Universiteit Utrecht

ORTEC
OPTIMIZE YOUR WORLD

Master's Thesis Mathematical Sciences

Synchronization in Simultaneous Vehicle and Crew Routing and Scheduling Problems with Breaks

Author:
Thomas R. Visser
3373487

Supervisor – UU:
Dr. ir. J.M. van den Akker

Second reader – UU:
Prof. dr. R.H. Bisseling

Supervisors – ORTEC:
Dr. ir. A.L. Kok
Dr. ir. S.W.A. Haneyah
Prof. dr. J.A. dos Santos Gromicho

February 23, 2015

Abstract

In this thesis, we investigate a practical simultaneous vehicle and crew routing and scheduling problem arising in distribution transportation: goods need to be delivered from a central depot to customers using trailers and truck+driver combinations. Trailers may need to be reloaded by other personnel at the depot during the planning. Allowing drivers to switch trailers during reloading has great savings potential but difficult synchronization constraints arise. The problem becomes even more complex if social legislation on driving times for drivers is considered. We solve this problem by two-stage decomposition: first generation of trailer routes and then assignment of route sections (trips) to truck+driver resource shifts, including a simplified driving rule for break planning in both stages. Different resource assignment solution methods able to handle synchronization constraints (temporal interdependencies) are investigated and compared, including construction heuristics and column generation methods with exact and heuristic pricing. The exact pricing problem is modelled as an ESPPRC with additionally linear node costs and it is solved exactly by a labelling algorithm. Mathematical properties can be used to significantly speed-up this exact pricing method. Further, different (predictive) break scheduling strategies during the trailer routing stage are investigated. Computational experiments on both modified benchmark instances (100–200 customers) and real-world data from a major Australian distributor show $\sim$ 10–25% less truck+drivers are needed when switching trailers during reloading is allowed.

Acknowledgements

I would like to thank everyone at ORTEC for the wonderful time I had during my internship. A special thanks goes to my (former) colleagues from the OSD PD ORD/OSP group: Bas, Chagiet, Erik, Gerben, Joep, and Joost for helping me with my minor ORD developments and (maybe more importantly) keeping our spirits very high (with candy on Friday, carnival techno tunes, VRP 'programming' in MIT Scratch, ...), making our 'vlek' on the 6th floor of the Zoetermeer HQ definitely the most fun place to go to each workday. I miss those days already.

I would also like to thank my ORTEC supervisor and dear colleague Sameh Haneyah for his continuous support on keeping this thesis and our plans structured, our very useful weekly discussions on scoping and keeping track of our projects through 'Jira'.

Also I would like to thank my ORTEC supervisor, professor and Innovations director Joaquim dos Santos Gromicho for his endless enthusiasm, very creative ideas, and his particular encouragement of my own made colourful graphs and pictures. Further, I would like to thank him for suggesting and supporting me for a PhD-programme in the field of real-world VRP solutions.

I would like to thank my daily ORTEC supervisor and group manager, Leendert Kok, for his infinite help, support and patience. Thanks for our countless inspiring discussions on the subject, helping me to understand the many intricate difficulties of solving real-life transportation problems, especially concerning driving/working time legislation and related break scheduling, but also inspiring me to develop new methods to solve these problems. I am very happy we were able to test some of the methods on customer data and I believe we gained very useful knowledge on potential solution methods for 'CRAS 2.0'.

I would like to thank my UU supervisor, Marjan van den Akker, for her great inspiration, patience, ideas and comments during our weekly/monthly talks. Especially thanks for the exciting ideas related to column generation methods and seemingly endless possibilities of mathematical programming methods in general. Also many thanks for taking the time to understand the difficult real-life problems faced by ORTEC and some of my attempted solution methods, written in this thesis.

Also I would like to thank my UU Second reader, Rob Bisseling, for his time and ability to read this thesis so quickly before the deadline.

These final months were particularly stressful because of a hard starting deadline set by the PhD-project. Besides thanks to all my supervisors for enabling me to graduate faster than originally planned, I would like to thank my parents, sister, close friends and none the least my girlfriend for their continuous love and support.

Contents

Chapter 1

Introduction

This thesis is done in fulfilment of the Mathematical Sciences master at the University of Utrecht and as part of an internship at ORTEC (http://www.ortec.nl), a large Netherlands based software company specialized in the development of Operations Research aided planning solutions. ORTEC has a large customer base, including well-known companies like Air-France KLM, Shell, Walmart, Coca-Cola Enterprises, DHL, PostNL, Albert.nl and Simon Loos. One core business of ORTEC is developing advanced transportation planning and scheduling software. Their routing product, ORTEC Routing and Dispatching (ORD), is used by large logistic companies to solve complex real-world puzzles arising in transportation and distribution networks, reducing operation costs, vehicle mileage, fuel consumption and CO_2-emissions, while also satisfying difficult requirements such as social legislation on driving/working time.

Recently, transportation customers of ORTEC have become increasingly interested in considering trailers and drivers separately instead of fixing them for a complete planning, due to their different characteristics. Trailers need to be (re)loaded with goods at a distribution center before delivering them to customers, but are not required to follow legislation on driving/working time. Often at distribution companies, which handle transportation in the retail industry, drivers do not have to participate in trailer loading at a distribution center (depot), since at this location loading personnel is already present, but drivers are obligated to follow driver's legislation and regularly take breaks/rests. Also, in general, operating costs of a single trailer are much less than those of a single driver. Treating trailers and drivers separately but optimizing their planning simultaneously can potentially reduce operation costs while gaining additional planning flexibility for fulfilling difficult requirements. Currently, sophisticated solution methods inside planning software ORD are able to simultaneously optimize non-fixed trailer routes and driver shifts, but require drivers to execute full trailer routes at once, allowing them to switch trailers only after they have completed the whole trailer route. In a typical distribution planning, a trailer needs to visits the depot multiple times during its route to be reloaded with goods. In ORD generated schedules, drivers have to wait during the reloading, which is not very realistic. Allowing drivers to switch trailers at the depot during the trailer route not only matches the real-world situation better, it also has the potential of eliminating those waiting times from the driver shifts and further increase planning flexibility. For these reasons, an increasing number of ORTEC's transportation customers likes to have such functionality. However, this makes the optimization problem significantly harder, since it introduces very difficult *synchronization* constraints between trailers and drivers. Only very recently are such problems being investigated in vehicle routing literature [22, 24]. The additional requirement of fulfilling social legislation on driving/working time makes this problem even more complex.

The aim of this thesis to investigate the additional benefit of drivers switching trailer during reloading, while we also investigate how to solve difficulties concerning synchronization effi-

ciently. We present and compare different solution methods able to deal with synchronization constraints and (simplified) break planning arising in simultaneously planning trailer routes and drivers shifts while allowing drivers to switch trailers during trailer reloading. The full problem is decomposed in two stages: trailer routing and resource (truck+driver) assignment. We focus primarily on different solution methods for the second stage problem, in which synchronization plays a major role. Construction heuristics, as well as Column Generation methods with exact and heuristic pricing are formulated and compared on benchmark instances modified from literature and on real-world data from an ORTEC customer. Additionally, the impact of different (predictive) break planning strategies in the first planning stage on the final solution is investigated.

The thesis is organized as follows. In Chapter 2, we formulate the full problem and propose a two-stage decomposition to solve this problem. Relevant literature will be discussed in Chapter 3. In Chapter 4, we present an overview of our proposed solution methods and how some methods work together. Construction based heuristic methods will be presented more elaborately in Chapter 5, while column generation based methods are presented in Chapter 6. Results from our computational experiments are presented in Chapter 7. The thesis concludes with a discussion and some directions for future research in Chapter 8.

Chapter 2

Problem description

In this chapter, detailed description of the problem and some of its variants considered in this thesis are presented. We first describe the complete problem, the *Simultaneous Vehicle and Crew Routing and Scheduling Problem* (SVCRSP) [24]. Important features of this problem include driving time legislation, which is described in more detail in Section 2.2. A mathematical description of the SVCRSP will be given in Section 2.3. A very general and promising way to solve the complete SVCRSP is to decompose the problem in two stages [24]. This idea is also supported by the way large transport companies like to do their planning and the way professional Routing and Dispatching software (ORD) from ORTEC currently is organized. We discuss such a decomposition by describing the relevant stage-one and stage-two problem descriptions in Section 2.5. Also we discuss the mathematical advantages and difficulties of this decomposition.

2.1 SVCRSP with breaks

Inspired by the problems faced by large distribution companies using trailers, trucks and truck drivers, we consider the following simultaneous vehicle and crew routing and scheduling problem. A transportation *order* or *request* consist of the delivery of certain goods from the distribution center (depot) to a customer. Each order has a fixed *servicing time* at the customer, as well as a *(hard) time-window* associated with the earliest- and latest possible time that servicing can start at the customer. It is possible for a vehicle to arrive early at a customer and wait until the service can start. A *homogeneous* fleet of *trailers* is available to transport the orders, with each having the same *maximum capacity*. We consider the orders to have a single capacity requirement *dimension*, such as volume, mass or number of pallets. All order are eligible to be transported by all trailers. Each trailer has a fixed start/end location, which it needs to be positioned at the start/end of the *planning horizon*. Usually, these locations are at the depots and the start and end location of a trailer are the same. The planning horizon, which is the complete period of time considered to be scheduled, can consist of one or multiple workdays. Although in this thesis, we consider a fleet of homogeneous trailers, the model can be fairly easy be adapted to work with a fleet of *heterogeneous* trailers.

We consider a *trip* to be a sequence of customer visits, starting and ending at the depot executed by a single trailer. During the planning horizon, a single trailer can execute multiple trips between which (re)loading at depot can take place. The sequence of trips of a single trailer, starting/ending at the trailer beginning/end location, is considered to be a trailer *route*. Before the start of each trip, the loading of goods in a trailer at the depot is needed. We consider a fixed *loading time* at the start of each trip.

The trailers represent so-called *passive* vehicles [22, 55], since they need *trucks* and *truck drivers* to be carried and driven, respectively. Trucks therefore represent *active* vehicles, while drivers can be considered *active* crew members. Since most transport companies consider (pre-determined) truck and driver *combinations*, such a combination can be modelled as a special kind of (active) crew member, while the trailers can be modelled as (passive) vehicles in the SVCRSP.

We consider crew members to each have a *(resource) shift*, which represents a period in time during which these crew members are available to work. Multiple trailer trips are assigned to a single resource shift. In this context, we refer to an already planned non-empty trailer trip which needs to be assigned to a single resource shift as a *(route) section*. Route sections represent the smallest consecutive pieces/jobs of a planned trailer route which need to be assigned to a driver. This explicitly excludes the depot (re)loading parts in a trailer route, since in most cases truck drivers are not assigned to do this loading. In contrast, service times at the customer are part of the section, since usually the driver is responsible for servicing a customer. Note, in principle trips and (route) sections are the same, but the term trip is used when planning trailer routes, while the term (route) section is used to denote already planned trailer trips when doing resource assignment.

Also, simplified social legislation on driving/work time is considered. In real-life, many countries have laws stating the maximum amount of time a truck driver can drive or work consecutively before he/she is forced to take a *break* (rest period). Note that such social legislation is only applicable to driving/working crew members, not to the passive forms of transport. This fact alone poses a mayor challenge in optimizing vehicle routes and crew schedules. To limit the complexity but to still model some important concepts of social legislation with its consequences, we provide a simplified concept of planning breaks after a maximum amount of consecutive driving time. More details on these *driving rules* in our model is presented in Section 2.2, where we also show how this simplification is related to the real-life EU social legislations and similar legislations in other countries.

In most vehicle routing and crew scheduling problems, crew member shifts are explicitly assigned to a single vehicle/route. As in [24], we explicitly consider the possibility of a crew member to switch passive vehicles inside a shift. To be more precise, a truck/driver combination can execute multiple trips (sections) of different routes, possibly switching trailers between the trips (sections) at the depots. As stated in [22, 24], the allowance for a driver to switch trailers poses a special kind of *synchronization* between the crew members and the passive vehicles, making the SVCRSP harder to solve than most general vehicle routing problems. The execution of a single order requires both an active and a passive means of transport, which need to be synchronized in time and place [55]. This is however motivated by the different characteristics/restrictions of the crew members versus the passive vehicles. As stated, crew members are affected by the driver's and work-time regulations, whereas the passive vehicles are for instance affected by the fixed loading time between trips. Therefore under certain circumstances, it could be beneficial to switch. This is discussed in more detail in Section 2.4.

In this problem, the objective is to minimize the total execution cost of a complete schedule. There could be fixed cost for using certain (resource) shifts and trailers, as well as cost depending on the distance travelled by the trailers/trucks. Also there could be cost induced for not executing a given order. Inspired by costs logistic companies have in real world, we assume that the fixed operational costs of using a trailer is (much) less than the fixed costs of using a (driver) shift. Therefore, the key is to reduce the number of (driver) resource shifts, while this reduction may possibly lead to a (small) increase in the number of trailers used.

2.2 Driving Rules

In many real-life applications, truck drivers are required to follow legislation on the maximum number of hours of driving time and on the maximum number of hours of working time before a break/rest period needs to be taken. Although this legislation differs from country to country, we shall give a brief overview of the European Union drivers' working hour regulations. Regulations of other important countries/continents, such as the US, Canada and Australia, have a similar structure.

2.2.1 European Driver Rules

In the European Union, there are currently two social directives concerning the accumulated driving/working time: Regulation (EC) No. 561/2006 [27] on *driving time* and Directive 2002/15/EC [26] on *working time*. We use a description from Kok et al. [48] to explain some of the important aspects of these rules. See their paper for a more detailed review of these rules.

Driving time is the time a truck driver spends driving a truck, but not the time the truck driver spends servicing or waiting (standby) at a customer/depot. Regulation (EC) No. 561/2006 [27] states that after 4.5 hours of accumulated driving time, the driver **must** take a break of at least 45 minutes. He/she may not work during this break, so servicing a customer during this time is prohibited. The duration of this break can be reduced to 30 minutes only if during the driving period an additional break of 15 minutes has been taken. This is generally referred to as *splitting* the driving time break [48]. There are also rules concerning maximum daily and weekly total driving time and daily and weekly rest periods. Since our planning problem usually concerns only one of multiple workdays, and drivers are assumed to start daily rested, these rules are not important to our problem.

Working time is the time a truck driver spends driving, but also the time working, which includes servicing or waiting at a customer when no break is scheduled. Directive 2002/15/EC [26] states that after 6 hours of accumulated working time, a break of at least 30 minutes **must** be taken. The duration of the break has to be extended to 45 minutes if the total working time between two daily rests exceeds 9 hours. Also working time break splitting is allowed when the breaks are split into parts of at least 15 minutes. There are also rules concerning maximum weekly working time, which we again neglect.

2.2.2 Simplified Driving Rules

Optimization under both social directives turns out to be very difficult, even when applied to a basic vehicle routing problem with time windows [48] or even a shortest path problem with time windows [23]. The following two aspects make planning these breaks especially hard:

- **Unforced breaks**
  Although it is in general better to plan breaks at the very end of a full 4.5 hour driving or full 6 hour working period, situations can occur when it is better to plan a break earlier (even without splitting). For instance, consider a driver must wait 45 minutes at a certain customer but has not accumulated 4.5 hours of driving time yet. Planning the break later, after the full 4.5 hour, could cause the driver to arrive too late at the next customer, while planning the break earlier fits nicely in the driver's waiting time. Breaks planned earlier than 'necessary' are usually called *unforced* breaks, while breaks planned at the end of a full driving period are called *forced* breaks [48]. Unforced breaks make the problem hard

since there are many possible positions for the breaks in the planning which must all be investigated in order to find a good solution. Allowing the splitting of breaks enlarges the number of possibilities even more, thus making the problem even harder.

- **Time-dependent working time breaks**
  Working time breaks generally pose additional problems: their optimal position in the planning is time-dependent. Even if only forced working time breaks are allowed, their position in the planning are highly dependent on the time a driver departs from his/her start location and subsequent locations. This poses additional challenges during optimization, especially in our problem of simultaneous vehicle routing and crew scheduling.

Since it is outside the scope of this thesis to consider a full set of legal rules of driving/working time and breaks, but we are interested in the behaviour of such breaks in our full simultaneous vehicle routing and crew scheduling problem, we pose the following simplified driving rule by neglecting the above two difficult aspects:

- **Forced and unsplit driving time breaks only**
  After **precisely** $T^{\text{driv}}_{\max} = 270$ minutes (4.5 hours) of accumulated driving time (not earlier nor later), the driver must take a break of $T^{\text{break}} = 45$ minutes. This break is therefore forced. We suppose the driver can always find a location en-route to have this break and we do not consider a special breaking location other than the driver's current location at the moment the breaks commences. No splitting of the breaks is allowed. Also no working time related breaks and other kind of daily/weekly rests are considered.

Although the EU and similar legislation is much simplified by this rule, the addition of such driving rule should already reflect more of the real-world planning situation that simply planning without any break rules. Also, certain algorithmic assumptions, such as static driving times or static trip/section durations cannot be made with the simple rule, like in more complex VRPTW solution procedures handling a complete set of rules.

2.3 Mathematical Formulation

Our SVCRSP with breaks can be mathematically formulated as follows: we consider a complete directed graph $\mathcal{G} = (V, A)$ with set of vertices $\mathcal{V}$ and set of directed arcs $\mathcal{A}$. The set of vertices consists of $\mathcal{V} = \{v_o, v_d\} \cup \mathcal{V}_C$, with vertices v_o, v_d denoting the depot (also called *distribution centre* (DC)) and the set of customers $\mathcal{V}_C = \{v_1, v_2, \ldots, v_n\}$. We have a set $\mathcal{R}$ of (homogeneous) trailers, each with a capacity Q_r, which are used for delivering goods from a central depot to the set of customers vertices $\mathcal{V}_C$. In this thesis we assume the trailers are homogeneous with fixed capacity $Q_r = Q$, but in general they can have different capacity (heterogeneous fleet). Further we have a set of $\mathcal{S}$ truck+driver resource shifts available to drive the trailers. Each customer vertex $v_i \in \mathcal{V}_C$ has a demand of q_i goods, service time T^{serv}_i and a time window $[a_i, b_i]$, with a_i the earliest time and b_i the latest time servicing may start at customer v_i. If a vehicle (truck+driver and trailer) arrives before time a_i at the customer, it must wait, but we assume it can wait at the customer's location without any cost penalty. Each arc $e_{ij} \in \mathcal{A}$ has a travel time t_{ij} and a travel cost c_{ij} when travelling with both truck+driver and trailer, which is usually based on the travel time t_{ij} or distance. In this thesis, we will assume $c_{ij} = t_{ij}$ on all arcs $e_{ij} \in \mathcal{A}$. Since the capacity of a trailer is limited, it can return to the depot during the planning and be re-loaded. We have a planning horizon $[0, T]$, in which each trailer makes one route consisting of a sequence of trips and also each truck+driver has one shift consisting of a sequence of trips. Let $\mathcal{T}$ denote the set of trips, $\mathcal{R}$ the set of trailer routes and $\mathcal{S}$ the set of truck+driver resource shifts. We will

refer to planned (non-empty) trips in a trailer route as (route) *sections*, representing jobs to be assigned to a truck+driver resource shift. We denote the depot by two vertices: v_o and v_d, with $q_o = q_d = 0$, $T_o^{\text{serv}} = T_d^{\text{serv}} = 0$, $a_o = a_d = 0$, $b_o = b_d = T$. Each trip consist of a sequence of vertices, starting at depot vertex v_o and ending at depot vertex v_d. A trip containing only the trivial arc e_{od} is considered empty, and further $t_{od} = c_{od} = 0$. Each non-empty trip needs to be assigned to both a single trailer route $r \in \mathcal{R}$ and a single truck+driver shift $s \in \mathcal{S}$. Between two consecutive trips k, l in a trailer route r, trailer loading takes place with a duration of Δ_{kl}^r. Note: in literature this loading time usually depends on (some fraction) of the service times of the customers visited by the consecutive trip. Although this could be possible in our model, we decided to leave the loading time fixed, $\Delta_{kl}^r = \Delta$, as this resembles more the practical situation where differences in partial/full trailer loading times do not differ much. Further, we assume without loss of generality that the first trips of each trailer route are loaded during the time interval $[-\Delta, 0]$. Drivers are able to switch trailers at the depot without waiting during the loading time. We assume this switching does not take additional time, since this time in real-life is usually negligible. But they need to take breaks according to our simple driving rule stated earlier. After $T_{\max}^{\text{driv}}$ of cumulative driving time (sum over t_{ij}) after their last taken break, they immediately take an en-route break with duration of T^{break}. Further waiting time at depot or en-route is not converted to/considered as breaking. Trailers are not subjected to this driving rule, although they wait en-route if the assigned driver takes a break. Let $\mathcal{S}$ be the set of truck+driver shifts. We can additionally define trailer availability time-windows $\left[a_s^{\text{tr}}, b_s^{\text{tr}}\right]$ for each trailer route $r \in \mathcal{R}$ and truck+driver shift availability time-windows $\left[a_s^{\text{sh}}, b_s^{\text{sh}}\right]$ for each shift $s \in \mathcal{S}$, meaning that all trips in that route/shift must start and end inside such time-window. In this thesis, we generally assume these time-windows span the whole planning horizon $[0, T]$, but note when it is otherwise. Let c_r^{tr} be the cost for using trailer route $r \in \mathcal{R}$ (using means executing a non-empty route), and c_s^{sh} for using truck+driver shift $s \in \mathcal{S}$. The primary objective is to first minimize the sum of fixed cost for using trailers and truck+driver shifts, and secondary objective is to minimize the sum of arc-costs c_{ij}.

We can formulate this Simultaneous Vehicle Routing and Crew Scheduling Problem with Breaks (SVCRSP) as a Mixed Integer Linear Program (MILP). The formulation is not intended to be as compact as possible, since we will not solve this MILP directly. It is stated here only to describe the problem and the decomposition approach mathematically, although the introduced notation will be used throughout this thesis.

We use the following variables:

- Binaries x_{ij}^τ indicate if arc $e_{ij} \in \mathcal{A}$ is used in trip $\tau \in \mathcal{T}$.
- Binaries γ_i^τ indicate if customer $i \in \mathcal{V}_C$ is serviced in trip $\tau \in \mathcal{T}$.
- Continuous variables T_i^τ represent the time vertex $i \in \mathcal{V}$ is serviced in trip $\tau \in \mathcal{T}$, 0 otherwise.
- Continuous variables $T^{\text{driv}}{}_i^\tau$ represent the accumulated driving time after the last break taken by the assigned truck+driver arriving at vertex $i \in \mathcal{V}$ in trip $\tau \in \mathcal{T}$, 0 otherwise.
- Integers β_{ij}^τ indicate how much breaks must be taken by the assigned truck+driver on arc $e_{ij} \in \mathcal{A}$ in trip $\tau \in \mathcal{T}$. Let $\beta_{\max}$ be the maximum number of breaks possible on a single arc: $\beta_{\max} := \lceil \frac{\max_{e_{ij} \in \mathcal{A}} t_{ij}}{T_{\max}^{\text{driv}}} \rceil$.
- Continuous variables S_τ, C_τ represent the start, completion time of trip $\tau \in \mathcal{T}$. These are only here to simplify notation: $S_\tau = T_o^\tau$ and $C_\tau = T_d^\tau$.

- Continuous variables S_τ^{driv}, C_τ^{driv} represent the accumulated driving time after last break taken by the assigned truck+driver at the start, completion of trip $\tau \in \mathcal{T}$. These are only here to simplify notation: $S_\tau^{\text{driv}} = T^{\text{driv}}{}_o^\tau$ and $C_\tau^{\text{driv}} = T^{\text{driv}}{}_d^\tau$.
- Binaries y_{kl}^r indicate if trip $l \in \mathcal{T} \cup \{o, d\}$ is immediately executed after trip $k \in \mathcal{T} \cup \{o, d\}$ $(k \neq l)$ by trailer route $r \in \mathcal{R}$. Here, dummy trips o and d represent the start/end of a trailer route, with y_{od}^r indicating if trailer route $r \in \mathcal{R}$ is empty.
- Binaries ρ_τ^r indicate if trip $\tau \in \mathcal{T}$ is assigned to trailer route $r \in \mathcal{R}$.
- Binaries m_r^{tr} indicate if trailer route $r \in \mathcal{R}$ is non-empty (contains at least one (non-empty) trip).
- Binaries z_{kl}^s indicate if trip $l \in \mathcal{T} \cup \{o, d\}$ is immediately executed after trip $k \in \mathcal{T} \cup \{o, d\}$ $(k \neq l)$ by truck+driver shift $s \in \mathcal{S}$. Here also, dummy trips o and d represent the start/end of a truck+driver shift, with z_{od}^s indicating if truck+driver shift $s \in \mathcal{S}$ is empty.
- Binaries σ_τ^s indicate if trip $\tau \in \mathcal{T}$ is assigned to truck+driver shift $r \in \mathcal{R}$.
- Binaries m_r^{sh} indicate if truck+driver resource shift $s \in \mathcal{S}$ is non-empty (contains at least one (non-empty) trip).

Let M be a large enough number. Our SVCRSP with breaks can then be formulated as follows. Roman numbers I and II between the brackets on the right indicate if constraints are solved in the first or second stage, respectively, of our two-decomposition, which will be described in Section 2.5.

$$\min. \sum_{\tau\in\mathcal{T}} \sum_{(i,j)\in\mathcal{A}} c_{ij} x_{ij}^\tau + \alpha \left(\sum_{r\in\mathcal{R}} c_r^{\text{tr}} m_r^{\text{tr}} + \sum_{s\in\mathcal{S}} c_s^{\text{sh}} m_s^{\text{sh}} \right) \tag{2.1}$$

$$\text{s. t.} \sum_{j\in\mathcal{V}} x_{ij}^\tau = \gamma_i^\tau, \qquad \forall i \in \mathcal{V}_C, \tau \in \mathcal{T}, \quad [\text{I}] \tag{2.2}$$

$$\sum_{\tau\in\mathcal{T}} \gamma_i^\tau = 1, \qquad \forall i \in \mathcal{V}_C, \quad [\text{I}] \tag{2.3}$$

$$\sum_{i\in\mathcal{V}, i\neq h} x_{ih}^\tau - \sum_{j\in\mathcal{V}, j\neq h} x_{hj}^\tau = 0, \qquad \forall h \in \mathcal{V}_C, \tau \in \mathcal{T}, \quad [\text{I}] \tag{2.4}$$

$$\sum_{j\in\mathcal{V}} x_{oj}^\tau = 1, \qquad \forall \tau \in \mathcal{T}, \quad [\text{I}] \tag{2.5}$$

$$\sum_{i\in\mathcal{V}} x_{id}^\tau = 1, \qquad \forall \tau \in \mathcal{T}, \quad [\text{I}] \tag{2.6}$$

$$\sum_{i\in\mathcal{V}_C} q_i \gamma_i^\tau \leq Q, \qquad \forall \tau \in \mathcal{T}, \quad [\text{I}] \tag{2.7}$$

$$a_i \gamma_i^\tau \leq T_i^\tau \leq b_i \gamma_i^\tau, \qquad \forall i \in \mathcal{V}_C, \tau \in \mathcal{T} \quad [\text{I, II}] \tag{2.8}$$

$$T_i^\tau + T_i^{\text{serv}} + t_{ij} + T^{\text{break}} \beta_{ij}^\tau - M\left(1 - x_{ij}^\tau\right) \leq T_j^\tau, \quad \forall e_{ij} \in \mathcal{A}, \tau \in \mathcal{T}, \quad [\text{I, II}] \tag{2.9}$$

$$0 \leq T^{\text{driv}}{}_i^\tau \leq T_{\max}^{\text{driv}} \gamma_i^\tau, \qquad \forall i \in \mathcal{V}_C, \tau \in \mathcal{T} \quad [\text{I}^*, \text{II}] \tag{2.10}$$

$$T^{\text{driv}}{}_i^\tau + t_{ij} - T_{\max}^{\text{driv}} \beta_{ij}^\tau - M\left(1 - x_{ij}^\tau\right) \leq T^{\text{driv}}{}_j^\tau, \quad \forall e_{ij} \in \mathcal{A}, \tau \in \mathcal{T}, \quad [\text{I}^*, \text{II}] \tag{2.11}$$

$$T^{\text{driv}}{}_i^\tau + t_{ij} - T_{\max}^{\text{driv}} \beta_{ij}^\tau + M\left(1 - x_{ij}^\tau\right) \geq T^{\text{driv}}{}_j^\tau, \quad \forall e_{ij} \in \mathcal{A}, \tau \in \mathcal{T}, \quad [\text{I}^*, \text{II}] \tag{2.12}$$

$$S_\tau = T_o^\tau, \; C_\tau = T_d^\tau, \qquad \forall \tau \in \mathcal{T}, \quad [\text{I, II}] \tag{2.13}$$

$$S_\tau^{\text{driv}} = T^{\text{driv}}{}_o^\tau, \; C_\tau^{\text{driv}} = T^{\text{driv}}{}_d^\tau, \qquad \forall \tau \in \mathcal{T}, \quad [\text{I}^*, \text{II}] \tag{2.14}$$

$$\sum_{i\in\mathcal{V}_C} x_{oi}^\tau = \sum_{r\in\mathcal{R}} \rho_\tau^r, \qquad \forall \tau \in \mathcal{T}, \quad [\text{I}] \tag{2.15}$$

$$\sum_{l\in\mathcal{T}\cup\{d\}} y_{\tau l}^r = \rho_\tau^r, \qquad \forall \tau \in \mathcal{T}, r \in \mathcal{R}, \quad [\text{I}] \tag{2.16}$$

$$\sum_{k\in\mathcal{T}\cup\{o,d\},k\neq\tau} y^r_{k\tau} - \sum_{l\in\mathcal{T}\cup\{o,d\},l\neq\tau} y^r_{\tau l} = 0, \quad \forall\tau\in\mathcal{T},\, r\in\mathcal{R}, \quad [\mathrm{I}] \quad (2.17)$$

$$\sum_{\tau\in\mathcal{T}\cup\{d\}} y^r_{o\tau} = 1, \quad \forall\tau\in\mathcal{T},\, r\in\mathcal{R}, \quad [\mathrm{I}] \quad (2.18)$$

$$\sum_{\tau\in\mathcal{T}\cup\{o\}} y^r_{\tau d} = 1, \quad \forall\tau\in\mathcal{T},\, r\in\mathcal{R}, \quad [\mathrm{I}] \quad (2.19)$$

$$C_k + \Delta^r_{kl} - M\left(1 - y^r_{kl}\right) \leq S_l, \quad \forall k,l\in\mathcal{T},\, k\neq l,\, r\in\mathcal{R}, \quad [\mathrm{I,II}] \quad (2.20)$$

$$a^{\mathrm{tr}}_r \leq S_\tau + M\left(1-\rho^r_\tau\right), \quad \forall\tau\in\mathcal{T},\, r\in\mathcal{R} \quad [\mathrm{I,II}] \quad (2.21)$$

$$C_\tau - M\left(1-\rho^r_\tau\right) \leq b^{\mathrm{tr}}_r, \quad \forall\tau\in\mathcal{T},\, r\in\mathcal{R} \quad [\mathrm{I,II}] \quad (2.22)$$

$$\sum_{\tau\in\mathcal{T}} y^r_{o\tau} = m^{\mathrm{tr}}_r, \quad \forall\tau\in\mathcal{T},\, r\in\mathcal{R} \quad [\mathrm{I}] \quad (2.23)$$

$$\sum_{i\in\mathcal{V}_C} x^\tau_{oi} = \sum_{s\in\mathcal{S}} \sigma^s_\tau, \quad \forall\tau\in\mathcal{T} \quad [\mathrm{II}] \quad (2.24)$$

$$\sum_{l\in\mathcal{T}\cup\{d\}} z^s_{\tau l} = \sigma^s_\tau, \quad \forall\tau\in\mathcal{T},\, s\in\mathcal{S}, \quad [\mathrm{II}] \quad (2.25)$$

$$\sum_{k\in\mathcal{T}\cup\{o,d\},k\neq\tau} z^s_{k\tau} - \sum_{l\in\mathcal{T}\cup\{o,d\},l\neq\tau} z^s_{\tau l} = 0, \quad \forall\tau\in\mathcal{T},\, s\in\mathcal{S}, \quad [\mathrm{II}] \quad (2.26)$$

$$\sum_{\tau\in\mathcal{T}\cup\{d\}} z^s_{o\tau} = 1, \quad \forall\tau\in\mathcal{T},\, s\in\mathcal{S}, \quad [\mathrm{II}] \quad (2.27)$$

$$\sum_{\tau\in\mathcal{T}\cup\{o\}} z^s_{\tau d} = 1, \quad \forall\tau\in\mathcal{T},\, s\in\mathcal{S}, \quad [\mathrm{II}] \quad (2.28)$$

$$C_k - M\left(1-z^s_{kl}\right) \leq S_l, \quad \forall k,l\in\mathcal{T},\, k\neq l,\, s\in\mathcal{S}, \quad [\mathrm{II}] \quad (2.29)$$

$$C^{\mathrm{driv}}_k - M\left(1-z^s_{kl}\right) \leq S^{\mathrm{driv}}_l, \quad \forall k,l\in\mathcal{T},\, k\neq l,\, s\in\mathcal{S}, \quad [\mathrm{II}] \quad (2.30)$$

$$C^{\mathrm{driv}}_k + M\left(1-z^s_{kl}\right) \geq S^{\mathrm{driv}}_l, \quad \forall k,l\in\mathcal{T},\, k\neq l,\, s\in\mathcal{S}, \quad [\mathrm{II}] \quad (2.31)$$

$$S^{\mathrm{driv}}_\tau - M\left(1-z^s_{o\tau}\right) \leq 0, \quad \forall\tau\in\mathcal{T},\, s\in\mathcal{S}, \quad [\mathrm{II}] \quad (2.32)$$

$$a^{\mathrm{sh}}_s \leq S_\tau + M\left(1-\sigma^s_\tau\right), \quad \forall\tau\in\mathcal{T},\, s\in\mathcal{S}, \quad [\mathrm{II}] \quad (2.33)$$

$$C_\tau - M\left(1-\sigma^s_\tau\right) \leq b^{\mathrm{sh}}_s, \quad \forall\tau\in\mathcal{T},\, s\in\mathcal{S}, \quad [\mathrm{II}] \quad (2.34)$$

$$\sum_{\tau\in\mathcal{T}} z^s_{o\tau} = m^{\mathrm{sh}}_s, \quad \forall\tau\in\mathcal{T},\, s\in\mathcal{S}, \quad [\mathrm{II}] \quad (2.35)$$

$$x^\tau_{ij} \in \{0,1\}, \quad \forall e_{ij}\in\mathcal{A},\tau\in\mathcal{T}, \quad [\mathrm{I}] \quad (2.36)$$

$$\gamma^\tau_i \in \{0,1\}, \quad \forall i\in\mathcal{V}_C,\tau\in\mathcal{T}, \quad [\mathrm{I}] \quad (2.37)$$

$$\beta^\tau_{ij} \in \{0,1,2,\ldots,\beta_{\max}\}, \quad \forall e_{ij}\in\mathcal{A},\tau\in\mathcal{T}, \quad [\mathrm{I}^*,\mathrm{II}] \quad (2.38)$$

$$0 \leq T^\tau_i \leq T, \quad \forall i\in\mathcal{V},\tau\in\mathcal{T}, \quad [\mathrm{I,II}] \quad (2.39)$$

$$0 \leq {T^{\mathrm{driv}}}^\tau_i \leq T^{\mathrm{driv}}_{\max}, \quad \forall i\in\mathcal{V},\tau\in\mathcal{T}, \quad [\mathrm{I}^*,\mathrm{II}] \quad (2.40)$$

$$0 \leq S_\tau \leq T, \quad \forall\tau\in\mathcal{T}, \quad [\mathrm{I,II}] \quad (2.41)$$

$$0 \leq C_\tau \leq T, \quad \forall\tau\in\mathcal{T}, \quad [\mathrm{I,II}] \quad (2.42)$$

$$0 \leq S^{\mathrm{driv}}_\tau \leq T^{\mathrm{driv}}_{\max}, \quad \forall\tau\in\mathcal{T}, \quad [\mathrm{I}^*,\mathrm{II}] \quad (2.43)$$

$$0 \leq C^{\mathrm{driv}}_\tau \leq T^{\mathrm{driv}}_{\max}, \quad \forall\tau\in\mathcal{T}, \quad [\mathrm{I}^*,\mathrm{II}] \quad (2.44)$$

$$y^\tau_{kl} \in \{0,1\}, \quad \forall k,l\in\mathcal{T}\cup\{o,d\},\, k\neq l,\, r\in\mathcal{R}, \quad [\mathrm{I}] \quad (2.45)$$

$$\rho^r_\tau \in \{0,1\}, \quad \forall\tau\in\mathcal{T},r\in\mathcal{R}, \quad [\mathrm{I}] \quad (2.46)$$

$$m^{\mathrm{tr}}_r \in \{0,1\}, \quad \forall r\in\mathcal{R}, \quad [\mathrm{I}] \quad (2.47)$$

$$z^\tau_{kl} \in \{0,1\}, \quad \forall k,l\in\mathcal{T}\cup\{o,d\},\, k\neq l,\, s\in\mathcal{S}, \quad [\mathrm{II}] \quad (2.48)$$

$$\sigma^s_\tau \in \{0,1\}, \quad \forall\tau\in\mathcal{T},s\in\mathcal{S}, \quad [\mathrm{II}] \quad (2.49)$$

$$m^{\mathrm{sh}}_s \in \{0,1\}, \quad \forall s\in\mathcal{S}. \quad [\mathrm{II}] \quad (2.50)$$

In the objective (2.1), α is chosen high enough to primarily minimize the number of used routes and shifts and secondary minimize the total travel cost. Constraints (2.2) and (2.3) state that

every customer needs to be visited exactly once. Constraints (2.4) represent the trip flow conservation. Constraints (2.5) state that each trip starts at depot vertex o and constraints (2.6) state that each trip ends at depot vertex d. Maximum trailer capacity for each trip is restricted by constraints (2.7), while customer start-of-service times are restricted by time-windows in constraints (2.8). Constraint (2.9) state the driving and breaking time between two consecutive start-of-service times, while constraints (2.10)–(2.12) concern the break planning by our simple breaking rule. Notice both constraints (2.11) ('$\leq$') and (2.12) ('$\geq$') are needed to update the accumulated driving time after break *precisely*, meaning: ${T^{\text{driv}}}_i^\tau + t_{ij} - T_{\max}^{\text{driv}} \beta_{ij}^\tau = {T^{\text{driv}}}_j^\tau$ only if $x_{ij}^\tau = 1$. This is opposed to the start-of-service times in constraints (2.9), in which waiting time is allowed (thus only an equation with '$\leq$' is needed). Trip start/completion time and accumulated driving time at trip start/completion variables for each trip are linked with constraints (2.13) and (2.14) respectively. Constraints (2.15) and (2.16) state that every non-empty trip (visiting at least one customer) needs to be assigned to exactly one trailer route. Constraints (2.17) represent the trailer route flow conservation. Constraints (2.18) state that each trailer route starts with dummy trip o and constraints (2.19) state that each trailer route ends with dummy trip d. Constraints (2.20) concern the loading time between two consecutive trips in a trailer route. Constraints (2.21) and (2.22) model the availability of a trailer. Constraints (2.23) link the non-empty trailer route binaries. Constraints (2.24) and (2.25) state that every non-empty trip also needs to be assigned to exactly one truck+driver resource shift. Constraints (2.26) represent the resource shift flow conservation. Constraints (2.27) state that each resource shift starts with dummy trip o and constraints (2.28) state that each route ends with dummy trip d. Constraints (2.29) make sure each trip in a shift start after its direct predecessor has completed. Constraints (2.30) and (2.31) make sure the accumulated driving time is precisely passed from each trip to the next trip of a resource shift, and constraints (2.32) state that each first trip of a resource shift starts with zero accumulated driving time after break. Constraints (2.33) and (2.34) model the availability of a resource shift. Constraints (2.35) link the non-empty resource shift binaries. Finally, variable domains are stated by the remaining constraints (2.36)–(2.50).

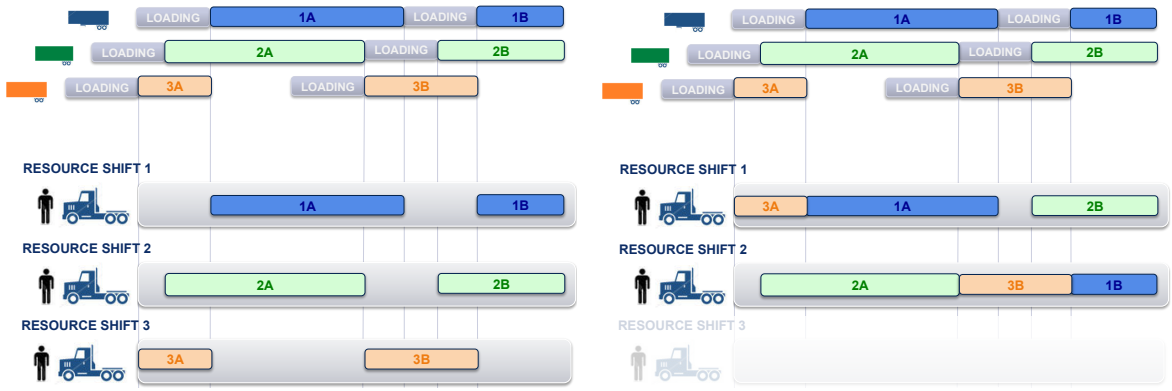
Although the above formulation is probably not as compact as possible, still the problem requires a very large number of constraints and variables (probably both in the order of $\mathcal{O}\left(|\mathcal{V}|^2 |\mathcal{T}|\right) \sim \mathcal{O}\left(|\mathcal{V}|^3\right) \sim \mathcal{O}\left(n^3\right)$). Also, a very large number of constraints are big-M type constraints, which makes the formulation's LP relaxation weak. For real world size problem instances, this formulation is probably too large and weak to be solved directly by an commercial MILP solver, such as Gurobi [39]. Therefore, we investigate a two-stage decomposition of this problem.

2.4 Relevance and potential gains

By considering non-fixed trailer and truck+driver assignment, opposed to more 'classical' fixed-assignment, our problem complicates tremendously by requiring the synchronization between the same trips in its trailer route and its truck+driver shift. However, this problem is very relevant to real-world problems logistic companies face. Furthermore, because trailers and truck+driver shifts have different characteristics, there is potential for gaining.

As stated in the above problem description, important differences between trailers and truck+driver shifts are:

- **Different fixed costs**
  In general, the fixed costs of using a trailer a single day is much less than the fixed cost of a truck+driver shift, the latter consisting mostly on driver salaries. It is thus beneficial to reduce the number of truck+driver shifts, while maybe slightly more trailers are needed.



(a) Solution with fixed Trailer and Truck+Driver Assignment

(b) Solution of our problem

Figure 2.1: Example of potential gains of separating trailers and truck+driver combinations to allow drivers to switch trailers between trips.

- **Loading time at depot**
  In our problem, we assume that different personnel is available at the depot for loading the trailer. Therefore, drivers do not need to participate and can immediately switch to an already loaded trailer. With fixed-assignment, this is not possible and drivers need to wait during the depot loading.

- **Break scheduling**
  Since the drivers need to take a break by law, but such driving rules do not hold for trailers, the drivers switching trailers may be beneficial for break planning. For instance, a driver may not be able to continue the next trip (route section) in a trailer route because no time (slack) is present to plan his required forced break. Switching to another trailer may be necessary.

- **Limited Shift durations**
  In some real-world cases, trailers are available the complete planning horizon (day), while truck+driver shifts are available for a much shorter duration in the planning horizon. For instance, there could be early morning shifts as well as late afternoon shifts. Since trailers are available the complete planning horizon, it could not be possible to fix the assignment of the whole trailer route to a single morning or afternoon shift. By allowing truck+drivers to execute only trips (route sections) solves this problem and increases flexibility of assignment to different shift kinds.

There are much more relevant benefits in real-world applications, but the above list states the most important benefits in our problem.

Figure 2.1 shows an example of switching benefits with depot loading time of non-fixed trailer and truck+driver assignment compared to the 'classical' fixed assignment. Figure 2.1(a) shows a solution with fixed assignment, while Figure 2.1(b) shows a solution of the same problem but allowing switching. Both show the trailer schedule (above) and the truck+driver resource shift schedule. Trips (route sections) are represented by notation starting with the trailer route number, followed by a letter which indicating its order in the trailer route. For instance, '3B' corresponds to the second trip (section) of trailer route number 3. Trips of the same trailer route are coloured similarly. We will use this notation often in this thesis. Also shown is the trailer loading times. The benefit of switching is clearly visible: Figure 2.1(a) needs three truck+driver resource shifts, while Figure 2.1(b) only needs two while doing the same work. The utilization of

the resources in Figure 2.1(b) is much higher, because unnecessary waiting during depot loading times is removed.

2.5 Two-stage decomposition

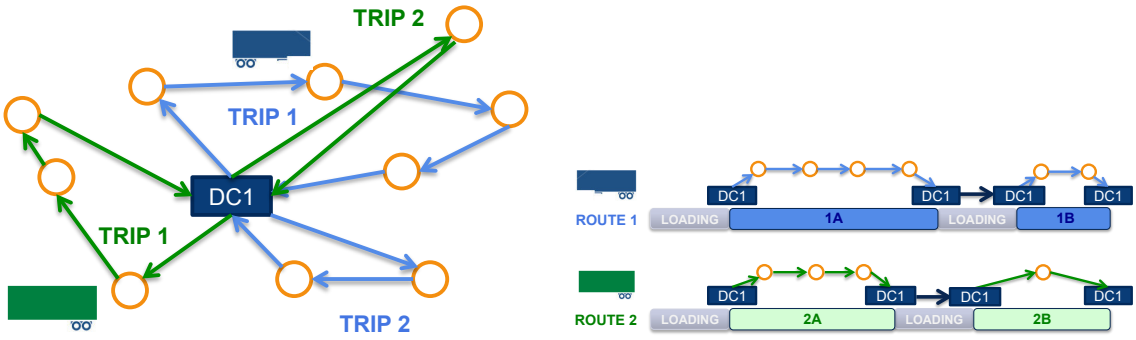
As mentioned before, the SVCRSP formulation is too large for real-world problem size instances. We propose a decomposition of the problem in two stages, inspired by Drexl et al. [24]. *Stage One* concerns only the generation of routing/scheduling of trailers, while *Stage Two* uses the generated trailer routes and assigns the generated trips, now called (route) sections, to truck+driver resources. The benefit of such decomposition is that the first Stage One problem is a general *Multi-Trip Vehicle Routing Problem with Time-Windows* (MT-VRPTW) [41], or sometimes also called *Vehicle Routing Problem with Time Windows and multiple use of vehicles* [2, 3, 51]. There is much literature available on solving this problem and also most commercial VRP solving software, like those developed by ORTEC, are capable of solving this Stage One problem without (a lot of) adapting. The more difficult Stage Two problem of truck+driver resource assignment which includes synchronization is restricted by the Stage One solution of trips and trailer routes (but not times). However, this restricting by the Stage One solution in the Stage Two problem can potentially be disadvantageous. If trips planned in Stage One turn out to be disadvantageous in Stage Two, this could increase the total number of needed resource shifts. A major contributor to this is the necessary break planning in Stage Two. Trailers are not required to follow the driving rules and break position depend only on the truck+driver shifts. But negating breaks in the Stage One trailer planning can cause trailer routes to be 'packed' with customers so no time slack is available for planning a break in Stage Two. Some kind of predictive break scheduling strategy may be necessary to allow some slack in the trailer route for break planning in Stage Two. We will investigate different Stage One break strategies, making the Stage One problem a MT-VRPTW with Breaks.

Apart from mathematical benefits of decomposing the full problem this way, it turns out such decomposition reflects how most logistic companies (like to) do their planning. Since the necessary goods must be available at the DC, the trailer routes usually must be known quite far in advance to allow for transportation/manufacturing. However, truck+driver resource assignment does not have to be done so far ahead, which is beneficial since usually driver shifts are not certain until only a day before the planning. Drivers can get sick, must take additional rests or just like to swap shift (times) with another driver. Definitive truck+driver resource assignment can thus only be done close to the planning date. This two-stage decomposition allows doing these different planning stages on different times. Professional ORTEC Routing and Dispatch software also uses these two separate planning stages.

2.5.1 Stage One: Vehicle Routing Problem (MT-VRPTW-Breaks)

The trailer routing problem is the first stage problem corresponding to a MT-VRPTW(-Breaks). In this stage, trailer routes are filled with multiple trips each consisting of a sequence of customers starting and ending at the depot, while respecting capacity and time-window constraints.

In the full SVCRSP problem formulation in Section 2.3, the constraints corresponding to the Stage One problem are marked by Roman number I between the brackets on the right. Constraints (2.2)–(2.9), when removing break related terms (variables β_{ij}^{τ}), correspond to the general VRPTW [16] by using multicommodity network flow formulation with capacity and time window restrictions. Constraints (2.13) and (2.15)–(2.22) model the 'flow' of trips in trailer routes, also by a multicommodity network flow formulation, now with time window and loading time



(a) Route Map (b) Route Schedule

Figure 2.2: Example of a Stage One solution

restrictions. In objective, equation (2.1), the sum over m_s^{sh} is removed and α is set high enough to minimize primary the number of trailers routes and secondary the distance related arc-costs. Note that more compact formulations for the MT-VRPTW exist, see for instance Azi et al. [2] or Hernandez et al [41].

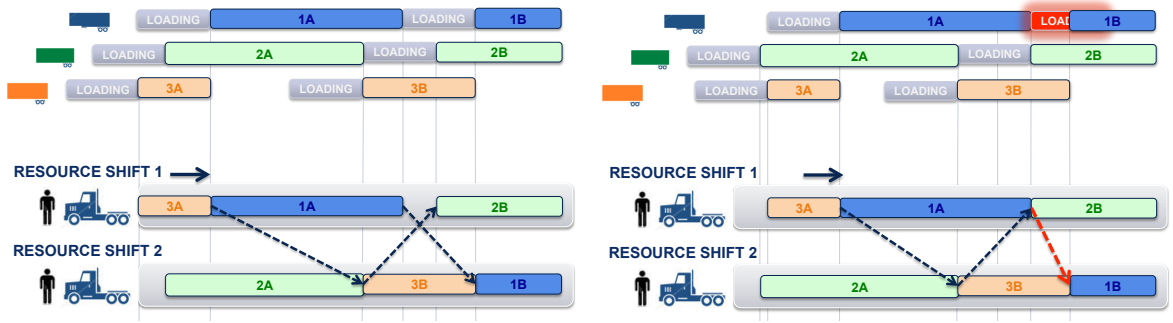
As mentioned, the precise positions of the breaks in the schedule cannot be determined yet, since their position depends on the truck+driver shift flow (opposed to the trailer route flow). However, it turns out to be beneficial to include some predictive break scheduling to generate slack for the exact break scheduling in Stage Two. In this case, constraints marked with Roman number I* are needed as well. Our used predictive break strategies will be described in Section 4.1.1.

To illustrate Stage One, a schematic overview of a Stage One solution is given in Figure 2.2. Customers are represented by orange dots and the depot (distribution center) is represented by the 'DC1' square. The blue (green) arrows represent the trips of the blue (green) trailer. Figure 2.2(a) shows a geographic representation of the trailer routes and the corresponding trailer route schedule is shown in Figure 2.2(b). Trips are denoted with route number and position letter ('2A': route 2 (green) trip 1) and between trips loading time is shown. Note however that exact start-of-service times are not yet fixed: the customer time-windows usually allow trips to move in time.

2.5.2 Stage Two: Resource Assignment Problem

The (truck+driver) resource assignment problem (RAP) [36] is the second stage problem of the full SVCRSP with Breaks. The Stage One solution of trailer routes and trips are now fixed (but not in time), and need to be assigned to truck+driver resource shifts. In this stage, we often refer to these fixed trips as route *sections*, representing the (smallest) 'section' or *job* of a planned trailer route which need to be assigned to a truck+driver resource shift.

In the full SVCRSP formulation in Section 2.3, variables corresponding to route sections (trips) (x_{ij}^{τ}, γ_i^{τ}) and trailer routes (y_{kl}^{r}, ρ_{τ}^{r}, m_r^{tr}) are fixed by the Stage One solution. The Stage Two problem corresponds to assigning trips to resource shifts (z_{kl}^{s}, σ_{τ}^{s}, m_s^{sh}) and finding all feasible start-of-service times (T_i^{τ}), trip start/completion times (S_{τ}, C_{τ}) and correct break planning (β_{ij}^{τ}, $T^{\text{driv}}{}_i^{\tau}$, S_{τ}^{driv}, C_{τ}^{driv}). In the full SVCRSP formulation, the Stage Two RAP problem corresponds to the non-fixed constraint equations, which are marked by Roman number II between the brackets on the right in Section 2.3. In the objective, equation (2.1), only the sum over non-empty resource shifts variables m_s^{sh} is not fixed. Other parts in this equation are fixed by the Stage One solution.



(a) Feasible solution (b) Infeasible solution

Figure 2.3: Example of synchronization dependencies between sections in Stage Two Resource Assignment.

Before starting Stage Two, time-windows of all customer are *reduced* to represent earliest/latest time service can start ensuring feasibility of the whole trailer route (making them narrower). This gives rise to earliest/latest start time-windows $\left[\underline{S}_j, \overline{S}_j\right]$ and earliest/latest completion time-windows $\left[\underline{C}_j, \overline{C}_j\right]$ for each route section (trip) j, again ensuring feasibility of the whole route. This reduces the complexity of the Stage Two problem, without loosing any feasible solution.

2.5.3 Synchronization of dependent route sections

As stated earlier, the drivers swapping trailers between route sections (trips) can potentially improve the solution. However, this causes sections in Stage Two problem to become *interdependent*. Constraints (2.20), ensuring the precedence relation (loading time) between two consecutive sections in a trailer route, causes *inter-shift* dependencies between these sections. This can make resource shifts dependent, which makes the Stage Two problem difficult. Most solution methods rely on the assumption of independent resource shifts (or routes) to quickly asses feasibility, but this does not hold for our Stage Two problem when we allow drivers to swap trailers.

Figure 2.3 shows an example of the difficulty arising in planning of resource shifts with dependent sections. Figure 2.3(a) shows a feasible Stage Two schedule of route sections in trailer routes (above) and in truck+driver resource shifts (below). Suppose we want to investigate if we can delay section '3A' in resource shift 1 to allow room for planning new section before it. Figure 2.3(b) shows what happens if resource shift are treated as being independent: the sections '3A' and '1A' are both delayed and moved tightly against section '2B', so resource shift 1 remains feasible. Even the trailer route 3 of the moved section '3A' remains feasible. However, there is a problem in trailer route 1 (blue): during the loading of the blue trailer between sections '1A' and '1B', resource shift 2 starts section '1B' (truck+driver 2 picks up the trailer) before the trailer loading is completed. The moving of section '3A' this way thus is infeasible, because it causes an infeasibility in another trailer route and resource shift.

The example shows the difficulty of feasibility calculation: multiple resource shifts and trailer routes need to be considered and calculated through to evaluate the movement of a single section. In this thesis, we will focus on different Stage Two solution methods able to work with these dependencies to allow the generation of high quality resource shift schedules in which drivers can swap trailers between sections.

Chapter 3

Literature Review

Vehicle routing and crew scheduling problems are very popular in literature, being used to model all kinds of rich, practical problems arising in the logistical industry. Although there are quite some studies considering problems similar to our Stage One Route planning or our Stage Two Resource Assignment problem, limited studies considers problems similar to our full Simultaneous Vehicle and Crew Routing and Scheduling (SVCRSP) problem. First, we discuss some literature related to the full problem. Then, we discuss some of the vast amount of literature related to either the Stage One and Stage Two problems. We also discuss some literature related to our proposed solution methods. Finally, we mention some literature related to the inclusion of driver's legislation in vehicle routing problems, which is needed in both our Stage One and Stage Two problems.

3.1 SVCRSP

In vehicle routing literature, until recently it was common to consider vehicles each consisting of a fixed assignment of truck, trailer and driver. In that case, a general vehicle routing problem with time windows (VRPTW) solution method can be used to solve both the vehicle routing and crew scheduling simultaneously. See the chapter of Cordeau et al. (2001) [16] in the book *The Vehicle Routing Problem* for a description of the VRPTW. See Cordeau et al (2007) [17], a chapter in the Handbook of Operations Research (HoOR), for a survey on many Vehicle Routing problems and various solution algorithms, see Laporte (2009) [52] for a historic overview of fifty years of vehicle routing literature. Many additions, inspired mostly by practice, have been made to the classic VRPTW, see for instance a very recent paper of Lahyani et al. (2015) [51] on a taxonomy of rich multi-attribute vehicle routing problems and Vidal et al. (2013) [71] providing an overview of popular solution algorithms for rich multi-attribute vehicle routing problems. Although very advanced method are used to solve very rich multi-attribute vehicle routing problems, vehicles are still considered to consist of fixed (truck+)driver and trailer pairs. In our problem, we like to investigate non-fixed (truck+)driver and trailer pairs to allow more flexible schedules inspired by practice.

Transportation crew scheduling literature is mainly focussed on solving problems arising in airline (survey Ball et al (2007) [4] in HoOR), public transit (survey Desaulniers et al. (2007) [19] in HoOR) and railway transportation (survey Caprara et al. (2007) [13] in HoOR), where crew members need to be assigned multiple blocks of work , tasks, duties or trips. In these areas, there are many solutions methods posed to simultaneously optimize routing and crew scheduling, for instance Freling et al (2003) [30] and Huisman et al. (2005) [42] using Column Generation (CG) to solve an urban mass transit routing and crew scheduling problem respectively with

a single and multiple depots, showing the benefit of simultaneous optimization. However, in many problems in these areas, there usually is a timetable with fixed start/completion times of tasks/duties/trips already given. In problems arising in logistical transportation, such as our problem of servicing distribution customers from a depot, no fixed start/completion times of tasks/duties/trips are given and no timetable exists, only customers have time-windows in which servicing must start.

In our SVCRSP problem, two resources are needed to service a customer: a trailer and a truck+driver combination, while in most vehicle routing problems only a single (vehicle) resource is needed. Also start/servicing/completion times are not fixed by for instance a timetable. In a recent survey of Drexl [22], the need of two or more independent resources to service a customer at a non-fixed time contributes to so-called *synchronization constraints* among these resources. Therefore, the SVCRSP can be seen as a specific instance of the more general *Vehicle Routing Problem with Multiple Synchronization Constraints* (VRPMS), which the survey [22] of 2011 provides an overview of related literature.

In a paper by Drexl et al. (2013) [24], a similar SVCRSP is defined for long-distance road transport in Europe. In this problem, pickup-and-delivery requests need to be fulfilled by both a vehicle (truck+trailer here) and driver. A multi-period planning horizon is considered, as well as European Union social legislation on driving/working times for the driver. During the planning, drivers may switch trucks at geographically dispersed relay stations and even an option to use a *shuttle van* between them. Unlike our problem, no central depot is used for switching and therefore no fixed loading time is included in this problem. Like in our proposed solution, this problem is decomposed in two stages. In his Stage One, pickup-and-delivery requests are assigned to truck routes, while also visits to relay stations are inserted. An Large Neighbourhood Search (LNS) algorithm inspired by work of Ropke and Pisinger (2006) [66] is used, which uses construction and destruction operators to iteratively build up and remove some parts of the solution to search a large neighbourhood of feasible solutions. Interestingly, Drexl et al. [24] use a strategy to already insert some breaks/rests for drivers in the truck routes, although the exact position of the breaks depends on the driver's schedule, which is not known yet. We will further investigate the use of such Stage One predictive break strategies. In Stage Two, the truck routes are split into multiple trips (acting as *super customers*) between depot and relay stations. Again a similar LNS algorithm is used to assign these trips to driver schedules and insert necessary breaks/rests. Now synchronization constraints appear to make sure the precedence relations of the truck routes are satisfied. Interestingly, Drexl at al. [24] 'solve' the difficult synchronization constraints heuristically by pre-processing the time-windows of the trips at the start of Stage Two to ensure the movement of the trips in time is independent. This narrowing of the trip time-windows allows some flexibility in trip start/completion times, but it is narrow enough that any movement in a trip satisfying the new time-windows will not affect the start/completion times of any other trip in that truck route. This effectively eliminates the use of synchronization constraints, although much flexibility is lost by the narrowing of the time-windows. We will compare the use of this 'Drexl' pre-possessing to an exact (full flexibility) method on our problem.

Meisel et al. (2014) [55] discusses a pickup-and-delivery problem in which servicing is both required by active (truck+drivers) and passive (trailers) resources, like in our problem, but switching is allowed at the customer instead of at (intermediate) depots. A Mixed Integer Linear Programming (MILP) formulation is given for the full problem but it is found to be intractable for large problem instances. Further a Large Neighbourhood Search meta-heuristic is investigated.

3.2 Stage One

By decomposing our full problem in the suggested two stages, the first stage corresponds to a Multi-Trip Vehicle Routing Problem with Time-Windows (MT-VRPTW), or Multi-Depot Multi-trip Vehicle Routing Problem with Time-Windows (MD-VRPTW) in case multiple depots are present. The earlier mentioned Vehicle Routing surveys contain many references to literature on these VRP variants, including construction heuristics in Campbell and Savelsbergh (2004) [12], exact branch-and-price (CG + branch-and-bound) based methods by Azi et al. (2010) [2] and Hernandez et al. (2014) [41] and LNS heuristics by Azi et al. (2014) [3]. Battarra et al. (2009) [7] propose an adaptive guidance approach which consists of iteratively constructing feasible trips and aggregating them to full routes. Guidance is done by examining *critical time intervals*, or bottleneck times in the route aggregation when most vehicles are simultaneously active executing trips. Iteratively new trips are generated by guiding them away from these critical time intervals, hopefully shifting the bottleneck and thereby to reduce the number of total vehicles needed. We will use similar ideas to evaluate resource bottlenecks, bottlenecks on the minimum number of drivers needed simultaneously, before solving the Stage Two problem. Also we investigate using bottleneck avoiding insertion strategies in construction heuristic for the Stage Two problem.

More on the use of Large Neighbourhood Search algorithms in vehicle routing problems can be found in Pisinger et al. (2007) [59] and the use in general can be found in a chapter of the Handbook Of Metaheuristics (2010) [60]. More on Column Generation and the related *Dantzig-Wolfe decomposition* of large (M)ILPs can be found in Lübecke and Desrosiers (2004) [54] and the book *Column Generation* (2005) [20], and the book *50 Years of Integer Programming 1958–2008* [15, 70], the latter giving a more historic introduction on the subject. A very good introduction of solving vehicle routing problems with column generation and related branch-and-price methods is given by Feillet (2010) [28].

3.3 Stage Two

In our decomposition of the full problem, Stage Two consists of crew scheduling, or also called resource assignment, of assigning drivers to trailer trips. Synchronization constraints arise as precedence constraints on the trips belonging to the same trailer (but possibly not executed by the same driver). In Local Search based solution methods, feasibility of the precedence constraints are usually directly evaluated. In Column Generation based solution methods, different ways of dealing with these precedence constraints are proposed in literature.

When precedence constraints act *intra-shift* (intra-column), meaning between actions inside a single driver's resource shift (single column), these precedence constraints can be solved completely inside the pricing subproblems. Columns in the master problem are independent and no explicit modelling of these constraints in the MP is needed. Recent work of Gschwind and Irnich (2012, Tech Report) [38] focusses on efficient pricing subproblems of the *dial-a-ride problem with temporal dependencies*. In the dial-a-ride problem, customers can call and request a ride between an origin and destination location. Although vehicles usually handle multiple customer requests at once, in order to guarantee a certain level of service the maximum *ride-time* between pickup and delivery of a customer is restricted. Branch-and-cut-and-price methods are formulated with temporal intra-route dependencies, being the customer maximum ride-times, handled purely by the pricing subproblems, which showed good results. In a more recent work, Gschwind (2014, Tech Report)[37] show for the more generalized *VRPTW and Temporal Synchronized Pickup and Delivery* that although dealing with intra-route (intra-column) dependencies severely complicate the pricing subproblems, it is still beneficial in branch-and-cut-and-price to have them in the pricing (instead of in the MP) to produce 'good' quality columns. Unfortunately, our prob-

lem deals with *inter-shift* (intra-column) precedence constraints: sections of (possibly) different driver resource shifts (but of same route) are temporal depended. As noted by Gschwind and Irnich [38], this severely complicates matters since these constraints cannot be solved inside the pricing subproblems.

Concerning also inter-route (inter-column) dependencies, a paper of Dohn et al. (2011) [21] reviews different mixed integer linear programming (MILP) formulations for the *VRPTW with Temporal Dependencies*, which is a generalization of our Stage Two problem. A full time-indexed model formulation was found to be very strong as LP in a branch-and-bound algorithm, but the number of constraints and variables explodes when large instances were considered, making the method intractable. Dantzig-Wolfe decompositions of this large ILP results in a number of different column generation methods are also investigated. Explicit modelling of synchronization constraints in the master problem (MP), one of the decompositions, is omitted by Dohn et al., since this makes the corresponding pricing problem cost objective time-dependent and therefore very difficult to solve. Also, Dohn et al. note that including the synchronization constraints in the MP very likely results in highly fractional LP solutions. So instead, Dohn et al. relax the synchronization constraints from the MP (leaving only set partition constraints), but instead the synchronization is enforced in branching and cut generation, as earlier done by Breström and Rönnqvist (2008) [11]. This makes the pricing problem much easier to solve, but the LP formulation of this 'relaxed' MP less strong. This method is used for instance by Rasmussen et al. (2012) [64] to solve a practical problem arising in Home Care when patients need to be regularly visited by multiple home carers with some temporal dependencies between those visits.

In both the theses of Groenendijk (2012) [36] and Baller (2013) [5], a different way of dealing with precedence constraints is proposed: a *Column-and-row generation* method, inspired by Muter et al. (2013) [56], to solve a Resource (Crew) Assignment problem with synchronization dependencies very similar to our Stage Two problem. Columns representing single driver schedules are generated without considering synchronization constraints, like in the pricing subproblem of the 'relaxed' master problem formulation of Dohn et al. [21]. Upon adding columns to the master problem, it is checked if there are incompatible pairs of columns, in which case an additional constraint is added to prevent the master problem from simultaneously selecting these incompatible columns. Baller [5] shows good results for this method, but as the number of columns grows, more columns need to be verified with increasing computation times.

In Van den Akker et al. (2010) [1], a column generation method with explicit modelling of the synchronization constraints in the master problem is introduced and solved for a Parallel Machine Scheduling problem with release/due times, and generalized precedence constraints. This problem is very similar to our Stage Two resource assignment problem, but does not include some of the underlying routing characteristics needed in the pricing subproblem(s). Columns are generated heuristically by Local Search and a full time-indexed formulation is used to strengthen the Objective value bounds. We will use similar explicit precedence constraint modelling in the MP of our column generation method for solving Stage Two. As mentioned (for instance) by Dohn et al. [21] and Gschwind and Irnich [38], and which we will also show in more detail, these constraints in the MP lead to the addition of time-dependent costs in the objective of the pricing subproblem(s), making solving these subproblems more difficult.

As we will show, it turns out the resulting pricing subproblem(s) resemble characteristics of both two special kinds of Shortest Path Problems, the *Shortest Path Problem with Time-Windows and Linear Node Costs* (SPPTWNC) and the *Elementary Shortest Path Problem with Resource Constraints* (ESPPRC). Both problems are discussed by Irnich and Desaulniers (2005) in the chapter Shortest Path Problems of the book Column Generation [46].

In a key paper, Ioachim et al. (1998) [44] studies the SPPTWNC problem, which arises as pricing subproblem of an airline routing and crew scheduling problem, described in Ioachim et al. (1999)

[43]. In this problem, flights need to be scheduled to crew members, with additional requirement that flight of the same type on different days are synchronized, which Ioachim et al. [43] model explicitly in the MP. By very clever use of convex piecewise linear functions, Ioachim et al. [44] solve the time-dependent costs by a Dynamic Programming algorithm on a acyclic graph.

However, our pricing subproblems each act on a graph containing negative cycles and therefore the *Elementary* condition needs to be enforced. Typically, in solving VRP, VRPTW and many of its rich variants by CG, the ESPPRC arises as 'natural' pricing subproblem [28, 29, 45, 46]. Beasley and Christofides (1989) [8] were the first to describe the ESPPRC and proposed a labelling algorithm. Feillet et al. (2004) [29] improved their labelling algorithm. We will use this newer algorithm as inspiration for our specific algorithm. Many constraints arising in (pricing subproblems of) VRP variants can be modelled as 'resources' being extended along paths in a graph, see Irnich (2008) [45] for a very extensive mathematical overview of many resource-extension functions. Drexl and Prescott-Gagnon (2010) [23] model many EU social legislation related constraints inside an ESPPRC.

Dror (2002) [25] proved that solving the ESPPRC on a graph containing negative cycles is $\mathcal{NP}$-hard. Therefore many exact solution methods become intractable when solving large problem instances and improvements are constantly sought-after by the research community. See Pugliese and Guerriero (2013) [63] for a recent survey on ESPPRC solution methods. Here, more advanced solution methods are mentioned, like the *Bi-directional* dynamic programming algorithm by Rinaldi and Salanti (2006) [65]. We did not investigated these newer ESPPRC solution methods, because it was not clear if they could be used (easily) to solve some of the complex (non-symmetric) constraints like those related to the driving rules/break planning. This also holds for the very recently formulated *Pulse* algorithm by Lozano et al. (2014) [53], although this may possibly be easier to adapt to our subproblems.

Combing the acyclic SPPTWNC and ESPPRC on a cyclic graph is difficult [21, 38, 45, 64], and thus consumes a reasonable part of this thesis. Spliet and Gabor (2015) [68] investigate the *Time-Window Assignment VRP*, in which customers have to be assigned time-windows while their demand is not yet known. The combination of SPPTWNC and ESPPRC arises as pricing subproblem in their branch-and-price method. To use the SPPTWNC algorithm of Ioachim et al. [44], they transform the cyclic graph to a acyclic graph by copying the graph a multiple times, and they relax the elementary conditions. They do include k–cycle elimination method of Irnich and Villeneuve (2006) [47] to eliminate cycles with length upto $k = 2$. Still, the solution visiting the same customer multiple times could occur, which lowers the RMP (LP) bound, but computation times are reduced significantly. Since we have a specific chain-like structure of precedence constraints, we will show that when using the full non-relaxed ESPPRC, we only need a limited number of label 'resources' to ensure elementary paths, and at the same time ensuring correct precedence of trips.

We know of only one paper actually applying the SPPTWNC and full ESPPRC combination: Tagmouti et al. (2007) [69] study a *Capacitated Arc routing problems with time-dependent service costs*. The problem, which is motivated from operations arising in winter road gritting (multiple vehicles can be used to clear roads of ice/snow), is transformed to an equivalent vertex-routing problem with time-dependent service costs. A branch-and-price algorithm is used with standard set partition MP. Interestingly, no inter or intra synchronization/temporal dependencies are present, only time-dependent service costs acting as generalized *soft time-windows*. Still, the pricing subproblem is similar to ours. Tagmouti et al. [69] use cost functions as 'label resource', which are extended along feasible paths. We will adopt this use of cost functions, but make more heavily use of mathematical observations related to linear node costs in the work of Ioachim et al. [44]. Although Tagmouti et al. allow vehicles to wait at the central depot before departing, waiting before servicing an arc is not allowed and costs are immediately incurred upon arrival

(often being a early/late arrival penalties). See Bhusiri et al. (2014) [9] for a more descriptive review on early/late arrival penalties in VRPs with Soft Time-Windows. In our problem, a driver may wait at a depot on a trailer if this is beneficial for synchronization, which is steered by the linear node costs. Waiting at a vertex before servicing and costs are incurred only when servicing starts. Another important difference is that we include (simple) driving rules for planning breaks.

As noted earlier, the exact ESPPRC solution methods can become intractable for large instances. Solving the pricing subproblems heuristically has become increasingly popular. Embedding column generation inside a meta-heuristic method, like Large Neighbourhood Search, is another kind of mathematical programming meta-heuristic *hybridization*. The book *Matheuristics* (2010) [14] provides an overview of all kinds of hybridizations of exact mathematical programming and meta-heuristic methods. Relevant applications include Prescott-Gagnon et al. (2010) [62] which study a VRPTW problem with EU social legislation and Parragh et al. (2012) [58], which study a dial-a-ride problem. Both works consider a combination of hybridizations: heuristic pricing inside CG inside LNS.

3.4 Driver's Legislation in VRPs

In real-world applications, driver's legislation is of vital importance to logistic companies in order to ensure legal and safe execution of crew schedules. Despite this importance, which also includes applications in commercial VRP/scheduling software like developed by ORTEC, there is very little research done by the VRP community on this subject. Possibly, because of extremely complicated modelling and algorithmic difficulties involved. Very important work done in this area includes Goel et al. (2010) [33] on VRPTW with EU legislation, Goel at al. (2012) [34] on Australian legislation and Goel and Kok (2012) [35] on USA legislation. As already mentioned, Drexl and Prescott-Gagnon (2013) [23] focusses on solving EU legislation inside ESPPRC, while Prescott-Gagnon (2010) [62] use a hybrid column generation large neighborhood search to solve the VRPTW with EU legislation. Kok et al. (2010) [48] provides an dynamic programming for heuristic EU break scheduling. Included is the consideration of more simplified driving rules (forced breaks only) like ours. Kok et al. (2010) [49] extended the DP with heuristic break scheduling to the case of time-dependent travel times arising from road congestion. Very interesting is the use of a piecewise linear function inside the labels which keeps track of the elapsed working time with respect to the depot departure time. In all of the above mentioned works, it is assumed that a driver is always able to break immediately at any time during his en-route travel, regardless of his location. This is opposed to Drexl et al. [24], which considers drivers to be able to break at relay stations.

For a recent survey on the matter, see the VRP taxonomy of Lahyani et al. [51], which includes a (short) survey on driver's legislation in VRPs and proposed solution methods.

Chapter 4

Solution Methods

In this chapter, we would like to give a very brief overview on our proposed and investigated solution methods for our Stage One and Stage Two problems. A schematic overview of the methods is given in Figure 4.1. As can be seen in the figure, we consider only one solution method type for Stage One (*PCI*) and three types of solution methods for Stage Two (*SPCI*, *CGSPCI* and *CGDP*). These will be discussed briefly in Section 4.1 (Stage One) and Section 4.4 (Stage Two), and more extensively the subsequent chapters. Furthermore, in this chapter, we formulate two Stage One break (predicting) strategies in Section 4.1.1, investigate a resource lower bound graph in Section 4.2, for detecting potential resource bottlenecks in a Stage One solution. Also in Section 4.3, we formulate a time-window preprocessing procedure based on work of Drexl et al. [24], to limit Stage Two computation times by limiting time-window flexibility.

4.1 Stage One: Trailer Routing

We consider only one solution method to solve the Stage One trailer routing problem:

- ***PCI*** – Parallel Cheapest Insertion (heuristic)
  This construction heuristic is based on the *Parallel Cheapest Insertion* construction heuristic of Potvin et al. (1993) [61], which is in turn based on the *Cheapest Insertion* construction heuristic of Solomon (1987) [67]. Trips in trailer routes are iteratively filled with customers having the cheapest insertion cost. This method is explained in more detail in Section 5.1.

More advanced solution method for the Stage One problem were not considered in this thesis for two reasons. First, our thesis focusses primarily on Stage Two solution methods with synchronization, for which not much literature exists. The Stage One MT-VRPTW problem is a very general problem, discussed in literature extensively (see literature Section 3.2). Furthermore, results indicate that if Stage One solutions are of very high quality, trailer routes are nicely 'packed' with customers, but the Stage Two problem now becomes very restricted. Most of the time, this results in low quality Stage Two solutions in which almost no switching can takes place and requiring a lot of resource shifts. Therefore, we decided to here investigate only the more 'simple' PCI method.

4.1.1 Breaking strategies

We did extend the PCI method with (predictive) break scheduling, to allow some slack for later Stage Two break planning. Since break position is based on resource shifts and thus only

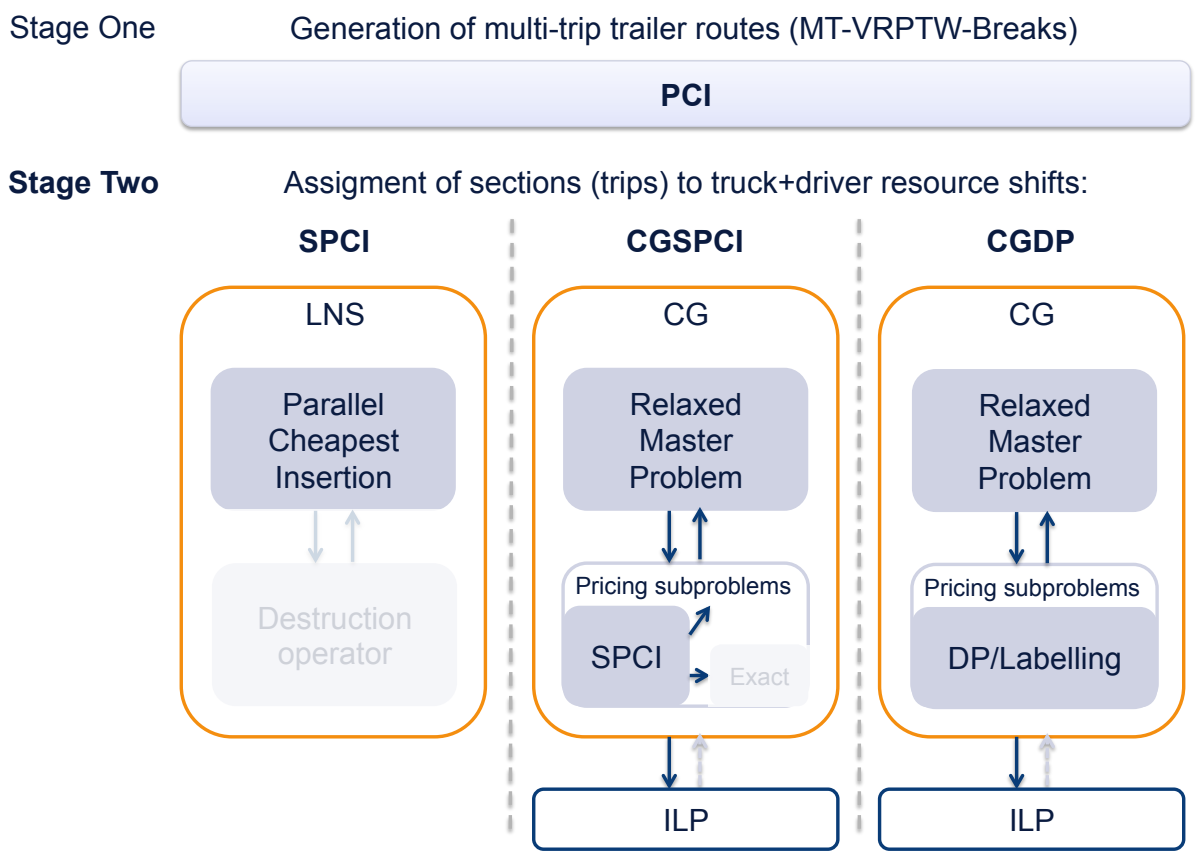


Figure 4.1: An schematic overview of the solution methods investigated in this thesis for solving the Stage One and Stage Two problems.

known in Stage Two, we consider the following two predictive *strategies* for planning these breaks already in Stage One:

- ***Fresh*** – Each trailer starts new trip fresh: with no driving time after last break.
  Using this strategy, each generated trip can in Stage Two be executed by an empty truck+driver resource shift without violating the driving rule. If trips do not contain much driving time, almost no breaks are scheduled using this strategy. This strategy is inspired on Stage One scheduling by Drexl et al. [24].
- ***NotFresh*** – Each trailer starts new trip not fresh: trailer is assumed to be driven by single driver.
  Using this strategy, each generated full trailer route can in Stage Two be executed by an empty truck+driver resource shift without violating the driving rule. If resource shift and trailer routes span the same planning horizon, breaks using this strategy tent to be planned on similar times. Usually more breaks are planned with this strategy than the *Fresh* strategy, adding more slack in trailer routes but also requiring slightly more trailers.

In the SVCRSP problem formulation in Section 2.3, break scheduling corresponds to constraint equations (2.9)–(2.12) and (2.14), which are added to the Stage One problem. The predictive strategies correspond to predicting constraint equations (2.30)–(2.32). The *Fresh* strategy predicts these constraint equation by adding only the following constraint equations in Stage One:

$$S_\tau^{\text{driv}} = 0, \ \forall \tau \in \mathcal{T}, \tag{4.1}$$

stating that each trip $\tau \in \mathcal{T}$ starts without any driving time before last break S^{driv}_{τ}. The *NotFresh* strategy predicts constraint equations (2.30)–(2.32) by the following constraint equations in Stage One:

$$C^{\text{driv}}_{k} - M(1 - y^{r}_{kl}) \leq S^{\text{driv}}_{l}, \ \forall k, l \in \mathcal{T},\ k \neq l,\ r \in \mathcal{R}, \tag{4.2}$$

$$C^{\text{driv}}_{k} + M(1 - y^{r}_{kl}) \geq S^{\text{driv}}_{l}, \ \forall k, l \in \mathcal{T},\ k \neq l,\ r \in \mathcal{R}, \tag{4.3}$$

$$S^{\text{driv}}_{\tau} - M(1 - y^{r}_{o\tau}) \leq 0, \qquad \forall \tau \in \mathcal{T},\ r \in \mathcal{R}, \tag{4.4}$$

stating that the driving times after break follow the trailer flow y^{r}_{kl} of consecutive trips $k, l \in \mathcal{T}$ in a single trailer route r, and that only the first trip $\tau \in \mathcal{T}$ of each route r (for which $y^{r}_{o\tau} = 1$) starts with zero driving time after last break: $S^{\text{driv}}_{\tau} = 0$.

4.2 Resource Lower Bound Graphs

Using a Stage One solution, we can define a *Resource Lower Bound graph* by calculation the minimum number of resources needed each time in the planning horizon. The idea is simple: at each time $t \in [0, T]$, if we know that for certain at least $\underline{m}(t)$ trailers are active executing route section (so not being loaded), this is a lower bound on the number of truck+driver shifts needed at that time, since each trailer route section needs one truck+driver shift. Let $\mathcal{J}$ be the set of route section: the non-empty trips in the Stage One solution. Calculation of this function can be done very easy, provided the earliest/latest start times $\underline{S}_j$, $\overline{S}_j$ and earliest/latest completion times $\underline{C}_j$, $\overline{C}_j$ of route section $j \in \mathcal{J}$ ensuring feasible start-of-service times of the whole route. Usually, these quantities are already deduced by the Stage One solution method, see for instance Campbell and Savelsbergh [12] for more details on using these quantities in solution methods.

Each non-empty trailer route section $j \in \mathcal{J}$ is being executed for *certain* during the time interval $\left[\overline{S}_j, \underline{C}_j\right]$. We can use this to 'count' the number of these certain intervals at each time using the following procedure.

Let vector $\delta = (\underbrace{1, 1, \ldots, 1}_{|\mathcal{J}|}, 0, \underbrace{-1, -1, \ldots, -1}_{|\mathcal{J}|}, 0)$ and vector $T^{\text{cer}} = \left(\overline{S}_1, \overline{S}_2, \ldots, \overline{S}_{|\mathcal{J}|}, 0, \underline{C}_1, \underline{C}_2, \ldots, \underline{C}_{|\mathcal{J}|}, T\right)$, with the planning horizon given by $[0, T]$. Let $\tilde{T}^{\text{cer}}$ be the incrementally sorted vector of T^{cer} and let vector I be the sorted index set: $\tilde{T}^{\text{cer}}_{i} = T^{\text{cer}}_{I_i}$ for $i = 1, \ldots, 2 \cdot |\mathcal{J}| + 2$. Let ties be split by favouring higher position in T^{cer}: completion times $\underline{C}_i$ are always considered 'earlier' than start times $\underline{S}_i$ when tied. By construction, $\tilde{T}^{\text{cer}}_{1} = 0$ and $\tilde{T}^{\text{cer}}_{2\cdot|\mathcal{J}|+2} = T$. The extreme points of the resource lower bound graph $\underline{m}(t)$ are now given by:

$$\underline{m}(\tilde{T}^{\text{cer}}_{i}) = \sum_{i'=1,\ldots,i} \delta_{I_{i'}}, \ \forall i = 1, \ldots, 2 \cdot |\mathcal{J}| + 2. \tag{4.5}$$

Now the complete graph is given by: $\underline{m}(t) = \underline{m}(\tilde{T}^{\text{cer}}_{i})$ for each $t \in [\tilde{T}^{\text{cer}}_{i}, \tilde{T}^{\text{cer}}_{i+1})$, for each $i = 1, \ldots, 2 \cdot |\mathcal{J}| + 1$.

Let T^{crit} be the critical bottleneck time, defined as the earliest time where the resource lower bound graph is maximal: $\underline{m}(T^{\text{crit}}) = \max_{i=1,\cdots,2\cdot|\mathcal{J}|+2} \underline{m}(\tilde{T}^{\text{cer}}_{i})$. We can conclude that **at least** $\underline{m}(T^{\text{crit}})$ resource shifts are needed to cover all trailer route sections, which is a lower bound on the actual number of resource shifts needed. Also on other times in the planning horizon, we now know how much resource shifts are minimally needed to cover all trailer route sections, which is useful when considering shifts of different kinds (morning shifts/afternoon shifts) which do not span the whole planning horizon.

However, we must be careful with the breaks in the Stage One solution when calculating the earliest/latest start/completion times $\underline{S}_j$, $\overline{S}_j$, $\underline{C}_j$, $\overline{C}_j$. Since breaks in Stage One are only pre-

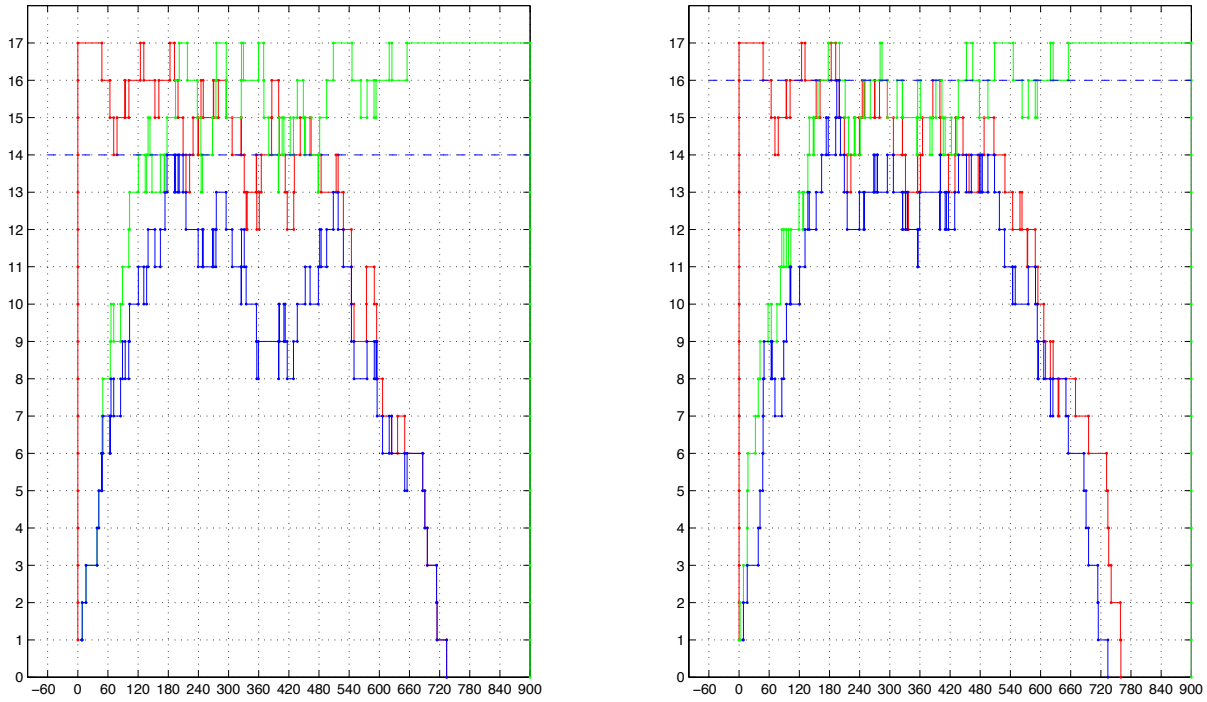


Figure 4.2: Resource lower bound graphs in blue for Stage One solution of Solomon instance 'C207' using *NotFresh* strategy. Red/Green lines show the resources needed when trips are planned as early/late as possible. Left: exact RLB using no breaks; Right: non-exact RLB using Stage One predicted breaks.

dicted and could be moved in Stage Two, the actual resource lower bound graph can only be determined with earliest/latest start/completion times $\underline{S}_j$, $\overline{S}_j$, $\underline{C}_j$, $\overline{C}_j$ of section $j \in \mathcal{J}$ without considering any Stage One solution break. This weakens the lower bound, since now the certain intervals $\left[\overline{S}_j, \underline{C}_j\right]$ are smaller (slack is larger) than those including the breaks in the Stage One solution. Therefore, we consider two variants of the resource lower bound graph: the exact *resource lower bound* (RLB) graph and the non-exact *resource lower bound* graph *with breaks* (RLB with breaks). The latter includes current breaks of the Stage One solution, which is only an exact lower bound if breaks remain at their current position in the schedule (on their current route arc). But since it includes breaks, it turn out that this graph provides a better estimate on the actual number of resource shifts needed.

Figure 4.2 shows an example of the resource lower bound graphs for a Stage One solution of a Solomon instance 'C207', which was planned using PCI with *NotFresh* break strategy and produced a total of 17 trailer routes. Left plot shows the exact RLB graph in blue, which shows that at least 14 resource shifts are needed to cover the planning, with critical bottleneck time T^{crit} is around time 180. Right plot shows the non-exact RLB with breaks of Stage One, which shows that at least 16 resource shifts are needed to cover the planning with breaks at current Stage One positions. Now critical bottleneck time T^{crit} is around time 200. For this particular instance and this Stage One solution, solving the Stage Two problem to optimality results in a schedule of 16 resource shifts. The bottleneck in the final schedule also is around time 200, although some breaks were moved.

We will use the RLB graphs in multiple ways: we will use it in some insertion strategies in our Stage Two construction heuristic to actively avoid planning sections on the critical time, similarly to the guidance of Barratta et al. [7]. Also, we use a *seeding* strategy to begin planning

by already assigning sections which are certain on the critical time to different empty resource shifts. Lastly, we use the exact RLB graph for all Stage Two heuristic methods to sometimes prove optimality. If a Stage Two heuristic gives us a schedule using only $\underline{m}(T^{\text{crit}})$ resource shifts, this Stage Two schedule must be optimal given the Stage One solution.

4.3 'Drexl' Prepocessing Time-Windows

An interesting Time-Window pre-processing step before solving Stage Two is described in Drexl et al. [24], although also related to work of Groenendijk [36]. Sections are made independent by adjusting their earliest/latest start/completing time-windows to not overlap. This decreases the complexity and the computation times of the Stage Two solution methods while sacrificing some flexibility and therefore solution quality.

Let us consider the feasible (reduced) start/completion time-windows $\left[\underline{S}_j, \overline{S}_j\right]$ and $\left[\underline{C}_j, \overline{C}_j\right]$ of all route sections j of a single route $r \in \mathcal{R}$ in a Stage One solution **including** current break positions. For ease of notation, let sections $j = 1, \ldots, n_r$ correspond to sections at position $1, \ldots, n_r$ in route r and let n_r be the total number of sections in route r. Let $\left[\underline{S}'_j, \overline{S}'_j\right]$ and $\left[\underline{C}'_j, \overline{C}'_j\right]$ denote the adjusted start/completion time-windows of section $j = 1, \ldots, n_r$. The time-windows of the last section $j = n_r$ are not adjusted: $\left[\underline{S}'_{n_r}, \overline{S}'_{n_r}\right] = \left[\underline{S}_{n_r}, \overline{S}_{n_r}\right]$, $\left[\underline{C}'_{n_r}, \overline{C}'_{n_r}\right] = \left[\underline{C}_{n_r}, \overline{C}_{n_r}\right]$. Looking at section $j = n_r - 1$, its time-windows are adjusted to: $\left[\underline{C}'_j, \overline{C}'_j\right] = \left[\underline{C}_j, \underline{S}'_{j+1} - \Delta_{j,j+1}\right]$, $\left[\underline{S}'_j, \overline{S}'_j\right] = \left[\underline{S}_j, \overline{S}^*_j\right]$, with $\overline{S}^*_j$ being the latest feasible start time of section j to complete the section at time $\overline{C}'_j = \underline{S}'_{j+1} - \Delta_{j,j+1}$, and $\Delta_{j,j+1}$ being the loading time between sections j and $j+1$ (which is usually fixed: $\Delta_{j,j+1} = \Delta$). Note this eliminates any overlap (dependence) between the completion time-window of section j and start time-window of $j+1$, accounting for the needed loading time in between, thus their movement in time is now independent. Note also that trailer loading times can now be fixed to begin at $\underline{S}'_{j+1} - \Delta_{j,j+1}$ and complete at $\underline{S}'_{j+1}$. The recursion continues backwards: $j \leftarrow j - 1$.

This recursion relation is similar to one which we will use later for exactly solving the CG pricing subproblem, equation (6.29), which can be used to determine all adjusted time-windows, including the values $\overline{S}^*_j$ exactly.

Note that using these new section start/completion time-windows, sections are independent since their movement in these time-windows does not influence movement of other sections in the same route. Only for routes with two or more sections ($n_r \geq 2$), some time-windows are (possibly) adjusted. This pre-processing method is especially restricting the time-windows for routes containing a lot of sections ($n_r \geq 3$). Figure 4.3 shows this effect of pre-processing on a Stage One solution of Solomon instance 'C207'. Notice the loading times, represented by dashed lines between two circles, can be scheduled independently after pre-processing.

Although not mentioned by Drexl et al. [24], we found that it is vital to include the Stage One predicted breaks in the calculation of the reduced start/completion time-windows. If breaks are not considered, the method could narrow time-windows of a section in such way no breaks can even be planned inside it by the Stage Two solution method, resulting in an infeasible solution. By including the Stage One breaks, we are sure these exact breaks are feasible also in Stage Two. Still, by narrowing the time-windows by the 'Drexl' pre-processing, chances these breaks can be moved to a better position in the Stage Two solution are reduced.

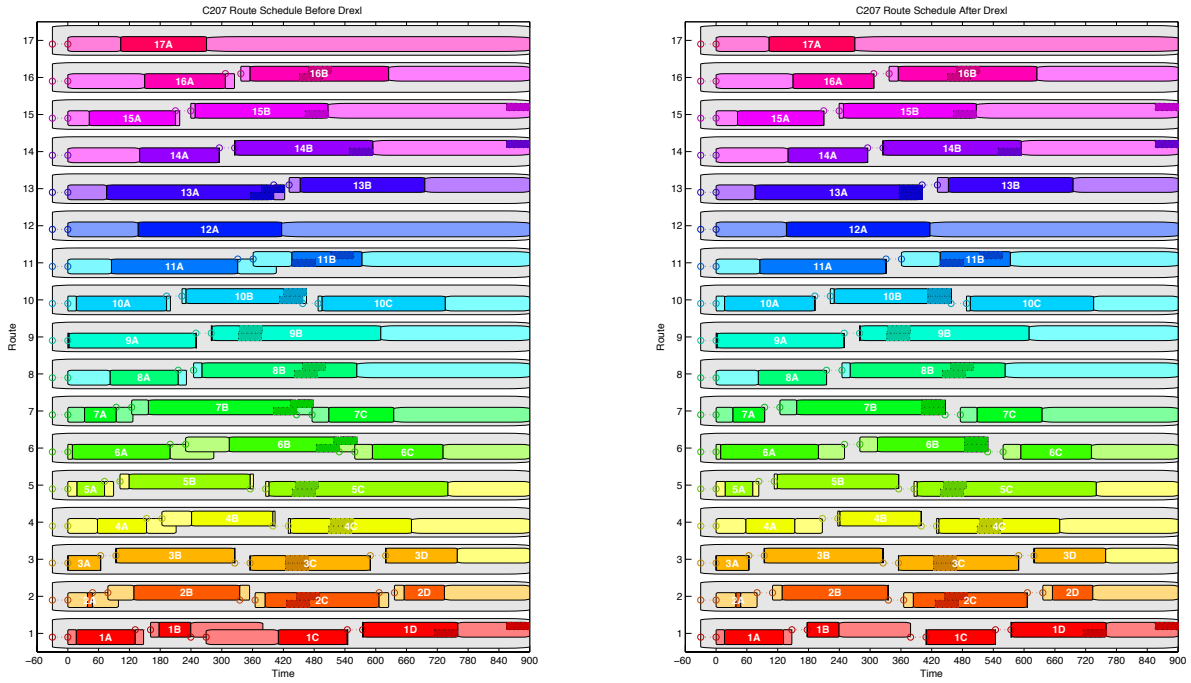


Figure 4.3: Example of the 'Drexl' pre-processing of the Stage One solution of Solomon instance 'C207' using *NotFresh* strategy. Left: original route schedule; Right: route schedule after 'Drexl' pre-processing. Sections are shown with earliest/latest start and earliest/latest completion times. Dashed lines between two circles represents the earliest time (re)loading can take place. Darker dashed areas show earliest/latest times of current Stage One predicted breaks.

4.4 Stage Two: Resource Assignment

We propose and investigate the following methods for solving the Stage Two problem of (truck+driver) resource assignment:

- ***SPCI*** – Section Parallel Cheapest Insertion (heuristic)
  This method generalizes the PCI construction heuristic to insert sections on resource shifts. We consider different insertion costs/sorting/seeding strategies, which are formulated in Chapter 5.2. Since sections are temporal dependent (synchronization is present), more elaborate feasibility checks are formulated.

- ***CGDP*** – Column Generation with Dynamic Programming/Labelling pricing (exact)
  This method uses a column generation method with explicit precedence constraints in the MP, like in van den Akker et al [1], described in Section 6.1. Pricing problem(s) exactly by a labelling algorithm, which is inspired on combining work of Ioachim et al.[44] and Feillet et al. [29], which is described in Section 6.3.

- ***CGDP 'Drexl'*** – Column Generation with Dynamic Programming/Labelling pricing and 'Drexl' pre-processing of time-window (heuristic)
  This method uses the above CGDP method, but the route section start/completion time-windows are pre-processed inspired by Drexl et al. [24], as explained in above Section 4.3. Route sections are made independent. This makes the method heuristic, since time-windows are limited, but the computation times are generally lower than the full exact CGDP method. Also no more synchronization is needed, since sections are independent. Therefore the CG MP and pricing subproblems are much easier.

- ***CGDP 'OnlyRoutes'*** – Column Generation with Dynamic Programming/Labelling pricing, but only assign full trailer routes (exact)
  This method used the full exact CGDP method, but a small adjustment is made to the pricing subproblems by only allowing drivers to execute full trailer route, so trailer swapping is only allowing swapping between full routes. The method is exact and even free from synchronization/precedence constraints, since full trailer routes are independent. We use this method to compare the optimal fixed trailer route assignment to the exact assignment when swapping is allowed between route section, so to evaluate the benefit of swapping trailers between route sections.

- ***CGSPCI*** – 'Hybrid' Column Generation with SPCI heuristic pricing (heuristic)
  This method uses the Column Generation Master Problem (CG MP) of the above CGDP method, but solves the pricing subproblem(s) heuristically by the use of the SPCI construction heuristic. Each pricing subproblem iteration is simultaniously solved by multiple SPCI runs with different strategies to diversify the generation of columns. We describe this connection between SPCI and the CG MP in Section 6.4.

Chapter 5

Construction Heuristics

In combinatorial optimization, where the number of feasible solutions to a problem is finite but very large, one is usually less interested in the exact optimal solution if the time required to compute it increases tremendously. A feasible, near optimal solution will in fact be better if this lowers the computation times significantly without sacrificing a lot in solution quality (on-average). Algorithms or methods able to provide feasible, close-to-optimal solutions are so-called *heuristics*. Heuristics can typically be divided into two types: *Construction Heuristics* and *Improvement Heuristics*.

Construction heuristics build a feasible solution from 'scratch' by making greedy choices which are computationally fast, but might not lead to a high quality solution. In vehicle routing literature, well known examples are the *Nearest-Neighbour*, the *Clarke and Wright Savings* and the *Solomon Cheapest Insertion* construction heuristics [10].

Improvement heuristics require a feasible solution as input and try to improve the solution. In vehicle routing and scheduling literature, the most used class of improvement heuristics is the class of *Local Search* heuristics. Local Search heuristics try to evaluate many local changes (called *moves*) to a solution, iteratively choosing the best and continuing the search. However, many improvement heuristics tend to get 'stuck' inside a local optimal solution. In many vehicle routing problems, these local optima can be significantly worse than the global optimum, so in general methods are needed to overcome this. So-called *Metaheuristic* methods use different strategies to try and guide the improvement heuristics out of their local optima [32, 57, 60], for instance by temporary allowing worse solutions or by searching a massive area with help of some random choices. Popular methods in the VRP literature using the first strategy are *Tabu Search* and *Simulated Annealing* [17, 32, 57], while popular methods using the second strategy are *Evolutionary Algorithms* [17, 71] and *Large Neighbourhood Search* [60] (particularly Pisinger and Ropke's *Destroy-and-Recreate* algorithms [59]). Although these methods are very successful in solving many different vehicle routing- and scheduling-like problems, we will focus primarily on the development of a construction heuristic for the second stage of resource assignment. Improvement heuristic, and metaheuristics, can be build 'on-top' of such a construction heuristic. We will discuss some of these possibilities.

We will use Solomon's Cheapest Insertion [67] heuristic in a parallel way as described in Potvin et al. [61], which we will denote by *Parallel Cheapest Insertion* (PCI). Note that parallel in this context does not mean using multiple CPUs, but rather that multiple routes are being constructed in parallel (possible insertions in multiple routes are considered at once). To clarify this PCI algorithm, we first discuss how to apply this algorithm to our Stage One problem of planning trailer routes. Then we also adapt this algorithm to our Stage Two problem of assigning/planning trailer route sections to driver resource shifts, which we will denote by *Section*

Parallel Cheapest Insertion (SPCI). Since the sections of Stage Two are dependent by precedence constraints on start/completion times, this requires a modified feasibility checking procedure. Also we discuss some different insertion cost and sorting/seeding strategies.

5.1 Stage One: Parallel Cheapest Insertion

Algorithm 5.1 Stage One: Parallel Cheapest Insertion

```
1:  procedure PCI(U)
2:      Input: set of unplanned customers U.
3:      Output: feasible solution x.
4:      x ← solution with a single empty route containing a single empty trip
5:      Ũ ← SortCustomers(U)
6:      repeat
7:          x_b ← ∅, c_b ← ∞
8:          continue ← false
9:          for s ∈ Ũ do
10:             for all positions pos of all trips t of all routes r in current solution x do
11:                 if IsFeasibleInsert(s, pos, t, r, x) then
12:                     c ← GetInsertCost(s, pos, t, r, x)
13:                     if c < c_b then
14:                         x_b ← InsertCustomer(s, pos, t, r, x)
15:                         c_b ← c
16:                         s_b ← s
17:         if c_b < ∞ then
18:             x ← x_b
19:             Ũ ← Ũ \ s_b
20:             // Ũ ← SortCustomers(Ũ)
21:             continue ← true
22:     until Ũ = ∅ or ¬continue
```

Algorithm 5.1 shows a schematic overview of our PCI construction heuristic. In the algorithm, we assume trailers are homogeneous. First, an empty solution x is initialised containing only a single empty route with one empty trip. The set of unplanned customers U is sorted by the procedure SortCustomers using a *sorting strategy*. The sorting strategy is used to break ties between feasible insertions with the same insertion costs. We use total travel time $t_{oi} + t_{id}$ from and to the depot as sorting, favouring highest total travel time. Inside the repeat-loop, the current best solution x_b and c_b are initialized to the dummy values of $\varnothing$ and ∞ respectively. Then for all customers s in the sorted customer list $\tilde{U}$, all positions pos of all trips t in all routes r in the current solution x are considered. Procedure IsFeasibleInsert checks if inserting customer s at position pos in trip t of route r is feasible. Efficient *pre-checks* are used first to quickly evaluate infeasibility regarding capacity and earliest/latest start-of-service times with use of Campbell and Savelsbergh [12]. If these pre-checks are passed, the exact earliest start-of-service times for the whole trailer route r starting at the previous customer (at $pos - 1$) are calculated (until a time-window is violated) and new breaks are inserted after $T_{\max}^{\text{driv}}$ of driving time. If passed, the insertion is feasible. Its insertion cost is then calculated by procedure GetInsertCost, using a *cost strategy*. Note that this insertion cost does not have to be equal to any exact cost derived from the problem formulation. The insertion costs can contain terms to influence the generation of 'nice' route structures. Insertion costs we used are inspired by Solomon [67], and can be found in the Appendix A.1. If the insertion cost c is lower than the best insertion cost c_b found so far, the current insertion will be the new temporary best. Procedure InsertCustomer will insert

the customer in the new best solution x_b. If the customer is inserted in an empty trip t, two new empty trips are added to solution x_b, one before and one after trip t (without any loading time). If the complete route r was also empty, a new empty route with an empty trip is added to the solution x_b. After all possible unplanned customer insertions in all positions in all trips in all routes are considered, the best solution x_b is taken as new solution x. Inserted customer s_b is removed from unplanned customer set $\tilde{U}$. The process repeats until all customers are planned or no customer could be feasibly inserted.

5.2 Stage Two: Section Parallel Cheapest Insertion

The PCI construction heuristic of Stage One can be modified to solve the Stage Two problem by inserting route sections in (truck+driver) resource shifts. This results in the SPCI (Section PCI) Algorithm 5.2.

Algorithm 5.2 Stage Two: Section Parallel Cheapest Insertion

```
1:  procedure SCPI(U)
2:      Input: set of unassigned sections U.
3:      Output: feasible schedule x.
4:      x ← schedule with a single empty resource shift
5:      if using seed strategy then
6:          (x, U) ← PlanSeedSections(x, U)
7:      Ũ ← SortSections(U)
8:      repeat
9:          x_b ← ∅, c_b ← ∞
10:         continue ← false
11:         for all s ∈ Ũ do
12:             for all positions pos of all resource shifts r in current solution x do
13:                 if IsFeasibleSectionInsert(s, pos, r, x) then
14:                     c ← GetSectionInsertCost(s, pos, r, x)
15:                     if c < c_b then
16:                         x_b ← InsertSection(s, pos, r, x)
17:                         c_b ← c
18:                         s_b ← s
19:         if c_b < ∞ then
20:             x ← x_b
21:             Ũ ← Ũ \ s_b
22:             Ũ ← SortSections(Ũ)
23:             continue ← true
24:     until Ũ = ∅ or ¬continue
```

An empty schedule x is initialized. If a seeding strategy is used, seed route sections are planned in resource shifts by the PlanSeedSections procedure, before the cheapest insertion starts. Next, the unassigned sections are sorted by SortSections. A cheapest insertion phase starts which repeatedly evaluates all positions in all resource shifts. Feasibility of each insertion is evaluated by procedure IsFeasibileSectionInsert, which after doing some quick pre-checks now needs to do a time calculation more sophisticated than that inside the PCI due to synchronization of dependent sections in different resource shifts. This can be done by using *Topological Sorting*, which is a general method for ordering vertices in a directed acyclic graph. We describe how to use topological sorting for time (feasibility) calculations in Section 5.2.1. The cheapest insertion

phase continues similarly to the PCI algorithm: if the insertion of section s at position pos in resource shift r is found feasible, its cost are determined by procedure GetSectionInsertCost. Our investigated cost/sort strategies are described in Section 5.2.2. If its insertion cost are better than the best found so far, the section is inserted in the best solution x_b by procedure InsertSection. If the resource shift was empty, InsertSection also adds a new empty resource shift to x_b. After all insertions of sections at all position in all resource shifts are evaluated, the best solution x_b is taken and inserted section s is removed from list $\tilde{U}$. The list is sorted again, since and the process repeats until no section can be inserted.

Here, we assume we have a homogeneous set of resource shifts (same start/completion availability), since this simplifies this heuristic. However, we will develop more advanced methods based on Column Generation which are able to use a heterogeneous set of resource shifts without much adoption.

5.2.1 Topological Sorting

Since route sections are inter-dependent, possibly between different resource shifts, updating start-of-service times cannot simply be done consecutively updating earliest start-of-service times along the resource shifts alone, or along the trailer routes alone. See the example on dependent sections used earlier in Figure 2.3. Suppose we have a current feasible Stage Two solution of 3 trailers (blue, green, orange) and 2 resource shifts with route sections (‘3A’, ‘1A’, ‘2B’) and (‘2A’, ‘3B’, ‘1B’) assigned to them as in Figure 2.3(a). Also suppose we need to evaluate the insertion of an independent section ‘4A’ of trailer 4, to be inserted at the start of resource shift 1 (position 1). Assume trailer 4 contains only section ‘4A’. Figure 2.3(b) shows that only updating earliest start-of-service times inside resource shift 1 is not enough. Some parts of trailer routes and parts of dependent resource shifts also need to be updated. To determine which sections need to be updated and of similar importance in which order the updating should take place, we can use *Topological Sorting* of the *dependency graph* [18].

The dependency graph of a temporary solution is given by $\mathcal{G}^{\text{dep}} = \left(\mathcal{V}^{\text{sec}}, A^{\text{dep}}\right)$, in which each section $j \in \mathcal{J}$ consists of a vertex $v_j \in \mathcal{V}^{\text{sec}}$ and each arc $e_{ij} \in A^{\text{dep}}$ indicates that route sections i is the direct predecessor of section j in a trailer route or resource shift. The arcs in the example are ‘1A’→‘1B’, ‘2A’→‘2B’, ‘3A’→‘3B’, ‘4A’→‘3A’→‘1A’→‘2B’, ‘2A’→‘3B’→‘1B’. Topologically sorting this graph, starting at ‘4A’, gives an ordered list of sections to be updated starting at ‘4A’. This gives the list (‘4A’, ‘3A’, ‘1A’, ‘3B’, ‘2B’, ‘1B’). Notice not all sections are present, simply because these not present are not affected in time by the insertion of ‘4A’. Further notice the topological sorting is not unique: we may end with ‘2B’, ‘1B’ or ‘1B’, ‘2B’. Finally, one can prove that every acyclic graph admits a topological sorting, while graphs containing cycles cannot be topological sorted. If inserting a section in a resource shift causes a cycle in the dependency graph $\mathcal{G}^{\text{dep}}$, there is a section which now needs to be planned ‘after itself’, which is impossible. In that case, the insertion is infeasible. Topological sorting algorithms will detect cycles, at which point they stop working.

We have implemented a topological sorting algorithm based on *depth-first search* [18] inside procedure IsFeasibleSectionInsert to, for an insertion which pasted pre-checks, obtain the correct order of sections to be updated for time calculations or to conclude infeasibility of the insertion by the detecting a cycle in the dependency graph. Our algorithm is based on the algorithm presented by Cormen et al. in chapter 22 of the book *Introduction to Algorithms* (2009, Third Edition) [18].

5.2.2 Insertion Strategies

Cost/Sort Functions

Suppose section $j \in \mathcal{J}$ is considered for insertion between sections k and l in resource shift $r \in \mathcal{S}$, with $k \in \mathcal{J} \cup \{o\}$ and $l \in \mathcal{J} \cup \{d\}$, $k = o$ and $l = d$ corresponding to respectively inserting section j as first or last section in the resource shift. Let $\underline{S}^r$, $\overline{S}^r$ be the earliest, latest start time of resource shift r and $\underline{C}^r$, $\overline{C}^r$ be the earliest/latest completion time of resource shift r before the insertion. Let $\underline{S}'^r$, $\overline{S}'^r$, $\underline{C}'_d$, $\overline{C}'_d$ be the same quantities after inserting section j in resource shift r. We define the insertion cost $c(k, l, j, r)$ for inserting section j between k and l in resource shift r by:

$$c(k,l,j,r) = \begin{cases} c_1(k,l,j,r) & \text{if resource shift } r \text{ is non-empty,} \\ \beta\left(b_r^{\text{sh}} - a_r^{\text{sh}}\right) & \text{if resource shift } r \text{ is empty,} \end{cases} \tag{5.1}$$

with $\left[a_r^{\text{sh}}, b_r^{\text{sh}}\right]$ being the availability time interval of resource shift r and α a factor discouraging the planning on empty resource shifts. We set $\beta = 2$, which showed good results. The cost function $c_1(k, l, j, r)$ is used when an insertion in non-empty resource shift is considered, while the costs for inserting in a empty resource shift is high and fixed for all sections. In this latter case, the sorting of the section will break these ties. Therefore the sort function steers the insertion of sections in empty resource shifts. The sorting will also break ties for non-empty resource shift insertions with the same c_1 insertion costs.

We consider the following two cost/sort function pairs.

- ***Compl. Time/Earliest Compl. Time***
  We use the following cost function based on difference in the earliest completion time of the resource shift before and after insertion:

  $$c_1(k,l,j,r) = \underline{C}'^r - \underline{C}^r. \tag{5.2}$$

  Sorting is done on earliest completion time $\underline{C}_j$ of section $j \in \mathcal{J}$, favouring sections completed earliest. This cost/sort function strategy will result in a section completing earliest being inserted on an empty resource shift and subsequently inserting sections completing early after it.

- ***Incr. Certain Makespan/Largest Certain Duration***
  We use the following cost function based on the difference in certain *makespan* or certain duration of a resource shift before and after insertion:

  $$c_1(k,l,j,r) = \overline{S}^r - \overline{S}'^r + \underline{C}'^r - \underline{C}^r. \tag{5.3}$$

  Sections are sorted decreasingly on their currently known certain duration: $t_j^{\text{cer}} = \underline{C}_j - \overline{S}_j$, with $\underline{C}_j$ the earliest completion time and $\overline{S}_j$ the latest start time of section $j \in \mathcal{J}$, favouring highest certain duration. This cost/sort strategy will result in a section with largest certain makespan being inserted in an empty resource shift and subsequently being filled with additional sections which each minimal increase in the total certain duration of the resource shift. Sections with the high certain duration are usually more difficult to insert, especially is already a lot of sections are planned. Therefore, it is reasonable to insert them first and try to fill sections 'around' them.

In using both strategies, after each insertion the sorting may change due to dependencies of unplanned sections with planned ones. Therefore, after each insertion, sorting (procedure SORT-SECTIONS) is done.

Seed Strategies

We also consider two seed strategies for inserting sections in resource shifts before the cheapest insertion phase starts and encourage planning sections away from bottlenecks. They are both based on the resource lower bound (RLB) graphs, described in Section 4.2.

- ***Bottleneck Res. LB***
  Before the cheapest insertion phase starts, sections which must be planned on the critical time T^{crit} are each planned as seed on a different resource shift. Let $\mathcal{J}^{\text{crit}} \subset \mathcal{J}$ be the set of sections which for certain overlap the (earliest) critical time T^{crit} of the RLB graph: $\mathcal{J}^{\text{crit}} = \left\{ j : \forall j \in \mathcal{J},\, T^{\text{crit}} \in \left[\overline{S}_j, \underline{C}_j\right] \right\}$. In every feasible schedule, each section in $\mathcal{J}^{\text{crit}}$ must be assigned to a different resource shift. It could be beneficial to do this already before cheapest insertion phase starts. Procedure PlanSeedSections in SPCI Algorithm 5.2 plans every section $j \in \mathcal{J}^{\text{crit}}$ on a different resource shift.

  During the cheapest insertion phase, sections which may overlap the critical time should be avoided to be inserted on the critical time, or equivalently are encouraged to be planned next to a seed section. Let $\mathcal{J}^{\text{ncrit}} \subset \mathcal{J}$ be the sections which could overlap the (earliest) critical time T^{crit} of the RLB graph, but not need to: $\mathcal{J}^{\text{ncrit}} = \left\{ j : \forall j \in \mathcal{J},\, T^{\text{crit}} \in \left[\underline{S}_j, \overline{S}_j\right] \cup \left[\underline{C}_j, \overline{C}_j\right] \right\}$. Inspired by the critical interval guidance strategies of Battarra et al. [7], we like to penalize planning sections in $\mathcal{J}^{\text{ncrit}}$ overlapping the critical time T^{crit}/encourage planning sections in $\mathcal{J}^{\text{ncrit}}$ completing before/starting after this time, or equivalently before/after a seed section in $\mathcal{J}^{\text{crit}}$. The latter was chosen: the insertion cost $c_1(k,l,j,r)$ for inserting section $j \in \mathcal{J}^{\text{ncrit}}$ in resource shift r which is already seeded with a section of set $\mathcal{J}^{\text{crit}}$ is lowered: $c_1(k,l,j,r) \leftarrow c_1(k,l,j,r) - c_{\text{avoid}}(k,l,j,r)$, with $c_{\text{avoid}}(k,l,j,r)$ an *avoidance* bonus. We set $c_{\text{avoid}}(k,l,j,r) = \underline{S}^r + \overline{S}^r$, which are the earliest/latest starting times of resource shift r before the insertion.

- ***Bottleneck Res. LB Breaks***
  We follow the same procedures as above, but use the RLB with (Stage One) breaks graph (blue line in right graph of Figure 4.2 instead of left graph). Since Stage One breaks might be moved in Stage Two, not all sections of $\mathcal{J}^{\text{crit}}$ have to be planned on different resource shifts if some breaks are changed. However, we found that since the RLB with breaks graph better predicts the actual usage of resource shifts, this strategy performed slightly better than the above Bottleneck Res. LB. Disadvantage of this seeding strategy is that a schedule requiring less than the RLB with breaks predictive lower bound $\underline{m}(T^{\text{crit}}) \equiv \left|\mathcal{J}^{\text{crit}}\right|$ cannot be found, while such schedule might exist.

5.3 Advanced LS methods

The Stage Two SPCI and especially the Stage One PCI are very basic construction heuristics. These construction heuristics can be used in combination with more advanced local search methods, such as *Variable Neighborhood Search* (VNS) [40] or Large Neighbourhood Search (LNS) improvement methods. After construction, the solution is improved by iteratively considering local moves/swaps (VNS) or by partially destroying the solution and re-constructing it (LNS). The feasibility checks of our SPCI method using Topological Sorting are very important in such methods as well. The construction methods can even be used directly for re-construction in LNS. In other words, there are many possibilities of improving the construction heuristic, which (unfortunately) lie outside the scope of this thesis. We do however investigate a simple 'Hybridization' of column generation and the SPCI heuristic as pricing, which will be explained in Chapter 6.4.

Chapter 6

Column Generation

The method of column generation has proven itself by good performances for solving very large ILPs [6, 54]. Besides popular for solving large scheduling problems, it is increasingly used in literature to solve various vehicle routing problems as well [28, 45, 46]. Inspired by a recent application of column generation for solving the general parallel machine scheduling problem with precedence relations by van den Akker et al. [1], we consider the use of column generation to solve our second stage problem of assigning sections to resources.

6.1 Master Problem

We now consider the master problem for the Stage Two Resource Assignment Problem defined in Section 2.5.2. It is assumed that we have a feasible solution for the stage one problem, e.g., that feasible trailer routes consisting of feasible trips are present.

As stated earlier in Section 2.5.2, a feasible Stage One solution consists of the following variables: trip flow variables x_{ij}^{τ}, which represent if arc $e_{ij} \in \mathcal{A}$ is used in trip $\tau \in \mathcal{T}$; γ_i^{τ}, which represent if customer $v_i \in \mathcal{V}_C$ is serviced in trip $\tau \in \mathcal{T}$; trailer route flow variables y_{kl}^{r}, which represent if trip $l \in \mathcal{T} \cup \{d\}$, is executed directly after $k \in \mathcal{T} \cup \{o\}$, $k \neq l$ by trailer route $r \in \mathcal{R}$; ρ_{τ}^{r}, which represent if trip $\tau \in \mathcal{T}$ is executed in route $r \in \mathcal{R}$; and m_r^{tr}, which represent if route $r \in \mathcal{R}$ is used (non-empty).

Let $R \subset \mathcal{R}$ be the set of used trailer routes in the Stage One solution: $R = \{r \in \mathcal{R} : m_r^{\text{tr}} = 1\}$. Let $\mathcal{J} \subset \mathcal{T}$ be the set of planned non-empty trips which are assigned to some trailer route in the Stage One solution: $\mathcal{J} = \{\tau \in \mathcal{T} : \exists r \in R \text{ with } \rho_{\tau}^{r} = 1\}$. We will call these planned non-empty trips assigned to trailer routes *(route) sections* to help clarify such trips are part of an already planned route. In Stage Two problem, each route section needs to be assigned to a single (truck+driver) resource shift $s \in \mathcal{S}$. In scheduling terms, each route section $j \in \mathcal{J}$ can be seen as a single *job*.

The Stage One solution gives rise to earliest/latest start times $[\underline{S}_j, \overline{S}_j]$ and earliest/latest completion times $[\underline{C}_j, \overline{C}_j]$ of each section $j \in \mathcal{J}$. These times are reduced to ensure feasible start/completion times of all other sections in a trailer route (see Section 4.2). However, here the Stage One predicted breaks are neglected, since breaks could be positioned differently in the Stage Two solution. Let variable y_{kl} denote if section $k \in \mathcal{J}$ is the direct predecessor of section $l \in \mathcal{J}$ in some trailer route: $y_{kl} = \sum_{r \in R} y_{kl}^{r}$ for all $k, l \in \mathcal{J}$, $k \neq l$. If $y_{kl} = 1$, the trailer of sections $k \in \mathcal{J}$ and $l \in \mathcal{J}$ must be (re)loaded between those two subsequent sections. Let Δ_{kl} be the loading time between subsequent sections $k, l \in \mathcal{J}$ with $y_{kl} = 1$. Note that there are exactly $|\mathcal{J}| - |R|$ pairs of sections with $y_{kl} = 1$, with $|R|$ being the number of trailer routes. This is because each

section $j \in \mathcal{J}$ is being preceded by exactly one section, except for the sections that start a route, of which there are $|R|$.

The column generation method allows us to group similar resource shifts in $\mathcal{S}$ into *resource shift kinds*. Two resource shifts belong to the same resource shift kinds if their characteristics are the same, which in our model corresponds to the both having the same availability time-window $\left[a_s^{\text{sh}}, b_s^{\text{sh}}\right]$ and the same operational cost c_s^{sh}. A resource shift $s \in \mathcal{S}$ can only belong to one resource shift kind $k \in \mathcal{K}$. Let $\mathcal{K}$ be the set of resource shift kinds and suppose there are u_k resource shifts in $\mathcal{S}$ available of kind $k \in \mathcal{K}$ (so $\sum_{k \in \mathcal{K}} u_k = |\mathcal{S}|$).

Let a *single resource schedule* be an assignment of sections to a resource shift (of a specific resource shift kind) with specific section start and completion times. Let Ω be the set of all such possible feasible single resource schedules. Let c_s denote the operational cost of a single resource schedule $s \in \Omega$. For our problem, $c_s = c_{s'}^{\text{sh}}$, with $s' \in \mathcal{S}$ being the corresponding resource shift for single resource schedule $s \in \Omega$. Let binary $a_{js} = 1$ if section $j \in \mathcal{J}$ is included in single resource schedule $s \in \Omega$, otherwise $a_{js} = 0$. Let binary $b_{ks} = 1$ if single resource schedule $s \in \Omega$ is a resource shift of kind $k \in \mathcal{K}$, otherwise $b_{ks} = 0$. Let S_{js}, $C_{js} \in [0, T]$ denote the start, completion times respectively of section $j \in \mathcal{J}$ in resource schedule $s \in \Omega$ if section $j \in \mathcal{J}$ is in schedule $s \in \Omega$, otherwise $S_{js} = C_{js} = 0$. A single resource schedule $s \in \Omega$ is characterized by a vector $(\mathbf{a}_s, \mathbf{b}_s, \mathbf{S}_s, \mathbf{C}_s)$, with $\mathbf{a}_s = (a_{1s}, \ldots, a_{|\mathcal{J}|s})$, $\mathbf{b}_r = (b_{1s}, \ldots, b_{|\mathcal{K}|s})$ and $\mathbf{S}_s$, $\mathbf{C}_s$ the vectors of respectively the start and completion times of the sections in resource schedule $s \in \Omega$.

Formally, a single resource schedule $s \in \Omega$ consists of the following values:

$$a_{js} = \begin{cases} 1 & \text{if schedule } s \text{ contains section } j, \\ 0 & \text{otherwise.} \end{cases} \tag{6.1}$$

$$b_{ks} = \begin{cases} 1 & \text{if schedule } s \text{ is for resource shift kind } k, \\ 0 & \text{otherwise.} \end{cases} \tag{6.2}$$

$$S_{js} \in \begin{cases} [\underline{S}_j, \overline{S}_j] & \text{starting time of section } j \text{ in schedule } s, \\ \{0\} & \text{if schedule } s \text{ does not contain section } j. \end{cases} \tag{6.3}$$

$$C_{js} \in \begin{cases} [\underline{C}_j, \overline{C}_j] & \text{completion time of section } j \text{ in schedule } s, \\ \{0\} & \text{if schedule } s \text{ does not contain section } j. \end{cases} \tag{6.4}$$

Let the binaries x_s indicate if single resource schedule $s \in \Omega$ is selected in the complete Stage Two solution schedule. We can now formulate the Stage Two resource assignment problem as a set partition problem with explicit precedence constraints. This ILP is called the *master problem* (MP) in column generation methods. The variables shown between the brackets on the right are the dual variables corresponding to the constraints.

$$\text{(MP)} \quad \min. \sum_{s \in \Omega} c_s x_s, \tag{6.5}$$

$$\text{s.t.} \sum_{s \in \Omega} a_{js} x_s = 1 \qquad \forall j \in \mathcal{J}, \qquad [\lambda_j] \tag{6.6}$$

$$\sum_{s \in \Omega} b_{ks} x_s \leq u_k \qquad \forall k \in \mathcal{K}, \qquad [\mu_k] \tag{6.7}$$

$$\sum_{s \in \Omega} S_{js} x_s - \sum_{s \in \Omega} C_{is} x_s \geq \Delta_{ij} \qquad \forall i, j \in \mathcal{J} \text{ with } y_{ij} = 1, \qquad [\delta_{ij}] \tag{6.8}$$

$$x_s \in \{0, 1\} \qquad \forall s \in \Omega. \tag{6.9}$$

The set partitioning constraints (6.6) state that in the complete solution schedule, each section should be contained in exactly one single resource schedule. Constraints (6.7) state that there can be at most u_k single resource schedules of kind $k \in \mathcal{K}$ in the complete solution, since their are u_k resource shifts of that kind. Precedence constraints (6.8) state that for two consecutive sections i and j, both in $\mathcal{J}$ with $y_{ij} = 1$, section j should start at least Δ_{ij} time after section i is finished. Notice these sections could be contained in different single resource schedules, which are different resource shifts of the complete solution. These *inter-shift* constraints are explicitly modelled in the master problem, just as in van den Akker et al. [1]. As stated earlier, there are $|\mathcal{J}| - |R|$ of these constraints. The MP consists of $2|\mathcal{J}| + |\mathcal{K}| - |R|$ constraints in total, which can be approximated by $\mathcal{O}(2n + |\mathcal{K}|)$, with $n = |\mathcal{V}_C|$ the number of customers and $\mathcal{O}$ the *big-o* notation.

Although the number of constraints is limited linearly, the MP does contain a very large number of variables x_s, or equivalently columns of the MP, since the set of all possible feasible single resource schedules Ω is combinatorially very large (possibly infinite). To solve the MP, we first solve its LP-relaxation by column generation. Column generation involves iteratively solving the LP-relaxed master problem over a subset $\Omega^* \subseteq \Omega$ of all possible single resource schedules, called the *restricted master problem* (RMP). Useful columns outside the subset Ω^* are generated and added iteratively to the RMP. In the RMP, the integer constraints are relaxed, so constraints (6.9) of the MP are replaced by

$$x_s \geq 0 \qquad \forall s \in \Omega^*. \tag{6.10}$$

6.1.1 Reduced cost

In the column generation method, the restricted master problem is first solved using a small subset of columns $\Omega^* \subseteq \Omega$. Then it is determined if there are any promising columns in $\Omega \setminus \Omega^*$ which need to be included to Ω^*. For this purpose, the so-called *reduced cost* $\tilde{c}_s$ of a single resource schedule $s \in \Omega \setminus \Omega^*$ is calculated using the dual variables of the current solution of the restricted master problem over Ω^*.

With the notation for the dual variables provided by the stated MP earlier, the reduced cost $\tilde{c}_s$ of $s \in \Omega \setminus \Omega^*$ can be written as:

$$\tilde{c}_s = c_s - \sum_{j \in \mathcal{J}} \lambda_j a_{js} - \sum_{k \in \mathcal{K}} \mu_k b_{ks} - \sum_{i,j \in \mathcal{J}, y_{ij}=1} \delta_{ij} \left(S_{js} - C_{is} \right), \tag{6.11}$$

or equivalently:

$$\tilde{c}_s = c_s - \sum_{j \in \mathcal{J}} \lambda_j a_{js} - \sum_{k \in \mathcal{K}} \mu_k b_{ks} - \sum_{j \in \mathcal{J}, y_{ij}=1} \delta_{ij} S_{js} + \sum_{j \in \mathcal{J}, y_{jl}=1} \delta_{jl} C_{js}. \tag{6.12}$$

Notice that the first part of this equation, without the right two sums over the precedence constraint dual values, is independent of time. The last two sum however are time-dependent, in particular dependent on the start and completion times of the sections.

Now the so-called *pricing* problem can be defined as the search for the most promising single resource schedules in $\Omega \setminus \Omega^*$. In terms of reduced cost, this means the search for resource schedules $s \in \Omega \setminus \Omega^*$ with the lowest reduced cost $\tilde{c}_s$. These columns are then added to the restricted master problem, which is then solved again to find new dual values. Only resource schedules with negative reduced cost have the potential to improve the current solution of the restricted master problem. Therefore, if at a certain stage the pricing problem determines the resource schedule s^* with the lowest reduced cost to have $\tilde{c}_{s^*} \geq 0$, then the current solution of the restricted master problem cannot be improved and thus the optimal solution to the RMP is found.

6.1.2 Initial Columns

To start the iterative process of solving the restricted master problem and then the pricing problem, we define an initial set of columns $\Omega_0^* \subset \Omega$. These represent the initial state of the master problem, but may also serve to detect infeasible solutions. Each section $j \in \mathcal{J}$ is represented by two dummy columns s_j^1, $s_j^2 \in \Omega_0^*$, the first containing the section planned as early as possible and the latter containing the section planned as late as possible. Both these dummy columns contain only one section $j \in \mathcal{J}$, so $a_{js_j^1} = a_{js_j^2} = 1$, $a_{j's_j^1} = a_{j's_j^2} = 0$, $\forall j' \in \mathcal{J}, j' \neq j$. They do not have a *physical* resource shift kind: $b_{ks} = 0$, $\forall k \in \mathcal{K}, s \in \Omega_0^*$. Further, their section start and completion times are given as early (as late) as possible: $S_{js_j^1} = \underline{S}_j$ ($S_{js_j^2} = \overline{S}_j$), $C_{js_j^1} = \underline{C}_j$ ($C_{js_j^2} = \overline{C}_j$). To make these dummy columns very unattractive (only to be chosen later iterations in case of infeasibility), their cost c_{dum} should be set high: $c_s = c_{\text{dum}} \gg c_{s'}$ for all $s \in \Omega_0^*$ and all $s' \in \Omega \setminus \Omega_0^*$. For our problem with fixed single-resource costs $c_{s'} = c_s^{\text{sh}}$, this can be easily satisfied.

The first iteration RMP over the dummy set of columns Ω_0^* will have a trivial feasible solution, in which either $x_{s_j^1} = 1$ or $x_{s_j^2} = 1$ for each $j \in \mathcal{J}$. Further, the dual solution is given by: $\lambda_j = c_{\text{dum}}$ for all $j \in \mathcal{J}$, $\mu_k = 0$ for all $k \in \mathcal{K}$ and $\delta_{ij} = 0$ for all $i, j \in \mathcal{J}$ with $y_{ij} = 1$. Although this solution could also be realized by including only half the columns: one column per section $j \in \mathcal{J}$, for instance only the earliest dummy column s_j^1 for every $j \in \mathcal{J}$, there is a benefit of both having 'earliest' and 'latest' section dummy columns. In case the Stage Two problem turns out to be infeasible in the first non-dummy iteration, the resulting schedule can indicate which section might cause this infeasibility without affecting the assignment of other sections of that trailer route. Such information could be useful for distribution companies to adjust their planning. Section $j \in \mathcal{J}$ might turn out to be infeasible, but the RMP can still 'move' it in time by choosing fractional values of both dummy columns: $x_{s_j^1} = 1 - x_{s_j^2} \in [0, 1]$, which effectively adjusts the *weighted* start and completion times between $\left[\underline{S}_j, \overline{S}_j\right]$ and $\left[\underline{C}_j, \overline{C}_j\right]$ respectively, as used in precedence constraints with the successor and predecessor section of j respectively. This allows the route predecessor and successor of j to be assigned without being affected by infeasibility of j. We will discuss the weighted start and completion times in more detail in Section 6.5.

6.2 Pricing Subproblems

The pricing subproblem consists of finding the column(s) of lowest reduced cost outside the RMP which could improve the RMP solution once added. Since a column in Ω corresponds to a single-resource schedule, the pricing problem is to find the best feasible single-resource schedule *priced* by the reduced cost $\tilde{c}_s$ of equation (6.12). It turns out to be convenient to divide this large problem into $|\mathcal{K}|$ subproblems: each subproblem concerning only one specific resource kind $k \in \mathcal{K}$. In each subproblem, the resource kind is fixed and therefore the contribution of the resource kind to the reduced cost equation, μ_k in (6.12), now becomes fixed in each subproblem.

A pricing subproblem can be described as finding the least reduced cost single-resource schedule, which is feasible under the following constraints: time-windows, driving rules (break planning), precedence/loading times. Such problem can be solved exact or heuristically. We will investigate solving the pricing subproblems exactly in Section 6.3, while a simple heuristic method by use of our SPCI algorithm is described in Section 6.4.

6.3 Exact Pricing

To solve the pricing problem exactly, one must optimize the reduced cost over all possible single resource schedules. This exact optimization problem is NP-hard and difficult since the reduced costs are time-dependent [1]. Van den Akker et al. [1] use Local Search heuristics first and then solve the pricing subproblem by formulating it as time-indexed ILP and using the heuristic solutions as (upper) bounds to speed-up the ILP solver. In our problem however, the time-indexed ILP will need a very large number of variables and constraints, since we additionally have the underlying route structures with customer time-windows in the sections and the driving rule to plan breaks.

Alternatively, the pricing subproblem with the constraints can be modelled as a variant of the Shortest Path Problem. Such problems consist of finding the shortest (or equivalently cheapest) path in a graph from an origin node to an destination node. The idea is to construct a single resource schedule as a path of sections, like the shift flow variables z^s_{kl} used in the SVCRSP formulation in Section 2.3. To find high quality single resource schedules, which will tighten the RMP bound, it is important their corresponding paths satisfy the customer time-windows, loading and precedence constraints. These can be modelled as *Resource Constraints*, with 'resources' being quantities representing the state of a path, which are extended through arcs by a *Resource Extention Function* (REF) [45]. Also important for high quality columns is the generation of only *Elementary* paths [46], which in our case means executing (visiting) a section in a single resource schedule only once. The reduced cost of executing a single schedule $j \in \mathcal{J}$ could be negative and we will see that our graph will contain negative cost cycles in that case, which could make it likely for a path to visit a section multiple times. Although including the elementary condition significantly increases the computational complexity, which is why it is often relaxed [47, 68], it has the benefit of producing high quality columns which tighten the bounds of the RMP [46]. Further, we will show that for our problem the elementary constraints can be modified to enforce precedence relations as well, and by the specific structure of our precedence constraint, they can be simplified.

The exact pricing subproblem is therefore modelled as an *Elementary Shortest Path Problem with Resource Constraints* (ESPPRC). First introduced by Beasley and Christofides [8] in 1989, it found many applications in solving vehicle routing (and/)or crew scheduling problems. It turns out that using column generation to solve such problems, the corresponding pricing (sub)problem(s) can be modelled as a SPPRC-variant and provide tight RMP lower bounds [46]. Still, the ESPPRC is NP-hard, as Dror [25] showed.

The algorithm of Feillet et al. (2004) [29] can be used to exactly solve these ESPPRCs. However, the main difficulty is to include the precedence constraint dual prices δ_{ij} of the reduced cost equation (6.12). The total reduced cost of a schedule depends on the start and completion times of its scheduled sections. To include these time-dependent cost, the ESPPRC model is extended to include *Linear Node Costs*, inspired by the *Shortest Path Problem with Time-Windows and Linear Node Costs* (SPPTWNC) of Ioachim et al. (1998) [44]. Another important property, which these models are extended with, is that single-resource schedules may contain *unforced waits*. It may be preferable for a driver to wait a bit before picking up a trailer (starting a section) to let another driver drop the trailer off (completing the trailers' previous section).

6.3.1 Auxiliary Single Resource Graph

Let us, for each pricing subproblem k, consider the following graph $\tilde{\mathcal{G}} = (\tilde{\mathcal{V}}, \tilde{\mathcal{A}})$, with $\tilde{\mathcal{V}}$ the set of vertices and $\tilde{\mathcal{A}}$ the set of arcs. This graph is often called the *auxiliary graph*. For simplicity, the dependence on resource shift kind k is omitted in the notation. In this graph, we model the

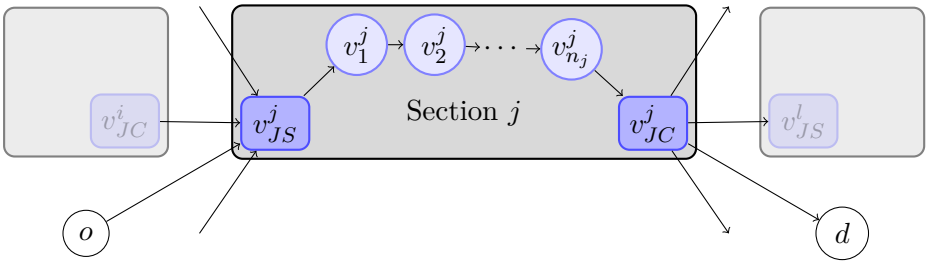


Figure 6.1: Schematic overview of a route section $j \in \mathcal{J}$, represented by vertices of subset the $\mathcal{V}_j$, in the auxiliary graph.

flow of sections in a resource shift as well with the inner flow of customers inside the sections to include break scheduling. Let $\tilde{\mathcal{V}} = \{o, d\} \cup \mathcal{V}_{JS} \cup \mathcal{V}_C \cup \mathcal{V}_{JC}$, with vertices o, d representing the start/end location of resource shift kind k, (functioning as the source/sink vertex of the graph respectively), $\mathcal{V}_{JS}$ the set of *section start* vertices, $\mathcal{V}_C$ the set of customers vertices and $\mathcal{V}_{JC}$ the set of *section completion* vertices. Each section $j \in \mathcal{J}$ has exactly one start vertex $v^j_{JS} \in \mathcal{V}_{JS}$ and one completion vertex $v^j_{JC} \in \mathcal{V}_{JC}$, as well as a number of unique customer vertices $v^j_1, v^j_2, \ldots, v^j_{n_j} \in \mathcal{V}_C$, with $v^j_i \in \mathcal{V}_C$ representing the ith customer in section j and n_j the total number of customers in section j. These customers are planned in the section (trip) of the Stage One solution. Let us use the notation $\mathcal{V}_j \subset \tilde{\mathcal{V}}$ as the (ordered) set of vertices of (in) section $j \in \mathcal{J}$: $\mathcal{V}_j = \left\{v^j_{JS}, v^j_1, v^j_2, \ldots, v^j_{n_j}, v^j_{JC}\right\}$. By definition, $\mathcal{V}_i \cap \mathcal{V}_j = \varnothing$ for all $i \neq j$ and $\tilde{\mathcal{V}} = \{o, d\} \cup \bigcup_{i=1,\ldots,|\mathcal{J}|} \mathcal{V}_j$.

Since the sequence of customers inside a section is fixed by the Stage One solution, the path between section start and section completion vertices is also fixed. Therefore inside each section, which is represented by the ordered set (sequence) of vertices $\mathcal{V}_j$, there is only a single forward directed arc in $\tilde{\mathcal{A}}$ between two subsequent vertices: $\left\{v^j_{JS} \to v^j_1 \to v^j_2 \to \cdots \to v^j_{n_j} \to v^j_{JC}\right\}$, which are dictated by the Stage One solution trip flow variables x^j_{pq} (See Section 2.5.2). The other arcs of $\tilde{\mathcal{A}}$ consist of connections between the sections: $e_{ij} \in \tilde{\mathcal{A}}$ for vertices $i \in \mathcal{V}_{JC} \cup \{o\}$ to vertices $j \in \mathcal{V}_{JS} \cup \{d\}$.

Let the driving time along arc $e_{pq} \in \mathcal{A}$ be given by t_{pq} and the servicing time duration at vertex $v_p \in \mathcal{V}$ by T^{serv}_p. The total time $\bar{t}_{pq}$ along arc $e_{pq} \in \mathcal{A}$ is given by:

$$\bar{t}_{pq} := \begin{cases} \Delta_{ij} & \text{if } v_p = v^i_{JC},\ v_q = v^j_{JS}, \text{ and } y_{ij} = 1, \\ 0 & \text{if } v_p = v^i_{JC},\ v_q = v^j_{JS}, \text{ and } y_{ij} = 0, \\ T^{\text{serv}}_p + t_{pq} & \text{if } v_p, v_q \in \mathcal{V}_j \text{ for one } j \in \mathcal{J}, \\ 0 & \text{otherwise.} \end{cases} \tag{6.13}$$

The arcs between vertices v^i_{JC} and v^j_{JS} $(i \neq j)$ with $y_{ij} = 1$ correspond to a special case where a driver is doing two subsequent sections i and j of the same trailer route. The total time along these arcs is equal to the trailer loading time Δ_{ij}, since the driver has to wait during the loading of the trailer. Further notice the total time $\bar{t}_{ij}$ does not include possible break time(s). These are modelled as label 'resources' in Section 6.3.2.

For each vertex $v_i \in \tilde{\mathcal{V}}$, let $[a_i, b_i]$ be the time window for which service can start at vertex v_i. These time windows are reduced in a sense that they represent the earliest and latest service start times at a vertex $v_i \in \tilde{\mathcal{V}} \setminus \{o, d\}$ also ensuring feasibility of the complete trailer route. Determining these reduced time-windows is done as a pre-processing step before the stage two resource assignment begins (See Section 2.5.2). Time windows of vertices o and d represent the earliest/latest start/completion time windows of the current resource shift kind k.

To model the propagation of the reduced cost equation (6.12) through the graph, we separate the time-independent cost part from the time-dependent cost part. Time-independent costs consist of the fixed resource shift cost c_s, resource shift kind availability dual value μ, and the set partition dual values λ_j for each section $j \in \mathcal{J}$. Let c_o be the fixed reduced resource shift cost $c_o = c_s - \mu$ corresponding to single resource schedule $s \in \Omega$. For each arc $e_{pq} \in \tilde{\mathcal{A}}$, the time-independent reduced cost $\bar{c}_{pq}$ along this arc is given by:

$$\bar{c}_{pq} := \begin{cases} -\lambda_j & \text{if } v_p \in \mathcal{V}_{JC} \cup \{o\} \text{ and } v_q = v^j_{JS}, \\ 0 & \text{otherwise.} \end{cases} \tag{6.14}$$

Notice the reduced cost could be negative for certain arcs, depending on the dual values of the RMP.

The time-dependent cost part of the reduced cost equation (6.12) consist only of the precedence constraint dual variables δ_{ij}. In the axillary graph, they can be modelled as *linear node costs* [44] on the section start and section end vertices. For each vertex $v_p \in \tilde{\mathcal{V}}$, the linear node cost w_p is given by:

$$w_p := \begin{cases} -\delta_{ij} & \text{if } v_p = v^j_{JS} \text{ and } y_{ij} = 1, \\ \delta_{ij} & \text{if } v_p = v^i_{JC} \text{ and } y_{ij} = 1, \\ 0 & \text{otherwise.} \end{cases} \tag{6.15}$$

The time-dependent reduced cost of starting or completing a section at vertex $v \in \mathcal{V}_{JS} \cup \mathcal{V}_{JC}$ is now given by $w_v \cdot T_v$, with T_v the time at which service starts at this vertex v. Also these costs can be negative, depending on the dual values.

The above defined quantities make sure that each single resource schedule $s \in \Omega$, which corresponds to an $\{o, d\}$-path P_s with start-of-service times T_v at each path-vertex, has a total reduced cost equal to reduced cost equation (6.12). The objective of finding the most promising column(s) to be added to the RMP is therefore equal to the objective of finding the lowest reduced cost feasible path(s) from o to d in the axillary graph.

6.3.2 Labels

To solve this special ESPPRC variant with linear node costs, which will we will denote by *ESPPRCNC*, an *labelling algorithm* is used based on the algorithm of Beasley, Chrostofides [8] and its improvement by Feillet [29]. We incorporate ideas based on the solution method of Ioachim [44] to handle the linear node costs and additionally incorporate the correct insertion of the driving breaks.

A labelling algorithm consists of generating/extending *labels*, which represent a certain state of a feasible partial solution, or equivalently a feasible partial path in the auxiliary graph. A label L is situated at the end-vertex of its partial path. Let $\mathcal{L}_p$ be the set of labels situated at vertex $v_p \in \tilde{\mathcal{V}}$. Each label L contains quantities called *label resources* (not to be confused with truck+driver resources) which represent the label's state. The following resources are used: $L = \left(G^c_L, T^{\text{driv}}_L, V^1_L, \ldots, V^{|R|}_L\right) \in \mathcal{L}_p$, which are defined in the next Sections and summarized in Table 6.1.

Cost Function

The first label resource, G^c_L, represents the label costs in terms of its current feasible start of service times at vertex v_p. It is not a quantity, but a set of extreme points. Since the presence of linear node costs, the cost of starting a service at a vertex could depend on time. Ioachim et al.

Table 6.1: Description of the different label resources used.

$L \in \mathcal{L}_p$ Label Resource	Description
$v_p \in \tilde{\mathcal{V}}$	Current vertex
$G_L^c \subset \mathbb{R}^3$	Cost Function Extreme Points in terms of start of service time
$T_L^{\text{driv}} \in \mathbb{R}$	Accumulated Driving Time after last break
$V_L^1, \ldots, V_L^{\lvert R\rvert} \in \mathbb{N}_0$	Last Visited Section for each Route variables
$L' \in \mathcal{L}_i$	Previous Label (pointer) (v_i being previous vertex)

[44] proved that the total cost of a partial path in the auxiliary graph, while visiting multiple vertices with linear node costs, is a continuous piecewise linear and convex function of time.

The following deduction of the cost functions is adapted from Ioachim et al [44]. Let P be a feasible partial path associated to label $L_P \in \mathcal{L}_p$, from vertex o to vertex $v_p \in \tilde{\mathcal{V}}$ in the auxiliary graph. Time-windows $[a_i, b_i]$ along the path can be reduced to eliminate start-of-service times where path P is not feasible. This can be done recursively using the following relations:

$$a_o^P = a_o,\ a_{i+1}^P = \max\left(a_{i+1},\ a_i^P + \bar{t}_{i,i+1} + T^{\text{break}} \cdot \left\lfloor \frac{T_i^{\text{driv}} + t_{i,i+1}}{T_{\max}^{\text{driv}}} \right\rfloor\right) \quad \forall e_{i,i+1} \in P \tag{6.16}$$

$$b_p^P = b_p, \quad b_i^P = \min\left(b_i,\ b_{i+1}^P - \bar{t}_{i,i+1} - T^{\text{break}} \cdot \left\lfloor \frac{T_i^{\text{driv}} + t_{i,i+1}}{T_{\max}^{\text{driv}}} \right\rfloor\right) \quad \forall e_{i,i+1} \in P \tag{6.17}$$

Here T_i^{driv} is the accumulated driving time after the last taken break arriving at vertex v_i. After $T_{\max}^{\text{driv}}$ time units of accumulated driving time (only the accumulated $t_{i,i+1}$ values), a break of T^{break} time units must taken between the travel of two vertices. For a given path P, these driving times are fixed and we assume here that they are already calculated (see equation (6.26)) for all path vertices.

We need to be able to calculate the total time-dependent reduced cost of partial path P. Consider the situation that all linear node costs along the path P are zero: $w_i = 0$, $\forall v_i \in P$. Since the total (reduced) cost of P in this case is independent of the start-of-service times T_i, it does not matter if service at a vertex is started early or late inside the reduced time-windows. When all linear node costs w_i along the path P are positive: $w_i > 0$, $\forall v_i \in P$, it will cost more when service is started later. Therefore it is optimal to start service as early as possible at all vertices : $T_i = a_i^P$, $\forall v_i \in P$. Similarly, when all linear node costs along the path are negative: $w_i < 0$, $\forall v_i \in P$, it will cost more when service is started earlier. Therefore it is optimal to start service as late as possible at all vertices: $T_i = b_i^P$, $\forall v_i \in P$. When there are vertices with positive linear node costs, and vertices with negative linear node costs along path P, which is generally the case for paths in the auxiliary graph, the optimal times T_i are balanced by the different linear node costs and must be determined explicitly.

The cost function $f^P(T)$, for path P starting service at v_p at time $T \in \left[a_p^P, b_p^P\right]$, is defined by the following linear program:

$$f^P(T) = \text{min.} \sum_{v_i \in P} w_i T_i + \sum_{e_{i,i+1} \in P} \bar{c}_{i,i+1} - \mu, \tag{6.18}$$

$$\text{s.t. } T_i + \bar{t}_{i,i+1} + T^{\text{break}} \cdot \left\lfloor \frac{T_i^{\text{driv}} + t_{i,i+1}}{T_{\max}^{\text{driv}}} \right\rfloor \leq T_{i+1} \quad \forall e_{i,i+1} \in P, \tag{6.19}$$

$$T_p = T, \tag{6.20}$$

$$a_i^P \leq T_i \leq b_i^P \qquad \forall v_i \in P. \tag{6.21}$$

Ioachim et al. [44] proved that this function is piecewise linear and convex over $\left[a_p^P, b_p^P\right]$ and that

the function has finite number of linear pieces (segments). Their proof is based on the results of (local) sensitivity analysis on right-hand side changes (in this case equation (6.20)) of linear programs.

By piecewise linearity of the cost function $f^P(T)$, it can be completely represented by its extreme points between its linear pieces (segments). Also by convexity, the piecewise linear function consists of three consecutive parts, each possibly infinite small: a strictly decreasing part, a flat part and a strictly increasing part. There is at least one extreme point (T^*, C^*) where $f^P(T)$ assumes its lowest cost C^*, which must lie in the flat part. Let T^* be the smallest time value at which $f^P(T)$ reaches its minimum of C^*. The cost function at $T < T^*$ is strictly decreasing, while for $T > T^*$ the cost function is strictly increasing after the flat part (linear piece of slope zero starting at T^*).

The key observation of Ioachim et al. [44] is that the increasing segments after T^* (the increasing part) can be neglected when finding the minimum-cost path. Therefore it is useful to define the following modified cost function $g^P(T)$ of path P, for starting service at v_p at time $T \in \left[a_p^P, b_p^P\right]$:

$$g^P(T) = \begin{cases} f^P(T) & \text{if } a_p^P \leq T < T^*, \\ f^P(T^*) & \text{if } T^* \leq T \leq b_p^P. \end{cases} \tag{6.22}$$

This modified cost function is also continuous, piecewise linear and convex, but additionally non-increasing. It takes into account the fact that when service needs to be started at a time $T > T^*$, the cost of starting at the optimal time T^* is used. This property is especially useful when used in dynamic programming/labelling algorithm. When extending a label/partial path to a new vertex, this modified cost function can be extended fairly easy to obtain the new modified cost function at the next vertex.

Since the function $g^P(T)$ is completely described by its extreme points, only these are stored in a label. The extreme points of the cost function $g^P(T)$, represented by a vector $\{T, C, m\}$, are stored inside the set G_L^c, sorted on times T. An extreme point contains: T the start-of-service time, C the accumulated cost for starting service at time T at the current vertex and m the slope of the cost function after this point ($m = 0$ if it has has no slope after it). This last quantity, the slope m, can be derived from the two other quantities but is used here for ease of notation. The start-of-service times T of the extreme points all lie inside the label vertex time-window: $T \in \left[a_p^P, b_p^P\right]$ and every feasible cost function G_L^c explicitly has its latest extreme point on $T = b_p^P = b_p$. Accumulated cost C can be positive or negative, while the slopes m can only be non-positive since the cost function is non-increasing.

Other resources

The second label resource, T_L^{driv}, represents the accumulated driving time arriving at v_p after the last break. It lies in the half-closed interval $\left[0, T_{\max}^{\text{driv}}\right)$, with $T_{\max}^{\text{driv}}$ representing the maximum accumulated driving time after which a break needs to be planned. Finally, the integer label resources V_L^r for each route $r \in R$ represent the position of the last visited section of route r by the path P of label L. They can assume the values $V_L^r \in \{0, 1, \ldots, n_{\text{sect}}^r\}$, with $n_{\text{sect}}^r = \sum_{j \in \mathcal{J}} \rho_j^r$ the total number of sections in route $r \in R$ and $V_L^r = 0$ meaning that none of the sections of route r are visited (yet) by the path. These integer resources are a modification of the binary *vertex visited/unreachable vertex* resources used by Feillet et al. [29]. Because of the special structure of the auxiliary graph, we only need $|R| \ll \left|\tilde{\mathcal{V}} \setminus \{o, d\}\right|$ label resources to ensure that the labels represent elementary (partial) paths.

To summarize the our label structure, we present the following definition:

Definition 6.1 (Labels)**.** A *label* $L_P \in \mathcal{L}_p$ is a feasible partial path P from origin o to vertex v_p, labelled with the following resources: $L_P = \left(G_L^c, T_L^{\text{driv}}, V_L^1, \ldots, V_L^{|R|}\right)$, with G_L^c the sorted set of extreme points $\{T, C, m\}$ of the start-of-service time–accumulated cost function at v_p, T^{driv} the accumulated driving time at v_p after the last break and the resources V_L^r for each route $r \in R$ representing the position of the last visited section of route $r \in R$ ($V_L^r = 0$ if none of the sections of route r are visited along path P).

6.3.3 Label Extension

To extend a label from its current vertex to a new vertex along an arc, first it is needs to be checked whether this extension is actually feasible. Suppose we want to check if label $L' \in \mathcal{L}_i$, representing a feasible partial path from origin o to vertex v_i, can be extended from v_i along $e_{ij} \in \tilde{\mathcal{A}}$ to vertex v_j, resulting in a new label $L \in \mathcal{L}_j$. Let $R(p)$ denote the route $r \in R$ of vertex $v_p \in \tilde{\mathcal{V}}$, $R(p) = \left\{r \in R : v_p \in \mathcal{V}_j, \rho_j^r = 1\right\}$, and $h(p)$ denote the position of the section $j \in \mathcal{J}$ of vertex $v_p \in \mathcal{V}_j$ in its route, with $h(p) = 1$ meaning section j of vertex v_p is the first section of its route. The following conditions must all hold for the extension to be feasible:

$$\min\{T : (T, C, m) \in G_{L'}^c\} + \bar{t}_{ij} + T^{\text{break}} \cdot \left\lfloor \frac{T_{L'}^{\text{driv}} + t_{ij}}{T_{\max}^{\text{driv}}} \right\rfloor \leq b_j, \tag{6.23}$$

$$V_{L'}^{R(j)} < h(j), \text{ only if } v_j \in \mathcal{V}_{JS}. \tag{6.24}$$

Equation (6.23) states that the earliest time service can start at v_j, determined by the left-hand side, cannot exceed latest start-of-service time b_j. The left-hand side consist of taking the earliest start-of-service time T at v_i, which is the first extreme point of $G_{L'}^c$, and adding the static total time $\bar{t}_{ij}$ along arc e_{ij} and adding T^{break} of break time a number of 'times' the accumulated driving time $T_{L'}^{\text{driv}} + t_{ij}$ exceeds the total driving time before breaking of $T_{\max}^{\text{driv}}$. This ensures feasibility of time. Equation (6.24) states that vertex $v_j \in \mathcal{V}_{JS}$ starting a new section can only be visited if its corresponding section has a higher position $h(j)$ in its route $R(j)$ than the position $V_L^{R(j)}$ of the last visited section of the same route. This ensures the precedence relations of the sections in a route: once the h-th section of a route is visited, only the $(h+1)$-th or later sections of that route can be visited next. This also ensures the elementary condition: each section can be visited at most once and therefore, by the structure of the auxiliary graph, each vertex can be visited at most once.

For example, consider a label L associated with the partial path $o \to 1\text{A} \to 2\text{A} \to 1\text{C}$, ending on the section end vertex $v_{JC}^{1\text{C}}$. It will have resources $V_L^1 = 3$, $V_L^2 = 1$ and $V_L^r = 0$ for all other routes $r \in R \setminus \{1, 2\}$. The label L can therefore only be extended to (section start vertices of) section 1D ($h(v_{JS}^{1\text{D}}) = 4 > 3$), 2B ($h(v_{JS}^{2\text{B}}) = 2 > 1$) or later subsequent sections of those routes. The extension to sections of other routes are not restricted by these label resources.

When the label extension is considered to be feasible, the label resources are updated as follows:

$$G_L^c = \textsc{ExtendCostFunction}\left(G_{L'}^c, \bar{t}_{ij} + T^{\text{break}} \cdot \left\lfloor \frac{T_{L'}^{\text{driv}} + t_{ij}}{T_{\text{break}}^{\text{driv}}} \right\rfloor, \bar{c}_{ij}, w_j, a_j, b_j\right), \tag{6.25}$$

$$T_L^{\text{driv}} = T_{L'}^{\text{driv}} + t_{ij} - T_{\max}^{\text{driv}} \cdot \left\lfloor \frac{T_{L'}^{\text{driv}} + t_{ij}}{T_{\max}^{\text{driv}}} \right\rfloor, \tag{6.26}$$

$$V_L^{R(j)} = h(j), \tag{6.27}$$

$$V_L^r = V_{L'}^r, \qquad \forall r \in R \setminus \{R(j)\}. \tag{6.28}$$

Algorithm 6.1 Extending a cost function extreme point set.

procedure ExtendCostFunction(G'^c, ΔT, ΔC, w, a, b)
  Input: Prev. cost extreme points G'^c, static additional time ΔT and cost ΔC, linear node cost w, time-window $[a, b]$.
  Output: Extension feasibility $feas$, new cost extreme points G^c.
  $feas \leftarrow$ `false`
  $F^c, F'^c, G^c \leftarrow \varnothing$
  for all $(T, C, m) \in G'^c$ **do**
    if $(T + \Delta T) \geq b$ **then**
      break
    else if $(T + \Delta T) \geq a$ **then**
      $F^c \leftarrow \{(T + \Delta T,\, C + \Delta C + w \cdot (T + \Delta T)\,, \min\{0,\, m + w\})\}$
    else
      $F^c \leftarrow F^c \cup \{(T + \Delta T,\, C + \Delta C + w \cdot (T + \Delta T)\,, \min\{0,\, m + w\})\}$
    if $m + w \geq 0$ **then**
      break
  if $F^c = \varnothing$ **then**
    return
  if $T_1 < a$ **then**
    $n \leftarrow |F^c|$
    $F'^c \leftarrow \{(T_2, C_2, m_2)\,, (T_3, C_3, m_3)\,, \ldots, (T_n, C_n, m_n)\} \subset F^c$
    $G^c \leftarrow \{(a, C_1 + m_1 \cdot (a - T_1)\,, m_1)\} \cup F'^c$
  else
    $G^c \leftarrow F^c$
  if $m_n < 0$ **then**
    $G^c \leftarrow G^c \cup \{(b, C_n + m_n \cdot (b - T_n)\,, 0)\}$
  $feas \leftarrow$ `true`

The set of cost function extreme points $G^c_{L'}$ is extended to G^c_L using the procedure ExtendCostFunction based on Ioachim et al. [44], described in algorithm 6.1. The static additional time $\Delta T = \bar{t}_{ij} + T^{\text{break}} \cdot \left\lfloor \frac{T^{\text{driv}}_{L'} + t_{ij}}{T^{\text{driv}}_{\max}} \right\rfloor$ and cost $\Delta C = \bar{c}_{ij}$ are added to the extreme points $\{T, C, m\}$, as well as the time-dependent linear node costs $w_j \cdot (T + \Delta T)$ and the slope is changed to $\min\{0, m + w_j\}$. At maximum one extreme point with new time $(T + \Delta T) \leq a_j$ before the time-window is kept, with other extreme points lying inside the time-window: $a_j < (T + \Delta T) < b_j$. Once a new slope greater or equal to zero is found or the new times lie beyond the time-window $(T + \Delta T) \geq b_j$, further (later) extreme points are discarded, since they are not useful. If the first extreme point lies before the start of the time-window, it is linearly interpolated to the start of the time-window $T_1 = a_j$. If the last extreme point $(T_n, C_n, m_n) \in F'^c$ (with a time $T_n < b_j$) has slope $m_n < 0$, an extra extreme point on b_j with zero slope is added, resulting in the new cost function extreme point set G^c_L. This latter situation is the only way the number of extreme points of a cost function set can increase when extending. In this case, at most one extreme point is added to the cost function set.

In equation (6.26), the new driving time after last break is calculated. Also, the last visited section variable for the vertex's current route $R(j)$ is updated in equation (6.27), while other last visited section variables of other routes remain the same. To enlighten the extention of a label, including updating the cost function (extreme points), we consider a brief example.

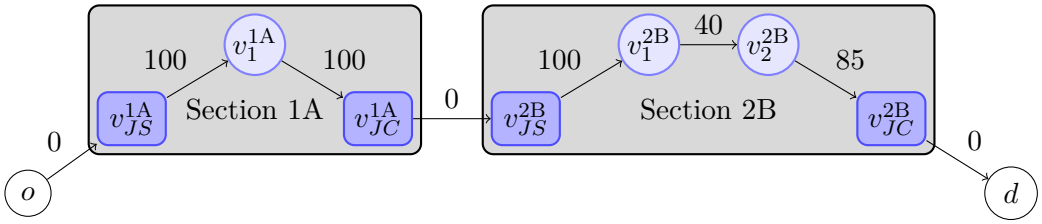


Figure 6.2: Schematic overview of path P used in the label extension example.

Example

Let us consider the following path P visiting two sections 1A and 2B in the auxiliary graph:

$$P = o \to v_{JS}^{1A} \to v_1^{1A} \to v_{JC}^{1A} \to v_{JS}^{2B} \to v_1^{2B} \to v_2^{2B} \to v_{JC}^{2B} \to d.$$

See Figure 6.2 for a schemetic overview of this path P in the auxiliary graph. We will use the following vertex indices for ease of notation:

$$P = o \to v_1 \to v_2 \to v_3 \to v_4 \to v_5 \to v_6 \to v_7 \to d.$$

Let index $i \in \{o, 1, 2, \ldots, 7, d\}$, with $i + 1 = 1$ for $i = o$ and $i + 1 = d$ for $i = 7$, also for ease of notation.

Let the driving times $t_{i,i+1}$ be given by: $t_{o1} = t_{12} = t_{45} = 100$, $t_{34} = t_{7d} = 0$, $t_{56} = 40$, $t_{67} = 85$. Let the service times T_i^{serv} be 15 (minutes) on only the customer visit vertices: $T_i^{\text{serv}} = 15$ $\forall v_i \in \mathcal{V}_C$ $(i \in \{2, 5, 6\})$, $T_i^{\text{serv}} = 0$ $\forall v_i \in P \setminus \mathcal{V}_C$ $(i \in \{o, 1, 3, 4, 7, d\})$. The total times $\bar{t}_{i,i+1}$ can now be determined as follows: $\bar{t}_{o1} = \bar{t}_{45} = 100$, $\bar{t}_{12} = 100 + 15 = 115$, $\bar{t}_{34} = \bar{t}_{7d} = 0$, $\bar{t}_{56} = 40 + 15 = 55$, $\bar{t}_{67} = 85 + 15 = 100$. Let $T_{\max}^{\text{driv}} = 270$ (minutes) and $T^{\text{break}} = 45$ (minutes).

Let the time-windows $[a_i, b_i]$ be given by: $[a_o, b_o] = [a_d, b_d] = [0, 1000]$.
Time-windows of section 1A: $[a_1, b_1] = [0, 100]$, $[a_2, b_2] = [100, 200]$, $[a_3, b_3] = [250, 350]$.
Time-windows of section 2B: $[a_4, b_4] = [300, 400]$, $[a_5, b_5] = [400, 600]$, $[a_6, b_6] = [455, 655]$, $[a_7, b_7] = [540, 740]$.

Let the fixed resource cost be $c_s = c_s^{\text{sh}} = c_o = 200$. Let the set partition shadow prices λ_j for the sections 1A and 2B be given by: $\lambda_{1A} = 50, \lambda_{2B} = 50$.
The reduced costs $\bar{c}_{i,i+1}$ can now be determined (using equation (6.14)) as follows: $\bar{c}_{o1} = -50$, $\bar{c}_{34} = -50$, $\bar{c}_{i,i+1} = 0$ $\forall 1 \leq i \leq 7, i \neq 3$.

Let the precedence constraint shadow prices δ_{ij} be given by: $\delta_{1AB} = 1$, $\delta_{2AB} = 2$. The linear node costs w_i can now be determined (using equation (6.15)) as follows: $w_1 = w_{JS}^{1A} = 0, w_3 = w_{JC}^{1A} = 1, w_4 = w_{JS}^{2B} = -2, w_7 = w_{JC}^{2B} = 0.5$ and $w_i = 0$ for all $i \in \{o, 2, 5, 6, d\}$. Notice that $w_1 = 0$ since corresponding vertex v_{JS}^{1A} has no predecessor in the route and therefore this vertex is not involved in a precedence relation.

We want to calculate the optimal start-of-service times T_i of path P and total (reduced) cost $\tilde{c}_P = f^P(T^*)$, using the earlier proposed method of extending labels. The initial label L_o is given by $L_o = \left(G_o^c, T_o^{\text{driv}}, V_o^1, V_o^2\right) = \left(\{(0, 200, 0)\}, 0, 0, 0\right)$. To calculate label $L_i = \left(G_i^c, T_i^{\text{driv}}, V_i^1, V_i^2\right)$ of vertex $v_i \in P$, the label extension equations (6.25)–(6.27) with the extend cost function algorithm 6.1 are successively applied. The resulting labels with their resource values are shown in Table 6.2. Also the resulting cost functions are plotted in Figure 6.3.

Notice a break was inserted between v_4 and v_5, resulting in 45 units of additional time along the arc $e_{4,5}$. Also notice that the label cost function sets G_i^c generally do not explicitly contain an extreme point on the right-side of the time-window $T = b_i$. Only in the cost function set of

L_i	v_i	G_L^c	T_L^{driv}	V_L^1	V_L^2
L_o	o	$(0, 200, 0)$	0	0	0
L_1	v_{JS}^{1A}	$(0, 150, 0)$	0	1	0
L_2	v_1^{1A}	$(100, 150, 0)$	100	1	0
L_3	v_{JC}^{1A}	$(250, 400, 0)$	200	1	0
L_4	v_{JS}^{2B}	$(300, -250, -2), (400, -450, 0)$	200	1	2
L_5	v_1^{2B}	$(445, -250, -2), (545, -450, 0)$	30	1	2
L_6	v_2^{2B}	$(500, -250, -2), (600, -450, 0)$	70	1	2
L_7	v_{JC}^{2B}	$(600, 50, -1.5), (700, -100, 0)$	155	1	2
L_d	d	$(600, 50, -1.5), (700, -100, 0)$	155	1	2

Table 6.2: Labels L_i with its resource values by successively extending L_o along path P of the example.

label L_4, an extreme point lies on $T = b_i$. This due to the fact that for that cost function, the optimal time-value $T^* = b_i$.

In Section 6.3.5, we will return to this example to show how to retrieve a full solution or equivalently a new column for the RMP using the above generated labels.

Observations

Some interesting statements about the label extension and the cost function sets can be derived, some of which are inspired by Ioachim et al [44]. For instance, in our model we can formulate an upper bound on the number of extreme points in the cost function sets.

First, we can derive some important properties of the label cost function sets.

Lemma 6.2 (Cost function set optimal extreme point)**.** Given a feasible label $L \in \mathcal{L}_i$ at $v_i \in \tilde{\mathcal{V}}$. The earliest optimal (of lowest cost) feasible start-of-service time T_i^* of the label is equal to $T_{|G_L^c|}$, with $\left(T_{|G_L^c|}, C_{|G_L^c|}, m_{|G_L^c|}\right) \in G_L^c$ **the last** extreme point. Furthermore, this last point has zero slope: $m_{|G_L^c|} = 0$.

Proof. The lemma is proven by induction. The cost function set G_o^c of the initial label L_o on origin vertex o contains only one extreme point: $G_o^c = (0,\, c_s - \mu,\, 0)$ with zero slope. Therefore the earliest optimal feasible start-of-service time T_o^* is indeed equal to the time of the last extreme point: $T_o^* = 0$. Now suppose we have a feasible label $L' \in \mathcal{L}_p$ at v_p which is extended to v_q, resulting in a feasible label $L \in \mathcal{L}_q$. Inside the for-loop of the ExtendCostFunction procedure (algorithm 6.1), extreme points of $G_{L'}^c$ are extended to F_L^c until the slope of the new cost function is non-negative. Suppose this occurs, then last slope is set to zero: $m_{|F^c|} = \min\{0, m + w\} = 0$, since $m + w \geq 0$. All previous slopes of the new cost function are negative, so this last extreme point will indeed be the earliest optimal feasible start-of-service time. This does not change when F_L^c is converted to G_L^c. The only way the for-loop can end with the last extreme point of the cost function set having a non-zero slope is when all extreme points of $G_{L'}^c$ are extended to F_L^c, and linear node cost w is negative. Then the last slope will be negative: $m_{|F_L^c|} = \min\{0, m + w\} < 0$, since $w < 0$ and $m = 0$, which is the last slope of $G_{L'}^c$. In this case however, at the end of the ExtendCostFunction procedure an additional extreme point is appended to the cost function set: $\left(T_{|G_L^c|}, C_{|G_L^c|}, m_{|G_L^c|}\right) = \left(b,\, C_{|F_L^c|} + m_{|F_L^c|} \cdot \left(b - T_{|F_L^c|}\right),\, 0\right)$. This last extreme point is the

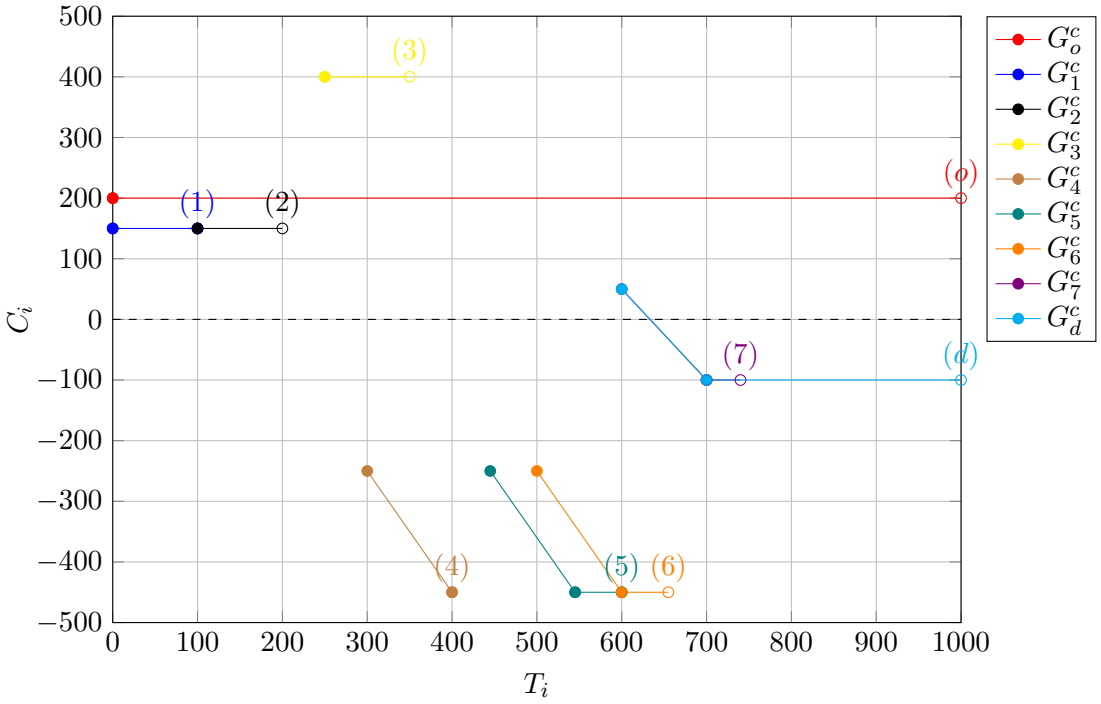


Figure 6.3: Cost functions corresponding to extreme point set G_i^c inside the labels L_i extended along path P of the example. An open marker indicates the end of time-window b_i, which is not explicitly included in the cost function extreme points set.

earliest optimal feasible start-of-service time, since its cost $C_{|G_L^c|} = C_{|F^c|} + m_{|F_L^c|} \cdot \left(b - T_{|F_L^c|}\right)$ is lower than $C_{|F_L^c|}$ and it has a zero slope. This proves the lemma that the earliest optimal feasible start-of-service time of the label always corresponds to the last extreme point of the cost function set. Also, this last point always has zero slope. □

Vertex with non-zero linear node cost play a special role in the label extension. Only these vertices have the potential to change the slopes (shape) of the cost function set. We will define these vertices to be *active*.

Definition 6.3 (Active section start/end vertices)**.** A section start or completion vertex $v_i \in \mathcal{V}_{JS} \cup \mathcal{V}_{JC}$ is *active* if it has non-zero linear node cost: $w_i \neq 0$.

Using this definition and the above lemma, we can formulate an upper bound on the number of extreme points in the cost function sets.

Theorem 6.4 (Number of extreme points of G_L^c)**.** The number of extreme points in set G_L^c of label $L_p \in \mathcal{L}_j$ representing a feasible partial path P from origin o to vertex $v_p \in \tilde{\mathcal{V}}$ is **at most** $\hat{n}_P^{JS} + 1$, with $\hat{n}_P^{JS}$ the number of *active* section start vertices visited along path P: $\hat{n}_P^{JS} = |\{v_i \in \mathcal{V}_{JS} \cap P : w_i \neq 0\}|$.

Proof. By examining the proof of Lemma 6.2, we see that extending a label $L' \in \mathcal{L}_p$ to $L \in \mathcal{L}_q$ generally reduces the number of extreme points in the cost function set, due to the (new) time-window $[a_q, b_q]$ on v_q and possible slope changes when linear node cost $w_q > 0$. However, only when all extreme points are feasibly extended to inside the time-window $[a_q, b_q]$ and the linear node cost $w_q < 0$, an additional extreme point is appended to the cost function set, increasing the number of extreme points by 1. The initial label L_o at origin vertex o starts with one extreme point, and only active section start vertices have negative linear node cost (active section end

vertices have positive linear node cost, see equation (6.15)). Therefore, the number of extreme points of the cost function set of label $L_p \in \mathcal{L}_j$ representing a feasible partial path P is at most $1 + \hat{n}_P^{JS}$, with $\hat{n}_P^{JS} = |\{v_i \in \mathcal{V}_{JS} \cap P : w_i \neq 0\}|$ the number of active section start vertices visited along path P. □

This upper bound on the number of extreme point is even tight. Consider the case when the (reduced) time-windows are very wide and therefore do not discard any extreme points when extending the labels. Also let the path P only consist of vertices with non-positive linear node cost. The number of extreme points of cost function set G_L^c of label L_p in this case will be exactly $1 + \hat{n}_P^{JS}$. If the (reduced) time-windows are however restrictive, the number of extreme points will be less due to infeasible start-of-service times.

A direct, but important consequence of the above theorem is the following corollary on feasible paths that do not contain active section start vertices.

Corollary 6.5. If a feasible path P contains no active section start vertices: $\hat{n}_P^{JS} = 0$, then all feasible labels along path P will have cost function sets containing only a single extreme point.

Proof. One can simply use $\hat{n}_P^{JS} = 0$ in theorem 6.4 to get $|G_L^c| \leq 1$ for label L_p. Since all $o - v_i$ subpaths $\forall v_i \in P$ neither contain active section start vertices, by the theorem also their corresponding labels $L \in \mathcal{L}_i$ contain only a single extreme point: $|G_L^c| \leq 1$ for all labels L along P. Since path P is feasible, all labels L along P must have at least one extreme point in their cost function set: $|G_L^c| \geq 1$. Combining these statements results in $|G_L^c| = 1$ for each label L along P. □

The corollary will be important when using the exact pricing method inside the column generation procedure, which will be discussed in Sections 6.3.5 and 6.3.6.

6.3.4 Dominance Relations

Although not a lot of labels are generated when extending a label along a single path, the full DP labelling algorithm needs to consider a possibly exponentially growing number of paths and can therefore possibly generate an exponentially growing number of labels. To help making the algorithm computationally tractable for large instances, one can reduce the number of labels by including so-called *dominance relations*. This is a very common method used in DP labelling algorithms posed in literature. Both Ioachim et al. [44] and Feillet et al. [29] state effective dominance relations for their labelling algorithms. However, we will see that we cannot simply use their relations directly in our case.

The concept of dominance is actually quite simple. Suppose we have two labels $L_p, L_q \in \mathcal{L}_j$, both residing on vertex $v_j \in \tilde{\mathcal{V}}$. If we could conclude that label L_p is *always strictly better* than label L_q, then there should be no point in keeping label L_q in memory and therefore calculating all feasible extensions of both L_p and L_q. Calculating all feasible extensions of L_p (as well as all other extensions) will still be sufficient to *guarantee* us finding an optimal $o - d$-path. If this is the case, we say that label L_p *dominates* label L_q. In a DP labelling algorithm, only non-dominated labels are maintained and at each label extension it must be checked if some labels are dominated and thus can be discarded.

We can adapt the dominance relation of Feillet et al. [29], which roughly states that label $L_p = (U_1^p, U_2^p, \ldots, U_l^p)$, with general label resources U_i and l the total number of label resources, dominates $L_q = (U_1^q, U_2^q, \ldots, U_l^q)$ on v_j if for each general label resource U_i it holds that $U_i^p \leq U_i^q$, with at least one inequality strict. When applying this relation to our case, note that a

comparison of the cost function sets G_p^c versus G_q^c is needed. We use the following notation: $G_p^c \preceq G_q^c$ to denote that cost function g_p^c associated with G_p^c lies at or below cost function g_q^c associated with G_q^c. To be precise: $G_p^c \preceq G_q^c$ means $g_p^c(T) \leq g_q^c(T)$ for all $T \in \left[T_q^{\min}, b_j\right]$, with $T_q^{\min} = \min\left\{T : (T, C, m) \in G_q^c\right\}$. This can be done by calculating the values of both cost functions at each unique time point in the interval: for all $T \in \left\{T : (T, C, m) \in G_p^c, T \geq T_q^{\min}\right\} \cup \left\{T : (T, C, m) \in G_q^c\right\}$. This procedure is related to the dominance relation used by Ioachim et al. [44] and quite similar to the comparison of cost functions in a more recent application by Tagmouti et al. [69]. In the latter work, time-dependent service cost in their problem causes labels to have time-dependent cost functions and a similar comparison of cost functions is used as dominance relation.

One further observation concerning the application of the Feillet et al. dominance relation to our label resources, is that the driving time resource T_L^{driv} should actually be taken care of differently. Because we only allow breaks to be planned as late as possible, when forced, two labels L_p and L_q cannot be compared if their driving time after break differs: $T_p^{\text{driv}} \neq T_q^{\text{driv}}$. Even if the driving time after break of label L^p is lower than that of L^q: $T_p^{\text{driv}} < T_q^{\text{driv}}$, still label L^q could be better (so not dominated by L_p). For instance, extending label L_q would at some point cause planning a break early which fits perfectly, while extending label L_p the same way causes planning the break later which results in violating a time-window restriction. Therefore, we can only check if label L_p dominates label L_q on v_j if $T_p^{\text{driv}} = T_q^{\text{driv}}$. This is similar to the reasoning of Drexl et al. [23] on dominance relations of labelling algorithms for the ESPPRC with EU drivers' rules.

The observations are summarized by the following proposition.

Proposition 6.6 (Dominance Relation)**.** Let two labels $L_p, L_q \in \mathcal{L}_j$ reside on the same vertex $v_j \in \mathcal{V} \setminus \{d\}$. Label L_p *dominates* label L_q **if** $G_p^c \preceq G_q^c$, $T_p^{\text{driv}} = T_q^{\text{driv}}$ and $V_p^r \leq V_q^r$, $\forall r \in R$, with at least one inequality being strict.

We will not formally prove this proposition, since its proof is already roughly sketched by our above observations. It turns out, similarly to observation made in [23], that the equality $T_p^{\text{driv}} = T_q^{\text{driv}}$ extremely weakens the dominance relation: almost none of the generated labels will be dominated by another until they are extended to destination vertex d. Note that on destination vertex d, the only relevant resource is total reduced cost, so comparison can be done easily and here, suddenly, a lot of labels are dominated. The reason of the labels not being dominated earlier (not at vertex d) is because the conditions of proposition 6.6 are rarely met. The condition $T_p^{\text{driv}} = T_q^{\text{driv}}$ means that the total driving time of both corresponding paths P and Q are equal up to a multiple of the maximum driving time before a break: $\sum_{e_{i,i+1} \in P} t_{i,i+1} = T_{\max}^{\text{driv}} \cdot m + \sum_{e_{i,i+1} \in Q} t_{i,i+1}$, for some integer $m \in \mathbb{Z}$. If also condition $V_p^r \leq V_q^r$, $\forall r \in R$ is imposed, besides in case of extremely specific driving time values, in general this only allows paths P and Q of labels L_p and L_q on v_j to have visited **exactly the same sections**, although not in the same order. Such permutations can only happen if not all sections come from the same route, since a single route only has one feasible visiting sequence of sections due to the precedence constraints.

For instance, consider two paths P and Q ending on vertex $v_j = v_{JS}^{3\text{A}}$, which both have visited only sections 1A and 2B previously: $P = o \to \mathcal{V}^{1\text{A}} \to \mathcal{V}^{2\text{B}} \to v_{JS}^{3\text{A}}$ and $Q = o \to \mathcal{V}^{2\text{B}} \to \mathcal{V}^{1\text{A}} \to v_{JS}^{3\text{A}}$. In this case $\sum_{e_{i,i+1} \in P} t_{i,i+1} = T_{\max}^{\text{driv}} \cdot 0 + \sum_{e_{i,i+1} \in Q} t_{i,i+1}$ (total driving times are equal), so $T_p^{\text{driv}} = T_q^{\text{driv}}$ is met. Since the visited sections are equal, $V_p^r = V_q^r$, $\forall r \in R$ is met as well. So in this case, the cost functions of both labels can be compared and dominance can be determined.

In computational experiments however, we found a very low number of such permutations that

were actually both feasible. The dominance checking slowed down the labelling algorithm to such degree that we decided to exclude it from the algorithm. We still mention it here to be complete and to provide some suggestions on overcoming this issue.

Inspired by the labelling algorithms of Drexl et al. [23], a possible solution is to use inequality $T_p^{\text{driv}} \leq T_q^{\text{driv}}$ in the dominance relation, but effectively making the algorithm heuristic instead of exact, since there is a slight possibility of removing labels corresponding to the optimal solution. Another idea is to also allow *unforced* breaks, so breaking may be done earlier than needed. Then also $T_p^{\text{driv}} \leq T_q^{\text{driv}}$ can be used in the dominance relation, but this changes the problem characteristics and also increases the number of labels tremendously. We refer the reader to Drexl et al. [23], who discuss both ideas in greater detail.

6.3.5 DP Labelling Algorithm

To calculate the lowest cost paths from vertex o to d in the auxiliary graph of a resource kind, a labelling algorithm based on the algorithm of Feillet et al. [29] is used. Using similar notation as in [29], the labelling algorithm is described in algorithm 6.2.

Algorithm 6.2 Labelling Algorithm

procedure ESPPTWNC($\tilde{\mathcal{G}}$)
  Input: graph $\tilde{\mathcal{G}} = \left(\tilde{\mathcal{V}}, \tilde{\mathcal{A}}\right)$.
  Output: feasible label sets $\left\{\mathcal{L}_i : v_i \in \tilde{\mathcal{V}}\right\}$.
  $\mathcal{L}_o \leftarrow \{(\{(a_o, c_o, 0)\}, 0, 0, \ldots, 0)\}$
  for all $v_i \in \tilde{\mathcal{V}} \setminus \{o\}$ **do**
    $\mathcal{L}_i \leftarrow \varnothing$
  $E = \{o\}$
  repeat
    choose $v_i \in E$
    for all $v_j \in$ SUCCESSORS(v_i) **do**
      $L_{ij} \leftarrow \varnothing$
      for all $L = \left(G_L^c, T_L^{\text{driv}}, V_L^1, \ldots, V_L^{|R|}\right) \in \mathcal{L}_i$ **do**
        if $v_j \notin \mathcal{V}_{JS}$ **or** $V_L^{R(j)} < h(j)$ **then**
          if $T_L^{\min} + \bar{t}_{ij} + T^{\text{break}} \cdot \left\lfloor \frac{T_{L'}^{\text{driv}} + t_{ij}}{T_{\max}^{\text{driv}}} \right\rfloor \leq b_j$ **then**
            $L_{ij} \leftarrow L_{ij} \cup \{\text{EXTENDLABEL}(L, v_j)\}$
      $\mathcal{L}_j \leftarrow \text{EFF}(\mathcal{L}_j \cup L_{ij})$
      if $\mathcal{L}_j$ has changed **then**
        $E \leftarrow E \cup \{v_j\}$
    $E \leftarrow E \setminus \{v_i\}$
  until $E = \varnothing$

As earlier, we use $\mathcal{L}_i$ to denote the set of labels residing on vertex $v_i \in \tilde{\mathcal{V}}$. We use E as a list of nodes which need to be treated, the procedure SUCCESSORS(v_i) to give the set of successors v_j of vertex v_i, $\left\{v_j \in \tilde{\mathcal{V}} : e_{ij} \in \tilde{\mathcal{A}}\right\}$. Some quick pre-checks are done in this procedure to ensure some infeasible extensions can be discarded quickly. The notation $T_L^{\min} = \min\{T : (T, C, m) \in G_L^c\}$ is used. Procedure EXTENDLABEL(L_i, v_j) is used to denote the extension of a label L at vertex v_i to vertex v_j by using the label extension equations (6.25)–(6.28). The set L_{ij} is used to maintain newly extended labels from vertex v_i to vertex v_j. Finally, the procedure EFF($\mathcal{L}_j \cup L_{ij}$) is suppose to keep only non-dominated labels in the list of labels $\mathcal{L}_j$ at vertex v_j by using the

dominance relation in proposition 6.6. As described earlier, the inclusion of the dominance related checks slowed down the algorithm without actually reducing the number of labels significantly. For this reason, we omitted the dominance related checks in our algorithm, but instead we made sure that each label is extended to a specific successor only once.

Extracting Columns

Once the labelling Algorithm 6.2 is completed, labels $L \in \mathcal{L}_d$ residing on destination vertex d form the set of feasible single-resource schedule solutions for a particular resource kind. By Lemma 6.2, we can find the optimal completion time T^* with corresponding total cost C^* of the path P corresponding to label L by simply looking at the last extreme point $(T^*, C^*, 0)$ of cost function set G^c_L. Since C^* represents the optimal reduced cost of a possible column, it will improve the relaxed master problem only if it has negative reduced cost: $C^* < 0$. If $C^* \geq 0$, the label and its associated path can therefore be discarded. Note that this only holds for the relaxed master problem (LP) and not in general for the non-relaxed master problem (ILP), which we will discuss in Section 6.5.

Solutions in the form of columns, consisting primarily of the start-of-service times T_i at the vertices $v_i \in P$, can be constructed by backwards visiting the generated labels of feasible path P. Let T^*_i denote the earliest time where the label cost function set G^c_i attains its minimum cost for each $v_i \in P$. By construction, this is the time of the last point of the label cost function set. Given these minimum cost times and the optimal completion time $T_d = T^*_d$, we can use a backwards recursion relation to determine the optimal feasible solution in terms of start-of-service times T_i:

$$T_d = T^*_d, \;\; T_i = \min\left(T^*_i,\, T_{i+1} - \bar{t}_{i,i+1} - T^{\text{break}} \cdot \left\lfloor \frac{T^{\text{driv}}_i + t_{i,i+1}}{T^{\text{driv}}_{\max}} \right\rfloor\right) \;\; \forall e_{i,i+1} \in P \tag{6.29}$$

This recursion relation is similar to the relation used earlier to determine the reduced time windows, equation (6.17).

We will demonstrate this procedure by generating the optimal start-of-service times of the labels generated in the example of Aection 6.3.3. The optimal completion time is: $T_d = T^*_d = 700$ and the total reduced cost is: $C^* = -100$. Since this completion time has negative total reduced cost, it will improve the relaxed master problem when added as column. Using the above recursion relation to the labels generated in the example gives the following solution to the start-of-service times (calculated from right-to-left) of path P:

$T_o = 0$, $T_1 = 0$, $T_2 = 100$, $T_3 = 250$, $T_4 = 400$, $T_5 = 545$, $T_6 = 600$, $T_7 = 700$, $T_d = 700$.

Notice the solution contains unforced waiting time between T_3 and T_4, which is a consequence of the *active* precedence constraint shadow price $\delta_{2\text{AB}} = 2 > 0$. Although normally undesirable in a single resource shift, in this case it will actually improve the solution of the relaxed master problem, which is indicated by the negative reduced cost. Notice that completing the path as early as possible, at $T_d = 600$, actually results in a solution of positive reduced cost $C = 50$. This solution will not improve the relaxed master problem.

From the start-of-service times T_i for $v_i \in P$, the section start times S_{js} and section completion times C_{js} for a new column s can be determined directly: $S_{js} = T_i$ for every section $j \in \mathcal{J}$ such that $v_i = v^j_{JS} \in P$ and $C_{js} = T_i$ for every $j \in \mathcal{J}$ such that $v_i = v^j_{JC} \in P$. For sections that are not visited in the path: $S_{js} = C_{js} = 0$. The columns generated this way are added to the restricted master problem, which is solved after all pricing problems are solved.

To be precise, column s, representing path P, is added to the master problem with the following

quantities:

$$a_{js} = \begin{cases} 1 & \text{if path } P \text{ visits section } j \ (\mathcal{V}_j \subset V(P)), \\ 0 & \text{otherwise.} \end{cases} \tag{6.30}$$

$$b_{ks} = \begin{cases} 1 & \text{if path } P \text{ is (in graph } \tilde{\mathcal{G}}_k) \text{ for resource shift kind } k, \\ 0 & \text{otherwise.} \end{cases} \tag{6.31}$$

$$S_{js} = \begin{cases} T_i & \text{if } v_i = v^j_{JS} \in P, \\ 0 & \text{otherwise.} \end{cases} \tag{6.32}$$

$$C_{js} = \begin{cases} T_i & \text{if } v_i = v^j_{JC} \in P, \\ 0 & \text{otherwise.} \end{cases} \tag{6.33}$$

6.3.6 Column Improvement Procedure

The complete DP labelling algorithm 6.2 is used to solve all pricing problems at each iteration of the column generation procedure. Generally, the restricted (relaxed) master problem is solved, providing a dual solution or shadow prices, which indicate what to look for in new columns. New columns are found by solving the pricing problems related to each resource kind. After all new columns are added, the master problem is again solved to provide new shadow prices. As soon as the pricing problems fail to find (new) columns, the relaxed master problem is solved to optimality. Solving the pricing problems every iteration however, can take a considerable amount of time to solve, especially if no dominance relation is used. There are some critical observations that can seriously reduces this computation time.

Lemma 6.7 (Shadow prices and path feasibility)**. None** of the shadow prices λ_j, δ_{ij}, and μ_k affect the feasibility of any path P in auxiliary graph $\tilde{\mathcal{G}}_k$.

Proof. The set partition and resource availability shadow prices λ_j and μ_k directly affect the cost of a path as constants. These shadow prices do not influence the start-of-service times T_i, so they do not modify the feasibility of any path P. The precedence constraints shadow prices δ_{ij}, which directly affect the linear node costs, influence only the optimal label start-of-service times T^*_i used in equation 6.29 to derive the path's optimal start-of-service times T_i. However, notice that the value of the earliest start-of-service time $T^{\min}_L$ of each label L, which are the first extreme point of the label cost function set G^c_L, is actually unaffected by any linear node cost along the path. Since the time feasibility criterion of extending a partial path, equation (6.23) only concerns the earliest start-of-service time, the feasibility of a complete path P from o to d is thus unaffected by the linear node costs. We conclude that the feasibility is unaffected by any of the shadow prices, since they only influence costs and the optimal start-of-service times. □

The above lemma is a very useful observation in the whole column generation method. Since the only difference between iterations of the column generation method is the value of the shadow prices, paths remain feasible in all iterations, although their times may change. In other words, if a certain feasible o–d path P in graph $\tilde{\mathcal{G}}_k$ is generated by the labelling algorithm at some iteration, then by the above lemma 6.7, it will also be feasible in all other iterations of the column generation method. Only the total (reduced) costs and optimal start-of-service times may change.

This leads to the following theorem.

Theorem 6.8. Using the dummy columns as initialization of the RMP, described in Section 6.1.2, the first iteration of the pricing problems will generate **all** feasible paths with optimal start-of-service times as early as possible. Subsequent iterations of the pricing problem only requires us to update the start-of-service times for **some** of the already generated feasible paths, delaying some of the sections.

Proof. First, notice that using the dummy column initialization the resulting shadow prices can already be deduced: $\lambda_j = c_{\text{dum}}$ for all sections $J_j \in \mathcal{J}$, $\delta_{ij} = 0$ for all sections $J_i \to J_j$ and $\mu_k = 0$ for all resource kinds $k \in \mathcal{K}$. Using Corollary 6.5, we see that the cost extreme point set of any feasible label now generated by the labelling algorithm will only contain a single extreme point, which will be as early as possible. Each feasible o–d path P visiting a number of $n \in \mathbb{N}$ sections will in this first iteration have a total reduced cost $\tilde{c}_s = c_s - n \cdot c_{\text{dum}} < 0$, since $c_{\text{dum}} \gg c_s$. Thus, any feasible o–d path P has negative reduced cost and is added to the master problem with sections start and completion times as early as possible. Notice that this does not hold if a dominance relation is used, since some labels may be dominated by others and thus not all feasible paths are generated. However, we focus here on the situation in which no dominance is used.

In subsequent iterations of the column generation, since all feasible paths with start/completion times as early as possible are already present, the only columns that will improve the RMP are columns of some feasible path P with some start/completion times later than as early as possible. By using theorem 6.4, we see this only happens if somewhere along P the number of extreme points is greater than one, so the optimal start-of-service time is not as early as possible. The theorem states that this can only happen if some section start vertex in P is active. Only paths that contain at least one section start vertex need to be recalculated to solve the complete pricing problem in that iteration. □

The above theorem with is proof leads to a potentially efficient column generation method, which is sketched in algorithm 6.3.

Procedure ImproveColumns examines the set of all feasible paths $\mathcal{P}_k$ of resource shift kind k (which is calculated in the first iteration pricing problem), and looks for paths with at least one active section start vertex under the current set partition shadow prices δ. Only along such a path $P \in \mathcal{P}_k$, labels are extended. If the resulting path has negative reduced cost, new section start/completion times are determined and a new column is added to the restricted (relaxed) master problem. This procedure is much faster than the complete labelling procedure ESPPRCNC, since only labels are generated for already determined feasible paths which have a potential to improve the RMP by visiting a section later. Usually the number of feasible paths with at least one active section start vertex is very low.

The bottleneck of this algorithm is usually the full labelling procedure ESPPRCNC, which without dominance relation generates all feasible paths of all resource kinds with sections starting as early as possible. Although we have shown that this procedure only has to be done in the first pricing iteration, still this procedure is responsible for the majority of the total computation time of the complete column generation method.

6.3.7 Loading Time Issues

Although the RMP handles the precedence constraints and in the pricing subproblem the labelling algorithm ensures no cycles can occur in the dependency graph, issues may arise when there is a section $j \in \mathcal{J}$ with a total duration less than the (maximum) loading time $\Delta_{ij}^{\max} = \max_{i,j \in \mathcal{J}, y_{ij}=1} \Delta_{ij}$. Extending a label after such short section may 'forget' loading time

Algorithm 6.3 Sketch of efficient Column Generation with exact pricing

```
1: procedure CGDP()
2:     InitializeRMP()
3:     λ, δ, μ ← SolveRMP()                ▷ Solve Initial Restricted Relaxed Master Problem
4:     for all k ∈ 𝒦 do                    ▷ Solve First Iteration Pricing Problems
5:         G̃_k ← GenerateAuxillaryGraph(k, λ, δ, μ)
6:         L^k ← ESPPRCNC(G̃_k, λ, δ, μ)    ▷ Without using a dominance relation
7:         AddFeasNegRedColumnsToRMP(L^k)
8:         P_k ← ExtractAllFeasiblePaths(L^k)
9:     repeat
10:        λ, δ, μ ← SolveRMP()            ▷ Solve Restricted Relaxed Master Problem
11:        if δ = 0 or λ, δ, μ are unchanged then
12:            break
13:        continue ← false
14:        for all k ∈ 𝒦 do                ▷ Solve Pricing Problems
15:            L^k ← ImproveColumns(k, P_k, G̃_k, λ, δ, μ)   ▷ Only recalculate feas. paths with
   active section start vertices
16:            AddFeasNegRedColumnsToRMP(L^k)
17:            if new columns were added then
18:                continue ← true
19:    until ¬continue
20:    SolveMP()                           ▷ Solve ILP Master Problem
```

of the section before. Our pricing problem may then generate infeasible columns (violating precedence constraints) when such small sections are present. Although such infeasible columns may actually help the (LP) RMP, since fractionally these columns could be feasible, they cause the 'ILP' gap to grow, since these columns are not feasible in the ILP MP.

In our problem, loading times are fixed $\Delta_{ij}^{\max} = \Delta$, and the sections were always larger than this loading time. Therefore, we did not have this problem in our cases and usually it does not happen in practice that route sections are shorter than the (maximum) loading time. However, dealing with such situations in the labelling procedure is tricky, requiring to keep additional information in the labels on cost function sets of some previously visited vertices. Although very interesting, solving such problems is difficult and not needed for our specific problem, and therefore outside the scope of this thesis.

6.4 Heuristic Pricing Methods

Exact pricing has the nice property of always finding the best feasible columns each pricing iteration and eventually solving the RMP (LP) exact, but unfortunately the number of labels needed to be generated grows exponentially, making the exact method intractable for solving large problem instances. Hybrid Column Generation [58, 62], or more general Matheuristic methods [14], are methods using combination of Column Generation (Mathematical Programming) and heuristics to efficiently and quickly generate high quality solutions. In this thesis, we investigate a very simple hybridization of our Column Generation master problem and our SPCI construction heueristic (Algorithm 6.4): we solve the CG pricing subproblem(s) heuristically by multiple runs of SPCI with different strategy. Algoritm 6.4 shows a sketch of this hybridization, CGSPCI.

We only used this algorithm in cases with a single resource kind, $|\mathcal{K}| = 1$, because our SPCI

Algorithm 6.4 Sketch of simple 'Hybridization' Column Generation with heuristic pricing

```
1:  procedure CGSPCI()
2:      InitializeRMP()
3:      λ, δ, μ ← SolveRMP()                ▷ Solve Initial Restricted Relaxed Master Problem
4:      for all strat in SPCI strategies do  ▷ Solve First Iteration Pricing Problem
5:          x ← SPCI(J) using strat
6:          AddFeasNegRedColumnsToRMP(x)
7:      repeat
8:          λ, δ, μ ← SolveRMP()            ▷ Solve Restricted Relaxed Master Problem
9:          if λ, δ, μ are unchanged or objective unchanged then
10:             break
11:         continue ← false
12:         Ũ ← SelectSections(J, λ, δ)     ▷ Solve Pricing Problem
13:         for all strat in SPCI strategies do
14:             x ← SPCI(Ũ) using strat
15:             AddFeasNegRedColumnsToRMP(x)
16:             if new columns were added then
17:                 continue ← true
18:     until ¬continue
19:     SolveMP()                           ▷ Solve ILP Master Problem
```

was not (yet) adapted to work with different resource shift kinds. Therefore, only a single pricing subproblem is considered in Algorithm 6.4. In the first pricing subproblem iteration, the standard SPCI algorithm is run planning all sections in $\mathcal{J}$ using multiple strategies to diversify the generation of columns. Procedure AddFeasNegRedColumnsToRMP now checks each planned resource shift in solution x. For each planned resource shift, two potential columns are examined: all sections as early/as late as possible. Columns with negative reduced cost are added to the RMP. Subsequent pricing problems are similarly solved, but now procedure SelectSections first makes a selection on which sections will be planned based on shadow prices λ, δ. To keep the method simple we choose a very basic selection criteria: only sections with positive set partition shadow price were selected: $\tilde{U} \leftarrow \{j \in \mathcal{J} : \lambda_j > 0\}$. A minor change was made to sorting procedure SortSections inside SPCI Algorithm 5.2 to first sort sections based on decreasing set partition shadow prices λ_j before sorting using the original sorting criteria. Shadow prices did not influence insertion cost functions.

We have ideas for more elaborate hybridization of CG and heuristic pricing, possibly with the use of the exact ImproveColumns procedure of Algorithm 6.3 to calculate optimal start-of-service times for paths exactly. However, preliminary results (some of which we will show) indicate that the above method already performs quite good, making researching different hybridization in the future particularly interesting.

6.5 Integer Solutions

In this thesis, when the RMP (LP) is solved to optimality, or no more columns can be found to improve it, the MP is solved directly as ILP by commercial solver Gurobi [39]. There might be a 'ILP' gap between the objective of the ILP and the (rounded up) LP, indicating the CG problem might not be solved to optimality. This usually happens with highly fractional final RMP LP solutions. Some columns may exists which are beneficial to the RMP ILP but not needed in the RMP LP. To find these columns, usually the CG method is embedded in a branch-and-

bound method, which results in a so-called *branch-and-price* method. We did not consider this method here, because the formulating efficient branching rules is difficult and lies outside our thesis scope. Furthermore, solving the ILP *at the root node* (equivalent to not using branch-and-price), could already solve almost all of our test instances without 'ILP gap', indicating the LP formulation was quite strong.

In light of future research, we make the following observation on interpreting or possibly 'fixing' highly fractional solutions of the LP RMP with explicit precedence constraints. Let $\mathbf{x} = (x_1, x_2, \ldots, x_{n_{\text{cols}}}) \in [0,1]^{n_{\text{cols}}}$ be the solution of the final LP RMP with $n_{\text{cols}} = |\Omega^*|$ columns with x_r corresponding to the *weight* of single resource schedule $r \in \Omega^*$ in the LP solution. Since solution $\mathbf{x}$ satisfies all RMP constraints, each section $j \in \mathcal{J}$ is assigned to single resource schedules with a sum of 1 (equation (6.6)), but possibly fractionally across multiple single resource schedules. In all cases, the precedence constraints (equation (6.8)) are satisfied by the *weighted* (average) start and completion times of a section $j \in \mathcal{J}$: $\hat{S}_j \equiv \sum_{s\in\Omega^*} S_{js}x_r$ and $\hat{C}_j \equiv \sum_{s\in\Omega^*} C_{js}x_r$, since equation (6.8) is equivalent to $\hat{S}_j - \hat{C}_i \geq \Delta_{ij}$, $\forall i,j \in \mathcal{J}$ with $y_{ij} = 1$.

Suppose the multiple columns used to cover section $j \in \mathcal{J}$ in the fractional solution (with each $a_{js}x_r > 0$) do not contain any other section. In that case, these covering columns can be replaced by a new single column s' with the section planned exactly at $S_{js'} = \hat{S}_j$, $C_{js'} = \hat{C}_j$, were we suppose these are actually feasible start/completion times for this section (does not always hold). By adding the new column, the LP RMP solution does not change, since choosing s' with weight 1 is equivalent of choosing the multiple columns covering the section before. However, the ILP RMP solution might improve since this new column can be chosen, while the multiple columns covering the section cannot. If the multiple columns covering a section also each cover very different sections, finding a useful *weighted* combination could be more difficult. Also more difficult is when a section is fractionally assigned to multiple resource shift kinds, or its *weighted* start/completion times may not be feasible at all. Still, the weighted start/completion times of a highly fractional RMP LP gives a good indication for start/completion times preserving the precedence constraints while possibly improving the RMP ILP solution. Exact pricing could for instance be used with start/completion time-windows narrowed around these weighed times, or they could be used somehow in a branching strategy. Further research is needed to develop such strategies.

Chapter 7

Computational Experiments

In this section, we present the computational results of the presented algorithms on benchmark instances modified from literature and a real-life case. Algorithms were implemented in MATLAB version R2007b (64-bit) and run in a single thread. Gurobi version 5.6.2 (64-bit) [39] was used for solving (integer) linear programs arising in the column generation methods. Computations were performed on a Dell Latitude E6530 laptop equipped with a Intel Core i7-3740QM CPU @ 2.70GHz (quad-core with 8 logical cores, single core Hyper-Threading up to 3.70GHz) with 8.00 GB RAM running Windows 7 (64-bit).

In section 7.1, we describe the different test instances used in the computational experiments which were obtained by adjusting well-known test instances from literature. Section 7.2 contains the results from our algorithms on these adjusted instances. In Section 7.3, we describe a slight modification of the adjusted test instances from literature and present results from our algorithms on these modified instances. We conclude the chapter with Section 7.3 on a practical real-life instance obtained from the daily planning of a large Australian logistics customer of ORTEC.

7.1 Adjusted Solomon/Homberger Test Instances

Unfortunately, our specific (full) problem is not well studied in the literature. Therefore, there are no clear instances available. We decided to use well known and widely used instances from vehicle routing problem with time-windows (VRPTW) literature and adjust some characteristics to match those of our problem and proposed application.

The Solomon 100 instances [67], and the related Gehringer & Homberger 200 – 1000 instances [31] are widely used in the field of vehicle routing problems with time-windows. The Solomon 100 instances consist of 56 instances with each 100 customers (orders) and a single depot. They are grouped as follows: 'C1' (9 instances), 'C2' (8 instances), 'R1' (12 instances), 'R2' (11 instances), 'RC1' (6 instances), and 'RC2' (6 instances). The letters 'C', 'R' and 'RC' represent the geographic properties of the customer locations: Clustered, Random and mixed Random-Clustered respectively. The numbers '1' or '2' after the letter(s) represent narrow- or wide time-windows respectively. The larger Homberger instances are grouped similarly, but consist of 60 instances with 200 customers each. Each group contains 10 instances. We did not use the even larger Homberger 400 – 1000 customer instances. All original instances are available on the recently launched vehicle routing repository *VRP-REP* (http://www.vrp-rep.org).

We consider the following adjustments to these instances:

Adjusted Solomon/Homberger instances

- **Distances/Travel Times**
  Original instance coordinates are first normalized to lie within a 100 × 100 square (normalization only needed for Homberger instances) and then scaled to a 400 × 400 square. Vehicle speed is set to 2 units per time unit of minutes. Euclidean distances/travel times are calculated with single precision ($\sim 10^{-8}$).

- **Time Windows/Planning Horizon**
  Customer time-windows $[a_i, b_i]$ are scaled to lie in [0, 720] (time unit of minutes, representing 12 hours). Depot time-window, planning horizon and trailer availability time-windows are set to [0, 900] to allow late driving breaks. Resource Shift availability is set to [0, 900]. Only a single resource shift kind is considered, with a fixed total cost of $c_s^{\text{sh}} = 200$ per resource shift.

- **Capacities, Service and Loading Times**
  To encourage the planning of multiple trip of a trailer, visiting the depot multiple times, the vehicle (trailer) capacity is reduced to $Q = 50$ units. Order capacities remain the same (10 – 50, multiples of 10). Service times of the customer orders are fixed at $T^{\text{serv}} = 15$ time units of minutes. The loading time at the depot needed at the start of a vehicle (trailer) trip is fixed at $\Delta_{ij} = 30$ time units of minutes (first section loading at $[-30, 0]$).

- **Driver Legislation**
  The simple driving rule presented in Section 2.2.2 is used with $T^{\text{driv}}_{\text{max}} = 270$ time units of minutes (4.5 hours) of accumulated driving after which a break of $T^{\text{break}} = 45$ time units of minutes is forced to be planned.

7.2 Computational Results Adjusted Instances

7.2.1 Results CGDP methods

In each adjusted Solomon/Homberger instance, first the stage one problem is solved by a parallel cheapest insertion method (described in) which creates trips and trailer routes. This stage one solution is then used as input for stage two, and we use multiple algorithms to solve this stage two resource assignment problem. Table 7.1 shows the results of all the adjusted Solomon 100 customer instances using the stage one *Fresh* breaking strategy. The upper half of the table shows the summed results over the instance groups, while the bottom half of the table shows the averaged results over the instance groups. The first columns show the number of sections (trips), the number of trailer routes of the stage one solution and the duration in seconds of computation time. Next to it, results from two algorithms are shown: the exact Column Generation method with exact pricing by labelling (CGDP) of Sections 6.1 – 6.3 and its variant denoted by CGDP 'Drexl', which uses the same algorithm but stage two procedure is started by the pre-processing of the time-windows so the sections are not depended any more (See Section 4.3). The column *Labels* shows the summed/average number of labels generated by the labelling algorithm in the first pricing iteration and *Duration* its duration in seconds. *Cols* shows the summed/average number of columns generated and used in the final ILP. This number includes dummy columns (always a total of $2 \cdot |\mathcal{J}|$ dummy columns: two dummy columns per section). Column *Iter* shows the summed/average number of iterations the column generation took to solve the relaxed master problem. This is equal to the number of LP master problems were solved, including the 'dummy' iteration and a final iteration. The minimum iterations (number of LP master problems) the column generation based methods need is 3. The column *ILPGap* shows the summed/average

value of $Z_{\text{ILPGap}} = \sum_{i=1,\ldots,|\Omega^*|} x_i^{\text{ILP}} - \left\lceil \sum_{i=1,\ldots,|\Omega^*|} x_i^{\text{LP}} \right\rceil$, with vector $\mathbf{x}^{\text{ILP}}$ the solution of the final ILP MP, and vector $\mathbf{x}^{\text{LP}}$ the solution of the final relaxed MP. It shows how much full resource shifts the solution of the ILP is worse than the LP rounded up. If this value is 0, for instance when $\sum_{i=1,\ldots,|\Omega^*|} x_i^{\text{ILP}} = 10$ resource shifts and $\sum_{i=1,\ldots,|\Omega^*|} x_i^{\text{LP}} = 9.1$ resource shifts, the found ILP solution is optimal. The column *Duration* shows the summed/average duration in seconds of the complete stage two algorithm. Column *ResShifts* shows the summed/average number of resource shifts used in the final ILP solution. Column *Above Best* shows the (summed/)average percentage the solutions are above our best found solutions for a particular problem resulted from a Stage One result.

Table 7.1 shows some interesting results. The full CGDP method, described in algorithm 6.3, non-surprisingly being the only exact method, produced all best found solutions. These instances were almost all solved to optimality ($Z_{\text{ILPGap}} = 0$) using a single final ILP MP (we did not include something more sophisticated like branch-and-pricing). The total summed ILPGap is only 2 resource shifts, in fact, there were only two instances with $Z_{\text{ILPGap}} = 1 > 0$. This shows the MP formulation is very strong. The table further shows that the first pricing iteration, which is the only iteration using the full ESPPRCNC labelling algorithm, takes the majority of the total duration of the complete algorithm. Large number of labels are generated especially for the '2' instances with wide time-windows, because the number of feasible section configurations on resource shifts in these instances is probably much larger than the narrow time-window '1' instances, in which the sections in routes are much more pinned by narrow individual customer time-windows. This also translates to more columns being generated for the '2' instances and the total duration being larger for these instances. We see the full CGDP method, although solving almost all instances to optimality, on average requires 1.5 more resource shifts then routes being planned in stage one. This is probably due to the *Fresh* breaking strategy used in stage one, which generally does not plans breaks at the times they are needed in the stage two assignment to resource shifts. Therefore the stage one solution lacks the required slack for breaks in the stage two, which worsens the overall solution.

In comparison, Table 7.1 shows similar results for the CGDP 'Drexl' method. The major differences are that the CGDP 'Drexl' method always needs the minimum of 3 iterations due to the missing of the precedence constraints (these are not needed since the time-windows are adjusted so the sections cannot violate them). Also the first pricing iterations generate significantly less labels and are much quicker than the full CGDP method, which in effect also lowers the total duration of the algorithm. The solution is however worse: on average 4 % more resource shifts are needed in the CGDP 'Drexl' compared to the full CGDP method. Especially on the wide time-window '2' instances, the 'Drexl' method performs much worse than the full method. This is probably due to the fact the narrowing of the time-windows by the 'Drexl' procedure had much more impact on wide time-windows instances than on narrow time-window instances.

Table 7.2 shows the results of the same algorithms on the same Adjusted Solomon 100 instances but with the *NotFresh* breaking strategy used in stage one. The number of routes being planned in stage one is higher than with the *Fresh*. This is due to more breaks being planned in stage one with *NotFresh* strategy, so the sections (trips) are less filled with customers and more trailer routes are needed to service all customers. The number of sections does not change significantly. Both the exact full CGDP and the CGDP 'Drexl' benefit from the *NotFresh* breaking rule used in stage one. The number of labels generated/columns generated is much higher, indicating more possible single resource schedules. This increases the total duration of the algorithms significantly, but also increases the quality of the solution significantly: much less resource shifts are needed. Even on average less resource shifts are needed than the number of trailer routes in the *Fresh* case. The 'Drexl' method performs slightly better in terms of resource shifts above the best in the *NotFresh* case, since it probably forces most breaks planned in stage one to remain there in the stage two solution, which is in general not feasible in the *Fresh* case.

To support our above observations on the performance of the full CGDP and CGDP 'Drexl' methods, we conduced the same experiments on the larger 200 customer adjusted Homberger instances. Tables 7.3 and 7.4 show the results using respectively the *Fresh* and *NotFresh* stage one breaking strategy. Many of the above observation also hold for these results, especially observation concerning '2' versus '1' instances and CGDP method versus CGDP 'Drexl' method. The exact full CGDP method again solves almost all of the test instances to optimality, apart from 3 *Fresh*-instances with $Z_{\text{ILPGap}} = 1$. However, the computation times increase significantly in comparison with the 100 customer instances. This is probably due to the exponentially growing number of possible single resource shifts, as shown by the increased number of labels being generated by the first pricing iterations for the 200 customer cases. This is especially apparent in the *NotFresh* case. Although the quality of the solution is very high: on average 40.9 resource shifts needed for 45.4 trailer routes (43.2 with *Fresh*), this method is not scalable to larger instances. With on average 3 % above the best solutions in the *NotFresh* case, the CGDP 'Drexl' performs similarly well compared to its performance on the adjusted Solomon 100 instances.

Table 7.1: Results of CGDP and CGDP 'Drexl' algorithms on the Adjusted Solomon 100 instances using *Fresh* Break Stage One Strategy.

	Stage One			Stage Two: CGDP								Stage Two: CGDP 'Drexl'							
				Iter 1 Pricing								Iter 1 Pricing							
Instances	nSections	nRoutes	Duration (s)	Labels	Duration (s)	Cols	Iter	ILPGap	Duration (s)	ResShifts	Above Best	Labels	Duration (s)	Cols	Iter	ILPGap	Duration (s)	ResShifts	Above Best
C1	40.56	20.33	3.18	6196.22	4.16	1139.44	3.22	0	6.18	19.78	0.00%	5464.89	3.47	971.67	3	0	4.88	20.22	2.65%
C2	40.88	16.38	3.36	7496.88	5.31	1376.88	3.38	0	7.77	16.75	0.00%	5910.5	3.83	1038.25	3	0	5.34	17.87	7.05%
R1	36.42	17.08	3.59	3709.58	2.07	664.08	3.08	0	3.06	18.17	0.00%	3330.25	1.79	583.67	3	0	2.56	18.5	1.87%
R2	35.91	14.36	3.88	5501.82	3.08	1005.55	3.45	0.18	5.22	16	0.00%	4392.55	2.31	723.18	3	0	3.33	16.73	5.05%
RC1	41.5	21.5	3.18	2997.25	1.86	598.88	3.25	0	2.67	23.38	0.00%	2652.62	1.62	530.12	3	0	2.27	23.87	2.06%
RC2	42.62	18.38	3.29	4687.5	3.15	921.38	3.12	0	4.48	22.5	0.00%	3593.38	2.27	683.88	3	0	3.15	24.37	8.82%
Mean:	39.23	17.79	3.45	5040.25	3.19	936.82	3.25	$4 \cdot 10^{-2}$	4.81	19.16	0.00%	4191.38	2.5	745.04	3	0	3.52	19.95	4.38%
Sum:	2197	996	193.04	$2.82 \cdot 10^5$	178.64	52462	182	2	269.12	1073	–%	$2.35 \cdot 10^5$	139.97	41722	168	0	197.26	1117	–%

Table 7.2: Results of CGDP and CGDP 'Drexl' algorithms on the Adjusted Solomon 100 instances using *NotFresh* Break Stage One Strategy.

	Stage One			Stage Two: CGDP								Stage Two: CGDP 'Drexl'							
				Iter 1 Pricing								Iter 1 Pricing							
Instances	nSections	nRoutes	Duration (s)	Labels	Duration (s)	Cols	Iter	ILPGap	Duration (s)	ResShifts	Above Best	Labels	Duration (s)	Cols	Iter	ILPGap	Duration (s)	ResShifts	Above Best
C1	40.78	21.44	3.07	10029.11	8.35	1881.89	3	0	11.55	18.56	0.00%	7533.67	5.37	1380.67	3	0	7.5	19.33	4.45%
C2	41.38	18.25	3.22	13890.75	12.72	2592	3	0	17.89	15.75	0.00%	8672.62	6.24	1541.12	3	0	8.67	16.25	3.46%
R1	36.67	18.17	3.49	6384.08	4.18	1271.33	3.25	0	6.31	16.58	0.00%	4791.42	2.9	927.42	3	0	4.24	16.92	2.39%
R2	35.64	15.36	3.71	9225.73	6.12	1692.73	3.18	0	9.35	14	0.00%	6413.64	3.68	1111.36	3	0	5.38	14.55	4.39%
RC1	41.62	23.25	3.07	4009.5	2.71	844	3	0	3.84	21.12	0.00%	3378.38	2.22	713.12	3	0	3.14	21.12	0.00%
RC2	42.5	19.5	3.17	8407.88	6.55	1770.38	3.12	0	9.38	17.88	0.00%	5776.62	4.07	1175.88	3	0	5.76	18.62	4.39%
Mean:	39.34	19.07	3.32	8550.34	6.58	1651.14	3.11	0	9.49	17.11	0.00%	6044.12	4	1128.95	3	0	5.68	17.59	3.21%
Sum:	2203	1068	185.97	$4.79 \cdot 10^5$	368.45	92464	174	0	531.39	958	–%	$3.38 \cdot 10^5$	223.86	63221	168	0	318.16	985	–%

Table 7.3: Results of CGDP and CGDP 'Drexl' algorithms on the Adjusted Homberger 200 instances using *Fresh* Break Stage One Strategy.

	Stage One			Stage Two: CGDP								Stage Two: CGDP 'Drexl'							
				Iter 1 Pricing								Iter 1 Pricing							
Instances	nSections	nRoutes	Duration (s)	Labels	Duration (s)	Cols	Iter	ILPGap	Duration (s)	ResShifts	Above Best	Labels	Duration (s)	Cols	Iter	ILPGap	Duration (s)	ResShifts	Above Best
C1_2	78.4	40.9	11.22	14357	18.91	2516.1	3.2	0.1	24.01	42.4	0.00%	11729	13.71	1963.6	3	0	16.74	44.3	5.03%
C2_2	82.4	34.5	12.15	51346.4	132.5	8904.4	3.4	0	170.2	34.7	0.00%	36884.2	75.92	6012.1	3	0	92.24	38.3	10.48%
R1_2	84.8	45.6	11.75	14438.8	20.99	2719.1	3.2	0.1	25.73	47.4	0.00%	10733	13.36	1945.1	3	0	16.26	49.5	4.60%
R2_2	85.2	41.8	12.25	18799.9	28.68	3432.9	3.2	0.1	35.77	45.4	0.00%	13423.3	17.06	2313.8	3	0	20.64	49.9	10.55%
RC1_2	82	47.9	11.59	18363.3	28.1	3504.7	3.2	0	34.8	47.7	0.00%	13595.9	18.13	2487.4	3	0	22.19	49.5	4.07%
RC2_2	82.3	48.5	11.24	21660.6	37.61	4041.1	3	0	46.21	49	0.00%	14654.7	19.96	2654.9	3	0	24.33	50.8	4.08%
Mean:	82.52	43.2	11.7	23161	44.47	4186.38	3.2	$5 \cdot 10^{-2}$	56.12	44.43	0.00%	16836.68	26.36	2896.15	3	0	32.07	47.05	6.47%
Sum:	4951	2592	702.01	$1.39 \cdot 10^6$	2667.93	$2.51 \cdot 10^5$	192	3	3367.13	2666	–%	$1.01 \cdot 10^6$	1581.46	$1.74 \cdot 10^5$	180	0	1924.01	2823	–%

Table 7.4: Results of CGDP and CGDP 'Drexl' algorithms on the Adjusted Homberger 200 instances using *NotFresh* Break Stage One Strategy.

	Stage One			Stage Two: CGDP								Stage Two: CGDP 'Drexl'							
				Iter 1 Pricing								Iter 1 Pricing							
Instances	nSections	nRoutes	Duration (s)	Labels	Duration (s)	Cols	Iter	ILPGap	Duration (s)	ResShifts	Above Best	Labels	Duration (s)	Cols	Iter	ILPGap	Duration (s)	ResShifts	Above Best
C1_2	78.3	43.9	10.94	26820.3	52.77	5083.8	3.1	0	66.42	39.3	0.00%	17569.1	25.87	3188	3	0	31.82	41.1	5.15%
C2_2	81.8	37.1	11.8	$1.13 \cdot 10^5$	522.84	21638.2	3.3	0	750.06	30.4	0.00%	58525.6	156.69	10468.3	3	0	198.69	32.1	5.64%
R1_2	84.4	48.3	11.39	25168.5	49.49	4965.4	3.1	0	61.67	44	0.00%	16988.4	26.01	3274.1	3	0	31.85	44.8	2.12%
R2_2	84.9	44.7	12	49191.5	153.88	9423.8	3.1	0	206.99	40.3	0.00%	27989.4	52.49	5141.3	3	0	64.6	40.9	1.64%
RC1_2	81.8	49.3	10.87	29495.4	60.93	5871.8	3	0	77.22	45.9	0.00%	19402	31.3	3760.8	3	0	38.5	46.6	1.96%
RC2_2	82.6	49.1	10.62	37271.4	91.09	7542.2	3.1	0	118.69	45.6	0.00%	24870.9	45.88	4827.7	3	0	56.81	46.6	2.48%
Mean:	82.3	45.4	11.27	46819.88	155.17	9087.53	3.12	0	213.51	40.92	0.00%	27557.57	56.37	5110.03	3	0	70.38	42.02	3.17%
Sum:	4938	2724	676.22	$2.81 \cdot 10^6$	9310.03	$5.45 \cdot 10^5$	187	0	12810.44	2455	–%	$1.65 \cdot 10^6$	3382.5	$3.07 \cdot 10^5$	180	0	4222.62	2521	–%

Table 7.5: SPCI insertion strategies

Strategy	Cost Function	Sort Function	Seed
1	Incr. Certain Makespan	Largest Certain Duration	–
2	Compl. Time	Earliest Compl. Time	–
3	Incr. Certain Makespan	Largest Certain Duration	Bottleneck Res. LB
4	Compl.Time	Earliest Compl Time	Bottleneck Res. LB
5	Incr. Certain Makespan	Largest Certain Duration	Bottleneck Res. LB Breaks
6	Compl.Time	Earliest Compl Time	Bottleneck Res. LB Breaks

7.2.2 Results SPCI methods

We also conducted computational experiments with the Section Parallel Cheapest Insertion stage two heuristic (described in 5.2). A total of 6 different insertion strategies were used, described in Table 7.5. The strategies are explained in Section 5.2.2. Tables 7.6 and 7.7 show the results of SPCI with the 6 different insertion strategies in stage two on the same Adjusted Solomon 100 customer instances with the same stage one solutions. The *Above Best* column represents the summed/average percentage of resource shifts above the number of resource shifts of the best solutions found by the exact CGDP method (see tables 7.1 and 7.2). Strategies 1, 3 and 5 using certain makespan/certain duration produce better solutions than strategies 2, 4, 6 using (earliest) completion times. Also using the bottleneck seeds seem to help the solutions in the *NotFresh* case, although it seems to worsen the solution in the *Fresh* case. This is probably due to the resource LB being much weaker in the *Fresh* case compared to the *NotFresh* case, since breaks planned with *Fresh* strategy are generally not on their final positions. Using the bottleneck with breaks seeds, strategy 6, seems to give the overall best results in the *NotFresh* case. The average duration of the strategies using the bottleneck seed (strat. 3 – 6) is much higher than those without (strat. 1, 2). Our implementation of the bottleneck seed insertion is far from optimal and we suspect a good implementation would need computation times similar to those of strategies 1 and 2.

In the table under *Best Of 6* are the combined results of using all 6 strategies each instance and then choosing the best solution. This increases the average performance of the algorithm from $\sim 19\%$ to $\sim 17\%$ (*Fresh*) and $\sim 15\%$ to $\sim 12\%$ (*NotFresh*) number of resource shifts above the best solution. Combining strategies this way seems really beneficial, although the total duration of the combined algorithm increases, since it becomes the sum of the individual SPCI runs.

Table 7.6: Results of SPCI algorithms with 6 insertion strategies on the Adjusted Solomon 100 instances using *Fresh* Break Stage One Strategy. Included the performance of the 'best' instances of using all 6 strategies.

	Stage Two: SPCI strat. 1			Stage Two: SPCI strat. 2			Stage Two: SPCI strat. 3			Stage Two: SPCI strat. 4			Stage Two: SPCI strat. 5			Stage Two: SPCI strat. 6			Stage Two: SPCI Best of 6	
Instances	Duration (s)	ResShifts	Above Best	Duration (s)	ResShifts	Above Best	Duration (s)	ResShifts	Above Best	Duration (s)	ResShifts	Above Best	Duration (s)	ResShifts	Above Best	Duration (s)	ResShifts	Above Best	ResShifts	Above Best
C1	8.72	21.11	24.75%	8.08	22.11	30.48%	21.68	21	24.14%	24.56	20.78	22.75%	21.77	21	24.14%	24.54	20.78	22.75%	20.78	22.75%
C2	9.82	18.62	17.40%	9.1	20.62	30.13%	24.31	19.88	25.41%	24.52	20.5	29.27%	23.69	20	26.19%	24.08	20.25	27.71%	18.62	17.40%
R1	7.27	19.67	18.06%	6.48	21.33	28.07%	14.93	20.17	20.94%	16.35	20.83	24.92%	14.95	20.17	20.94%	16.42	20.75	24.40%	19.25	15.47%
R2	7.65	18.18	21.78%	6.68	19	26.81%	16.12	18.64	24.32%	17.38	18.36	22.61%	16.06	18.64	24.32%	17.41	18.27	22.04%	17.73	18.39%
RC1	9.55	24.38	9.71%	8.87	26.12	17.51%	21.38	24.75	11.41%	22.68	25.75	15.99%	21.44	24.75	11.41%	22.81	25.75	15.99%	24	7.95%
RC2	11.13	24.5	19.34%	10.38	25.38	23.80%	24.58	25.12	23.08%	24.69	25	22.22%	24.09	25.12	23.08%	24.85	25	22.22%	23.88	16.62%
Mean:	8.82	20.82	18.76%	8.05	22.16	26.39%	19.89	21.32	21.70%	21.13	21.59	23.08%	19.75	21.34	21.81%	21.13	21.52	22.63%	20.45	16.58%
Sum:	493.95	1166	–%	450.85	1241	–%	1113.68	1194	–%	1183.44	1209	–%	1105.76	1195	–%	1183.3	1205	–%	1145	–%

Table 7.7: Results of SPCI algorithms with 6 insertion strategies on the Adjusted Solomon 100 instances using *NotFresh* Break Stage One Strategy. Included the performance of the 'best' instances of using all 6 strategies.

	Stage Two: SPCI strat. 1			Stage Two: SPCI strat. 2			Stage Two: SPCI strat. 3			Stage Two: SPCI strat. 4			Stage Two: SPCI strat. 5			Stage Two: SPCI strat. 6			Stage Two: SPCI Best of 6	
Instances	Duration (s)	ResShifts	Above Best	Duration (s)	ResShifts	Above Best	Duration (s)	ResShifts	Above Best	Duration (s)	ResShifts	Above Best	Duration (s)	ResShifts	Above Best	Duration (s)	ResShifts	Above Best	ResShifts	Above Best
C1	8.7	19.89	23.84%	8.22	20.67	28.86%	24.77	19.33	20.24%	27.52	19.44	21.11%	25.39	19.56	21.84%	28.21	19.56	22.07%	19.22	19.62%
C2	9.53	17.62	22.05%	9.17	18.12	25.26%	28.9	17.38	20.11%	28.82	18.62	28.51%	33.05	17.62	22.06%	32.12	19.5	34.82%	16.88	16.71%
R1	7.2	18.17	17.73%	6.51	18.5	19.51%	19.31	18.25	18.35%	20.46	18.08	17.22%	20.59	17.92	15.92%	21.45	18.33	18.75%	17.33	11.82%
R2	6.84	15.18	16.33%	6.4	16.09	23.83%	19.36	14.91	14.06%	20.14	15.64	20.25%	20.24	15	14.86%	20.88	15.36	17.90%	14.73	12.58%
RC1	9.14	21.88	5.75%	8.9	23.62	14.17%	24.57	22.25	7.54%	26.03	22.25	7.69%	27.99	22.12	6.84%	28.45	23.5	13.86%	21.75	5.09%
RC2	10.09	19.5	9.80%	9.71	20.25	14.56%	29.56	19.62	10.84%	31.21	20	13.28%	31.25	18.88	6.89%	32.6	19.25	9.16%	18.5	4.44%
Mean:	8.39	18.5	16.21%	7.94	19.3	21.21%	23.78	18.41	15.44%	25.06	18.77	18.13%	25.65	18.3	14.95%	26.54	18.98	19.35%	17.86	11.91%
Sum:	469.95	1036	–%	444.78	1081	–%	1331.9	1031	–%	1403.2	1051	–%	1436.6	1025	–%	1486.3	1063	–%	1000	–%

7.2.3 Preliminary results Hybrid CG SPCI method

We also conducted some preliminary experiments with the column generation method using SPCI as heuristic pricing (described in 6.4). Pricing problems were heuristically solved using all of the 6 strategies described in Table 7.5, which each generate a complete schedule of selected sections. The first iteration consist of running the SPCI methods to generate full schedules of all sections, similary to the SPCI methods on their own. Subsequent pricing iterations, only sections with positive set partition shadow prices (sections J_i with $\lambda_i > 0$) were planned each pricing iteration. The each resource shift of the complete schedules was translated to two columns: one with every section planned as early as possible, and another with every section planned as late as possible (but still respecting the complete schedule solution). These columns were added to the relaxed master problem only if its reduced cost (equation (6.12)) was in fact negative. Column generation method stopped when after 2 consecutive iterations the objective did not improve or the shadow prices stayed the same.

Table 7.8: Preliminary results of CGSPCI the Adjusted Solomon 100 instances using *Fresh* (upper) and *NotFresh* (lower) Break Stage One Strategy.

	Stage Two: CGSPCI strat. 1–6					
Instances	Cols	Iter	ILPGap	Duration (s)	ResShifts	Above Best
C1	231.11	5.22	0	192.38	19.78	0.00%
C2	285.88	6.88	0.5	304.59	17.38	3.87%
R1	218.25	5	0	153.14	18.25	0.44%
R2	228.82	6.55	$9 \cdot 10^{-2}$	199.54	16.36	2.48%
RC1	230.38	5.13	0	195.47	23.63	1.09%
RC2	229.88	4.25	0	152.79	23	2.43%
Mean:	235.45	5.52	$9 \cdot 10^{-2}$	196.19	19.45	1.64%
Sum:	13185	309	5	10986.85	1089	–%

	Stage Two: CGSPCI strat. 1–6					
Instances	Cols	Iter	ILPGap	Duration (s)	ResShifts	Above Best
C1	248.22	4.67	0	211.13	18.56	0.00%
C2	307.63	5.13	0.13	296.85	16.13	2.31%
R1	244.75	4.67	0	171.37	16.75	1.11%
R2	238.73	5	0.27	164.96	14.27	2.23%
RC1	278.5	4.63	0	243.45	21.13	0.00%
RC2	270.25	4	0.13	170.8	18	0.83%
Mean:	261.57	4.7	$9 \cdot 10^{-2}$	204.64	17.27	1.13%
Sum:	14648	263	5	11459.99	967	–%

Table 7.8 shows the preliminary results of the CGSPI method on the adjusted Solomon 100 customer instances. In both *Fresh* and *NotFresh* cases, 5 instances showed an Z_{ILPGap} of 1. Nonetheless, the schedules produced by the method are of surprisingly high quality ($\sim$ 1.6% above best *Fresh* of CGDP, $\sim$ 1.1% above best *NotFresh* of CGDP). The duration of the method is highly depended on the duration of the individual SPCI methods used in pricing.

7.3 Computational Results Modified Adjusted Instances

From the earlier results on the CGDP 'Drexl' method versus the full exact CGDP method, one might conclude that the CGDP 'Drexl' method is quite good (on average $\sim$ 3% above best) while it needs significantly lower computation times. After more detailed analysis of the 'Drexl' method and its results, we found that the impact of the time-window narrowing 'Drexl' method depends highly on the number of sections (trips) in a route. If a route in a stage one solution only contains a single section, no narrowing is done by the method. The single section is already

independent on its own. If a route contains two sections, minor adjustments to the time-windows are done to only the first section in the route. The time-windows of the last section remain the same. Only when there are three or more sections in a route of the stage one solution, the narrowing procedure of the 'Drexl' method seriously adjusts the time-windows.

To show this dependency of the 'Drexl' method on the number of sections, we consider a slight variant on the adjusted Solomon instances by changing only capacity characteristics:

Modification

- **Capacities**, Service and Loading Times
  To encourage the planning of multiple trip of a trailer, visiting the depot multiple times, the vehicle (trailer) capacity is reduced to **26** units. **Order capacities are scaled to lie in the closed interval** [8, 10] **and rounded to the nearest integer.**

The idea of the modification is that less customers (only 1–3) will be planned in a section, so more sections (trips) will be planned in a trailer route.

Table 7.9: Comparison of the Stage One results on sections per route of the Adjusted Solomon 100 instances and with the Modification using *Fresh* Break Stage One Strategy.

	Stage One Adjusted Solomon							Stage One Modified Adjusted Solomon					
				% of routes with:							% of routes with:		
Instances	Sections	Routes	Sec./Route	1 Sec.	2 Sec.	≥ 3 Sec.	Instances	Sections	Routes	Sec./Route	1 Sec.	2 Sec.	≥ 3 Sec.
C1	40.56	20.33	2.03	38.87	31.38	29.75	C1	39.44	18.22	2.18	33.61	31.52	34.88
C2	40.88	16.38	2.52	15.16	38.22	46.62	C2	38.62	14.75	2.64	16.06	34.71	49.23
R1	36.42	17.08	2.18	26.56	36.83	36.61	R1	40.67	16.75	2.47	23.97	27.07	48.95
R2	35.91	14.36	2.53	20.11	24.78	55.12	R2	40.82	14.09	2.92	16.01	20.08	63.91
RC1	41.50	21.50	1.95	28.57	50.30	21.13	RC1	48.25	22.25	2.18	18.11	54.74	27.15
RC2	42.62	18.38	2.35	18.27	35.42	46.31	RC2	46.88	18.75	2.52	15.20	30.90	53.89
Mean:	39.23	17.79	2.26	24.74	35.51	39.75	Mean:	42.18	17.25	2.50	20.74	32.00	47.26

Table 7.10: Comparison of the Stage One results on sections per route of the Adjusted Solomon 100 instances and with the Modification using *NotFresh* Break Stage One Strategy.

	Stage One Adjusted Solomon							Stage One Modified Adjusted Solomon					
				% of routes with:							% of routes with:		
Instances	Sections	Routes	Sec./Route	1 Sec.	2 Sec.	≥ 3 Sec.	Instances	Sections	Routes	Sec./Route	1 Sec.	2 Sec.	≥ 3 Sec.
C1	40.78	21.44	1.92	42.24	32.15	25.61	C1	39.78	18.44	2.18	33.16	30.96	35.88
C2	41.38	18.25	2.28	20.78	42.97	36.25	C2	39.88	16.25	2.47	18.67	34.63	46.70
R1	36.67	18.17	2.04	27.58	43.45	28.97	R1	40.33	17.58	2.33	23.24	34.64	42.12
R2	35.64	15.36	2.35	22.60	32.38	45.01	R2	40.45	14.45	2.82	14.64	23.52	61.84
RC1	41.62	23.25	1.80	33.50	54.28	12.22	RC1	48.75	24.12	2.04	23.44	55.41	21.15
RC2	42.50	19.50	2.20	20.15	41.53	38.32	RC2	47.50	20.00	2.41	17.73	34.20	48.07
Mean:	39.34	19.07	2.11	27.77	40.66	31.56	Mean:	42.43	18.20	2.39	21.73	34.77	43.50

Both tables 7.9 and 7.10 show the effect of the modification on the stage one solutions. Table 7.9 shows results of the stage one solutions of the original Adjusted Solomon 100 instances in comparison with the instances using the Modification, using the *Fresh* breaking strategy. Table 7.10 shows similar results using the *NotFresh* breaking strategy. Both tables show the average number of sections per route and also the distribution of routes with 1, 2 and ≥ 3 sections. The modification increases the total number of sections being planned, while a slightly decreasing number of trailer routes are needed. The average number of sections per route indeed increases when the modification is used (in each of the *Fresh*/*NotFresh* cases). The distribution of sections per route changes: a larger fraction of routes have ≥ 3 sections while the fraction of routes with 1 or 2 sections decreases.

Tables 7.11 and 7.12 show the results of the modified Adjusted Solomon 100 instances solved by the full CGDP method in comparison with the CGDP 'Drexl' method, *Fresh* and *NotFresh*

stage on breaking strategy respectively. The full method again produces good results ($\sim$ 16 resource shifts for $\sim$ 18 routes, *NotFresh*), while the computation times increase significantly with respect to the computation times needed for the original Adjusted Solomon 100 instances of Tables 7.1 and 7.2. Since the sections are smaller, the routes contain on average more sections and more sections can on average be planned on a single resource shift. Therefore there are a lot more feasible possibilities the exact pricing method needs to consider (a lot more labels are generated), so the computation times increases.

As expected, the CGDP 'Drexl' method now performs worse compared to its performance on the original Adjusted Solomon 100 instances. The average percentage of resource shifts above best is significantly higher in all instance groups except the *NotFresh* 'C2' group. Although the modification did increase the number of ≥ 3 section routes for this group, the CGDP 'Drexl' did especially well in on this instance group. An possible explanation is that the routes of stage one solutions of group 'C2' are now tightly packed with sections that the reduced time-windows of stage two become narrow. The 'Drexl' does not alter these narrow time-windows much. This is supported by the apparent low number of labels being generated by both the full CGDP method and the CGDP 'Drexl' method, in comparison with the modified 'C1' group but also to the original 'C2' groups. In the original Adjusted Solomon 100 results, the average number of labels generated in the first pricing iteration in each instance group was less than the average number of labels generated in the results of the modified instances, with again the exception being the *NotFresh* 'C2' group. Also in the original Adjusted Solomon 100 results, the number of labels generated for all the '2' instances was on average more ~ 2 times the number of labels generated for all the '1' instances. In the modified instances, this is a bit less, with the exception of the 'C2' group having an average number labels generated even less than the 'C1' group. This still suggests that the CGDP 'Drexl' method performed much worse on stage one solutions with wide time-windows and a lot of routes with ≥ 3 sections, but not so much if the (reduced) time-windows are in fact narrow by this high number of sections.

Table 7.11: Results of CGDP and CGDP 'Drexl' algorithms on the Modified Adjusted Solomon 100 instances using *Fresh* Break Stage One Strategy.

	Stage One			Stage Two: CGDP								Stage Two: CGDP 'Drexl'							
				Iter 1 Pricing								Iter 1 Pricing							
Instances	nSections	nRoutes	Duration (s)	Labels	Duration (s)	Cols	Iter	ILPGap	Duration (s)	ResShifts	Above Best	Labels	Duration (s)	Cols	Iter	ILPGap	Duration (s)	ResShifts	Above Best
C1	39.44	18.22	3.26	7463.44	4.89	1316.78	3	0	6.94	17	0.00%	6554	4.12	1157	3	0	5.87	17.44	2.98%
C2	38.62	14.75	3.4	6686.25	4.02	1124.75	3.25	0.12	6.02	15.87	0.00%	5378.38	2.94	867.75	3	0	4.18	17	7.50%
R1	40.67	16.75	3.73	8706.5	6.21	1567.08	3.5	0	9.67	16.75	0.00%	7331.33	4.79	1261	3	0	6.71	17.17	2.61%
R2	40.82	14.09	3.98	13654.09	10.68	2474.91	4	$9 \cdot 10^{-2}$	18.89	15.09	0.00%	9773.36	6.47	1537	3	0	8.88	16.45	10.00%
RC1	48.25	22.25	3.26	5328.75	3.9	1049.5	3.5	0	5.52	22.25	0.00%	4696.62	3.4	914.38	3	0	4.62	22.75	2.16%
RC2	46.88	18.75	3.37	8159.38	6.17	1520.88	3.38	0.12	8.66	20.87	0.00%	5215.75	3.44	920	3	0	4.68	24.87	19.82%
Mean:	42.18	17.25	3.54	8629.27	6.23	1561.45	3.46	$5 \cdot 10^{-2}$	9.79	17.71	0.00%	6728.48	4.36	1144.09	3	0	6.05	18.95	7.21%
Sum:	2362	966	198.03	$4.83 \cdot 10^5$	348.63	87441	194	3	548	992	–%	$3.77 \cdot 10^5$	243.97	64069	168	0	338.92	1061	–%

Table 7.12: Results of CGDP and CGDP 'Drexl' algorithms on the Modified Adjusted Solomon 100 instances using *NotFresh* Break Stage One Strategy.

	Stage One			Stage Two: CGDP								Stage Two: CGDP 'Drexl'							
				Iter 1 Pricing								Iter 1 Pricing							
Instances	nSections	nRoutes	Duration (s)	Labels	Duration (s)	Cols	Iter	ILPGap	Duration (s)	ResShifts	Above Best	Labels	Duration (s)	Cols	Iter	ILPGap	Duration (s)	ResShifts	Above Best
C1	39.78	18.44	3.19	13837.56	13.28	2730	3.11	0	19.55	16.11	0.00%	9299.11	7.09	1763.56	3	0	10.01	16.89	5.18%
C2	39.88	16.25	3.29	13186.12	11.28	2549	3.38	0	18	14.5	0.00%	8339.12	5.6	1479.75	3	0	7.95	14.75	1.88%
R1	40.33	17.58	3.64	13901.42	13.16	2679.33	3.33	0	19.64	15.58	0.00%	8966.33	6.58	1624.67	3	0	9.2	16.42	5.81%
R2	40.45	14.45	3.85	26442.18	31.01	5118.45	3.45	$9 \cdot 10^{-2}$	45.86	13.09	0.00%	15287.91	12.88	2676.91	3	0	17.69	14.09	8.06%
RC1	48.75	24.12	3.19	9663.12	8.51	1985.62	3.5	0	12.52	21	0.00%	6740.12	5.24	1325	3	0	7.12	21.37	1.89%
RC2	47.5	20	3.29	16397.12	16.85	3401.88	3.25	0	24.07	18.25	0.00%	10037	8.24	1927.75	3	0	11.25	19.12	5.41%
Mean:	42.43	18.2	3.44	16003.39	16.28	3152.09	3.34	$2 \cdot 10^{-2}$	24.16	16.18	0.00%	10006.88	7.8	1833.46	3	0	10.82	16.89	4.97%
Sum:	2376	1019	192.84	$8.96 \cdot 10^5$	911.54	$1.77 \cdot 10^5$	187	1	1352.82	906	–%	$5.6 \cdot 10^5$	437.06	$1.03 \cdot 10^5$	168	0	605.69	946	–%

Table 7.13: Some characteristics of the real life test instance.

Delivery Orders	175
Depots	1
Trailer Routes	36
Sections (trips)	57
Orders per Sections	1 – 6 (av. 3.18)
Sections per Route	1 – 4 (av. 1.58)
Available Resource Shifts	35 (early & late shifts)
Planning Horizon	25.5 h (1 day)
Fixed Loading Time	0.5 h
Max Driving Time before Break	6 h
Break Duration	0.5 h

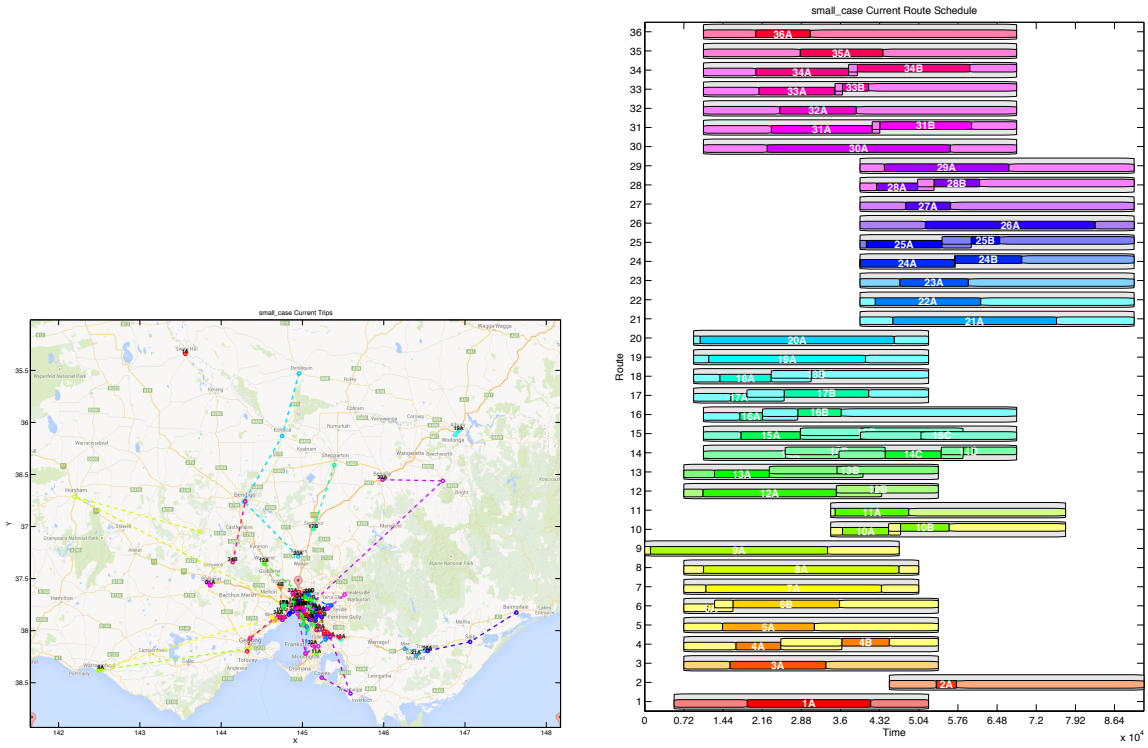
7.4 Real-life cases

To show the real-life potential of our algorithms, we also conducted experiments on a single instance obtained from a large Australian logistics customer of ORTEC. The test instance consist of a distribution planning of a single depot for a single (full) day. The stage one solution trailer routes were already planned by Route Scheduling software of ORTEC, using a custom configuration regularly used by the customer. We build a conversion tool to convert the instance from a format used by ORTEC to a format which could be imported by our MATLAB prototype suite. Figure 7.1 shows the Stage One solution as imported in our MATLAB prototype: Figure 7.1(a) shows a map of the Melbourne area (source: Google Maps, retrieved 7 jan '15). Overlayed are the trailer trips, but with the arcs from and to the depot (the red square in north Melbourne) omitted. Figure 7.1(b) shows the route schedules with sections labelled '1A'/'2B' for the first/second section of the first/second route respectively. The sections are shown with three coloured parts (light, dark, light) cut by the following 4 points in time: the section's earliest, latest start and earliest, latest completion times. The light areas are large and overlapping, which shows the sections have a lot of slack, meaning they have wide reduced time-windows and can be moved a lot in time.

Table 7.13 shows some more characteristics of the test instance. Especially interesting is the low average number of sections per route (1.58). Also notice we changed the simple driving rule to $T^{\text{driv}}_{\max} = 6$ hours and $T^{\text{break}} = 0.5$ hours. This reflects more the Australian social drivers legislation [34], opposed to the earlier used driving rule reflecting the European drivers legislation.

As can be seen in the route schedule, Figure 7.1(b), the trailer routes were not planned for the full planning horizon, but with some predetermined start/completion times reflecting already different available resource shift kinds (early and late resource shift kind). The original case had 35 resource shifts, each start early or late with an allowable range of start and completion times, but with the extra requirement that the total duration of a resource shifts does not exceed 13 hours. This is a very difficult constraint, mathematically equivalent to the working time rules of Directive 2002/15/EC discussed in section 2.2, which our algorithms cannot (yet) handle. Therefore we investigate three other cases, which can be solved by our algorithms.

- **Case 1: One Resource Shift kind**
  We consider only one resource shift kind, which is available for the entire planning horizon, so without maximum duration. This is similar to the resource shift kind used in the earlier discussed adjusted Solomon/Homberger instances. In this case, resource shifts can have a duration up to 25.5 hours of work for a single driver, which is somewhat unrealistic. Still the results of the performance of our different algorithms are interesting.



(a) Route Map (background src: Google Maps)

(b) Route Schedule

Figure 7.1: Real-life case Stage One results after import.

- **Case 2: Two Resource Shift kinds**
  To emulate the early and late resource shifts, we consider two different resource shift kinds: an early resource shift kind starting (earliest) at 0 h and completing (latest) at 16 h, and a late resource shift kind starting (earliest) at 9.5 h and completing (latest) at 25.5 h. We do not limit the amount of resource shifts (variables u_k of master problem resource availability constraints (6.7) are set to a high value).

- **Case 3: Three Resource Shift kinds**
  We use the early and late resource shift kinds of case 2, but now add the following middle resource shift kind: starting (earliest) at 3.5 h and completing (latest) at 17.5 h. This resource shift kind is shorter (14 h opposed to 16 h), since being in the middle of the planning, it could be tightly packet with sections. This way, the planned resource shifts will have a higher chance that they approximately satisfy the maximum working duration of 13 h. We still make sure the amount of resource shift kinds available is not limiting.

Using the stage one solution, the resource lower bound (discussed in) was calculated. It is shown as the blue line in Figure 7.2. Red/green lines correspond to the minimum number of resource shifts needed when sections are planned as early/as late as possible respectively. The bottleneck is around $3.0 \cdot 10^4$ seconds, which is ~ 8.5 hours, with at least 19 resource shifts needed at that time. This is a global minimum resource shifts needed to assign all sections in case 1 to resource shifts of a single resource shift kind. Also the different resource shifts kinds of cases 2 and 3 are displayed in the figure. For case 2, when we have only resource shifts of the bottom two kinds, we see we need at least 19 of the early kind and at least 8 of the late kind. For case 3, when we have all 3 resource shift kinds, at least 19 shifts of kinds early and middle are needed and at least 5 shifts of the late kind are needed. We will show that such 'pre-analysis' is valuable in evaluating the quality of the final solutions.

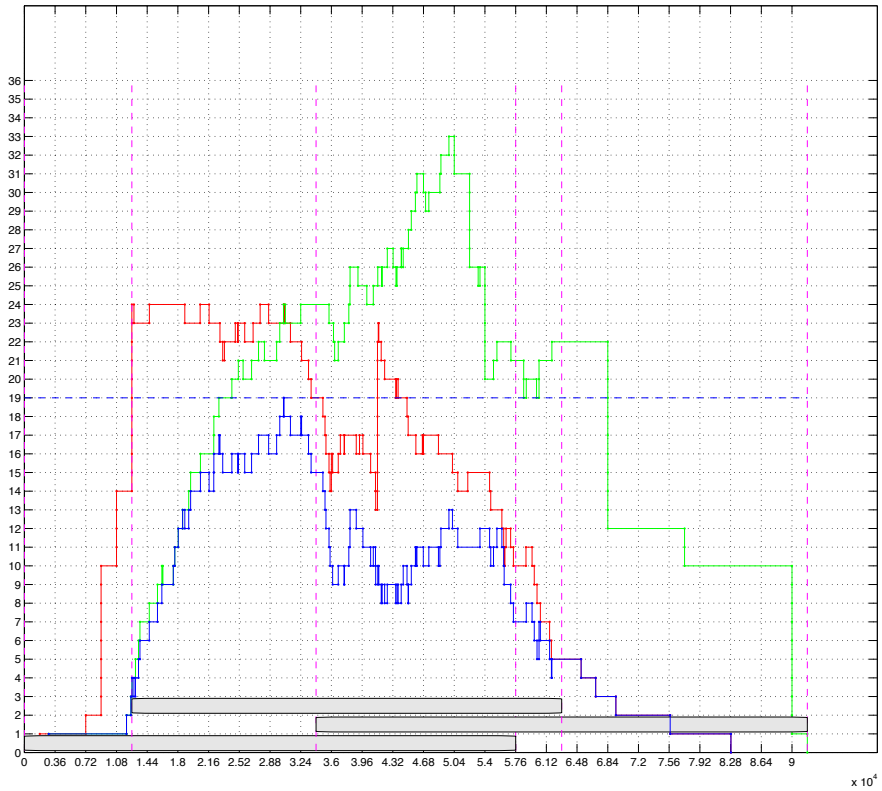


Figure 7.2: Real-life case resource lower bound in blue. Red/Green lines show the resources needed when sections are planned as early/late as possible. Also the position of Resource Shift Kinds in the planning horizon of cases 2 (bottom 2) and 3 (all 3) are shown.

To investigate the benefit of driver resources being able to switch trailers during the trailer routes, we tested another CGDP variant: CGDP 'Only Routes'. In this variant, the label extension of the first pricing iteration labelling algorithm is altered to only allow the planning of full trailer routes. This can be easily achieved by altering the SUCCESSORS(v_i) procedure of the labelling algorithm 6.2. Resource Shift paths always start a new route by its first section and visit all subsequent sections. Only after completing the full route, a new route may start. Notice this will cause the fixed loading time between the sections to be always planned to the resource shift: the driver has to wait at the depot while the trailer is being loaded for the next section/trip. We did not use this CGDP variant in the previous adjusted literature instances, since its outcome is (approximately) equal to the *NotFresh* number of routes being planned by the stage one result. This is because in these instances, the generated routes generally fill the complete planning horizon, therefore in almost all cases a resource shift cannot be assigned multiple full routes for these instances. However, the real-life instance schedule in Figure 7.1(b) suggests that resource shifts could be filled by multiple full routes.

7.4.1 Results Case 1: One Resource Shift kind

Table 7.14 shows the results of the different stage two algorithms on the real-life instance with full planning horizon spanning resource shifts. SPCI 1,2 show the average results of the cheapest insertion heuristic using strategy 1 and 2, which both produced similar results. SPCI 3 – 6 shows the average results of strategies 3, 4, 5 and 6, which also produce quite similar results. The CGSPCI 1–4 is the column generation method using SPCI heuristic pricing running strategies 1,2,3 and 4.

Table 7.14: Results of different algorithms on real-life instance case 1 with one resource shift kind.

Method	CGDP 'Only Routes'	CGDP	CGDP 'Drexl'	SPCI 1,2	SPCI 3–6	CGSPCI 1–4
ResShifts	24	19	19	21	20	19
Above Best	26.31%	0.00%	0.00%	10.52%	5.26%	0.00%
Duration (s)	1.00	4160	1800	19	51	203
Labels it. 1	1948	386731	257635	–	–	–
Duration (s) it. 1	0.90	4158.24	1823.62	–	–	145.62
Cols.	348	74239	48636	–	–	334
Iter.	3	3	3	–	–	5
ILPGap	0	0	0	–	–	0

The 36 trailer routes were optimally assigned to 24 resource shifts in case of no switching between sections (CGDP 'Only Routes'), but allowing switching between section only needs 19 resource shifts in the optimal assignment (CGDP), which makes the first method's result $\sim$ 26% worse than the latter. Not only the exact CGDP method found an optimal schedule, also the CGDP 'Drexl' and even the CGSPCI method found an optimal schedule using only 19 resource shifts. Since the resource lower bound graph in 7.2 stated this 19 resource shifts is the absolute minimum necessary to assign all sections, optimality can even be concluded for these later algorithms, even though these methods are not exact. The resource lower bound graph is a powerful tool to view absolute minimum needed number of resource shifts over time and bottlenecks, and in some cases, like here, can be used to prove optimality of a heuristically produced schedule. The cheapest insertion methods SPCI produce quite good results, especially strategies 3, 4, 5 and 6 (3 – 6) using the resource lower bound bottleneck seed. The CGSPCI method takes advantage of simultaneously using the SPCI methods and can even produce an optimal schedule after only 5 iterations, using only 334 columns. The duration of this latter method is highly depended on the duration of the individual SPCI methods used in pricing.

7.4.2 Results Case 2: Two Resource Shift kinds

Table 7.15: Results of CGDP algorithms on real-life instance case 2 with two resource shift kinds.

Method	CGDP 'Only Routes'	CGDP	CGDP 'Drexl'
ResShifts	INF: 29 (21+8) + 3 dum.	28 (19+9)	28 (19+9)
Above Best	–%	0.00%	0.00%
Duration (s)	0.3	329	134
Labels it. 1	764	109577	67301
Duration (s) it. 1	0.25	327.98	134.00
Cols.	198	11394	6626
Iter.	3	3	3
ILPGap	0	0	0

Table 7.15 shows the results of the column generation with pricing by labelling on the real-life instance case 2 with two resource shift kinds (one early, one late). We did not include other methods, since they were not ready to be used planning on different resource shift kinds in our prototype. Especially SPCI methods would require additional/extending strategies on how/when to plan on a particular resource shift kind, which was outside the scope of this thesis.

Interestingly, the CGDP 'Only Routes' was not able to find a feasible assignment of all routes to the two resource shift kinds. Three routes lay in the 'middle' of the planning horizon, which could therefore not be each fully assigned to a single early or single late resource shift kind, without splitting the route. Allowing switching between sections solves this problem, and both

CGDP and CGDP 'Drexl' methods can find the optimal assignment of 28 resource shifts. The optimal schedule consists of 19 early- and 9 late resource shifts. As can be seen in the resource lower bound graph in Figure 7.2, 19 resource shifts of early kind is lowest possible optimal, but 9 resource shifts of the late kind is not (8 is the minimum needed). Therefore, using only the results of the CGDP 'Drexl' method with the resource lower bound in this case is not enough to prove optimality.

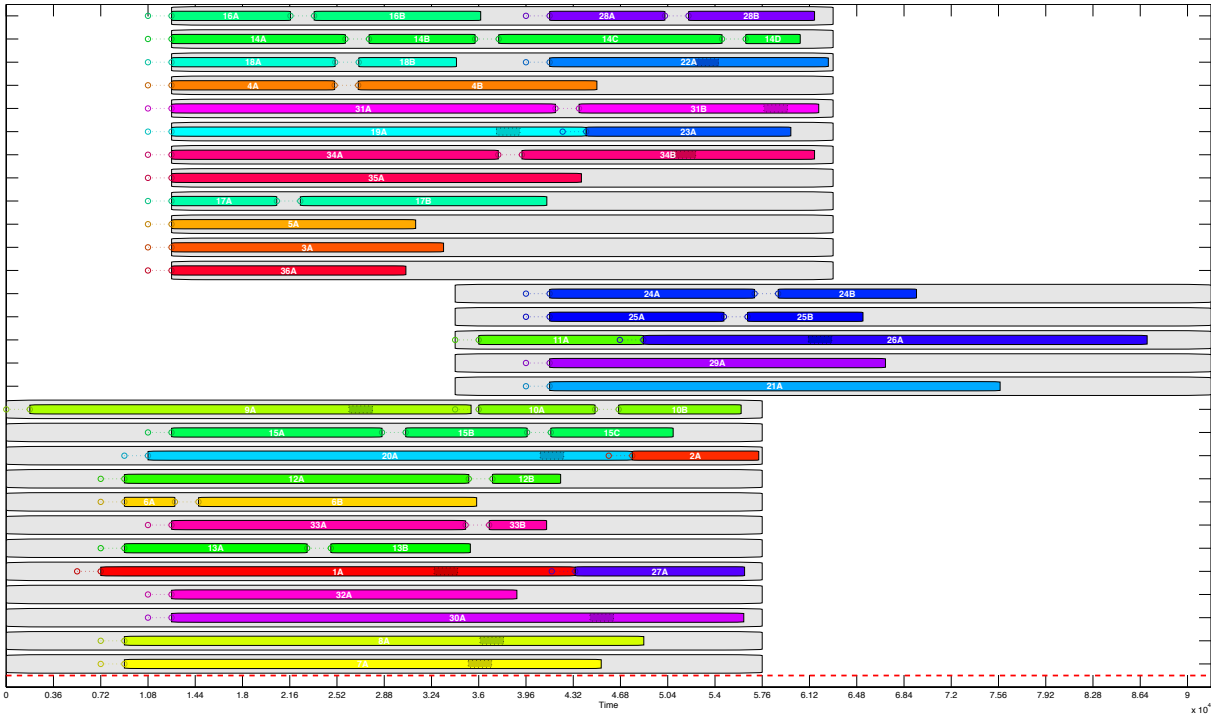
7.4.3 Results Case 3: Three Resource Shift kinds

Table 7.16: Results of CGDP algorithms on real-life instance case 3 with three resource shift kinds.

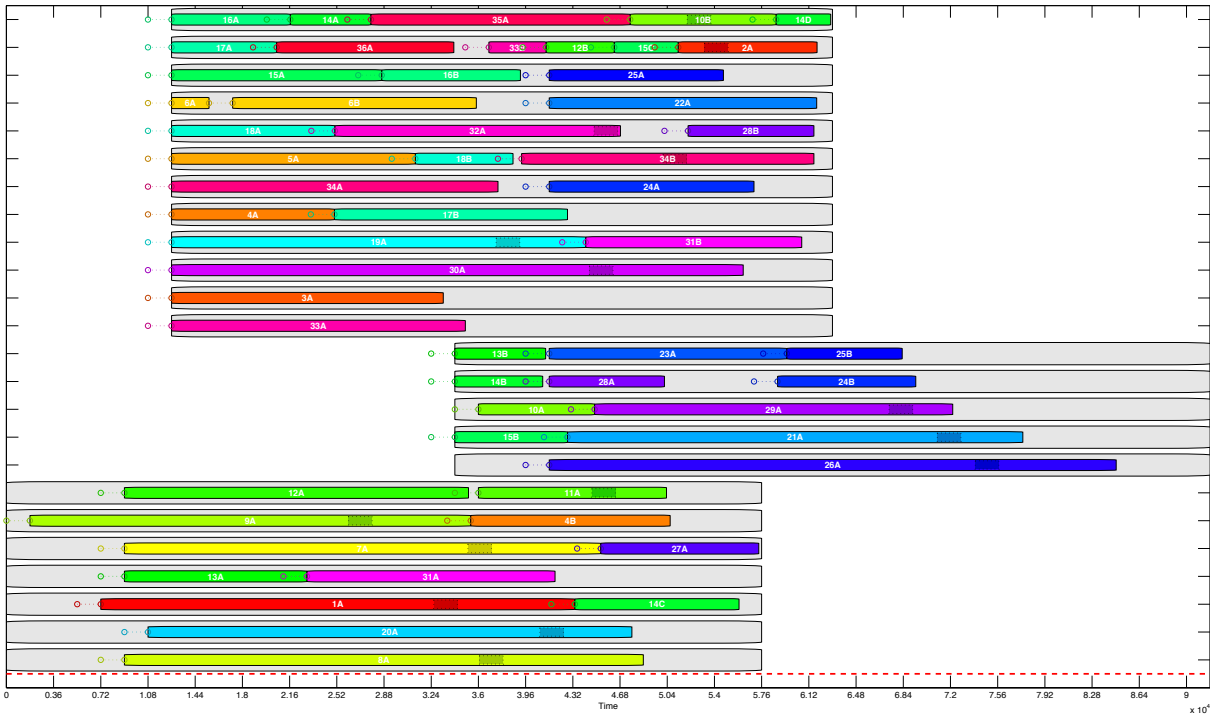
Method	CGDP 'Only Routes'	CGDP	CGDP 'Drexl'
ResShifts	29 (12+12+5)	24 (8+11+5)	24 (7+12+5)
Above Best	20.83%	0.00%	0.00%
Duration (s)	0.7	1285	398
Labels it. 1	1694	291580	158708
Duration (s) it. 1	0.61	1280.00	395.23
Cols.	303	40445	20701
Iter.	3	3	3
ILPGap	0	0	0

Finally, Table 7.16 shows the results of the column generation with pricing by labelling on the real-life instance case 3 of three resource shift kinds (early, middle and late). Introduction of the middle resource shift kind allows the CGDP 'Only Routes' to find a feasible solution of 29 resource shifts (12 early, 12 middle and 5 late). Figure 7.3(a) shows this solution. It is $\sim$ 20% worse than the optimal solution of 24 (8 early, 11 middle and 5 late) found by CGDP when allowing switching between sections, which is shown in Figure 7.3(b). The figures show how some of the gaps cased by the fixed loading time in the full route assignment (fig 7.3(a)) can be filled quite nicely by allowing drivers to switch trailers between sections (fig 7.3(b)). Also in the latter figure, it is shown how some routes are not only split over different resource shifts but even over different resource shift kinds (for instance 'B' sections planned at the start of the late resource shifts). This greatly increases the utilisation of the resource shifts.

An optimal solution was also found by the CGDP 'Drexl' (7 early, 12 middle and 5 late). Using once again the resource lower bound graph show in Figure 7.2, we see that a minimum of 19 resource shifts of kinds early plus middle is needed and at least 5 resource shifts of the late kind is needed to plan all sections. This proves optimality of also the result produced by the CGDP 'Drexl' method, and even shows optimality of the late resource shift kind assignment of the CGDP 'Only Routes' solution. Introduction of the middle resource shift kind greatly reduces the number of total resource shifts needed (24 opposed to the 28 in case 2), but also greatly increases computation times needed. This is due to the very large number of possible assignment of sections on this middle resource shift kind.



(a) Optimal 'Only Routes' schedule



(b) Optimal schedule

Figure 7.3: Optimal schedules for the real-life instance case 3 with three resource shift kinds. Breaks are shown by the darker area inside sections. Fixed loading time before each section is shown by dashed line starting at a circle. The upper CGDP 'Only Routes' solution schedule uses 29 resource shifts, while the lower exact CGDP solution schedule only needs 24 resource shifts for the same sections.

Chapter 8

Discussion and Conclusions

We have discussed various algorithms for solving the Stage Two resource assignment problem of the two-stage Simultaneous Vehicle Routing and Crew Scheduling Problem with Breaks arising in delivery transportation planning of various logistic companies. We also presented results of these methods on adjusted benchmark instances from literature and a case from a real-life daily planning of a large Australian ORTEC customer. Although the additional flexibility of enabling truck drivers to switch trailers between sections/trips causes additional (difficult) synchronization constraints and therefore additional computation time, we have shown that this enables the number of (expensive) truck+driver resources to be reduced by $\sim 10\% - 25\%$ in our instances. The fixed loading time needed in the trailer routes but not in the truck-driver resource shifts seems to be mostly responsible for this reduction, although the increased flexibility for planning the required breaks also seems to help. Both in solving the Stage One and Stage Two problems, we have found some interesting results.

In Stage One, a break planning strategy which can predict the position of the breaks in Stage Two can really be effective in lowering the number of (expensive) resource shifts needed, although this added slack can cause the number of (less expensive) trailers to increase. We found that the *NotFresh* outperformed the *Fresh* break strategy on our adjusted Solomon/Homberger instances with a single resource shift kind. When more resource shift kinds are considered, or more realistic driving rules are posed, a different predictive break strategy may be needed in Stage One to have a similar effect. In Stage Two, we formulated and tested a number of different solution methods for the resource assignment problem with dependence sections. Column generation with explicit precedence constraints in the master problem was formulated like van den Akker et al. [1] to relieve the pricing problems of these nasty synchronization constraints. Solving the pricing problem exact with a labelling based algorithm showed very good results on medium-sized instances (100 - 200 customers), solving almost all instances to optimality with a single final ILP solver call. Although including a dominance relation was showed to be ineffective, other mathematical observations led to the speed-up of the second and higher pricing iterations. The number of labels generated in the first iteration explodes for large instances with wide time-windows, making the exact method intractable. Narrowing the time-windows of the sections before solving Stage Two, like in Drexl et al. [24], showed effective in reducing the number of labels and therefore computation time while sacrificing a bit of solution quality. However, when the number of sections per route increases, this variant seems to worsen in solution quality. Insertion based construction heuristics performed quite good, especially using the resource lower bound bottleneck seed strategy. The resource lower bound graph also proved to be a valuable tool in estimating the solution quality or even proving the optimality of heuristic generated solutions, even when multiple resource shift kinds are present. Finally, we presented a simple hybrid method based on column generation and the use of multiple constructing heuristics as

pricing. Preliminary results on this hybrid method look very promising, making such hybrid methods a very interesting topic for future research.

We make the following observations in comparing our results to recent advances in literature. The paper of Drexl et al. [24], which studies a similar simultaneous vehicle and crew routing and scheduling problem with breaks and similar two-stage decomposition, introduced the *Fresh* Stage One break strategy and the use of narrowing the section time-windows to make them independent in Stage Two. Concluded in that work is that allowing crew members to switch trailers during the planning does not have a savings potential in real-life long-haul logistic planning. However, it is suggested that in different settings, it could be beneficial. We show in this thesis that in the slightly different setting concerning the daily planning of a (supermarket) distribution problem with a single depot (single location where switching takes place), the savings potential does seem to be present. The difference is in our problem, the (geographical) central place where all the driver switching trailers takes place, unlike the problem of Drexl et al [24], where a considerable number of geographically spread relay stations are used where switching needs to take place. We suspect the savings potential to decrease as the number of depot/switching location increases, since trailer routes using these different depots need to be planned already in Stage One without driver assignment is known.

Groenendijk [36] and Baller [5] independently suggested to solve the difficult synchronization constraints arising in the Stage Two problem by means of a column-and-row generation method, which Baller also showed to give good results. Breström and Rönnqvist [11], Dohn et al. [21] and Rasmussen et al. [64] also avoided the explicit synchronization constraints in the master problem by using them in a branching scheme, therefore also avoid dealing with a more difficult pricing problem. However, we did solve the synchronization dependencies by explicitly including precedence constraints in the master problem, extending the work of van den Akker et al. [1], which introduces this explicit modelling for solving general parallel machine scheduling problems. This does make the pricing problem more difficult since additional shadow prices included in the reduced cost are now time-dependent. By combing work of Ioachim et al. [44] and Feillet [29], we formulated a method to solve this pricing problem exactly and showed it was able to solve most of our (medium sized) instances to optimality. However, in deriving this method, we made heavily use of various specific problem characteristics, such as the specific chain structure of the precedence constraints arising from the trailer routes. This might make our exact method less powerful in general synchronization problems, but it seems to work quite well for our problem.

Very recently, Kool (master's thesis, 2014) [50] studied feedback mechanisms in various two-stage decomposition solution methods, allowing the Stage Two solution to affect the (re)planning of Stage One and doing this iteratively. He also studied this on our problem, using our code of Stage One PCI and Stage Two exact CGDP method and extending the resource lower bound graph in a very interesting way to his framework for connecting the current Stage Two solution to the new replanning in Stage One, like adaptive guidance of Barratta et al. [7]. Kool tested his iterative feedback mechanism on our Adjusted Solomon 100 instances using *NotFresh* Stage One break strategy. For quite some instances, Kool [50] improved our best solutions by iteratively finding better Stage One solution which give an even better Stage Two solution. On average he managed to reduce the number of resource shift needed by 7.55% after running 10 iterations. This shows the use of such feedback mechanisms between Stage Two and Stage One could be very beneficial in producing good Stage One solutions of trailer routes with enough slack for breaks in Stage Two. Interestingly, in his solutions the number of trailers needed (although often less then our best solution) is still $\sim 14\%$ higher than the number of resource shifts needed, which is similar to our figures ($\sim 12\%$) of these instances. This shows similar benefit of allowing drivers to switch trailers between section.

8.1 Future Research

We conclude this thesis by making some suggestions for future research directions.

- **Impact of better Stage One algorithms**
  In our computational experiments, we used a very basic PCI heuristic construction method for planning multi-trip trailer routes in Stage One. However, in multi-trip vehicle routing problem with time-windows literature a tremendous number of more advanced methods are described and even in most commercial planning software, such more advanced method are used. We justify the use of a simple, less powerful Stage One heuristic by the observation that 'better', tightly planned trailer routes actually result in lower quality Stage Two resource assignment schedules. However, it is interesting to verify this impact exactly and to study strategies which both improve the Stage One and Stage Two solutions.

- **Impact of fixed loading time on assignment**
  We have seen that in our problem, the fixed loading time a trailer needs before starting a section but a driver does not, is very important in the benefit of drivers allowing to switch trailers between sections. Interesting is to measure the benefit switching has when drivers are actually needed to load a trailer. So then switching is allowed between trailers, but also the fixed loading time before the section needs to be assignment to the drivers. We expect the switching benefit in this case to be very minimal, suggesting switching is only useful when the fixed loading time is not needed to be assigned to drivers. Additionally, the impact of the length of the fixed loading time can be student. Preliminary results show that very large fixed loading times drastically increase the savings potential of drivers switching trailers between sections.

- **Multiple Depots, shuttling and balancing**
  In our computational experiments, we studied cases with only a single depot. Since most distribution companies need to simultaneously schedule orders from multiple depots, it is interesting to consider this case. Quite some literature considers advanced methods for solving the Stage One problem of trailer routes with multiple depots, and most of our Stage Two methods can be adapted to work with multiple depots (especially the exact CGDP method). However, we expect the savings potential to decrease, since drivers are now spread over multiple depots decrease the number of possibilities for switching. The use of *Shuttle vans* or *deadheading* could be introduced, allowing drivers to travel between depots without a trailer, increasing the number of possibilities. Real-life cases may also require to balance the number of trailers over these depots at the start and end of the planning horizon. All of the above additions to the problem are very important in real-life applications and it is therefore interesting to measure the performance of our methods on these cases.

- **Unforced/depot breaking**
  By using our simplified driving rule, breaks were only planned when needed (forced) en-route. Assumed is that the driver can stop his vehicle en-route to take this break. As noted by for instance Drexl and Prescott-Gagnon [23], it can be very beneficial for a driver to break earlier than needed (unforced). Also, as noted by Drexl et al. [24], (intermediate) depots could be a very natural place for drivers to take such breaks, especially if they need to wait there anyway. Our model lacks this possibility of breaking at the depot, resulting sometimes in unrealistic schedules. However, unforced breaks increases the number of possibilities tremendously, which increases the computation times of exact (pricing) methods. Heuristic methods or heuristic break strategies could be employed to reduce the computation times.

- **Full driving & work-time legislation/maximum shift duration**
  In this thesis, we considered a very simplified version of the full driver's legislation imposed by many countries around the world. Since schedules in real-life should be legal, it is in real-life cases important to use the full driving and work time legislation, as well as taking into account maximum resource shift durations. Including the full set of rules is very difficult, especially in our Stage Two problem, but really needed in practice. Possible use of full ESPPRC model of Drexl and Prescott-Gagnon [23] or additional piecewise linear functions in the labels like Kok et al. [49] may be needed. The interaction between synchronization constraints (associated linear node costs) and seemingly opposing maximum shift duration/working time restrictions could be extremely difficult but very interesting and important.

- **Resource shift kind SPCI strategies**
  In our results of the real-life cases, we showed that our column generation method with labelling pricing could be also used to solve problems with multiple resource shift kinds. It should be very interesting to adapt the SPCI construction heuristic procedures to be able to plan sections on multiple resource shift kinds as well.

- **Large Neighbourhood Search: Destruction operators**
  Our construction heuristic procedures could also be used in a more advanced local search iterative improvement method. Currently popular in the vehicle routing literature is the use of Large Neighbourhood Search with destruction operators [3, 24, 59, 60, 66]. After having constructed a feasible solution, this solution is partially 'destroyed' by a destruction operator, which removes some of the sections from the schedule. Our construction heuristics, possibly using multiple strategies, should be used to re-insert the sections again in the schedule.

- **Heuristic Pricing by Construction with Labels**
  We saw the number of labels generated by our first iteration exact pricing to explode for large instances. Although we investigated a very simple hybridization of column generation with heuristic pricing by the construction heuristics, hybridization could also be done by taking the exact labelling algorithm and limiting the labels actually being explored.

  The following construction paradigm could be used to limit the number of labels:
  Calculate labels of all single section $o - d$ paths. Add all negative reduced single section-resource schedules to master problem and choose lowest cost section. Next calculate/update labels for all two section $o - d$ paths containing the previous best section (calculating effectively the best section to insert to the schedule). Add all negative reduced two section-resource schedules to master problem and choose lowest cost two section-resource shift. Repeat calculating the best section to insert next until the total reduced cost is not improving.

  This method seems to guarantee the finding of a feasible negative reduced cost single resource schedule if it exists, although not all labels are searched for the best single resource shift to be added to the master problem. This speeds up the pricing iterations, making it very useful in large instances. Other such *Matheheuristic* (combination of metaheuristics and mathematical programming, see [14]) procedures could also be very interesting for solving large instances.

- **Incremental Topological Ordering for construction methods**
  In our Stage Two full scheduling construction methods, at every insertion topological sorting is important to evaluate the feasibility of the precedence constraints in the schedule. Recent advances in literature on topological sorting suggest the use of *Incremental* Topological Ordering, which keeps track of the topological ordering of the precedence network

and requiring only small updates when dependency arcs are inserted (when a route section is planned in a resource shift).

- **Feedback between Stage One and Stage Two**
  Results of Kool (master's thesis, 2014) [50] on using feedback mechanisms in our two-stage decomposition are very promising. The use of such methods for improving the quality of the Stage One solution and better break prediction by feedback could be explored further.

- **Branch-and-price**
  In this thesis, we solved the CG methods by solving the RMP directly as ILP after LP was solved (at root node). Although we solved almost all instances to optimality, a few instances had an ILP gap. Our data does not suggest the number of instances with ILP gap to grow with instance size, still branch-and-prize methods may be needed for solving larger and more complex instances. In Section 6.5, we made some suggestions on making the ILP formulation stronger by adding *weighted* columns representing a *weighted* combination of multiple fractionally selected columns. Branching on narrowed time-windows could also be considered.

- **Solving the full SVCRSP by column(-and-row) generation**
  We formulated our full problem as large ILP in Section 2.3. Although this formulation probably needs a very large number of constraints and variables for the medium sized instances we considered, it could be possible to solve this problem directly without decomposition as column generation or possibly column-and-row generation [56]. One type of column could represent a single trailer route, while another could represent a single resource shift schedule. Sections in both type of columns should be directly synchronized by the master problem. Since the number of possible sections is combinatorially large, they could possibly be generated as rows. The efficiently of this full solution method is questionable, since direct synchronization constraints to match exactly start-of-service times in two columns are possibly resulting in highly fractional RMP solutions and possibly not so strong RMP LP-bounds.

- **Parallel/Distributed Solution methods**
  Computers are nowadays equipped with an increasing number of CPU-cores, enabling speed-ups to algorithms if parallizing can be done efficiently. CG methods could benefit from distributing different pricing subproblems, different SPCI strategy runs, or even distributing parts of the exact ESPPRCNC labelling search tree.

- **Pulse framework for ESPPRCNC?**
  Very recently, a new framework called *Pulse* was developed by Lozano et al. [53] to solve Elementary Shortest Path Problems with Resource Constrains. It does not uses dominance relations, but bounding and pruning techniques in order to find good solutions quickly. Authors state its performance is on average better than currently the most advanced ESPRRC solution methods. Since we also did not use the dominance relations in our problem, it would be very interesting to see if our ESPPRCNC labelling algorithm can be improved by incorporating bounding and pruning techniques inspired by the new Pulse framework.

References

[1] J.M. van den Akker, J.A. Hoogeveen, and J.W. van Kempen, *Using column generation to solve parallel machine scheduling problems with minmax objective functions*, Journal of Scheduling **15** (2012), no. 6, 801–810.

[2] Nabila Azi, Michel Gendreau, and Jean-Yves Potvin, *An exact algorithm for a vehicle routing problem with time windows and multiple use of vehicles*, European Journal of Operational Research **202** (2010), no. 3, 756 – 763.

[3] ———, *An adaptive large neighborhood search for a vehicle routing problem with multiple routes*, Computers & Operations Research **41** (2014), no. 0, 167 – 173.

[4] Michael Ball, Cynthia Barnhart, George Nemhauser, and Amedeo Odoni, *Chapter 1 air transportation: Irregular operations and control*, Transportation (Cynthia Barnhart and Gilbert Laporte, eds.), Handbooks in Operations Research and Management Science, vol. 14, Elsevier, 2007, pp. 1 – 67.

[5] A.C. Baller, *Column-and-row generation and dependency between columns*, Master's thesis, VU, Amsterdam, 2013.

[6] Cynthia Barnhart, Ellis L. Johnson, George L. Nemhauser, Martin W. P. Savelsbergh, and Pamela H. Vance, *Branch-and-Price: Column Generation for Solving Huge Integer Programs*, Operations Research **46** (1996), 316–329.

[7] M. Battarra, M. Monaci, and D. Vigo, *An adaptive guidance approach for the heuristic solution of a minimum multiple trip vehicle routing problem*, Computers & Operations Research **36** (2009), no. 11, 3041 – 3050.

[8] J. E. Beasley and N. Christofides, *An algorithm for the resource constrained shortest path problem*, Networks **19** (1989), no. 4, 379–394.

[9] Narath Bhusiri, Ali Gul Qureshi, and Eiichi Taniguchi, *The trade-off between fixed vehicle costs and time-dependent arrival penalties in a routing problem*, Transportation Research Part E: Logistics and Transportation Review **62** (2014), no. 0, 1 – 22.

[10] Olli Bräysy and Michel Gendreau, *Vehicle Routing Problem with Time Windows, Part I: Route Construction and Local Search Algorithms*, Transportation Science **39** (2005), no. 1, 104–118.

[11] David Bredström and Mikael Rönnqvist, *Combined vehicle routing and scheduling with temporal precedence and synchronization constraints*, European Journal of Operational Research **191** (2008), no. 1, 19 – 31.

[12] Ann Melissa Campbell and Martin Savelsbergh, *Efficient insertion heuristics for vehicle routing and scheduling problems*, Transportation Science **38** (2004), no. 3, 369–378.

[13] Alberto Caprara, Leo Kroon, Michele Monaci, Marc Peeters, and Paolo Toth, *Chapter 3 passenger railway optimization*, Transportation (Cynthia Barnhart and Gilbert Laporte, eds.), Handbooks in Operations Research and Management Science, vol. 14, Elsevier, 2007, pp. 129 – 187.

[14] Marco Caserta and Stefan Voß, *Metaheuristics: Intelligent problem solving*, Matheuristics (Vittorio Maniezzo, Thomas Stützle, and Stefan Voß, eds.), Annals of Information Systems, vol. 10, Springer US, 2010, pp. 1–38 (English).

[15] William Cook, *Fifty-plus years of combinatorial integer programming*, 50 Years of Integer Programming 1958-2008 (Michael Jünger, Thomas M. Liebling, Denis Naddef, George L. Nemhauser, William R. Pulleyblank, Gerhard Reinelt, Giovanni Rinaldi, and Laurence A. Wolsey, eds.), Springer Berlin Heidelberg, 2010, pp. 387–430.

[16] J.-F. Cordeau, G. Desaulniers, J. Desrosiers, M. M. Solomon, and F. Soumis, *The Vehicle Routing Problem*, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2001, pp. 157–193.

[17] Jean-François Cordeau, Gilbert Laporte, Martin W.P. Savelsbergh, and Daniele Vigo, *Chapter 6 vehicle routing*, Transportation (Cynthia Barnhart and Gilbert Laporte, eds.), Handbooks in Operations Research and Management Science, vol. 14, Elsevier, 2007, pp. 367 – 428.

[18] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein, *Introduction to Algorithms, Third Edition*, 3rd ed., The MIT Press, 2009.

[19] Guy Desaulniers and Mark D. Hickman, *Chapter 2 public transit*, Transportation (Cynthia Barnhart and Gilbert Laporte, eds.), Handbooks in Operations Research and Management Science, vol. 14, Elsevier, 2007, pp. 69 – 127.

[20] Jacques Desrosiers and Marco E. Lübbecke, *A primer in column generation*, Column Generation (Guy Desaulniers, Jacques Desrosiers, and Marius M. Solomon, eds.), Springer US, 2005, pp. 1–32 (English).

[21] Anders Dohn, Matias Sevel Rasmussen, and Jesper Larsen, *The vehicle routing problem with time windows and temporal dependencies*, Networks **58** (2011), no. 4, 273–289.

[22] Michael Drexl, *Synchronization in Vehicle Routing – A Survey of VRPs with Multiple Synchronization Constraints*, Transportation Science **46** (2012), no. 3, 297–316.

[23] Michael Drexl and Eric Prescott-Gagnon, *Labelling algorithms for the elementary shortest path problem with resource constraints considering EU drivers' rules*, Logistics Research **2** (2010), no. 2, 79–96.

[24] Michael Drexl, Julia Rieck, Thomas Sigl, and Bettina Press, *Simultaneous Vehicle and Crew Routing and Scheduling for Partial- and Full-Load Long-Distance Road Transport*, BuR - Business Research **6** (2013), no. 2, 242–264.

[25] Moshe Dror, *Note on the Complexity of the Shortest Path Models for Column Generation in VRPTW*, Operations Research **42** (1994), no. 5, 977–978.

[26] European Union, *Directive 2002/15/EC of the European Parliament and of the Council of 11 March 2002 on the organisation of the working time of persons performing mobile road transport activities*, (2002), Available at http://eur--lex.europa.eu/legal--content/EN/ALL/?uri=CELEX:32002L0015. Accessed May 2, 2014.

[27] ———, *Regulation (EC) No 561/2006 of the European Parliament and of the Council of 15 March 2006 on the harmonisation of certain social legislation relating to road transport and amending Council Regulations (EEC) No 3821/85 and (EC) No 2135/98 and repealing Council Regulation (EEC) No 3820/85*, (2006), Available at http://eur--lex.europa.eu/legal--content/EN/ALL/?uri=CELEX:32006R0561. Accessed May 2, 2014.

[28] Dominique Feillet, *A tutorial on column generation and branch-and-price for vehicle routing problems*, 4OR **8** (2010), no. 4, 407–424 (English).

[29] Dominique Feillet, Pierre Dejax, Michel Gendreau, and Cyrille Gueguen, *An exact algorithm for the elementary shortest path problem with resource constraints: Application to some vehicle routing problems*, Networks **44** (2004), no. 3, 216–229.

[30] Richard Freling, Dennis Huisman, and Albert P. M. Wagelmans, *Models and algorithms for integration of vehicle and crew scheduling*, Journal of Scheduling **6** (2003), no. 1, 63–85 (English).

[31] Hermann Gehring and Jörg Homberger, *A parallel hybrid evolutionary metaheuristic for the vehicle routing problem with time windows*, Proceedings of EUROGEN99, vol. 2, 1999, pp. 57–64.

[32] Michel Gendreau and Jean-Yves Potvin, *Tabu search*, Handbook of Metaheuristics (Michel Gendreau and Jean-Yves Potvin, eds.), International Series in Operations Research & Management Science, vol. 146, Springer US, 2010, pp. 41–59 (English).

[33] Asvin Goel, *Truck driver scheduling in the european union*, Transportation Science **44** (2010), no. 4, 429–441.

[34] Asvin Goel, Claudia Archetti, and Martin Savelsbergh, *Truck driver scheduling in Australia*, Computers & Operations Research **39** (2012), no. 5, 1122 – 1132.

[35] Asvin Goel and Leendert Kok, *Truck driver scheduling in the united states*, Transportation Science **46** (2012), no. 3, 317–326.

[36] Gerben Groenendijk, *Resource assignment problem with sections*, Master's thesis, VU, Amsterdam, 2012.

[37] Timo Gschwind, *A Comparison of Column-Generation Approaches to the Vehicle Routing Problem with Time Windows and Temporal Synchronized Pickup and Delivery* , Tech. Report LM-2014-01, Chair of Logistics Management, Gutenberg School of Management and Economics, Johannes Gutenberg University Mainz, Mainz, Germany, 2014.

[38] Timo Gschwind and Stefan Irnich, *Effective Handling of Dynamic Time Windows and Synchronization with Precedences for Exact Vehicle Routing*, Technical Report LM-2012-05, Chair of Logistics Management, Johannes Gutenberg University Mainz, Mainz, Germany, October 2012, Available online at http://logistik.bwl.uni-mainz.de/158.php.

[39] Inc. Gurobi Optimization, *Gurobi optimizer reference manual*, 2015, http://www.gurobi.com.

[40] Pierre Hansen, Nenad Mladenović, Jack Brimberg, and José A. Moreno Pérez, *Variable neighborhood search*, Handbook of Metaheuristics (Michel Gendreau and Jean-Yves Potvin, eds.), International Series in Operations Research & Management Science, vol. 146, Springer US, 2010, pp. 61–86 (English).

[41] F. Hernandez, D. Feillet, R. Giroudeau, and O. Naud, *A new exact algorithm to solve the multi-trip vehicle routing problem with time windows and limited duration*, 4OR **12** (2014), no. 3, 235–259 (English).

[42] Dennis Huisman, Richard Freling, and Albert P. M. Wagelmans, *Multiple-depot integrated vehicle and crew scheduling*, Transportation Science **39** (2005), no. 4, 491–502.

[43] Irina Ioachim, Jacques Desrosiers, François Soumis, and Nicolas Bélanger, *Fleet assignment and routing with schedule synchronization constraints*, European Journal of Operational Research **119** (1999), no. 1, 75 – 90.

[44] Irina Ioachim, Sylvie Gélinas, François Soumis, and Jacques Desrosiers, *A dynamic programming algorithm for the shortest path problem with time windows and linear node costs*, Networks **31** (1998), no. 3, 193–204.

[45] Stefan Irnich, *Resource extension functions: properties, inversion, and generalization to segments*, OR Spectrum **30** (2008), no. 1, 113–148 (English).

[46] Stefan Irnich and Guy Desaulniers, *Shortest path problems with resource constraints*, Column Generation (Guy Desaulniers, Jacques Desrosiers, and MariusM. Solomon, eds.), Springer US, 2005, pp. 33–65.

[47] Stefan Irnich and Daniel Villeneuve, *The Shortest-Path Problem with Resource Constraints and k-Cycle Elimination for $k \geq 3$*, INFORMS Journal on Computing **18** (2006), no. 3, 391–406.

[48] A. L. Kok, C. M. Meyer, H. Kopfer, and J. M. J. Schutten, *A Dynamic Programming Heuristic for the Vehicle Routing Problem with Time Windows and European Community Social Legislation*, Transportation Science **44** (2010), no. 4, 442–454.

[49] A.L. Kok, E.W. Hans, J.M.J. Schutten, and W.H.M. Zijm, *A dynamic programming heuristic for vehicle routing with time-dependent travel times and required breaks*, Flexible Services and Manufacturing Journal **22** (2010), no. 1-2, 83–108 (English).

[50] W. Kool, *Optimization of two-phase methods using simple feedback mechanisms*, Master's thesis, VU, Amsterdam, 2014.

[51] Rahma Lahyani, Mahdi Khemakhem, and Frédéric Semet, *Rich vehicle routing problems: From a taxonomy to a definition*, European Journal of Operational Research **241** (2015), no. 1, 1 – 14.

[52] Gilbert Laporte, *Fifty years of vehicle routing*, Transportation Science **43** (2009), no. 4, 408–416.

[53] Leonardo Lozano, Daniel Duque, and Andrés L. Medaglia, *An Exact Algorithm for the Elementary Shortest Path Problem with Resource Constraints*, Transportation Science (2015).

[54] Marco E. Lübbecke and Jacques Desrosiers, *Selected Topics in Column Generation*, Operations Research **53** (2005), no. 6, 1007–1023.

[55] Frank Meisel and Herbert Kopfer, *Synchronized routing of active and passive means of transport*, OR Spectrum **36** (2014), no. 2, 297–322.

[56] Íbrahim Muter, Ş.Ílker Birbil, and Kerem Bülbül, *Simultaneous column-and-row generation for large-scale linear programs with column-dependent-rows*, Mathematical Programming **142** (2013), no. 1-2, 47–82 (English).

[57] Alexander G. Nikolaev and Sheldon H. Jacobson, *Simulated annealing*, Handbook of Metaheuristics (Michel Gendreau and Jean-Yves Potvin, eds.), International Series in Operations Research & Management Science, vol. 146, Springer US, 2010, pp. 1–39 (English).

[58] Sophie N. Parragh and Verena Schmid, *Hybrid column generation and large neighborhood search for the dial-a-ride problem*, Computers & Operations Research **40** (2013), no. 1, 490 – 497.

[59] David Pisinger and Stefan Ropke, *A general heuristic for vehicle routing problems*, Computers & Operations Research **34** (2007), 2403–2435.

[60] David Pisinger and Stefan Ropke, *Large neighborhood search*, Handbook of Metaheuristics (Michel Gendreau and Jean-Yves Potvin, eds.), International Series in Operations Research & Management Science, vol. 146, Springer US, 2010, pp. 399–419 (English).

[61] Jean-Yves Potvin and Jean-Marc Rousseau, *A parallel route building algorithm for the vehicle routing and scheduling problem with time windows*, European Journal of Operational Research **66** (1993), no. 3, 331 – 340.

[62] Eric Prescott-Gagnon, Guy Desaulniers, Michael Drexl, and Louis-Martin Rousseau, *European Driver Rules in Vehicle Routing with Time Windows*, Transportation Science **44** (2010), no. 4, 455–473.

[63] Luigi Di Puglia Pugliese and Francesca Guerriero, *A survey of resource constrained shortest path problems: Exact solution approaches*, Networks **62** (2013), no. 3, 183–200.

[64] Matias Sevel Rasmussen, Tor Justesen, Anders Dohn, and Jesper Larsen, *The home care crew scheduling problem: Preference-based visit clustering and temporal dependencies*, European Journal of Operational Research **219** (2012), no. 3, 598 – 610, Feature Clusters.

[65] Giovanni Righini and Matteo Salani, *Symmetry helps: Bounded bi-directional dynamic programming for the elementary shortest path problem with resource constraints*, Discrete Optimization **3** (2006), no. 3, 255 – 273, Graphs and Combinatorial Optimization The Cologne/Twente Workshop on Graphs and Combinatorial Optimization.

[66] Stefan Ropke and David Pisinger, *An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows*, Transportation Science **40** (2006), no. 4, 455–472.

[67] Marius M. Solomon, *Algorithms for the vehicle routing and scheduling problems with time window constraints*, Operations Research **35** (1987), no. 2, 254–265.

[68] Remy Spliet and Adriana Gabor, *The Time Window Assignment Vehicle Routing Problem*, Tech. Report EI 2012-07, Econometric Institute, Erasmus University Rotterdam, April 2012, Accepted to be published in Transportation Science (2014).

[69] Mariam Tagmouti, Michel Gendreau, and Jean-Yves Potvin, *Arc routing problems with time-dependent service costs*, European Journal of Operational Research **181** (2007), no. 1, 30 – 39.

[70] François Vanderbeck and LaurenceA. Wolsey, *Reformulation and decomposition of integer programs*, 50 Years of Integer Programming 1958-2008 (Michael Jünger, Thomas M. Liebling, Denis Naddef, George L. Nemhauser, William R. Pulleyblank, Gerhard Reinelt, Giovanni Rinaldi, and Laurence A. Wolsey, eds.), Springer Berlin Heidelberg, 2010, pp. 431–502 (English).

[71] Thibaut Vidal, Teodor Gabriel Crainic, Michel Gendreau, and Christian Prins, *Heuristics for multi-attribute vehicle routing problems: A survey and synthesis*, European Journal of Operational Research **231** (2013), no. 1, 1 – 21.

Appendix A

Insertion Costs

A.1 Stage One: PCI

In the Stage One PCI algorithm, we use the following insertion cost $c(i,j,k)$ for inserting unplanned customer $k \in \tilde{U}$ between consecutive nodes $i \in \mathcal{V}_C \cup \{o\}$ and $j \in \mathcal{V}_C \cup \{d\}$ already planned in the same trip:

$$c_1(i,j,k) = t_{ik} + t_{kj} - t_{ij}, \tag{A.1}$$

$$c_2(i,j,k) = \overline{T}_i - \min\left\{\overline{T}_i, \overline{T}'_k - t_{ik} - T_i^{\text{serv}}\right\}, \tag{A.2}$$

$$c_3(i,j,k) = \max\left\{\underline{T}_j, \underline{T}'_k + T_k^{\text{serv}} + t_{kj}\right\} - \underline{T}_j, \tag{A.3}$$

$$c(i,j,k) = c_1(i,j,k) + \beta\left[c_2(i,j,k) + c_3(i,j,k)\right], \tag{A.4}$$

with $\underline{T}_i$, $\overline{T}_i$ the current earliest/latest start-of-service time at planned node i, $\underline{T}'_k$, $\overline{T}'_k$ the with IsFeasibleInsert calculated earliest/latest start-of-service time at unplanned customer k if inserted between i and j and β a weight factor. These insertion costs are based on those used by Solomon [67]. Additional travel time is penalized, as well as changes in the new earliest (latest) start-of-service time at the next (previous) node. Costs c_2 and c_3 are high when inserting k into an empty route, making this very expensive. We set $\beta = \frac{1}{2}$, which showed good results.