computation time of 0.5 seconds.

Chapter 7

Conclusion

In this thesis we have looked at the problem of departure time optimization in a network of dependent routes. The main goal of the research was to construct an algorithm in which all routes can be optimized simultaneously, such that dependencies between routes do not form a problem. Other constraints which should be taken into account are the drivers' legislation and multiple time windows. Furthermore, the developed algorithm should be fast, as in practice it is applied often.

The main contribution as compared to the algorithm currently used by ORTEC, is that by using the labeling algorithm, the complete network is optimized simultaneously rather than route by route. The dependencies do not form any problem in this solution approach and do not add anything to the complexity, as was shown in Section 5.9. However, one of the other constraints, the drivers' legislation, makes the problem still hard to solve. Because of the drivers' legislation rules, in our solution approach, we need to construct a lot of labels, thereby increasing computation times. The other characteristic which increases complexity is the presence of multiple time windows.

From the numerical experiments, we find that for small instances the labeling algorithm can find the optimal solution. However, the instances become too large to solve within a reasonable computation time for the exact approach rather quickly. Instances of reasonable size can therefore never be solved using this approach. The heuristics can solve larger instances, with varying computation times and varying solution quality. The first heuristic which was

presented, only looking at one route in the dominance rules, does not find a feasible solution at all for about half of the instances. When it does find a feasible solution, the computation time is generally very low and the solution quality quite varying, reaching the optimum in 9 out of the 15 solved cases. The second heuristic, considering only one start time in the dominance rules, does find a feasible solution in almost all cases. However, the computation time still increases so fast with the increase of the number of nodes that the largest instances cannot be solved within two hours of computation time whenever the instances have two or three time windows. The third heuristic performs quite well when the instances have one time window. It reaches the optimal solution in eight out of ten instances and the computation times are clearly lower than for the exact approach. However, whenever more time windows are added to the instances, the third heuristic performs very badly. With average deviations of 12 and 18 percent from the optimal solution when there are two or three time windows, respectively.

Besides the different sets of dominance rules, we also presented two versions of a construction algorithm. These algorithms show a more promising performance than the initial labeling algorithm, with lower computation times and good solution quality. The overall best-performing algorithm, is the second construction algorithm with 75 percent as a parameter. This algorithm not only outperforms the exact labeling algorithm, but also the slack algorithm, both on solution quality and computation times. For the instances with single time windows, this construction algorithm always finds the optimal solution in an average computation time of 0.3 seconds, whereas the slack algorithm shows an average deviation from the optimum of 2.6 percent with an average computation time of 0.9 seconds.

Chapter 8

Further research

In this chapter some recommendations are given for possible extensions of the algorithms provided in this paper. First of all, some constraints which were not taken into account, may be interesting to be part of future research. Other than that, it may also be beneficial to consider slightly different algorithms, which are still based on the presented labeling algorithm to try and reach a good solution quality in low computation times.

8.1 Time-dependent travel times

In the algorithm as presented in this paper, the travel times between customers are assumed to be fixed. However, in reality the travel time between two locations may be affected by the time of the day and the area in which the traveling takes place. Time-dependent travel times are taken into account in the model of Kok et al. (2011). The authors make use of a travel speed step function for every link in the network, resulting in a piecewise linear travel time function. With this way of travel time modeling, the non-passing property holds, meaning that a vehicle which departs later in time than a different vehicle also arrives later in time than the first vehicle. When time-dependent travel times are taken into account in the labeling algorithm presented, one change would take place in the drivers' legislation functions. These functions will consist of more pieces in case of time-dependent travel times as for different

starting times the travel time is different and so, the values will be updated to different values. As a result, more computations need to be made for the construction of the drivers' legislation functions, most likely resulting in an increase in computation time.

8.2 Drivers' legislation

The drivers' legislation rules which are taken into account in this paper are only the daily rules of the European drivers' legislation. An extension to this could be to also consider weekly rules and rules concerning night shifts, as these rules of course also need to be adhered to in reality. Additionally, the rules in the European drivers' legislation have some exceptions, which may be taken into account. The main rules and exceptions not taken into account in this paper are:

- The daily driving time is not allowed to exceed 9 hours, except twice a week when it can be extended to 10 hours.
- The total weekly driving time may not exceed 56 hours. The total driving time in a fortnight may not exceed 90 hours.
- The daily rest period should be at least 11 hours. This may be reduced to 9 hours for a maximum of three times a week. The daily rest may also be split into a 3 hours rest followed by a 9 hours rest, making a total of 12 hours daily rest.

When different types of rest periods are taken into account, like the split of an 11 hour rest period, this would imply that more solutions need to be created in the algorithm, as more possibilities now exist. For example, the rule stating that the daily rest may be split into a 3 hours rest followed by a 9 hours rest, would imply that we need to try all possibilities to establish those rests. In this case, we need to try both the possibility to take a rest of 11 hours and to split the rest, trying out every possibility of scheduling the 3- and 9-hour rests. This will highly increase the number of possible solutions and will lead to increased memory usage and higher computation times. It is likely that

only smaller instances can be solved and that the computation times will rise. On the other hand, the possibilities for a rest are now more flexible, making it possible to find a solution with a shorter network duration.

8.3 Construction algorithms

Besides the construction algorithms already presented in this thesis, other types of construction algorithms could be investigated. One aspect of the presented construction algorithms is that it decides before an action whether we will try to plan a break or a rest. However, we also have the information of the duration of the action. As a result, we can also calculate the values of the drivers' legislation variables at the end of this action. If we only plan a rest when strictly needed, or a break if strictly needed, very few solutions are created, which will highly benefit the computation time. The risk in this approach is that because of waiting time the solutions can still become infeasible. It may then be beneficial to always schedule a break or rest in a travel, like in the second construction algorithm presented.

Moreover, the parameters in the presented construction algorithms can be optimized. In the numerical experiments, the same parameter was used for each of the drivers' legislation variables. However, maybe it benefits the performance if different values are used.

Finally, in the numerical experiments for the given construction algorithms, the exact set of dominance rules was used. However, it may be interesting to see what happens to the solution quality when stricter dominance rules are applied, like the sets of dominance rules as they are presented in this paper for three heuristics.

8.4 Linear number of labels

In the heuristic approaches presented in this thesis, the number of labels is dependent on the dominance rules applied. Because of this approach the number of labels can grow exponentially. A different approach could be to limit the number of labels to either a fixed constant, or to some amount linear to the

number of nodes. In this way, we have a bigger influence on the number of labels throughout the algorithm and computation times will be limited. The challenge of this approach is to find a way as to which labels will be stored and which ones will be deleted. The labels can be selected on for example the cost functions, the drivers' legislation functions, or a combination of those two.

8.5 Optimize route by route

In this thesis, we focused on optimizing a complete network simultaneously. From the numerical results, we could see that computation times grow exponentially with the number of nodes in the network and that small instances can be solved within very low computation times. It may, therefore, potentially be beneficial to apply the labeling algorithm to separate routes. In order to still take into account the dependencies between routes, we can still use the time windows of a predecessor when calculating the feasible start times of a node. Once the start time of a node is fixed, we can adjust the time windows of its direct predecessors and successors in other routes to assure feasibility.

When optimization is done route by route, a decision needs to be made on the order in which the optimization is done. Whether the order of optimization has a big influence on the performance of the algorithm could be a topic of interest for further research, just like the order of optimization itself.

Bibliography

Ascheuer, N., Jünger, M., and Reinelt, G. (2000). A branch & cut algorithm for the asymmetric traveling salesman problem with precedence constraints. *Computational Optimization and Applications*, 17(1):61–84.

Baptiste, P., Le Pape, C., and Nuijten, W. (2012). *Constraint-based scheduling: applying constraint programming to scheduling problems*. Springer Science & Business Media.

Bredström, D. and Rönnqvist, M. (2007). A branch and price algorithm for the combined vehicle routing and scheduling problem with synchronization constraints. *NHH Dept. of Finance & Management Science Discussion Paper*, (2007/7).

Bredström, D. and Rönnqvist, M. (2008). Combined vehicle routing and scheduling with temporal precedence and synchronization constraints. *European journal of operational research*, 191(1):19–31.

Dohn, A., Rasmussen, M., and Larsen, J. (2011). The vehicle routing problem with time windows and temporal dependencies. *Networks*, 58(4):273–289.

Drexl, M. (2012). Synchronization in vehicle routing-a survey of vrps with multiple synchronization constraints. *Transportation Science*, 46(3):297–316.

Goel, A. (2012). The minimum duration truck driver scheduling problem. *EURO Journal on Transportation and Logistics*, 1(4):285–306.

Ioachim, I., Gelinas, S., Soumis, F., and Desrosiers, J. (1998). A dynamic programming algorithm for the shortest path problem with time windows and linear node costs. *Networks*, 31(3):193–204.

Kok, L., Hans, E., and Schutten, J. (2011). Optimizing departure times in vehicle routes. *European Journal of Operational Research*, 210(3):579–587.

Kolisch, R. and Hartmann, S. (2006). Experimental investigation of heuristics for resource-constrained project scheduling: An update. *European journal of operational research*, 174(1):23–37.

Labadie, N., Prins, C., and Yang, Y. (2014). Iterated local search for a vehicle routing problem with synchronization constraints. In *ICORES*, pages 257–263.

Rousseau, L., Gendreau, M., and Pesant, G. (2003). *The synchronized vehicle dispatching problem*. Citeseer.

Solomon, M. (1987). Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations research*, 35(2):254–265.

Stojković, M. and Soumis, F. (2001). An optimization model for the simultaneous operational flight and pilot scheduling problem. *Management Science*, 47(9):1290–1305.

Valls, V., Ballestín, F., and Quintanilla, S. (2004). A population-based approach to the resource-constrained project scheduling problem. *Annals of Operations Research*, 131(1-4):305–324.

Appendix A

Mathematical Programming Formulation

In this appendix a mathematical programming formulation of the problem considered in this paper is given. In this formulation we make use of locations and dummy locations. At every location some service needs to be executed, where at the dummy locations this service has duration zero. At every (dummy) location we can decide to take a break or a rest before or after service. The purpose of the dummy locations is that in this way we can schedule a break or a rest during a travel.

In the next sections we first present the notation on sets, parameters, and variables, followed by the model.

A.1 Sets

$R = \{1, ..., T\}$	Set of routes
$L_r = \{0, ..., N_r + 1\}$	Set of locations of route r
$C_r = \{1, ..., N_r\}$	Set of customer locations of route r
$D_r = \{2, 4, ..., N_r\}$	Set of dummy locations in route r
$P^{prec} = \{((r, n), (u, v)) \mid r \in R, n \in L_r, u \in R, v \in L_u\}$	Set of dependencies between nodes

A.2 Parameters

$t_{r,n}$ = number of time windows at location n in route r
$l^j_{r,n}$ = time window j lower bound at location n of route r, $j \in \{1, ..., t_{r,n}\}$

$u^j_{r,n}$ = time window j upper bound at location n of route r, $j \in \{1, ..., t_{r,n}\}$
$k_{r,n}$ = service duration at location n of route r
b^{break}_{min} = minimum duration of break
b^{rest}_{min} = minimum duration of rest period
d^{acc}_{max} = maximum accumulated driving time
d^{daily}_{max} = maximum daily driving time
w^{acc}_{max} = maximum accumulated working time
w^{daily}_{max} = maximum daily working time
$c_{r,(n,n+2)}$ = traveling time from location n to location $n+2$ in route r, where both locations are original locations
$adt_{r,0}$ = start value of the accumulated driving time for route r
$awt_{r,0}$ = start value of the accumulated working time for route r
$ddt_{r,0}$ = start value of the daily driving time for route r
$dwt_{r,0}$ = start value of the daily working time for route r
M = arbitrarily large number

A.3 Variables

$C_{r,(n,n+1)}$ = traveling time from location n to location $n+1$ in route r, where both locations can be either an original location or a dummy location
$X^s_{r,n}$ = start time of service at location n of route r
$X^d_{r,n}$ = departure time from location n of route r
$W^s_{r,n}$ = waiting time before service at location n of route r
$W^d_{r,n}$ = waiting time after service at location n of route r
$B^{s,l}_{r,n}$ = break or rest time before service at location n of route r, $l \in \{break, rest\}$
$B^{d,l}_{r,n}$ = break or rest time after service at location n of route r, $l \in \{break, rest\}$

$$Y^{p,l}_{r,n} = \begin{cases} 1, & \text{if a break or rest is taken at location } n \text{ of route } r,\ p \in \{s,d\},\ l \in \{break, rest\} \\ 0, & \text{otherwise} \end{cases}$$

$$I^j_{r,n} = \begin{cases} 1, & \text{if, the } j\text{th time window at location } n \text{ of route } r \text{ is used} \\ 0, & \text{otherwise} \end{cases}$$

A.4 Model

$$\text{Min.} \sum_{r \in R} (X^s_{r,N+1} - X^d_{r,0}) \tag{A.1}$$

$$\text{s.t.} \quad X^s_{r,n} + k_{r,n} \leq X^s_{u,v} \qquad \forall ((r,n),(u,v)) \in P^{prec} \tag{A.2}$$

$$X^s_{r,n} = X^d_{r,n-1} + C_{r,(n-1,n)} + W^s_{r,n} \qquad \forall r \in R, \forall n \in \{1, ..., N_r + 1\} \tag{A.3}$$

$$X^d_{r,n} = X^s_{r,n} + k_{r,n} + W^d_{r,n} \qquad \forall r \in R, \forall n \in \{1, ..., N_r + 1\} \tag{A.4}$$

$$\sum_{j=1}^{t_{r,n}} I^j_{r,n} = 1 \qquad \forall r \in R, \forall n \in C_r \tag{A.5}$$

$$X^s_{r,n} \geq l^j_{r,n} I^j_{r,n} \qquad \forall r \in R, \forall n \in C_r, \forall j \in \{1, ..., t_{r,n}\} \tag{A.6}$$

$$X^s_{r,n} \leq u^j_{r,n} + M(1 - I^j_{r,n}) \qquad \forall r \in R, \forall n \in C_r, \forall j \in \{1, ..., t_{r,n}\} \tag{A.7}$$

$$W^p_{r,n} \geq b^l_{min} Y^{p,l}_{r,n} \qquad \forall r \in R, \forall n \in C_r, p \in \{s,d\}, l \in \{break, rest\} \tag{A.8}$$

$$B^{p,l}_{r,n} \leq M Y^{p,l}_{r,n} \qquad \forall r \in R, \forall n \in C_r, p \in \{s,d\}, l \in \{break, rest\} \tag{A.9}$$

$$B^{p,l}_{r,n} \leq W^p_{r,n} \qquad \forall r \in R, \forall n \in C_r, p \in \{s,d\}, l \in \{break, rest\} \tag{A.10}$$

$$\sum_{n=i}^{j} C_{r,(n,n+1)} \leq d^{acc}_{max} + M\Big(\sum_{l \in \{break,rest\}} \sum_{p \in \{s,d\}} \sum_{n=i+1}^{j-1} Y^{p,l}_{r,n}\Big) \qquad \forall r \in R, \forall i,j \in C_r, i < j \tag{A.11}$$

$$\sum_{n=i}^{j} C_{r,(n,n+1)} \leq d^{daily}_{max} + M\Big(\sum_{p \in \{s,d\}} \sum_{n=i+1}^{j-1} Y^{p,rest}_{r,n}\Big) \qquad \forall r \in R, \forall i,j \in C_r, i < j \tag{A.12}$$

$$X^d_{r,j} - X^s_{r,i} \leq w^{acc}_{max} + M\Big(\sum_{l \in \{break,rest\}} Y^{d,l}_{r,i} + Y^{s,l}_{r,j} + \sum_{p \in \{s,d\}} \sum_{n=i+1}^{j-1} Y^{p,l}_{r,n}\Big) \quad \forall r \in R, \forall i,j \in C_r, i < j \tag{A.13}$$

$$X^d_{r,j} - X^s_{r,i} \leq w^{daily}_{max} + M\Big(Y^{d,rest}_{r,i} + Y^{s,rest}_{r,j} + \sum_{p \in \{s,d\}} \sum_{n=i+1}^{j-1} Y^{p,rest}_{r,n}\Big) \qquad \forall r \in R, \forall i,j \in C_r, i < j \tag{A.14}$$

$$adt_{r,0} + \sum_{n=0}^{j} C_{r,(n,n+1)} \leq d^{acc}_{max} + M(\sum_{l \in \{break,rest\}} \sum_{p \in \{s,d\}} \sum_{n=1}^{j-1} Y^{p,l}_{r,n}) \qquad \forall r \in R, \forall j \in C_r \quad \text{(A.15)}$$

$$ddt_{r,0} + \sum_{n=0}^{j} C_{r,(n,n+1)} \leq d^{daily}_{max} + M(\sum_{p \in \{s,d\}} \sum_{n=1}^{j-1} Y^{p,rest}_{r,n}) \qquad \forall r \in R, \forall j \in C_r \quad \text{(A.16)}$$

$$awt_{r,0} + (X^{d}_{r,j} - X^{s}_{r,0}) \leq w^{acc}_{max} + M(\sum_{l \in \{break,rest\}} \sum_{p \in \{s,d\}} \sum_{n=1}^{j-1} Y^{p,l}_{r,n}) \qquad \forall r \in R, \forall j \in C_r \quad \text{(A.17)}$$

$$dwt_{r,0} + (X^{d}_{r,j} - X^{s}_{r,0}) \leq w^{daily}_{max} + M(\sum_{p \in \{s,d\}} \sum_{n=1}^{j-1} Y^{p,rest}_{r,n}) \qquad \forall r \in R, \forall j \in C_r \quad \text{(A.18)}$$

$$C_{r,(n,n+1)} + C_{r,(n+1,n+2)} = c_{r,(n,n+2)} \qquad \forall r \in R, \forall (n+1) \in D_r \quad \text{(A.19)}$$

$$C_{r,(n,n+1)} \geq 0 \qquad \forall r \in R, \forall n \in L_r \quad \text{(A.20)}$$

$$X^{p}_{r,n} \geq 0 \qquad \forall r \in R, \forall n \in L_r, p \in \{s,d\} \quad \text{(A.21)}$$

$$W^{p}_{r,n} \geq 0 \qquad \forall r \in R, \forall n \in C_r, p \in \{s,d\} \quad \text{(A.22)}$$

$$I^{j}_{r,n} \in \{0,1\} \qquad \forall r \in R, \forall n \in L_r, \forall j \in \{1,...,t_{r,n}\} \quad \text{(A.23)}$$

$$Y^{p,l}_{r,n} \in \{0,1\} \qquad \forall r \in R, \forall n \in C_r, p \in \{s,d\}, l \in \{break,rest\} \quad \text{(A.24)}$$

$$B^{p,l}_{r,n} \geq 0 \qquad \forall r \in R, \forall n \in C_r, p \in \{s,d\}, l \in \{break,rest\} \quad \text{(A.25)}$$

The objective (A.1) of this problem is to minimize the overall route duration. The possible dependencies are modeled by constraints (A.2). They state that the servicing of a customer at location n in route r must be done before the service at location v in route u can start. Constraints (A.3) give the definition of the start time of servicing a customer at location n in route r. The start time of service is the departure time from the previous location plus the traveling time between the two locations plus the waiting time before service at the customer starts. Similarly, constraints (A.4) give the definition of the departure time from a location, which is the start time of servicing the customer plus the service duration plus the waiting time at the location after servicing. In constraints (A.5) it is made sure that exactly one of the time windows at a location is used. Constraints (A.6) and (A.7) make sure that service at a location starts within one of the specified time windows. In constraints (A.8) it is checked whether a waiting period can be considered a break or a rest. If this is not the

case, the variable $Y_{r,n}^{p,l}$ is forced to become zero. In this case the break or rest time is also forced to be zero by constraints (A.9). By constraints (A.10) the break or rest time cannot exceed the waiting time.

Constraint sets (A.11) - (A.18) are all about checking the drivers' legislation. First, by constraints (A.11) it is checked whether the driving time never exceeds the maximum allowed driving time until a break. Similarly, constraints (A.13) check the working time until a break is taken. Constraints (A.12) and (A.14) check whether the daily driving time and daily working time never exceed the set limits. In constraints (A.12) and (A.16) the same happens, except it is now checked whether the daily driving time is not exceeded. Constraints (A.14) and (A.18) check whether the daily working time is not exceeded. Constraints (A.15) - (A.18) also check the drivers' legislation rules, but now before the first break or rest is taken. In this case the amount of driving and working time present at the start of the route are taken into account. This is relevant in case the driver has worked before starting this route, without having a break or rest in between.

Constraints (A.19) assure that the driving time from one customer to the next customer, via a dummy location, is the same as the initial driving time between the two customers.

Finally, constraints (A.24) - (A.25) model the domains of the variables.