Dissertation

Topics in Integrated Vehicle and Crew Scheduling in Public Transport

Dipl. Wirt.-Inform. Ingmar Steinzen

Schriftliche Arbeit zur Erlangung des akademischen Grades
doctor rerum politicarum (dr. rer. pol.)
im Fach Wirtschaftsinformatik

eingereicht an der
Fakultät für Wirtschaftswissenschaften der
Universität Paderborn
Paderborn, im Juli 2007

Datum der mündlichen Prüfung: 23.11.2007

Gutachter:
1. Prof. Dr. Leena Suhl
2. Prof. Dr. Knut Haase

It is not the mountain
we conquer – but ourselves.
Sir Edmund Hillary

Acknowledgements

The thesis in front of you is a result of the research that I conducted as member of the *International Graduate School of Dynamic Intelligent Systems* and the *Decision Support & Operations Research Lab (DSOR)* at the University of Paderborn. This dissertation would not have been possible to complete in the last three years without the precious support of so many people.

First and foremost, I am very grateful to my supervisor Prof. Dr. Leena Suhl for giving me the opportunity to work in her working group and to write this thesis. It was her valuable guidance, support, and ever-friendly nature that added a great deal to the completion of this work.

Furthermore, I thoroughly enjoyed the friendly atmosphere and innumerable interesting discussions that I had with my colleagues at the working group. In particular, I would like to thank Vitali Gintner for providing such a great company and the general support during the first two years. Moreover, several students and student assistants made valuable contributions in the implementation of the optimization system. Among others I wish to thank Bastian, Boris, and Viktor.

I am indebted to the International Graduate School of Dynamic Intelligent Systems, its director PD Dr. Eckhard Steffen, and its support staff who accompanied and financially supported my work. I hope many future students will have the opportunity to participate in their excellent PhD program.

Finally, I wish to sincerely thank my father and his wife for their constant support and encouragement in all my private and professional endeavors. Most importantly, I thank Andrea. I am very fortunate to have the opportunity to share everything with you that worries me, bothers me, delights me, or just makes me laugh.

Ingmar Steinzen
Paderborn, July 2007

Contents

1. Introduction

Vehicle and crew scheduling are two major planning problems arising in public bus transport companies. Briefly stated, these problems aim at assigning vehicle itineraries to scheduled trips and crews itineraries to tasks resulting from the vehicle schedule.

For several years now, Operations Research (OR) has been successful in solving vehicle and crew scheduling optimization problems in public transport. Several commercial software systems have been developed and used by public transport companies to support planning and to run their operation. Many companies in Germany and other countries have adapted these tools, mainly for three reasons:

- These companies face an increasing cost pressure and competition due to market deregulation. Furthermore, the level of subsidy is being gradually reduced. As a consequence, efficient utilization of available resources is more and more important.

- Scheduling vehicles and crews has become increasingly complex due to larger problem sizes and an increased complexity of labor rules.

- The power of computers and algorithms has advanced remarkably.

The main objective of a vehicle and crew schedule is to offer a given service that allows passengers to travel easily at a low fare while minimizing asset and operating costs. Traditionally, both planning steps have been approached sequentially where vehicle schedules are determined before crew schedules. In this thesis, however, we focus on the integrated consideration of vehicle and crew scheduling. The integration of both planning steps discloses additional flexibility that can lead to gains in efficiency compared to sequential planning.

However, we do not only focus on how to conduct operations at minimum cost but also on another aspect which is related to the quality of crew schedules. In particular, we consider the case where schedules consist of trips serviced every day as well as trips that do not repeat daily. The traditional approach to crew scheduling usually produces irregular crew schedules which are undesired in practice. Regularity is an important aspect for crew schedules in public transport

since regular solutions are less error-prone and are easier to implement and to manage.

In the remainder of this chapter, we describe the planning process of a public transport company (Section 1.1), the integrated treatment of vehicle and crew scheduling in Section 1.2, and the effect of irregular timetables on the regularity of crew schedules (Section 1.3). In Sections 1.4 and 1.5 we discuss some well-known combinatorial optimization problems and techniques, respectively. The chapter is concluded in Section 1.6 with a discussion of the scope and purpose of the thesis.

1.1. Planning Process of Public Transport Companies

Most public transport companies aim to offer a service of good quality that allows passengers to travel easily at a low fare while utilizing the resources at their disposal as efficient as possible. The complete planning process in public transport is very complex and is computationally not tractable as a whole. Consequently, it is traditionally divided into a *strategic*, *tactical*, and *operational* phase. Each phase is further split into several subproblems that are successively solved. Figure 1.1 depicts the entire planning process. In the following, we briefly describe each subproblem.

In strategic planning, we are concerned with network design and line planning. The planning horizon is typically several years. It is generally assumed that an origin-destination (O-D) matrix is available for strategic planning. Each entry in the matrix gives the number of passengers that want to travel between any two points in the network by time of the day. Based on this demand data, the *network design problem* is to determine the links of the network such that construction costs are minimized. Of course, these links must provide sufficient capacity to transport the estimated number of passengers. The *line planning problem* consists of choosing a set of line routes (see Figure 1.2) and their frequencies for a given transportation network such that passenger demand can be satisfied. There are two conflicting objectives: maximize passenger comfort and minimize operating costs of the lines. Passenger comfort can be measured by the total transit time or the number of direct connections.

The tactical planning phase aims at timetable construction. The *timetabling problem* is solved on a seasonal basis for given line routes, their frequencies, traveling times along the lines, and any potential layover times at stations. The task is to convert the desired frequency of a line into a detailed timetable. A timetable

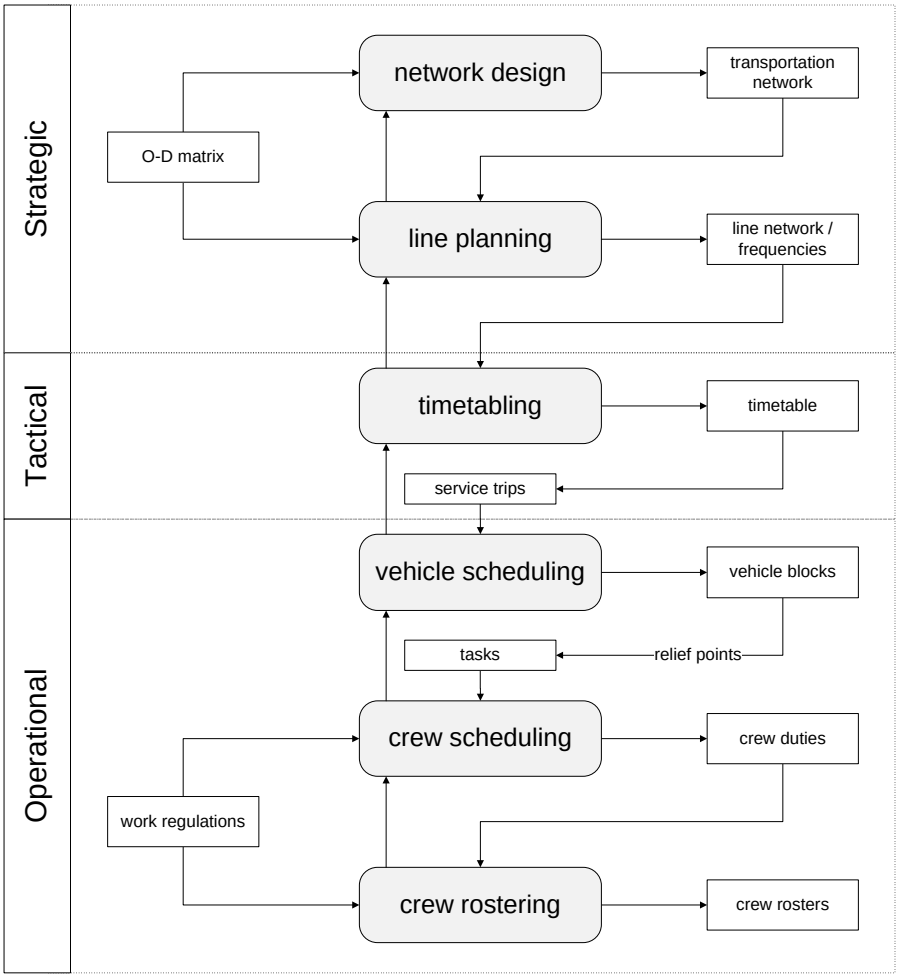

Figure 1.1.: Planning Process of a Public Transport Company

corresponds to a set of *(service) trips* with start and end locations and times. A common objective is maximum synchronization of trips such that transfers within the network are well-timed and passengers can transfer between lines with minimum waiting time. Similar to [Desaulniers and Hickman, 2006] we include the timetabling step in the tactical level (and not in the operational phase), since it is primarily focused on service quality (and not on cost minimization).

At the operational level, planning is concerned with constructing vehicle and crew itineraries that minimize total costs while considering all operational constraints and work regulations. In other words, this phase solves the problem of how to conduct operations to offer the proposed service at minimum cost.

The *vehicle scheduling problem* is to assign vehicles to trips resulting in vehicle

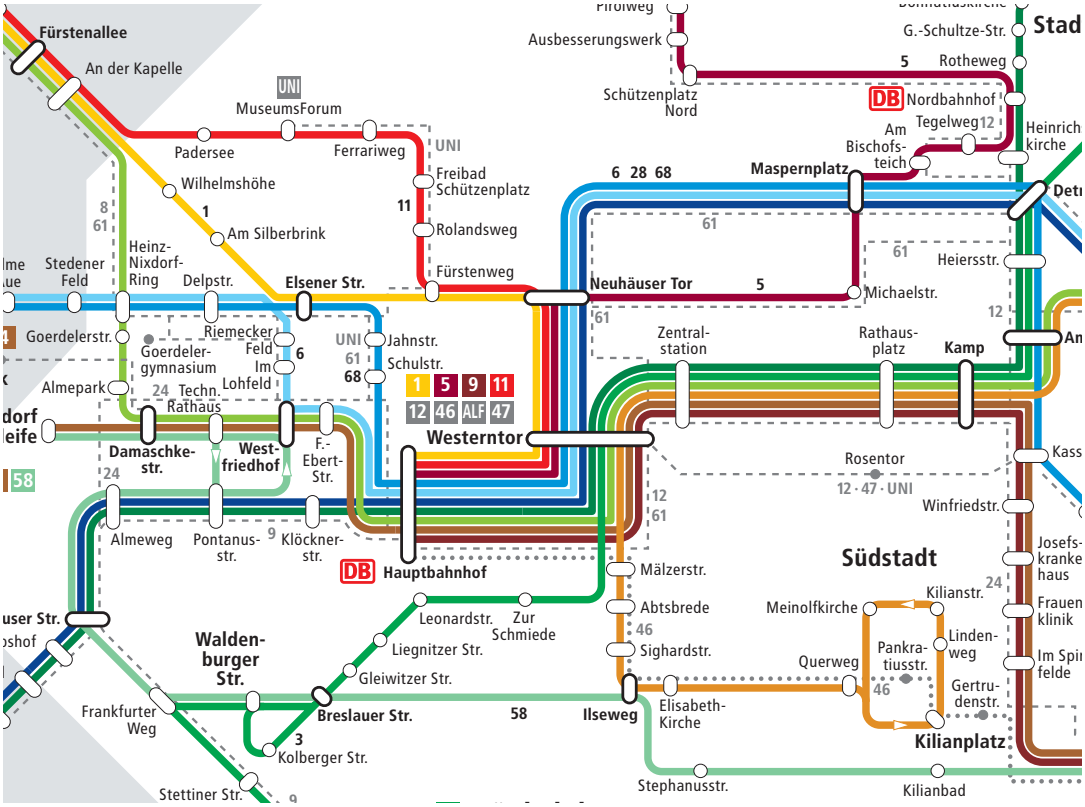

Figure 1.2.: Excerpt from line network of PaderSprinter, Paderborn (Germany)

blocks. A vehicle block is the daily schedule for one vehicle. Each vehicle block is a sequence of tasks where each task needs to be covered by a crew duty in the *crew scheduling problem*. A crew duty is the workload of an anonymous driver for one day that must satisfy a number of work regulations. The *crew rostering problem* consists of constructing long-term (monthly) work schedules (called crew rosters) from short-term (daily) crew duties considering several work regulations. Unlike crew duties, crew rosters are assigned to individual drivers. The planning horizon for both vehicle and crew scheduling is usually one day, while crew rostering problems are solved on a monthly basis. In the following subsections, we describe vehicle and crew scheduling in more detail since this thesis is mainly focused on these problems.

1.1.1. Vehicle Scheduling

A bus *depot* is basically a maintenance and storage facility where buses may be parked and serviced when not in use. A depot may have a maximum storage capacity. Vehicles start and terminate their daily schedule in a depot. A fleet

may consist of different *vehicle types* that differ in capacity, speed or, equipment. In practice, there are often trips that must be operated by a specific vehicle type or a subset of vehicle types. Furthermore, the number of vehicles in a facility of a particular type must be equal at the beginning and end of a day. As a consequence, we will treat a combination of facility and vehicle type as depot.

A timetable defines a set of trips that are used to carry passengers. Generally, it is assumed that start and end location for all trips are fixed as well as their start and end times. Given a set of timetabled trips, the vehicle scheduling problem (VSP) can be stated as follows: find an assignment of trips to vehicles such that

- each trip is assigned exactly once,
- each vehicle performs a feasible sequence of trips,
- each sequence starts and ends at the same depot, and
- asset and operational costs are minimized.

Two trips are said to be *compatible* if they can be covered by the same vehicle. Trips operated in sequence by the same vehicle are linked by *deadheads*. Deadheads are vehicle movements or idle times (or both) without carrying passengers. A vehicle is idle if it stands (idle) at a location other than the depot. A *vehicle block* is a sequence of compatible trips that starts with a *pull-out trip* and ends with a *pull-in trip*. A pull-out trip connects the depot with the start location of the first trip while a pull-in trip moves a vehicle from the end location of the last trip to the depot. A daily schedule for one vehicle can thus include several vehicle blocks. Figure 1.3 depicts an example of a daily schedule for one vehicle with two blocks.

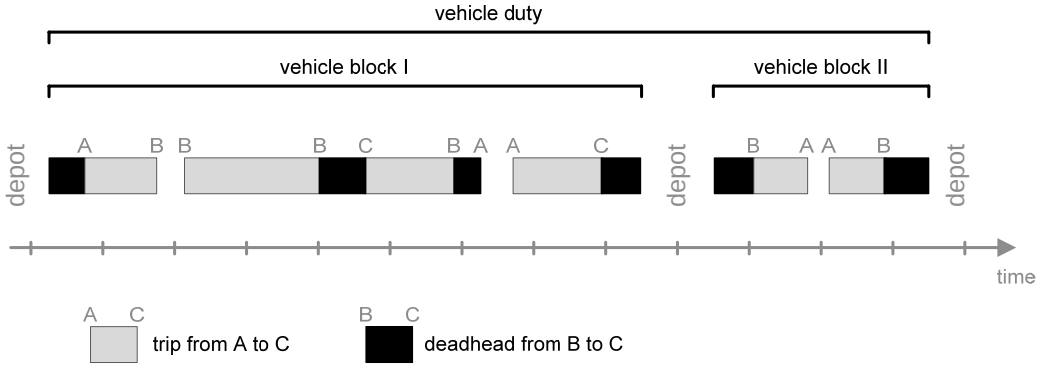

Figure 1.3.: Schedule of one vehicle consisting of two blocks

While asset costs usually correspond to investment and maintenance costs, public transport companies define operational cost in different ways such as distance driven without passengers or waiting time. In most practical situations, companies try to minimize their asset costs first and leave operational cost minimization as a secondary objective.

In the case that there is only one depot, a homogeneous fleet, and no route constraints, we have the standard *single depot vehicle scheduling problem* (SDVSP). If there are multiple depots, we have the *multiple-depot vehicle scheduling problem* (MDVSP). In the latter case, each vehicle is assigned to a given depot and, possibly, some trips have to be serviced by vehicles from a certain subset of depots.

It is well known that SDVSP corresponds to a minimum cost flow problem that can be solved in polynomial time while its multiple-depot counterpart MDVSP is NP-hard (see [Bertossi et al., 1987]). Additionally, Löbel [Löbel, 1997] shows that an ϵ-approximation of MDVSP is also NP-hard. The complexity of a specific problem instance mainly depends on the number of depots, the number of trips, and the number of potential deadheads. Of course, additional constraints, e.g. depot capacities or route time constraints, can be imposed that make instances more challenging.

1.1.2. Crew Scheduling

Crew scheduling plays an important role in the operational planning process since crew costs generally dominate vehicle costs (see [Bodin et al., 1983], [Leuthardt, 1998]). Instead of assigning trips to vehicles as in the preceding phase, we now assign tasks to crews. A basic assumption is that all crews are equal since individual crew members are not considered.

The crew scheduling problem (CSP) is defined as follows: find a set of *duties* for a given set of *tasks* such that

- each task is covered by a duty that can be performed by a single driver,
- each duty satisfies a wide variety of federal laws, safety regulations, and (collective) in-house agreements, and
- labor costs are minimized.

A task is a sequence of activities (such as performing trips or deadheading) between two consecutive *relief points* and represents an elementary portion of work that can be assigned to a driver. A relief point defines a location and time where a driver may change his vehicle. In traditional crew scheduling, i.e., a vehicle first

- crew second approach, relief points subdivide vehicle blocks that were obtained in the preceding phase.

A *piece of work* is a sequence of tasks without a (long) break for which a driver stays with the same vehicle. Consequently, duties are composed of pieces of work separated by breaks. Duties start with a *sign-on* and end with a *sign-off* activity. Typically, there are several *duty types* in practical applications, each with a different rule set. Examples of working rules are minimum/maximum driving time, minimum break length, allowed start and end time, or maximum spread (length) of a duty. Moreover, companies often limit the (minimum/maximum) number or percentage of duties of a particular type. For instance, the percentage of split duties that have two pieces of work - one in the early morning and another in the late afternoon with a long break in the middle - is often restricted. Figure 1.4 shows the schedule of one crew that consists of two pieces of work. Note that the first two tasks remain unassigned.

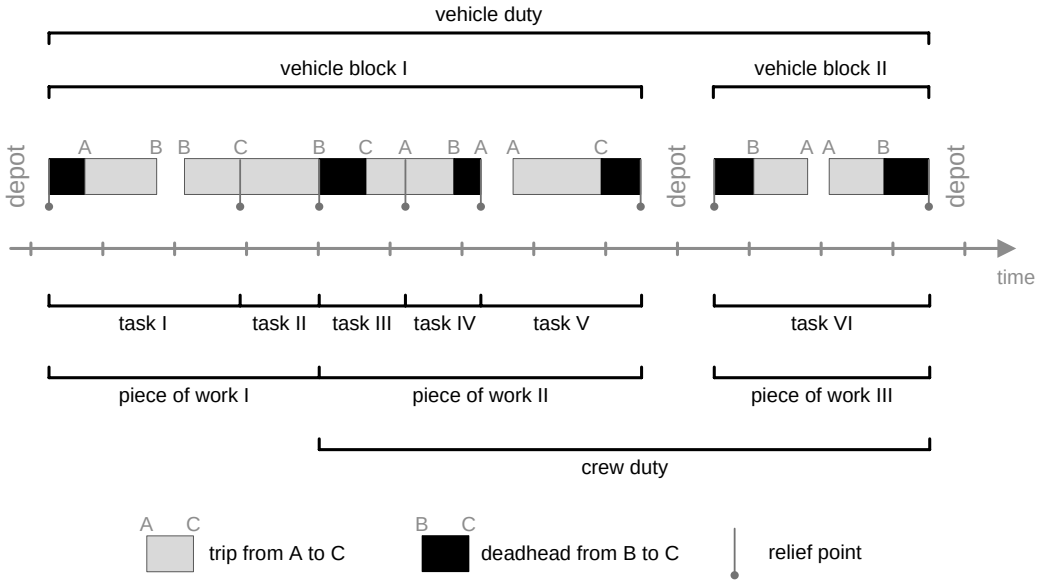

Figure 1.4.: Schedule of one vehicle and one crew where a piece of work remains unassigned

The objective is often to first minimize the number of duties and second the total working time. Therefore, high fixed crew costs and an hourly rate for working time are taken into account. Crew scheduling problems, however, are often subject to non-linear costs, e.g. overtime bonuses.

[Fischetti et al., 1987] and [Fischetti et al., 1989] show that the CSP with either working time or spread time constraints is NP-hard. Although duty constraints

differ from application to application, we assume that the CSP has at least one of these constraints and is therefore NP-hard.

1.2. Integrated Vehicle and Crew Scheduling

Vehicle and crew scheduling are traditionally approached in a sequential manner which means that vehicle schedules are determined before crew schedules. In this section, we discuss potential benefits of integrating vehicle and crew scheduling.

Although scheduling vehicles independently of crews was seriously criticized in the early eighties by [Bodin et al., 1983], most commercial software packages still use the sequential approach or offer integration at user level. However, an integrated approach as sketched in Figure 1.5 discloses additional flexibility in crew scheduling leading to savings as we will show in the following with a small example.

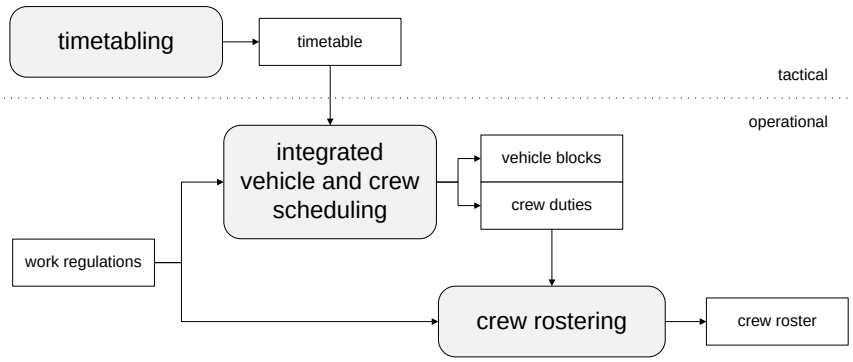

Figure 1.5.: Planning process for integrated vehicle and crew scheduling

Example 1 *Let us assume a timetable with five trips f_1, f_2, f_3, f_4, and f_5 is given as follows: f_1 from A to B (08:15-09:40), f_2 from B to A (09:50-10:15), f_3 from A to C (10:15-10:55), f_4 from B to A (11:15-12:15), and f_5 from C to C (10:45-11:30). Furthermore, the travel times in minutes between all stations and both depots (deadhead matrix) are given in Table 1.1. Note that the travel time between two locations is shorter than a service trip between the same locations since the bus does not need to stop for passengers. A duty is feasible if the duration of each piece of work is less than or equal to 4 hours. A duty has either one or two pieces of work with a break of at least 45 minutes in between. A duty may only contain tasks of a single depot and must start and end in that depot. If a*

	depot 1	depot 2	A	B	C
depot 1	–	15	15	41	23
depot 2	15	–	22	35	10
A	15	22	–	40	17
B	41	35	40	–	20
C	23	10	17	20	–

Table 1.1.: Deadhead matrix

duty does not start/end in its depot, additional walking time (as defined in the deadhead matrix) is added to the time of the duty. Both depots and station B are relief points. Finally, we define the cost structure as follows: a fixed cost of 1000 per vehicle, 1000 per duty, and variable vehicle costs of 60 per hour that a vehicle is without passengers outside its depot.

The optimal vehicle schedule (see Figure 1.6) consists of two vehicles where one vehicle operates trips f_1, f_2, f_3 and f_4 and the other trip f_5. Obviously, the first vehicle block cannot be covered by a single crew leading to a crew schedule with three duties.

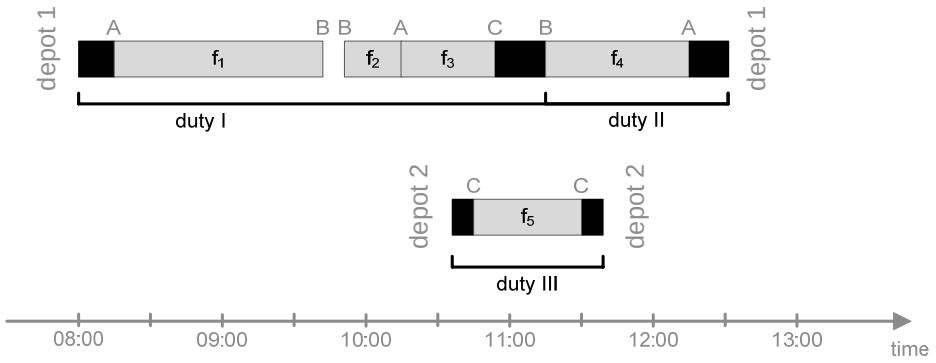

Figure 1.6.: Optimal vehicle and crew schedule for sequential approach consist of two blocks and three duties.

However, scheduling vehicles and crews together leads to an overall optimal solution of two blocks and two duties (see Figure 1.7). Instead of one long and one short vehicle block as illustrated in Figure 1.6, we obtain two vehicle blocks of almost same length that allow a better assignment to duties. Notice that the overall optimal solution incurs higher operational costs for the vehicle schedule but saves one duty.

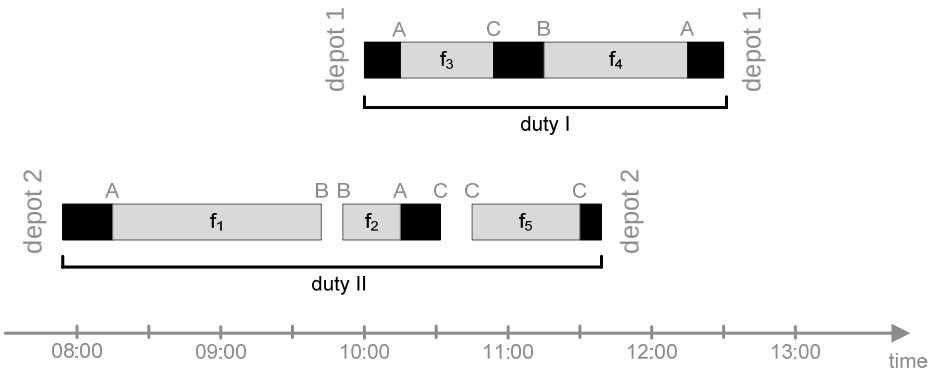

Figure 1.7.: Optimal vehicle and crew schedule for integrated approach consist of two blocks and two duties.

Of course, problems are much more complex in reality than in this example. Nevertheless, this example shows that scheduling vehicles independently of crews can lead to inefficient solutions and that the overall solution can be improved by using an integrated approach.

Moreover, applying an integrated approach is essential when relief opportunities are rare, i.e., in extra- or sub-urban public transport systems, since efficient vehicle schedules may lead to poor or even infeasible crew scheduling solutions. If we construct an optimal vehicle schedule in an extra-urban scenario, it is likely to contain pieces of work that are too long to meet break regulations. In an urban setting with many relief points drivers can often move to another relief point by foot or other means of transport. Thus, many pieces of work can be combined to form a duty. In other settings, distances are such that drivers are virtually tied to their vehicle in order to reach relief opportunities. As a consequence, integrating vehicle and crew scheduling is essential in sub- or extra-urban settings since vehicle scheduling strongly affects crew scheduling.

The *integrated vehicle and crew scheduling problem* (VCSP) for a given set of timetabled trips, depots, and relief points can be stated as follows: find minimum cost sets of vehicle blocks and crew duties such that both vehicle and crew schedule are feasible and mutually *compatible*. Vehicle and crew schedule are compatible if each trip is covered and each deadhead used in the vehicle schedule is also covered by exactly one duty while all deadheads not contained in the vehicle schedule are not part of any duty. Feasible vehicle and crew schedules are defined in Section 1.1.1 and 1.1.2, respectively. VCSP is NP-hard since (at least) the crew scheduling part is NP-hard.

1.3. Irregular Timetables

In the preceding section, we focused on how to conduct operations of a given timetable at minimum cost. In this section, however, we will address another aspect which is related to the quality of crew schedules.

In practice, timetables consist of many trips serviced every day and some exceptions that do not repeat daily. In particular, service trips to schools, production facilities, or public swimming baths are often subject to change, e.g., trips may be operated on every day except Sunday or on Monday only. Unless specifically imposed, traditional vehicle and crew scheduling usually produces *irregular* crew schedules which are undesired in practice. A crew schedule is called irregular if it cannot be repeated many times. Similar to airline crew scheduling, *regularity* is an important aspect for crew schedules in public transport since regular solutions can improve operational reliability and reduce training costs. Furthermore, regular solutions are less error-prone and crews often prefer to repeat itineraries. In current practice, companies often try to increase regularity of crew scheduling solutions by applying one of the following heuristic procedures:

- *All first - irregular second*: First, the planner solves a crew scheduling problem for a particular period with both regular and irregular trips. In a second step, he fixes the subset of crew duties that can be operated the whole period and reoptimizes all unfixed trips. Notice that the second problem can also contain some regular trips.

- *Regular first - irregular second*: The set of service trips is divided into regular and irregular trips. First, a crew scheduling problem for the set of regular trips is solved while the irregular trips are left for subsequent optimization.

In both cases, the second problem has a sparse schedule and, thus, likely requires extensive deadheading and even its optimal solution yields a high costs. On the other hand, if the second problem contains many trips, the corresponding solution has low cost but low regularity as well.

In the following, we will evaluate the impact of irregular timetables on the regularity of vehicle and crew scheduling solutions. We consider the timetables for Wednesday and Thursday of a small-town in Germany where 5% of the trips are different on Thursday. A reference crew schedule is given and we seek a crew schedule for Thursday that is similar to Wednesday's schedule. First, we perform traditional vehicle and crew scheduling (vehicle first – crew second). Second, we solve an independent crew schedule problem for Thursday where the set of tasks

corresponds to the set of service trips (see Section 6.1 for a detailed description). In both cases, we compare the crew schedule for Thursday with Wednesday's crew schedule.

In Figure 1.8 we depict the impact of an irregular timetable on the regularity of the crew scheduling solution if independent crew scheduling is performed. In

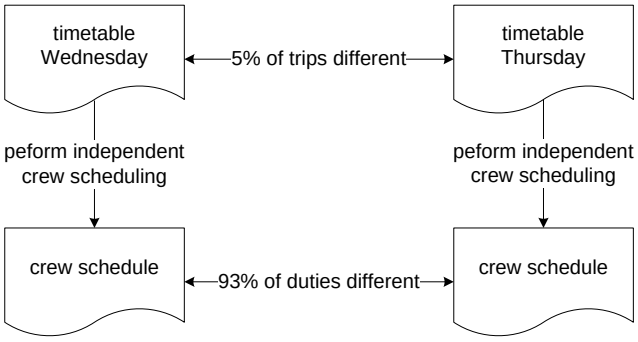

Figure 1.8.: Impact of an irregular timetable on the regularity of the crew scheduling solution if independent crew scheduling is performed.

our setting, 5% of trips are either in the Wednesday or the Thursday timetable, but not in both. As we can see in Figure 1.8, only 7% of the duties in the Wednesday crew scheduling solution are also part of the Thursday solution. The impact is even stronger for traditional vehicle and crew scheduling: none of the duties in the Wednesday solution could be preserved in the Thursday solution. Therefore, we conclude that, unless specifically imposed, small modifications of the timetable can destroy the structure of the crew scheduling solution.

Finally, notice that public transport companies face a similar situation whenever they change their timetable, e.g., scheduled timetable changes in summer or winter. Typically, the changes involve only a small portion of the complete timetable. Again, a traditional approach to vehicle and crew scheduling is likely to produce solutions for the new timetable that do not have much in common with the former solution.

1.4. Selected Combinatorial Optimization Problems

For more than 50 years now, many researches have studied the field of *combinatorial optimization*. It involves the problem of minimizing or maximizing a function of discrete *decision variables* subject to equality or inequality constraints. Vehicle and crew scheduling problems have been formulated as combinatorial optimization problems for more than thirty years now. Therefore, we review selected

well-known combinatorial optimization problems that we will use in this thesis. We do not describe all mathematical theory, but rather provide the necessary background for the models and algorithms used in the remainder of this thesis. For extensive surveys of integer and combinatorial optimization we refer to [Nemhauser and Wolsey, 1988] and [Wolsey, 1998].

1.4.1. Network Flow Problems

In network flow problems "we wish to move some entity (electricity, a consumer product, a person or a vehicle, a message) from one point to another in an underlying network, and to do so as efficiently as possible, both to provide good service to the users of the network and to use the underlying (and typically expensive) transmission facilities effectively" ([Ahuja et al., 1993]).

In this subsection we describe three related network flow problems that will occur as subproblems in the remainder of the thesis, namely the minimum cost flow, the multicommodity cost flow, and the resource constrained shortest path problem.

Minimum Cost Flow Problem

The *minimum cost flow problem* (MCFP) is a very fundamental network flow problem. The problem aims at finding a minimum cost shipment of a commodity through a network that satisfies the demand at certain nodes from available supplies at other nodes (see [Ahuja et al., 1993]).

Let $G = (N, A)$ be a directed graph with N as the set of nodes and A as the set of directed arcs. We associate a cost c_{ij} with each directed arc $(i, j) \in A$ that corresponds to the cost per unit flow on that arc and varies linearly with the amount of flow. Furthermore, we define an upper (lower) bound u_{ij} (l_{ij}) on the amount of flow for each arc $(i, j) \in A$. Let b_i denote the supply/demand of node $i \in N$. If $b_i > 0$ ($b_i < 0$), node i is a supply (demand) node. We call each node i with $b_i = 0$ transshipment node. Finally, we associate a decision variable x_{ij} with the amount of flow on arc $(i, j) \in A$ and formulate MCFP as follows:

$$\sum_{(i,j)\in A} c_{ij}x_{ij} \quad \rightarrow \quad \min \tag{1.1}$$

$$s.t. \quad \sum_{\{j:(i,j)\in A\}} x_{ij} - \sum_{\{j:(j,i)\in A\}} x_{ji} \quad = \quad b_i \qquad \forall i \in N, \tag{1.2}$$

$$l_{ij} \leq x_{ij} \quad \leq \quad u_{ij} \qquad \forall (i, j) \in A. \tag{1.3}$$

The objective function (1.1) aims at minimizing total costs such that total outflow minus total inflow of each node is equal to the supply/demand of that node (1.2).

The flow on each arc must satisfy the lower and upper bound of that arc (1.3). We refer to constraint set (1.2) as *flow conservation constraints* and to (1.3) as *flow bound constraints*.

Notice that it is not necessary to impose integrality on the flow variables if all data (supplies, demands, and bounds) are integral since the constraint matrix is *totally unimodular* (see e.g. [Nemhauser and Wolsey, 1988]) and, thus, each solution to the linear program above is integral. However, algorithms that exploit the structure of the underlying network, such as capacity scaling or network simplex algorithms, are often considerably faster than general purpose linear programming algorithms (e.g. primal or dual simplex). Many of these algorithms run in polynomial time.

The *shortest path problem* (SP) can be stated as a minimum cost flow problem and aims at finding a path of minimum cost (length) from a source s to a sink t in network G. If we set $b_s = 1$, $b_t = -1$ and $b_i = 0$ for all other nodes, we will send unit flow from node s to t. Additionally, we set $u_{ij} = 1$ for all $(i, j) \in A$ if G contains directed cycles of negative length. The *all-pairs shortest path problem* is a generalization of the shortest path problem where we would like to find the shortest paths between all pairs of nodes.

Multicommodity Flow Problem

The *multicommodity flow problem* (MFP) is a generalization of the minimum cost flow problem. MFP is composed of several commodities each with its origin and destination that use the same underlying network whereas MCFP considers only a single commodity. Each commodity has separate flow conservation constraints while all commodities share the same flow bound constraints.

The formulation of the multicommodity flow problem is related to the minimum cost flow problem in the previous subsection. However, we introduce K as the set of commodities and separate flow variables, supply/demand, and costs by commodity $k \in K$. Furthermore, we have a set of flow bound constraints that are separated by commodity and another set for all commodities with lower bounds

L_{ij} and upper bounds U_{ij}, respectively.

$$\sum_{k\in K}\sum_{(i,j)\in A} c_{ij}^k x_{ij}^k \quad \rightarrow \quad \min \tag{1.4}$$

$$s.t. \quad \sum_{\{j:(i,j)\in A\}} x_{ij}^k - \sum_{\{j:(j,i)\in A\}} x_{ji}^k = b_i^k \qquad \forall i \in N, \forall k \in K \tag{1.5}$$

$$l_{ij}^k \leq x_{ij}^k \leq u_{ij}^k \qquad \forall (i,j) \in A, \forall k \in K \tag{1.6}$$

$$L_{ij} \leq \sum_{k\in K} x_{ij}^k \leq U_{ij} \qquad \forall (i,j) \in A \tag{1.7}$$

$$x_{ij}^k \in \mathbb{N} \qquad \forall (i,j) \in A \tag{1.8}$$

As opposed to minimum cost flow problems, solutions to multicommodity flow problems are not necessarily integral. We must impose integrality on the flow variables in order to obtain integral solutions - even if all data is integral. The integral multicommodity flow problem has been proven to be NP-complete (see [Garey and Johnson, 1979]) if there are at least two commodities. As stated earlier, the single commodity flow problem can be solved in polynomial time.

Resource Constrained Shortest Path Problem

The *resource constrained shortest path problem* (RCSP) is an extension of the shortest path problem. It consists of finding the minimum cost path between a source s and a sink t while respecting constraints on resource consumption.

Again, we define a directed network $G = (N, A)$ as in the previous subsections. However, we do not only associate a traversal cost c_{ij} with each arc $(i, j) \in A$, we also define a resource consumption $d_{ij}^r \geq 0$ for each resource $r \in R$. Consequently, each path P accumulates $\sum_{(i,j)\in P} d_{ij}^r$ of resource $r \in R$. We say that a path is *resource feasible* if and only if the resource consumption along the path is greater or equal to lower bound l^r and less or equal to upper bound u^r. RCSP can be formulated as follows:

$$\sum_{(i,j)\in A} c_{ij} x_{ij} \quad \rightarrow \quad \min \tag{1.9}$$

$$s.t. \quad \sum_{\{j:(i,j)\in A\}} x_{ij} - \sum_{\{j:(j,i)\in A\}} x_{ji} = \begin{cases} 1 & \text{for } i = s \\ 0 & \text{for } i \in N \setminus \{s,t\} \\ -1 & \text{for } i = t \end{cases}, \tag{1.10}$$

$$l^r \leq \sum_{(i,j)\in A} d_{ij}^r x_{ij} \leq u^r \qquad \forall r \in R, \tag{1.11}$$

$$x_{ij} \in \{0,1\} \quad \forall (i,j) \in A. \tag{1.12}$$

Note that a resource constrained shortest path problem with cost and resource consumption on nodes can be easily transformed to a problem with consumption on arcs. The resource constrained shortest path problem is NP-hard if there is least one resource (see [Garey and Johnson, 1979]), even though it can be solved in pseudo-polynomial time with a simple dynamic programming formulation. [Hassin, 1992] shows that a fully polynomial ϵ-approximation scheme exists.

1.4.2. Set Partitioning/Covering Problem

The *set partitioning problem* (SPP) can be defined as follows: given a finite set M, constraints defining a set N of feasible subsets of M, and a cost associated with each member in N, find a minimum cost subset of N that is a partition of M (see [Balas and Padberg, 1976]).

Let x_j be a binary decision variable that equals 1 if subset $j \in N$ is part of the solution and 0 otherwise. We associate a cost c_j with each variable x_j. Furthermore, we set $a_{ij} = 1$ if subset $j \in N$ contains element $i \in M$ and $a_{ij} = 0$ otherwise. Now, the set partitioning problem can be expressed as follows:

$$\sum_{j \in N} c_j x_j \quad \rightarrow \quad \min \tag{1.13}$$

$$s.t. \quad \sum_{j \in N} a_{ij} x_j \quad = \quad 1 \qquad \forall i \in M, \tag{1.14}$$

$$x_j \quad \in \quad \{0,1\} \qquad \forall j \in N. \tag{1.15}$$

The *set covering problem* (SCP) is a relaxation of SPP that does not require to partition set M but to cover it. In other words, a feasible solution may consist of several (but at least one) subsets $j \in N$ that contain a particular element $i \in M$. Clearly, each solution that is feasible for SPP is also feasible for SCP. The optimal solution of the set covering problem is a lower bound of the set partitioning problem. SCP can be obtained from SPP by replacing the equality sign in constraints (1.14) by a greater or equal sign "$\geq$".

A great variety of scheduling problems from practice can be formulated as set partitioning or covering problems. Well-established applications that use this formulation are (bus/airline) crew scheduling and vehicle routing problems. Typically, these problems have a huge number of variables.

It is well known that both the set partitioning and set covering problem are NP-hard (see [Garey and Johnson, 1979]).

1.5. Selected Combinatorial Optimization Techniques

The purpose of this section is to discuss algorithms and techniques to solve combinatorial optimization problem that will be used in the remainder of this thesis.

1.5.1. Lagrangian Relaxation

The general idea behind *Lagrangian relaxation* is to remove complicating constraints from a (combinatorial) optimization problem by penalizing their violation in the objective function. In the following we will briefly discuss this relaxation method and refer to [Geoffrion, 1974] and [Fisher, 1981] for an extensive discussion of the theory of Lagrangian relaxation. We consider the following problem P:

$$
\begin{aligned}
Z(P) = \quad \min \quad & \sum_{j \in N} c_j x_j && && (1.16)\\
s.t. \quad & \sum_{j \in N} a_{ij} x_j = d_i && \forall i \in M_1, && (1.17)\\
& \sum_{j \in N} b_{ij} x_j = e_i && \forall i \in M_2, && (1.18)\\
& x_j \in \mathbb{Z}_+ && \forall j \in N. && (1.19)
\end{aligned}
$$

Suppose that constraints (1.17) are hard constraints in the sense that the optimization problem without these constraints is easy to solve. Dualizing the hard constraints leads to the *Lagrangian subproblem*

$$
\begin{aligned}
\Phi(\pi) = \quad \min \quad & \sum_{j \in N} c_j x_j + \sum_{i \in M_1} \pi_i (d_i - \sum_{j \in N} a_{ij} x_j) && && (1.20)\\
s.t. \quad & \sum_{j \in N} b_{ij} x_j = e_i && \forall i \in M_2, && (1.21)\\
& x_j \in \mathbb{Z}_+ && \forall j \in N, && (1.22)
\end{aligned}
$$

where $\pi = (\pi_i)_{i \in M_1}$ represents the Lagrangian multipliers associated with the dualized hard constraints. When we dualize inequality constraints of the form $\sum_{j \in N} a_{ij} x_j \leq d_i$ ($\sum_{j \in N} a_{ij} x_j \geq d_i$) for all $i \in M_1$, the corresponding Lagrangian multipliers are restricted in sign $\pi_i \leq 0$ ($\pi_i \geq 0$), $i \in M_1$. Note that $\Phi(\pi)$ can be

rewritten as

$$\Phi(\pi) = \Bigg\{ \quad \min \quad \sum_{j \in N} \bar{c}_j x_j + \sum_{i \in M_1} \pi_i d_i | \\ \sum_{j \in N} b_{ij} x_j = e_i, \forall i \in M_2; x_j \in \mathbb{Z}_+, \forall j \in N \Bigg\} \tag{1.23}$$

where we call $\bar{c}_j = c_j - \sum_{i \in M_1} a_{ij}\pi_i$ *Lagrangian cost* of column $j \in N$.

$\Phi(\pi)$ defines a lower bound on the original problem P for any fixed vector π since each feasible solution for the original problem is also feasible for the Lagrangian subproblem (but not vice versa) and $\Phi(\pi)$ equals the objective value of such feasible solution in the original problem.

We obtain the best possible lower bound by solving the *Lagrangian dual problem* (LDP)

$$Z(LDP) = \max_{\pi} \Phi(\pi). \tag{1.24}$$

It can be shown (see [Geoffrion, 1974]) that the optimal solution of the Lagrangian dual problem always provides a lower bound on the original problem that is at least as good as the objective value of the linear relaxation LP:

$$Z(LP) \leq Z(LDP) \leq Z(P). \tag{1.25}$$

Furthermore, the objective values of Lagrangian and linear relaxation are equal when the integrality constraints (1.22) of the Lagrangian subproblem can be replaced with $x_j \geq 0, j \in N$ and the solution of the subproblem remains unchanged for all possible multipliers π. Then, the Lagrangian subproblem is said to have the *integrality property*.

The Lagrangian dual problem maximizes a piecewise linear concave, but nondifferentiable function $\Phi(\pi)$ which implies that LDP is also nondifferentiable. In the context of combinatorial optimization LDP is typically solved by a subgradient algorithm that was introduced by [Held and Karp, 1971] and will be described in the following.

Subgradient Algorithm

The *subgradient algorithm* is an iterative search procedure to optimize nondifferentiable functions. It is well-known that a differentiable function f can be optimized by an iterative gradient method like the steepest ascent method: starting with an initial solution u^0 the sequence

$$u^{t+1} = u^t + w^t \nabla f(u^t) \tag{1.26}$$

eventually converges to an optimal solution with $\nabla f(u^t)$ as the gradient of f at u^t and w^t as suitable step length. However, for nondifferentiable functions we cannot use a gradient method since some points do not have a gradient. Instead we use a subgradient method which is a generalization of the gradient method to the nondifferentiable case where gradients are replaced by subgradients.

A *subgradient* at π_0 of a concave function $\Phi : \mathbb{R}^{|M_1|} \rightarrow \mathbb{R}^1$ is a vector $s \in \mathbb{R}^{|M_1|}$ such that $\Phi(\varphi) \leq \Phi(\pi_0) + s(\varphi - \pi_0)$ for $\varphi \in \mathbb{R}^{|M_1|}$. In other words, a subgradient in π_0 is the slope of a straight line through point $(\pi_0, \Phi(\pi_0))$ that runs above function Φ for $|M_1| = 1$. Let $\partial\Phi(\pi_0)$ denote the non-empty set of all subgradients (*subdifferential*) of Φ at π_0 that reduces to the gradient if Φ is differentiable at π_0. At points where the function is nondifferentiable the subgradient method chooses an arbitrary subgradient from the subdifferential. Figure 1.9 shows a subgradient s_k of $\Phi(\pi)$ at the nondifferentiable point π_0 (arbitrarily) chosen from the subdifferential which is represented by the grey area. It is easy to see that

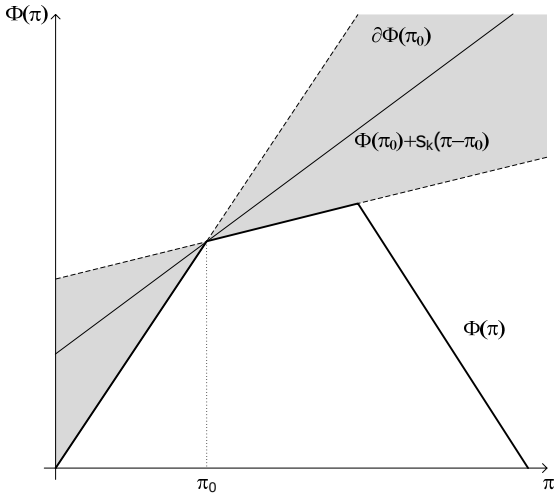

Figure 1.9.: Subdifferential and subgradient s_k of a concave, nondifferentiable function $\Phi(\pi)$ at π_0 for $|M_1| = 1$.

a subgradient $s \in \mathbb{R}^{|M_1|}$ for the Lagrangian subproblem is given by $s_i = d_i - \sum_{j \in N} a_{ij} x_j, i \in M_1$ with x as optimal solution to this problem.

A basic version of the subgradient method to solve the Lagrangian dual problem is depicted in Algorithm 1. Notice that the formula used to compute the step size in step 4 does not assure that LDP converges to the global maximum. However, an empirical justification for the rule used in step 4 is given by [Held et al., 1974]. Furthermore, this rule has a low computational burden as opposed to other step

length rules with proven convergence (e.g. [Polyak, 1967]). As proposed by [Held and Karp, 1971] we initially set the step size parameter $\lambda = 2$ and halve it whenever $\Phi(\pi^t)$ fails to improve in a certain number of iterations. Parameter ϵ basically steers the accuracy of the solution and, thus, the number of iterations.

Algorithm 1: Subgradient Algorithm

(Step 1) **Initialization**
Initialize multipliers π^0, parameter ϵ and set $t = 0$.
Compute upper bound UB.

(Step 2) **Solve Lagrangian subproblem**
Solve $\Phi(\pi^t)$ and store optimal solution x^t

(Step 3) **Compute search direction δ^t**
Compute subgradients $s_i^t = d_i - \sum_{j \in N} a_{ij} x_j^t$ for all $i \in M_1$.
Set $\delta^t = s^t$.

(Step 4) **Compute step size w^t**
$w^t = \lambda \frac{UB - \Phi(\pi^t)}{||\delta^t||^2}$

(Step 5) **Update Lagrangian multipliers**
$\lambda_i^{t+1} = \lambda_i^t + w^t \delta_i^t$ for all $i \in M_1$

(Step 6) **Check termination criteria**
Terminate if one of the following criteria is satisfied:
$s^t = 0$,
$\sum_{i \in M_1} (\delta_i^t)^2 \leq \epsilon$
$\lambda \leq \epsilon$
$t \geq t_{max}$
$UB - \Phi(\pi^t) \leq \epsilon$
otherwise set $t = t + 1$ and return to step 2

1.5.2. Dantzig-Wolfe Decomposition and Column Generation

Column generation is a method to solve linear programs that involve a large number of columns. Formulations of problems with a large number of variables arise in many real-life situations, e.g crew scheduling or vehicle routing, where one likes to select a minimum cost (maximum profit) subset from a very wide choice of subsets. Typically, the chosen subset must satisfy a number of constraints. Below we show how such formulations (*master problems*) of a combinatorial optimization problem may arise by reformulation and how linear relaxations of master problems can be solved by (delayed) column generation.

Consider the following (*compact*) integer program P

$$Z(P) = \min\{c(\mathbf{x}) : A\mathbf{x} = \mathbf{d}, \mathbf{x} \in X\} \tag{1.27}$$

where $X = \{\mathbf{x} \in \mathbb{Z}_+^{|N|} : B\mathbf{x} = \mathbf{e}\}$. We assume that X is non-empty and contains a large (but finite) number of elements. It is well known that replacing X by the convex hull $\text{conv}(X)$ does not change the optimal objective value $Z(P)$. The Minkowski-Weyl theorem (see [Nemhauser and Wolsey, 1988]) states that each $\mathbf{x} \in X$ can be represented as a convex combination of extreme points $\{\mathbf{x}_p\}_{p\in\mathcal{P}}$ plus a non-negative combination of extreme rays $\{\mathbf{x}_r\}_{r\in\mathcal{R}}$ of $\text{conv}(X)$ (see Figure 1.10). $\mathcal{P}$ denotes the set of extreme points and $\mathcal{R}$ the set of extreme rays. As we assumed

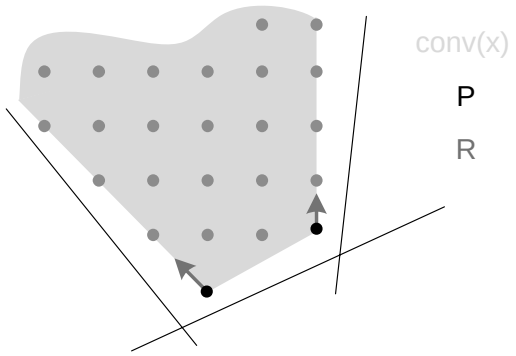

Figure 1.10.: The convex hull $\text{conv}(X)$ of the unbounded polyhedron X with two extreme points and two extreme rays.

polyhedron X to be bounded, the convex hull $conv(X)$ can be represented by a convex combination of a finite number of extreme points $\{\mathbf{x}_p\}_{p\in\mathcal{P}}$, i.e. $\mathbf{x} = \sum_{p\in\mathcal{P}} \mathbf{x}_p\vartheta_p$ with $\sum_{p\in\mathcal{P}} \vartheta_p = 1$ and $\vartheta \in \{0,1\}^{|\mathcal{P}|}$. Next, we replace $\mathbf{x}$ in the original problem with the internal representation of X, define $c_p = c(\mathbf{x}_p)$ as well as $\mathbf{a}_p = A\mathbf{x}_p$ with $p \in \mathcal{P}$, and obtain the following *extensive formulation* P' with

a large number of columns

$$Z(P') = \min \sum_{p \in \mathcal{P}} c_p \vartheta_p \tag{1.28}$$

$$s.t. \sum_{p \in \mathcal{P}} \mathbf{a}_p \vartheta_p = \mathbf{d} \tag{1.29}$$

$$\sum_{p \in \mathcal{P}} \vartheta_p = 1 \tag{1.30}$$

$$\vartheta \in \{0,1\}^{|\mathcal{P}|} \tag{1.31}$$

which we refer to as *master problem*. Solving the extensive formulation is equivalent to solving the original compact formulation. Any fractional solution to the LP relaxation of the compact formulation is a feasible solution to the LP relaxation of the extensive formulation if and only if it can be expressed by a convex combination of extreme points of $\text{conv}(X)$, but not vice versa. In particular, [Geoffrion, 1974] shows that the extensive formulation provides a tighter LP relaxation if $\text{conv}(X)$ does not only have integral extreme points.

The reformulation process is also known as *Dantzig-Wolfe decomposition* (see [Dantzig and Wolfe, 1960]) and is one way of obtaining a formulation with a large number of variables. Problems may also have a "natural" formulation with a huge number of variables. However, [Villeneuve et al., 2005] show that a compact formulation to such a natural (extensive) formulation exists under very mild assumptions.

Below we will describe a column generation algorithm to solve the linear programming relaxation of the following master problem MP

$$Z(MP) = \min \sum_{j \in N} c_j x_j \tag{1.32}$$

$$s.t. \sum_{j \in N} a_{ij} x_j = d_i \qquad \forall i \in M, \tag{1.33}$$

$$x_j \in \{0,1\} \qquad \forall j \in N. \tag{1.34}$$

where $|N|$ is huge. Notice that in the linear programming (LP) relaxation of MP the integrality constraints (1.34) are replaced by $x_j \geq 0, \forall j \in N$. As in most practical situations we assume that it is impossible to explicitly keep all columns in main memory and, as a result, to solve the master linear program from scratch. Instead, we solve a sequence of *restricted* master linear programs (RMP) where each problem contains only a small subset of all columns.

We initialize the algorithm (see Algorithm 2) with an initial column set $N^0 \subseteq N$ that contains a feasible solution. This initial solution can either be generated

Algorithm 2: Column Generation Algorithm

(Step 1) **Initialization**
Choose initial column set N^0.
Set $t = 0$.

(Step 2) **Solve (restricted) master problem**
Solve $RMP(N^t)$ and store duals π^t.

(Step 3) **Solve pricing problem**
Solve pricing problem $P(\pi^t)$ and obtain columns N' with negative reduced costs.
If $|N'| = 0$ terminate.

(Step 4) **Add column(s) to restricted master problem**
Set $N^{t+1} = (N' \cup N^t)$ and $t = t + 1$.
Goto Step 2.

by a heuristic or by adding appropriate artificial variables to the RMP. After the restricted problem is solved, the dual information of the solution is used to *price out* new columns with negative reduced costs, i.e., columns that can improve the objective value of the RMP. Given an optimal dual solution π of the current RMP the *pricing problem* (subproblem) can be stated as follows

$$\bar{c}^* = \min \{c_j - \sum_{i \in M_1} a_{ij}\pi_i : j \in N\}. \tag{1.35}$$

If the pricing problems returns $\bar{c}^* < 0$, the column j with least negative reduced costs $\bar{c}_j = c_j - \sum_{i \in M_1} a_{ij}\pi_i$ is added to the RMP. The process iterates until $\bar{c}^* \geq 0$. Then, the current solution $\mathbf{x}$ to the RMP solves the linear relaxation of MP without having all columns explicitly available. The complete set of columns N is only implicitly available since new columns are only generated (and kept in memory) when needed. The pricing problem (1.35) is an optimization problem itself that in our application corresponds to a resource constrained shortest path problem (see Section 3.1).

The solution to the restricted master problem satisfies all constraints of our master problem except for the integrality constraints. We obtain integer solutions by integrating column generation and *branch-and-bound* (see Section 1.5.4). A column generation algorithm is used to solve the linear relaxation of the master problem in each node of the branch-and-bound tree. This solution approach is often referred to as *branch-and-price*. For a general discussion on integer programming (IP) column generation see [Barnhart et al., 1998].

Excellent surveys on column generation are given by [Desaulniers et al., 2005]

and [Lübbecke and Desrosiers, 2005]. Acceleration techniques often applied in column generation algorithms are described in [Desaulniers et al., 2002].

1.5.3. Lagrangian Relaxation based Column Generation

Typically, column generation is used to solve the LP-relaxation of the master problem, but it can also be combined with Lagrangian relaxation as we will discuss in this section. In this thesis Lagrangian relaxation in combination with column generation will be used to solve integrated vehicle and crew scheduling problems (see Section 2.4).

There is a strong relationship between Dantzig-Wolfe decomposition and Lagrangian relaxation. Let us consider the Lagrangian relaxation of the compact formulation (1.27) with constraints $A\mathbf{x} = \mathbf{d}$ relaxed. The corresponding Lagrangian subproblem reads

$$\Phi(\pi) = \{ \quad \min \quad \sum_{j \in N} (c_j - \sum_{i \in M_1} a_{ij}\pi_i)x_j + \sum_{i \in M_1} \pi_i d_i | \mathbf{x} \in X \}. \tag{1.36}$$

It is well known (see [Nemhauser and Wolsey, 1988]) that the associated Lagrangian dual is the dual of the LP relaxation of the extensive formulation (1.28)-(1.31). In other words, we can solve the Lagrangian dual either by applying the subgradient method to it or by solving the linear relaxation of the extensive formulation (RMP) with a column generation approach. Consequently, the optimal lower bound of the restricted linear master program (RMP) and the best Lagrangian dual are the same. Moreover, the Lagrangian subproblem is of the form of the column generation pricing problem and, consequently, solutions of the Lagrangian subproblem can be added as new columns to the RMP. Both solution methods for the Lagrangian dual have advantages and disadvantages, hence [Barahona and Jensen, 1998] use a hybrid method that combines the advantages of both approaches. Note that the Lagrangian multiplier vector π corresponds to the dual variables associated with the linking constraints $\sum_{p \in P}(A\mathbf{x}_p)\vartheta_p = \mathbf{d}$ of the LP relaxation of the extensive formulation.

Lagrangian relaxation can also be applied to the extensive formulation in order to obtain approximate dual solutions and lower bounds to the restricted master problem. Instead of solving restricted problems with the simplex method to optimality, a subgradient method is used to solve the Lagrangian dual approximately. At the end of the subgradient phase, the Lagrangian multipliers are an approximation of the optimal dual variables for the current restricted master problem. The multipliers can be used to price out new columns. There are different reasons to choose this approach:

- The subgradient method is fast, easy to implement, and does not require a commercial LP solver.

- When solving the RMP with a simplex method, we obtain a basic dual solution that corresponds to a vertex of the optimal face of the dual polyhedron. Basically, a new column of the RMP may cut that vertex while a dual solution interior (in the center) of the dual face allows stronger dual cuts (i.e. better primal columns). [Bixby et al., 1992] and [Barnhart et al., 1998] note that this may improve the convergence of a column generation algorithm and reduce degeneracy. The subgradient method naturally provides non-basic solutions with many non-zero elements. [Jans and Degraeve, 2004] provide computational results indicating that Lagrangian multipliers are beneficial.

- During the subgradient phase possibly feasible solutions are generated.

However, since the Lagrangian multipliers are not exact, columns in the restricted master problem may have negative reduced costs. Thus, they should be modified before generating new columns in order to prevent that columns are generated twice. [Carraresi et al., 1995] and [Freling, 1997] describe a greedy heuristic that modifies Lagrangian multipliers in such a way that all columns in the RMP have non-negative reduced costs and that the lower bound does not decrease. In Algorithm 3 we describe their heuristic with π as approximate dual variables to the linear relaxation of master problem (1.32)-(1.34) and N^t as the current column set.

Algorithm 3: Lagrangian multiplier adjustment heuristic

(Step 1) **Find negative reduced cost columns**

Define negative reduced cost columns $j \in R \subseteq N^t$ with $c_j - \sum_{i \in M_1} a_{ij}\pi_i < 0$

(Step 2) **Update multipliers and reduced costs**

foreach $r \in R$ **do**

  Define $\delta = \frac{c_r - \sum_{i \in M_1} a_{ir}\pi_i}{\sum_{i \in M_1} a_{ir}}$

  foreach $i \in M_1$ *with* $a_{ir} = 1$ **do**

    Update multiplier $\pi_i = \pi_i + \delta$

  foreach $j \in R$ *with* $j > r$ **do**

    Update reduced costs $\bar{c}_j = c_j - \sum_{i \in M_1} a_{ij}\pi_i$

A thorough discussion on how to combine Lagrangian relaxation and column generation with some examples can be found in [Huisman et al., 2005b].

1.5.4. Branch-and-Bound

Branch-and-bound is an exact solution approach for combinatorial optimization problems that is based on the *divide-and-conquer* principle (see [Wolsey, 1998]). Basically, the problem is decomposed into a series of smaller problems that are easier to solve than the original. These smaller problems are solved and their solution information is later put together to solve the original problem. *Branching* divides the original solution space into two or more parts. Each of the smaller solution spaces is split again into two or more parts and so on. As a result we obtain a *search tree* where each solution space is represented by a node. Without *bounding* this procedure corresponds to a complete enumeration which is impossible for most problems of practical size. Bounding is used to prune nodes of the search that cannot contain a solution better than the best solution (*incumbent*) found so far. For minimization problems we need to find lower bounds on each solution space (node). Typically, lower bounds are calculated by solving the LP relaxation of each node. A combinatorial optimization problem can be solved with branch-and-bound based on the LP relaxation by successively separating fractional solutions from the feasible solution space. Algorithm 4 depicts a branch-and-bound method to solve minimization problem P.

Basically, branch-and-bound algorithms leave two choices: how to branch (Step 6) and which (sub)problem to select next (Step 3). In LP-based branch-and-bound algorithms branching can only be performed by adding linear inequalities to the problem or by modifying bounds on variables. Inequalities or bound modifications correspond to a *valid branching rule* if they split the problem, cut the current fractional feasible solution (but no integer solution), and result in integer solutions in each leaf node after a finite number of separations. Ideally, a good branching rule is not only valid but also takes the performance of the algorithm into account. Recall that nodes are pruned either by optimality, by bound, or by infeasibility. Thus, a branch-and-bound algorithm works in two directions: construct an integer solution and provide a tight lower bound that possibly proves optimality of the incumbent. Of course, we do not seek infeasible subproblems. Consequently, we like to select that branching rule among several alternatives that maximizes the minimum lower bound and has a good chance to generate many integer solutions. The same reasoning holds for good *node selection strategies*. For a recent comparison of general branching rules see [Achterberg et al., 2005]. For a comprehensive survey on general branch-and-bound strategies

Algorithm 4: Branch-and-bound

(Step 1) **Initialization**
Set upper bound $Z^* = \infty$.
Add original problem to set of unprocessed nodes N.

(Step 2) **Check termination criteria**
If $N = \emptyset$ terminate and output incumbent $\mathbf{x}^*$ with objective value Z^*.

(Step 3) **Select next node**
Choose node $p \in N$ and delete it from N.

(Step 4) **Calculate bound of current node** p
Solve LP relaxation of current problem p with dual bound $\widehat{Z}$ and store solution $\widehat{\mathbf{x}}$.
If $\widehat{\mathbf{x}}$ is empty prune by infeasibility and goto Step 2.

(Step 5) **Bounding**
If $\widehat{Z} \geq Z^*$ prune by bound and goto Step 2.
If $\widehat{\mathbf{x}}$ is integer and $\widehat{Z} < Z^*$ set $Z^* = \widehat{Z}$ and store new incumbent $\mathbf{x}^* = \widehat{\mathbf{x}}$. Then, prune by optimality and goto Step 2.

(Step 6) **Branching**
Create two subproblems p_1 and p_2 that separate the current LP solution and add both to list of unprocessed nodes N.
Goto Step 3.

we refer to [Linderoth and Savelsbergh, 1999]. We will discuss problem-specific branching rules for the crew scheduling problem in Section 6.4.1.

In a *branch-and-cut* algorithm we tighten lower bounds by adding cutting planes to a subproblem. A *branch-and-price* algorithm uses a column generation algorithm to solve linear relaxation with an enormous number of columns in each node (see Section 1.5.2). A *branch-and-cut-and-price* method combines both extensions.

1.5.5. Metaheuristics

A *heuristic* applied to a combinatorial optimization problem seeks to find a good approximate solution in reasonable time while an exact method guarantees to find the global optimum in a potentially long time. [Osman and Laporte, 1996] define a *metaheuristic* formally as "an iterative generation process which guides a subordinate heuristic by combining intelligently different concepts for exploring and exploiting the search space, learning strategies are used to structure information in order to find efficiently near-optimal solutions." Well-known metaheuristics include ant colony optimization (ACO), tabu search (TS), simulated annealing (SA), and evolutionary algorithms (EA). For an overview of metaheuristics the reader is referred to [Reeves, 1993]. In this thesis, we will propose a novel hybrid evolutionary algorithm for the integrated vehicle and crew scheduling problem (see Chapter 4). Moreover, we will use different metaheuristics to solve combinatorial optimization problems with two competing objectives (see Section 6.4.2).

1.6. Scope and Purpose of the Thesis

In the previous sections we have introduced both the practical and the theoretical foundation for the remainder of this thesis. We will now define the scope and purpose of the thesis.

Until recently it was not possible to solve real-world integrated vehicle and crew scheduling problems with several depots within reasonable time and guaranteed solution quality. Still, large instances with complex duty feasibility rules cannot be tackled in an integrated manner. In addition to cost reduction the quality of crew schedules is an important aspect. Therefore, we will consider the regularity of crew schedules as one aspect of quality. All together, our main research objectives are threefold:

- To develop models and techniques for the integration of vehicle and crew scheduling in public transit that allow to tackle large problem instances.

Moreover, to efficiently model complex crew duty feasibility rules arising from German federal laws, safety regulations, and (collective) in-house agreements.

- To develop models and techniques to increase the regularity of crew schedules for the integration of vehicle and crew scheduling with irregular timetables.

- To test the applicability of the proposed techniques in practice.

The thesis comprises seven chapters and is set up as follows. The first chapter corresponds to this introduction which outlines both the practical and the theoretical foundation of the thesis.

In Chapter 2 we define the integrated multiple-depot vehicle and crew scheduling problem in public transport. We review models and solution techniques that are used in literature for sequential, partially integrated, and fully integrated vehicle and crew scheduling. Furthermore, we thoroughly describe the modeling approach, mathematical formulation, and solution approach that provides the starting point for the following chapters of the thesis. The solution approach is based on column generation in combination with Lagrangian relaxation.

Chapter 3 presents new approaches for the integrated vehicle and crew scheduling problem. More specifically, we propose a novel approach for the column generation pricing problem that includes both modeling approach and solution method. Furthermore, we discuss different solution approaches to construct integer solutions. Finally, we propose a new model where drivers are allowed to change their vehicle whenever there is a relief point. The chapter is concluded with a computational study using real-world and randomly generated benchmark instances in order to evaluate the effectiveness of our approaches.

In Chapter 4 we deal with a novel hybrid evolutionary algorithm to tackle integrated vehicle and crew scheduling problems. Our method combines mathematical programming techniques with an evolutionary algorithm. We compare different versions of the evolutionary algorithms with each other, with the traditional sequential approach, and with an integrated treatment of both planning steps.

In Chapter 5 we consider practical rules and regulations arising in public transport companies in Germany. We suggest enhancements and modifications of our modeling and solution approach from Chapter 3 to cover these practical extensions. Furthermore, we give an overview of how our implementation is being integrated in the commercial software tool interplan®. Finally, we test the applicability of the proposed techniques using randomly generated and real-life data

instances.

We address the ex-urban vehicle and crew scheduling problem with irregular timetables in Chapter 6. We discuss the impact of irregular timetables on the regularity of crew scheduling solutions. Regularity is an aspect which is related to the quality of crew schedules. We suggest a novel combination of local and follow-on branching to construct cost-effective and regular crew schedules. Furthermore, we show how bi-objective metaheuristics can be used to quickly estimate the quality of the solution generated with the latter approach. The chapter is concluded with a computational study that involves real-world and randomly generated ex-urban scenarios with a single depot.

Finally, in Chapter 7 we conclude the thesis with a summary and some final remarks.

2. Integrated Vehicle and Crew Scheduling: State-of-the-Art

In this chapter, we define the integrated vehicle and crew scheduling problem with multiple depots. Furthermore, we review state-of-the-art models and solution methods for traditional (sequential), partially integrated and integrated vehicle and crew scheduling. In particular, we thoroughly describe the modeling approach, mathematical formulation and solution approach that we will use in the remainder of this thesis.

This chapter is organized as follows. In Section 2.1, we give a problem definition for the integrated vehicle and crew scheduling problem and discuss the state-of-the-art for both sequential and simultaneous treatment of vehicle and crew scheduling in Section 2.2. In the next section, we describe a mathematical formulation from literature for the integrated vehicle and crew scheduling that we will use in the following chapters. Finally, we describe the column generation algorithm in Section 2.4 that we use to solve integrated models.

2.1. Problem Definition

The *integrated vehicle and crew scheduling problem* (VCSP) for a given set of timetabled trips within a fixed planning horizon, given traveling times between all pairs of locations, and a fleet of vehicles assigned to several depots can be stated as follows: find minimum cost sets of vehicle blocks and crew duties such that both vehicle and crew schedule are feasible and mutually *compatible*. Vehicle and crew schedule are compatible if each vehicle activity in the vehicle schedule is also covered by exactly one duty while all deadheads not contained in the vehicle schedule are not part of any duty.

A vehicle schedule is feasible if each trip is assigned to exactly one vehicle and each vehicle performs a feasible sequence of trips. A sequence of trips (*vehicle block*) is feasible if each pair of consecutive trips can be executed in sequence and each block starts and ends at the same depot. The vehicle costs comprise fixed (asset) costs for every vehicle and variable costs for travel and idle time outside

the depot.

A crew schedule is feasible if each task (either deadhead or trip) of the vehicle schedule is covered by a duty that can be performed by a single driver, and if each duty complies with a wide variety of federal laws, safety regulations, and collective in-house agreements. A task is a sequence of activities on a vehicle (such as performing trips and/or deadheading) between two consecutive *relief points* (see Section 1.1.2) and represents an elementary portion of work that can be assigned to a driver. A relief point defines a location and time where a driver may change his vehicle. A *piece of work* is a sequence of tasks without a (long) break for which a driver stays with the same vehicle. Consequently, duties are composed of pieces of work separated by breaks. The duty cost usually consists of a fixed component for wages and variable costs for working time or overtime bonuses.

If there are multiple depots some trips possibly have to be assigned to vehicles and drivers from a certain (sub)set of depots. It is easy to see that a problem with multiple depots reduces to several single depot problems if every trip can only be serviced from a single depot.

We make the same assumptions as [Huisman, 2004] in order to obtain comparable results in Chapter 3. However, we will relax and change some of the assumptions in Chapter 5 in order to apply our approach on practical scenarios arising in Germany.

- Each vehicle is assigned to the depot where its daily schedule starts and ends. Each depot is unlimited in capacity. That is, it can store an unlimited number of vehicles.

- Each crew is assigned to a depot and may only conduct tasks on vehicles from this particular depot. However, a duty does not necessarily start and end in this depot.

- A piece of work is only restricted by its duration. It may have a minimum and maximum duration.

- A vehicle returns to its depot if the idle time between two consecutive trips is long enough to perform a round-trip to the depot.

- Each trip has exactly two relief points: one at the beginning and the other at the end of the trip.

- A driver is required to be present if a bus is outside of a depot (*continuous attendance*) while there is no driver needed inside a depot.

- A driver may only change his or her vehicle during a break, i.e., between two pieces of work. The change of a vehicle of a driver is called *changeover*.

Notice that the last two assumptions imply that a second driver must be present during the break of a driver, if the original driver has no changeover and the vehicle is outside the depot. Otherwise, nobody would attend the vehicle during the break.

2.2. Literature Review

The purpose of this section is to review state-of-the-art models and approaches for solving sequential and (partially) integrated vehicle and crew scheduling problems.

2.2.1. Sequential Vehicle and Crew Scheduling

In the sequential or traditional planning procedure, the crew scheduling problem is solved after the vehicle scheduling problem. That is, we first assign trips to vehicles and schedule crews based on the vehicle blocks obtained before.

A thorough understanding of traditional vehicle and crew scheduling provides a useful introduction to the integrated problem since the integrated problem includes traditional vehicle and crew scheduling problems as subproblems. Furthermore, we will use the traditional approach to evaluate the efficiency gain by an integrated treatment. We will first discuss single- and multiple-depot vehicle scheduling problems and then review models and approaches for the crew scheduling problem.

Single-Depot Vehicle Scheduling

The single-depot vehicle scheduling problem (SDVSP) arises for small to medium-sized public transport companies that have a single depot and a homogeneous fleet of vehicles. Additionally, it may appear as a subproblem for the multiple-depot case. It is well known that the SDVSP corresponds to a minimum cost flow problem (see [Bodin et al., 1983]) that can be solved in polynomial time. The SDVSP has also been formulated as linear assignment problem [Orloff, 1976] and transportation problem [Gavish and Shlifer, 1978]. The formulation as transportation problem is also known as quasi-assignment formulation. All formulations can be solved in polynomial time.

[Löbel, 1996] describes an efficient implementation of the network simplex method for a minimum cost flow formulation. [Paixão and Branco, 1987] propose

an algorithm based on the Hungarian method for a quasi-assignment and assignment formulation. Their algorithm performs better on the quasi-assignment formulation and outperforms all other algorithms at that time. [Freling, 1997] and, subsequently, [Freling et al., 2001b] solve a quasi-assignment formulation efficiently with a combined forward and reverse auction algorithm (see [Bertsekas and Castañon, 1992]). Additionally, they propose a two phase approach that is valid when a special cost structure can be used. They test their algorithms on both real-world and artificial data with up to 1,500 trips and show that their algorithms outperform other approaches proposed before. [Silva et al., 1999] present an arc generation approach for a quasi-assignment formulation that is initialized with short deadhead arcs. Further arcs are added to the master problem by a column generation approach until optimality is proven.

For surveys on the SDVSP and its practical extensions we refer to [Daduna and Paixão, 1995], [Desrosiers et al., 1995], and [Bunte and Kliewer, 2006].

Multiple-Depot Vehicle Scheduling

The multiple-depot vehicle scheduling problem (MDVSP) often arises in medium-sized public transport companies and is inevitable in larger ones. The company operates its (homogeneous) fleet out of several depots where each vehicle is assigned to a single depot. The MDVSP can be extended by multiple vehicle types and by the constraint that some trips have to be serviced by vehicles from a certain subset of depots. The MDVSP has been extensively studied for more than 25 years now. Since this is a NP-hard problem (see [Bertossi et al., 1987]), early works mainly focused on heuristic algorithms. For an overview on heuristic methods for the MDVSP we refer to [Dell'Amico et al., 1993] and [Löbel, 1997]. A recent comparison and computational tests of different heuristic approaches to the MDVSP can be found in [Pepin et al., 2006].

[Fischetti et al., 2001] categorize exact solution approaches to the MDVSP by the mathematical formulation used:

1. Single-commodity formulations,
2. Multicommodity formulations,
3. Set partitioning formulations.

The first exact solution approach is proposed by [Carpaneto et al., 1989] and belongs to the first category. They add subtour breaking constraints and derive lower bounds by an additive lower bounding method (see [Fischetti and Toth, 1989]). Subsequently, they use a branch-and-bound method with user-defined

branching rules to obtain integer feasible solutions. The weak lower bound of the formulation can be improved by adding path elimination cuts in a branch-and-cut framework (see [Fischetti et al., 1999]). This approach is further extended by [Fischetti et al., 2001]. A single-commodity flow formulation with special assignment variables is discussed in [Mesquita and Paixão, 1992].

The multicommodity flow formulation is a generalization of the network flow approach for single-depot problems to the multiple-depot case where a network is set up for each depot. The multicommodity formulation combines these networks to form a multigraph that contains an arc for each depot between two nodes. Several authors including [Bodin et al., 1983], [Bertossi et al., 1987], [Forbes et al., 1994], [Ribeiro and Soumis, 1994], [Löbel, 1997], [Löbel, 1998], and [Kliewer et al., 2006b] use this type of formulation. Some authors use two types of variables (one for the assignment of trips to depots and another to obtain a feasible flow, e.g., [Bertossi et al., 1987]) while other propose a more compact model with only one type of variables (e.g. [Ribeiro and Soumis, 1994]). [Mesquita and Paixão, 1999] show that both variants lead to the same LP-relaxation. Moreover, they prove that multicommodity flow formulations have a tighter LP-relaxation than single-commodity formulations. Multicommodity flow formulations can be further classified by the underlying network structure. Among others, [Löbel, 1997] and [Löbel, 1998] use a *connection-based network* where each feasible connection between two trips corresponds to an explicit arc in the network. Notice that the size of the network grows quadratically with the number of trips. Since the number of connection arcs can be vast, they propose an arc generation approach with a special Lagrangian pricing technique to solve large instances to proven optimality. Recently, [Kliewer et al., 2006b] propose a multicommodity flow model based on a *time-space network* that does not explicitly consider all possible connections between trips. The network structure exploits the transitivity property of partial ordered sets and aggregates connections between groups of compatible trips. In fact, this approach reduces the number of connection arcs dramatically (by 97 to 99%) if the number of start and end locations is small compared to the number of trips. They report solving large scale real-world instances with up to 7,068 trips and five depots to optimality with an off-the-shelve mixed-integer programming (MIP) optimization software. In [Gintner et al., 2005] the authors propose a two-phase heuristic that first fixes some connections based on the solutions of easier subproblems and optimizes the reduced model with a standard MIP solver. Their results indicate that close to optimal solutions can be found for very large scale instances in reasonable time, e.g., for an instance with 11,062 trips and 55 depot/vehicle type combinations in about five hours.

In contrast to the latter formulations columns in set partitioning models for the

MDVSP correspond to feasible vehicle routes. The basic idea is to enumerate all feasible vehicle routes and choose a subset of routes that partitions the set of trips. Set partitioning formulations can be derived from multicommodity flow formulations by applying the Dantzig-Wolfe decomposition principle (see [Ribeiro and Soumis, 1994]). Recently, [Hadjar et al., 2006] discuss a branch-and-bound algorithm for a set partitioning formulation that includes column generation, variable fixing, and cutting planes. As originally proposed by [Ribeiro and Soumis, 1994] they use a column generation algorithm to solve the LP-relaxation in each node of the branch-and-bound tree since the number of feasible columns is enormous. Furthermore, they apply a variable fixing strategy similar to [Bianco et al., 1994]. Their experiments on randomly generated instances involve up to 850 trips and 6 depots. However, their results are difficult to compare with results obtained from real-world instances since the average number of trips per vehicle route is small (approximately 4). Apparently, such instances are not very realistic.

For a recent survey on models for multiple-depot vehicle scheduling problem the reader is referred to [Bunte and Kliewer, 2006].

Crew Scheduling

Crew scheduling is similar to vehicle scheduling but is more complex due to duty feasibility constraints. It has received considerable attention in the operations research literature since the late eighties. An overview of earlier works on crew scheduling can be found in [Carraresi and Gallo, 1984]. In the following we will not only review literature from public bus transport, we will rather include sources from airline crew scheduling since similar models and solution approaches are used there. Recall from section 1.1.2 that the crew scheduling problem (CSP) is NP-hard even if only working time or spread time constraints are imposed (see [Fischetti et al., 1987] and [Fischetti et al., 1989]).

The CSP is usually formulated as a set partitioning problem and solved with a column generation approach since the number of columns is huge in real-world problems. In such a model columns represent feasible crew duties while the constraints ensure that each vehicle activity (task) outside the depot is covered by exactly one driver. Consequently, duty feasibility constraints have to be considered in the pricing problem only. Several authors formulate the CSP as set covering problem that allow tasks to be over-covered. In practice, this over-covering is often not acceptable, but solutions of this model often contain little or no over-covers (since it is cheaper to assign only one driver to a task). The main advantage of a set covering over a set partitioning formulations is that continuous and integer solutions can be easier computed.

In [Desrochers and Soumis, 1989] column generation was applied to the CSP in public transport for the first time. They propose to solve the LP relaxation of a set covering formulation with column generation and model the pricing problem as a resource constrained shortest path problem. A dynamic programming algorithm similar to [Desrochers and Soumis, 1988] is used to generate new negative reduced cost columns. Moreover, in order to obtain integer solutions, the column generation algorithm is used to solve the linear relaxation of the master problem in each node of the branch-and-bound tree. A similar approach for airline crew scheduling (pairing) is proposed by [Lavoie et al., 1988] where the pricing problem corresponds to a pure shortest path problem on a specific state-expanded network structure. Other successful applications of a column generation approach to solve the LP relaxation of a set partitioning/covering formulation are [Falkner and Ryan, 1992] and [Desrochers et al., 1992]. In the following fifteen years, branch-and-price approaches have been further refined by acceleration strategies (see [Desaulniers et al., 2002]) , stabilization (see [Du Merle et al., 1999] and [Ben Amor et al., 2006]), and heuristics (see [Barnhart et al., 1998], [Vance et al., 1997a], and [Danna and Le Pape, 2005]) in order to tackle huge real-world crew scheduling problems.

Instead of solving the linear relaxation of the master problem by column generation, [Carraresi et al., 1995] and [Freling, 1997] approximately solve the Lagrangian relaxation by column generation. A branch-and-price heuristic with speed-up techniques is proposed by [Grötschel et al., 2003]. Instead of solving the Lagrangian dual with a subgradient method, they solve the dual of the master problem by a coordinate ascent method (e.g. [Wedelin, 1995]) in combination with a boxstep stabilization technique (see [Marsten et al., 1975]). Both subgradient and coordinate ascent methods are based on Lagrangian relaxation. The pricing problem is a resource constrained shortest path problem that is solved by a two-phase algorithm. First, Lagrangian distance labels are generated. In a second step, these labels serve as backtracking criterion in an enumerative algorithm. Finally, they propose a heuristic variable fixing strategy within a *branch-and-generate* framework that does not allow to backtrack.

However, duty feasibility constraints often arise in practice that cannot be covered with a resource constrained shortest path formulation. A simple workaround in order to deal with this problem is to ignore these rules in the RCSP pricing problem and skip all duties that violate them in a second phase. Another way of overcoming this difficulty has been suggested by [de Silva, 2001], [Fahle, 2002], and [Yunes et al., 2005] who apply *constraint programming* techniques to solve the pricing problem. Constraint programming models allow to model a wide variety of complex work rules that cannot be covered by a resource constrained shortest

path formulation. In particular, the results of [Yunes et al., 2005] indicate that in a column generation context a combination of mathematical programming and constraint programming performs better than isolated approaches.

Many heuristic approaches to the crew scheduling problem have been suggested: see [Wren and Rousseau, 1995] for a survey and, more recently, [Fores et al., 2002]. Most of the approaches heuristically generate a subset of feasible duties and solve a set partitioning/covering model with these duties afterwards. Furthermore, several metaheuristics have been proposed for solving the crew scheduling problem. Genetic algorithms are used by, among others, [Wren and Wren, 1995], [Kwan et al., 1999], [Kwan et al., 2001], [Marchiori and Steenbeek, 2003], and [Li and Kwan, 2005]. [Cavique et al., 1999] and [Shen and Kwan, 2001] suggest tabu search algorithms. [Lourenço et al., 2001] propose a genetic and tabu search algorithm that involves multiple objectives.

[Barnhart et al., 2003] and [Gopalakrishnan and Johnson, 2005] provide extensive surveys on the state-of-the-art of airline crew scheduling.

2.2.2. Partial Integration

Scheduling vehicles independently of crews was seriously criticized in the early eighties by [Bodin et al., 1983] since crew costs mostly dominate vehicle costs. Although integrated vehicle and crew scheduling problems have been proposed in literature at that time (see [Ball et al., 1983] and [Patrikalakis and Xerocostas, 1992]), it was not until recently that problems of considerable size and multiple depots could be solved in an integrated manner (see [Huisman et al., 2005a]). Consequently, most approaches until the late nineties were based on a heuristic integration of both problems since a fully integrated consideration was computationally intractable. Such a heuristic integration is called a *partial integration* of vehicle and crew scheduling. Similar to [Freling, 1997] we distinguish between two types of partial integration:

- perform crew scheduling but include vehicle scheduling considerations and construct a feasible vehicle schedule afterwards (crew first - vehicle second),
- perform vehicle scheduling but include crew scheduling considerations and subsequently generate a feasible crew schedule (vehicle first - crew second).

Most approaches of the first category are inspired by the landmark contribution [Ball et al., 1983]. The authors define a multigraph that shares the same set of nodes but contains two types of arcs: one type for combined *vehicle-crew* activities and another for *crew-only* activities (such as waiting or walking). The

set of nodes corresponds to a source, a sink, and the set of tasks (called *d-trips*), i.e., elementary portions of work that must be operated by one driver and one vehicle. The single depot is represented by the source and the sink. Each vehicle schedule corresponds to a set of node disjoint paths from the source to the sink that partitions the node set and that uses only combined vehicle-crew arcs. Similarly, a feasible crew schedule corresponds to a set of node disjoint paths from the source to the sink that partitions the node set and that may contain both types of arcs. Furthermore, both sets of paths must be compatible to make an overall feasible solution. Vehicle and crew schedule are compatible if each vehicle-crew arc in the vehicle schedule is also covered in the crew schedule while all vehicle-crew arcs not contained in the vehicle schedule are not part of the crew schedule. However, the integrated model is not practical due to prohibitive network dimensions. Thus, the solution process is decomposed into three parts that emphasize the crew scheduling phase: heuristically construct a set of pieces of work, improve the set of pieces by recombination, and generate a feasible crew schedule. The pieces of work are constructed in such a way that a feasible vehicle schedule can always be derived. To sum up, the model is integrated while the solution approach is sequential. Other heuristic approaches of the first category have been suggested by [Tosini and Vercellis, 1988], [Falkner and Ryan, 1992], and [Patrikalakis and Xerocostas, 1992]. All these approaches use a network structure similar to [Ball et al., 1983].

Heuristic approaches of the second category have been suggested by [Scott, 1985] and [Darby-Dowman et al., 1988]. In the latter, an interactive decision support system is described that allows to include crew scheduling considerations while performing vehicle scheduling. However, no details on the models and algorithms are provided. The system is part of a planning system developed for the Rome transport agency.

Recently, another approach of the second category is proposed by [Borndörfer et al., 2002]. The authors modify the costs in the vehicle scheduling problem in such a way that pull-in/out trips are encouraged while connections between long service trips are discouraged. Furthermore, the authors impose constraints on the length of vehicle blocks. These modifications aim at generating vehicle blocks with many relief opportunities for drivers. As a consequence, there is more flexibility when crews are scheduled.

Another interesting approach that belongs to neither category was proposed by [Gintner et al., 2004, Gintner et al., 2006a] and [Gintner, 2007]. The basic idea is to change a given optimal vehicle schedule without loss of optimality in the crew scheduling phase. They set up a time-space network that allows to recombine parts of vehicle blocks in order to disclose additional flexibility in crew

scheduling while maintaining the optimality of vehicle schedule. In other words, the crew scheduling approach does not only consider a single optimal vehicle schedule, but a set of optimal vehicle schedules with minimum fleet size and minimum operational costs. The authors report savings of 8-24% on real-world instances as compared with a sequential approach.

2.2.3. Complete Integration

As reviewed in the previous subsection, only few partially integrated approaches have been suggested for the integrated vehicle and crew scheduling problem in the eighties and the early nineties. However, the problem has lately attracted several researchers who developed models and solution techniques mainly based on decomposition approaches for mathematical programming. In the following, we will first discuss integrated models for the single depot and then for the multiple-depot case.

Single Depot Case

The first mathematical formulation for the integrated vehicle and crew scheduling problem with a single depot is given in [Patrikalakis and Xerocostas, 1992]. However, the model is computationally intractable and, thus, the authors resort to a partially integrated solution method.

[Freling, 1997] propose the first integrated treatment of vehicle and crew scheduling in terms of model and solution approach. The model consists of three components: a quasi-assignment formulation for vehicle scheduling, a set partitioning formulation for crew scheduling, and additional linking constraints that ensure the compatibility of vehicle and crew schedule. He suggests a solution approach that is based on column generation in combination with Lagrangian relaxation. The set partitioning and linking constraints are relaxed such that two independent Lagrangian subproblems remain: a single depot vehicle scheduling problem and an easy selection problem. The Lagrangian dual problem is solved by a subgradient algorithm while a novel two-phase pricing method is proposed to generate new columns (duties). Finally, he applies several heuristics to obtain feasible integer solutions. The approach is tested on real-world and randomly generated instances with up to 148 trips on a Pentium 90 PC with 32MB of main memory. The largest instance is solved in approximately one hour where 96% of the time is spent on the column generation pricing problem. The solution methodology of [Freling, 1997] has inspired a series of publications (e.g. [Freling et al., 2001a], [Freling et al., 2003], and [Huisman, 2004]) and also forms the basis

of our solution approach (see Section 2.4).

[Friberg and Haase, 1999] propose an integrated mathematical formulation that combines two set partitioning models: the vehicle scheduling model of [Ribeiro and Soumis, 1994] with the crew scheduling model of [Desrochers and Soumis, 1989]. They propose an exact branch-and-cut-and-price algorithm where they solve the LP-relaxation in each node of the search tree by column generation. Furthermore, clique cuts (see [Hoffman and Padberg, 1993]) are derived to tighten the LP-relaxation. However, only small sized instances with up to 20 trips are solved within one hour of computational time on a SUN Sparc-10/40.

[Haase et al., 2001] introduce an interesting crew scheduling formulation with side constraints that involves duty flow variables and a bus counter variable. The set partitioning formulation with flow conservation and bus count constraints (similar to the plane count constraints of [Klabjan et al., 2002]) guarantees that an optimal vehicle schedule can always be derived afterwards. They propose an elaborate branch-and-price algorithm that relies on several acceleration strategies, e.g., dynamic generation of bus count constraints and appropriate substitution of partitioning constraints in order to reduce column density. Computational results with randomly generated data show that instances with at most 150 trips (300 tasks) can be solved in 82 minutes on a SUN Ultra-10/400 and an average optimality gap of 0.3%. In order to tackle larger instances they suggest a heuristic version where multiple branching decisions are made at every node of the search tree. With this approach they solve instances with up to 350 trips (700 tasks) in approximately two hours on average and with an average integrality gap of 0.3%.

[Borndörfer et al., 2002] suggest a formulation for the integrated problem with a single depot that combines the vehicle scheduling model of [Löbel, 1998] with the crew scheduling approach of [Borndörfer et al., 2001, Grötschel et al., 2003]. They propose a column generation algorithm based on Lagrangian relaxation to solve a linked multicommodity network flow and set partitioning formulation. The Lagrangian dual problem is solved with a subgradient method. They apply two primal heuristics to compute feasible solutions. Their computational tests involve three scenarios with up to 1,457 trips and one depot and is performed on a dual Intel Xeon PC 1.7GHz with 1GB of main memory. They solve the largest instance in approximately 6.5 hours where 50% of the cpu time is spent on the column generation pricing problem. However, the authors do not compute a valid lower bound, and, thus, cannot assess the quality of their solutions.

[Valouxis and Housos, 2002] describe a combined vehicle and crew scheduling problem that is actually a crew scheduling problem since drivers are tied to their vehicle. They propose a fast heuristic which is based on column generation and solve instances with up to 350 trips within a given 30 minute timeframe. Another

elaborate heuristic is proposed by [Rodrigues et al., 2006] where vehicle and crews are scheduled in an integrated way. Unlike other approaches mentioned so far, the set of trips is not given in advance. Instead, the timetable is heuristically constructed to meet an estimated passenger demand. They test their algorithm on a real-world scenario from Sao Paulo in Brazil with up to 395 trips. However, they do not compute lower bounds and, consequently, the quality of their solutions cannot be assessed.

Multiple-Depot Case

The main difference between single and multiple-depot integrated vehicle and crew scheduling is that the vehicle scheduling subproblem with multiple depots is NP-hard unlike the single depot case (see Section 2.2.1).

The integrated vehicle and crew scheduling problem with multiple depots has been introduced by [Gaffi and Nonato, 1999]. Their approach is developed for ex-urban public transport systems where crews are virtually tied to their vehicle or crew-deadheading (by foot) is highly constrained. In particular, crews may only be relieved in depots and, thus, vehicle blocks correspond to pieces of work. These assumptions make the problem computationally much more attractive than the general case that we consider in this thesis. Their formulation consists of two parts: a quasi-assignment model for scheduling vehicles and a set of linking constraints to ensure compatibility of the crew schedule. Furthermore, the number of vehicle blocks per depot and the number of duties of a specific type can be restricted. The authors develop a heuristic column generation algorithm based on a Lagrangian relaxation with all linking constraints relaxed. They test their approach on a Power PC 604/180MHz with real-world ex-urban and sub-urban instances. Their results with instances of up to 257 trips and 28 depots in the ex-urban setting show that their algorithm finds feasible solutions to all instances while the planning system currently used could not. The cpu time is greater than 24 hours for the largest instance and between 2 and 6 hours on average. Their approach seems to be less suitable for the sub-urban setting with more relief opportunities and smaller distances between the depots. However, they do not compute lower bounds which makes it difficult to assess the quality of their solutions.

[Huisman, 2004, Huisman et al., 2005a] investigate two formulations that generalize the single depot models of [Freling, 1997, Freling et al., 2003] and [Haase et al., 2001] to the multiple depot case. Moreover, they propose two similar adaptations of the solution approach that was suggested for the single depot case by [Freling, 1997]. In the following we will review the formulation of [Huisman,

2004,Huisman et al., 2005a] that is based on the single depot formulation of [Freling, 1997]. We will compare this formulation with the formulation of [Gintner, 2007] in Section 2.3.

Let $\mathcal{T} = \{1, 2, \ldots, n\}$ be the set of n timetabled trips where trip i starts at time st_i and ends at time et_i. We assume that the set of trips is ordered by increasing start times with $st_i \leq st_{i+1}$. Furthermore, we denote by τ_{ij} the travel time between the end location of trip i and the start location of trip j. Two trips i and j are said to be *compatible* if they can be covered consecutively by the same vehicle, that is $et_i + \tau_{ij} \leq st_j$ holds. Now, let us define $\mathcal{H} = \{(i,j) | i < j, i \text{ and } j \text{ compatible}, i \in \mathcal{T}, j \in \mathcal{T}\}$ as the set of deadheads (including waiting activities outside the depot). Let $\mathcal{D} = \{1, 2, \ldots, m\}$ be the set of depots. For each depot $d \in \mathcal{D}$ we define an acyclic vehicle scheduling network $G^d = (N^d, A^d)$ with nodes $N^d = \mathcal{T}^d \cup \{r^d, t^d\}$ and arcs $A^d = \mathcal{H}^d \cup (r^d \times \mathcal{T}^d) \cup (\mathcal{T}^d \times t^d)$ where both r^d and t^d represent depot d. The set of trips and deadheads that can be serviced from depot $d \in \mathcal{D}$ is denoted by $\mathcal{T}^d$ and $\mathcal{H}^d$, respectively. *Pull-out* (*pull-in*) trips are denoted by $r^d \times \mathcal{T}^d (\mathcal{T}^d \times t^d)$. We associate vehicle costs c^d_{ij} with each arc $(i,j) \in A^d$ that are typically a function of travel and idle time. Moreover, we add a fixed (asset) cost for using a vehicle to the cost of each pull-out arc. As stated earlier, we assume that a vehicle returns to its depot if the idle time between two trips is long enough to perform a round-trip to the depot. Deadhead arcs between trips that allow a round-trip to the depot are called *long arcs* $A^{ld} \subset A^d$. All other deadhead arcs between trips are *short arcs* $A^{sd} \subset A^d$. Finally, we introduce two types of decision variables: flow variables and duty variables. Flow variable y^d_{ij} indicates whether arc $(i,j) \in A^d$ is used and assigned to depot $d \in \mathcal{D}$ or not. Likewise, duty variable $x^d_k \in K^d$ with associated cost f^d_k indicates whether duty k is selected for depot $d \in \mathcal{D}$ or not. Furthermore, $K^d(i)$ denotes the set of duties that cover trip $i \in \mathcal{T}^d$ while $K^d(i,j)$ denotes the set of duties covering deadhead task $(i,j) \in A^{sd}$. Note that this implicitly assumes that a trip corresponds to exactly one task. [Huisman, 2004] proposes to state the integrated vehicle and crew scheduling problem with multiple depots (MDVCSP-H) as follows.

$$\sum_{d \in \mathcal{D}} \sum_{(i,j) \in A^d} c^d_{ij} y^d_{ij} + \sum_{d \in \mathcal{D}} \sum_{k \in K^d} f^d_k x^d_k \quad \rightarrow \quad \min \tag{2.1}$$

$$s.t. \qquad \sum_{d \in \mathcal{D}} \sum_{j:(i,j) \in A^d} y^d_{ij} = 1 \qquad \forall i \in \mathcal{T} \tag{2.2}$$

$$\sum_{d \in \mathcal{D}} \sum_{i:(i,j) \in A^d} y^d_{ij} = 1 \qquad \forall j \in \mathcal{T} \tag{2.3}$$

$$\sum_{i:(i,j) \in A^d} y^d_{ij} - \sum_{i:(j,i) \in A^d} y^d_{ji} = 0 \qquad \forall d \in \mathcal{D}, \forall j \in N^d \tag{2.4}$$

$$\sum_{j:(i,j)\in A^d} y_{ij}^d - \sum_{k\in K^d(i)} x_k^d = 0 \qquad \forall d \in \mathcal{D}, \forall i \in N^d \tag{2.5}$$

$$y_{ij}^d - \sum_{k\in K^d(i,j)} x_k^d = 0 \qquad \forall d \in \mathcal{D}, \forall (i,j) \in A^{sd} \tag{2.6}$$

$$y_{it^d}^d + \sum_{j:(i,j)\in A^{ld}} y_{ij}^d - \sum_{k\in K^d(i,t^d)} x_k^d = 0 \qquad \forall d \in \mathcal{D}, \forall i \in N^d \tag{2.7}$$

$$y_{r^d j}^d + \sum_{i:(i,j)\in A^{ld}} y_{ij}^d - \sum_{k\in K^d(r^d,j)} x_k^d = 0 \qquad \forall d \in \mathcal{D}, \forall j \in N^d \tag{2.8}$$

$$y_{ij}^d \in \{0,1\} \quad \forall d \in \mathcal{D}, \forall (i,j) \in A^d \tag{2.9}$$

$$x_k^d \in \{0,1\} \quad \forall d \in \mathcal{D}, \forall k \in K^d \tag{2.10}$$

The objective function (2.1) minimizes the sum of vehicle and duty costs. Constraint sets (2.2)-(2.4) correspond to a multicommodity flow formulation for the vehicle scheduling problem. Constraint set (2.5) imposes that each trip will be covered by a duty from a depot if and only if the trip is covered by a vehicle from the same depot. Similarly, constraints (2.6)-(2.8) establish the link between vehicle and crew deadheads where deadheads corresponding to short and long arcs are considered separately.

The solution approach to solve MDVCSP-H consists of two phases: the first phase computes a lower bound on the optimal solution value while a feasible solution is constructed in the second phase. To obtain a lower bound he solves the linear relaxation of model MDVCSP-H using a column generation method in combination with Lagrangian relaxation. The author relaxes constraints (2.4)-(2.8) and, hence, obtains a large single depot vehicle scheduling problem and a trivial selection problem as Lagrangian subproblems. The Lagrangian dual problem is solved with a subgradient method while the vehicle scheduling subproblem is solved with the combined forward and reverse auction algorithm of [Freling, 1997]. As originally proposed in [Freling, 1997] he uses a two phase (column generation) pricing problem to generate new negative reduced cost columns where pieces of work are generated in the first and feasible duties in the second phase. Finally, a heuristic solution to model MDVCSP-H is constructed where constraints (2.5)-(2.8) are relaxed in a Lagrangian way. Again, the Lagrangian dual problem is solved with a subgradient method where the Lagrangian subproblem corresponds to a multiple-depot vehicle scheduling problem. Hence, each feasible solution to the subproblem constitutes a feasible vehicle schedule. Feasible crew schedules are constructed by solving a crew scheduling problem based on the current solution of the Lagrangian subproblem. However, only columns generated in the lower bounding phase are considered, that is no new columns are constructed in the second phase.

A series of tests on real-world and randomly generated data with up to 653 trips indicates that both integrated approaches lead to efficiency gains compared to sequential planning (vehicle first - crew second). The average number of depots which a trip can be assigned to lies between 1.27 and 2.47 for real-world and is either 2 or 4 for artificial instances. The cpu times to compute lower bounds for the real-world instances varies a lot and takes at most six hours on an Intel Pentium III PC 450MHz with 128 MB main memory. The results on artificial data show that instances with up to 200 trips and 2 depots can be solved in approximately 2 hours on average while it takes about 2.5 hours on average for 160 trips and 4 depots. The cpu time is limited to 3 hours per randomly generated instance. About 85% percent of the cpu time is spent in the column generation pricing problem. The gap between best lower and best upper bound lies between 5.31% and 8.11% for the approach based on model MDVCSP-H. Additionally, neither of the integrated approaches can outperform the other. However, the approach based on model MDVCSP-H regularly provides tighter lower bounds. In order to tackle large instances, [de Groot and Huisman, 2004] suggest several heuristics that split large instances into several smaller ones which can be solved by an integrated or sequential vehicle and crew scheduling method. They use the same formulation and solution approach for integrated problems as stated above. They test their heuristics on real-world instances with up to 1,372 trips and 6 depots on a Intel Pentium IV PC 1.8GHz/512MB main memory. In their setting each trip can be assigned to 1.27-3.64 depots on average. Their results show that large (previously unsolved) instances can be solved now. Furthermore, they show that their heuristics disclose efficiency gains compared to a simple sequential approach. Interestingly, the best heuristic outperformed an integrated approach with a given time limit in terms of solution quality and time. The largest instance is solved in about 1 hour with a heuristic.

Another approach that relies on model MDVCSP-H is proposed by [Borndörfer et al., 2004]. They use a solution approach similar to the one sketched above since they aim at computing a lower bound first and subsequently generate an integer feasible solution. However, the authors solve the Lagrangian dual problem with an inexact adaptation of a proximal bundle method (see [Kiwiel, 1995]) that produces dual and additional primal information as opposed to a subgradient algorithm. The inexact bundle method is embedded in a backtracking procedure to produce integer solution in the second phase. The procedure utilizes the primal information produced by the bundle method to iteratively fix deadhead (flow) variables until the complete vehicle schedule is fixed. A compatible crew schedule is generated as described in [Grötschel et al., 2003]. The authors report computational results with both real-world and artificial data. All tests are

performed on a dual Intel Xeon PC 3GHz/4GB main memory. The largest real-world instance contains 1414 trips with 1 depot and 3 vehicle types and is solved in about 125 hours. Furthermore, they compare their approach with [Huisman, 2004] on the same set of artificial instances. Using the same assumptions their approach clearly outperforms Huisman's method and solves instances with up to 400 trips and 2 (4) depots in 3.3 (12) hours.

Very recently, [Mesquita and Paias, 2006] propose two mathematical formulations similar to MDVCSP-H but with fewer constraints. Both models involve a multicommodity network flow model for vehicle scheduling while the crew scheduling part either relies on a pure set partitioning or on a combined set partitioning/covering formulation. They develop a column generation algorithm where the LP relaxation of their formulation is solved with a commercial LP solver. The column generation subproblem corresponds to a resource constrained shortest path problem that is either solved exactly or approximately by a dynamic programming algorithm similar to [Desrochers and Soumis, 1988]. If the optimal solution of the LP relaxation is not integer, they use a commercial IP (branch-and-bound) solver to find an integer solution over a subset of feasible crew duties. They show that integer solutions can be obtained by branching on one type of decision variables, i.e., either flow or duty variables. They report computational results for the randomly generated instances that have also been used by [Borndörfer et al., 2004] and [Huisman, 2004, Huisman et al., 2005a]. However, it should be mentioned here that they make different assumptions and, therefore, their results cannot be directly compared for the following reasons.

- They do not assume that a vehicle returns to its depot if the idle time between two consecutive trips is long enough to perform a round-trip to the depot. Consequently, it is easy to construct a piece of work that is feasible in their definition but not in Huisman's.

- The authors only provide computational results for the case where they do not assume that a crew may only conduct tasks on vehicles from a single depot, i.e., changeovers are allowed during a piece of work (and not only during a break). This makes the problem computationally much more attractive. Furthermore, they provide computational results where drivers may walk on deadhead connections that are not part of the vehicle schedule.

- A different set of duty types is used that expands the solution space.

Nevertheless, all computational tests are performed on an Intel Pentium IV 3.2GHz. The authors are able to solve instances with up to 400 trips and 4 depots in less than 4 hours. It still remains an open question whether their

method is also effective under the assumptions as stated by [Huisman, 2004]. In [Mesquita et al., 2006] the authors compare different branching strategies for the model and solution approach sketched above. They are able to solve the same randomly generated instances as above with up to 100 trips in about 3.5 hours using an exact branch-and-price algorithm. The results of the heuristic branching schemes correspond to the results presented in [Mesquita and Paias, 2006].

[Hollis et al., 2006] present a new set covering formulation with side constraints for an integrated vehicle and crew scheduling problem with multiple depots faced by Australia Post mail distribution. The main difference to all approaches discussed before is that the set of trips is not given in advance. Instead the set of trips (routes) is heuristically constructed by solving a vehicle routing problem prior to the actual integrated problem. Furthermore, as opposed to other formulations they include crew and vehicle capacity constraints separated by depot. They use a heuristic column generation procedure to solve instances with up to 1,181 shipments. We do not provide more details on their results since they are not comparable with those given above.

Finally, [Gintner, 2007] proposes a model that is based on a time-space network and leads to a mathematical formulation with fewer constraints and variables compared to approaches previously exposed in literature. Since we will use this formulation in the remainder of this thesis, we devote the next section to describe this model in detail. Moreover, we describe his solution approach in Section 2.4.

2.3. Modeling approach

In this section we discuss a modeling approach and mathematical formulation for the integrated vehicle and crew scheduling problem with multiple depots under the assumptions stated in Section 2.1. The formulation is introduced by [Gintner, 2007] and combines a multicommodity network flow formulation for vehicle scheduling with a set partitioning formulation for crew scheduling. The main advantage of this formulation is the structure of the underlying vehicle scheduling network that leads to models with fewer constraints and variables compared to approaches previously exposed in literature. Recall from Section 2.2.1 that multicommodity network flow formulations for multiple-depot vehicle scheduling problems can be classified by the underlying network structure. In a *connection-based network* (CBN), each feasible connection between two trips corresponds to an explicit arc in the network while in a *time-space network* (TSN) only connections between groups of compatible trips are considered. In fact, a time-space network approach reduces the number of connection arcs dramatically if

the number of start and end locations is small compared to the number of trips. A time-space network structure for the multiple depot vehicle scheduling problem in public transport has been introduced by [Kliewer, 2005, Kliewer et al., 2006b] and adapted for integrated vehicle and crew scheduling problems by [Gintner, 2007]. In the following, we will first describe the time-space network used and, subsequently, present the mathematical formulation that we will use in the remainder of this thesis. The exposition in this section follows [Gintner, 2007].

In a time-space network each node represents a specific location at a particular time while each arc corresponds to a transition in time and, possibly, space. In order to ease the exposition we first assume that there is only one depot and one vehicle type. The vehicle scheduling solution must satisfy flow conservation constraints that force the vehicles to circulate through a network of service trips where each vehicle must return to the depot where its daily schedule has started. In a time-space network, flow conservation is enforced by modeling the activity at each station (including the depot) with a timeline. Each timeline contains nodes that either represent arrivals or departures from the station. Each departure (arrival) splits an edge of the timeline and adds a node to the timeline at the departure (arrival + minimum layover) time (see Figure 2.1). Notice that each

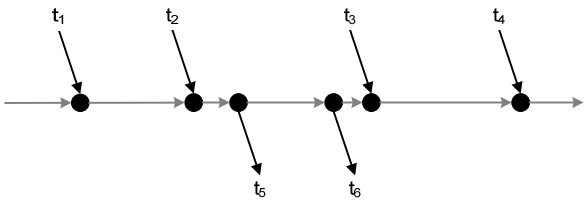

Figure 2.1.: Timeline of a station with four arrivals and two departures.

arrival node has only one outgoing waiting arc in a time-space network while in a connection-based network each arrival node entails an arc to each (feasible) subsequent departure node. A trip arc is used to connect the corresponding departure and arrival nodes at the start and end location of a trip. Furthermore, a pull-out (pull-in) trip arc is added from (to) the depot for each trip including the corresponding departure (arrival) nodes in the depot timeline. The timeline of the depot is made a cycle to force a circulation through the network that arises from the flow conservation constraints and the lack of source and sink nodes. Since the network has a timespan of one day, the circulation flow defined by a solution defines a daily vehicle schedule. A flow along a timeline (on waiting arcs) represents a vehicle waiting at the station while a flow on pull-in/out and

trip arcs corresponds to vehicle movements. Each unit of flow on the *circulation arc* from the last to the first node of the depot timeline corresponds to a vehicle used. Consequently, each path from the first to the last depot node represents a daily schedule for one vehicle.

Vehicle movements without passengers (deadheads) are virtually unrestricted in public bus transport. Thus, a deadhead arc is added between two compatible trips that require a deadhead from the end location of the first trip to the start location of the second one. However, it is not necessary to connect all pairs of compatible trips explicitly as in a connection-based network. Instead each arrival node is connected with the next compatible departure node of the start location of the second trip. All subsequent connections at the station are implicitly included by traversing the timeline.

As an example, Figure 2.2 shows the deadhead arcs of a connection-based and time-space network for 11 trips that arrive at station A or depart from station B. In time-space networks, there is at most one deadhead arc to connect a trip with all subsequent trips at a different station as opposed to a connection-based network where there is an explicit arc for each pair of compatible trips. Moreover, in time-space networks, not all arrival nodes at station A have an outgoing deadhead arc to station B, e.g., arrival node of trip t_2. There is no benefit of adding a deadhead arc between the arrival node of t_2 and the departure node of t_9 since both trips can be connected by following the timeline at station A (and using the deadhead between t_3 and t_9). The same reasoning holds for connecting trip t_4 and t_{10}. In other words, if a group of arrivals at station A has the same first compatible departure at station B, only the latest arrival of the group must be connected with the first compatible departure. In our example, we have 14 deadhead arcs in the connection-based network, but only 3 in the time-space network. In time-space networks, the small number of deadhead arcs generally outweighs the overhead generated by introducing waiting arcs in timelines. We will discuss the network complexity of both network structures in the next paragraph.

Let n be the number of service trips and m the number different start and end locations. Typically, the number of different start and end locations is small compared to the number of service trips in real-world settings. As stated earlier there is at most one deadhead arc to connect a trip with all subsequent trips at a different station in time-space networks as opposed to a connection-based network where there is an explicit arc for each pair of compatible trips. While the number of deadhead arcs in a connection-based network is $\mathcal{O}(n^2)$, time-space networks only contain $\mathcal{O}(nm)$ deadhead arcs with $n \gg m$. Notice that the number of waiting arcs grows linearly with the number of tasks. As discussed in [Gintner,

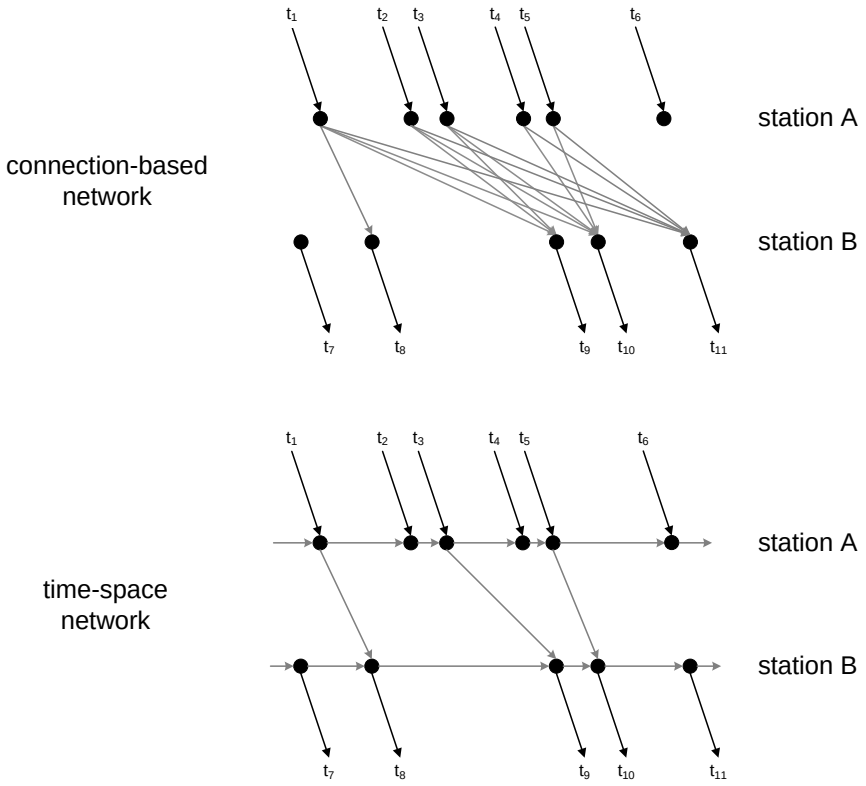

Figure 2.2.: Deadhead arcs in a connection-based and time-space network between two stations

2007], Table 2.1 provides the number of deadhead arcs for instances with 100 to 2,000 trips for both network formulations. It is easy to see that the number of deadhead arcs can be dramatically reduced (up to 99%) with a time-space network formulation. It is important to mention that both networks are directed acyclic graphs and contain all compatible connections between trips.

Finally, the costs and capacities associated with each arc are defined as follows. Generally, vehicle costs consist of both fixed and variable costs where variable costs reflect operating time outside the depot and distance covered. Thus, the sum of operating time and distance costs is assigned to pull-in/out, trip, and deadhead arcs while only operating time costs are considered for waiting arcs outside the depot. A vehicle waiting inside the depot does not incur any costs. In order to reflect asset costs of vehicles, the circulation arc takes a fixed cost for each unit of flow. The maximum capacity of pull-in/out and trip arcs is set to 1

network type	# trips				
	100	200	400	800	2,000
connection-based network	4,043	16,396	65,788	269,462	1,879,262
time-space network	362	946	2,106	4,589	17,086

Table 2.1.: Number of deadhead arcs of a connection-based and time-space network structure as presented in [Gintner, 2007]

while all other arcs have a maximum capacity equal to the number of available vehicles.

Figure 2.3 depicts an example of a time-space network with three stations, six trips, and one depot. Notice that there is no waiting arc at station B between

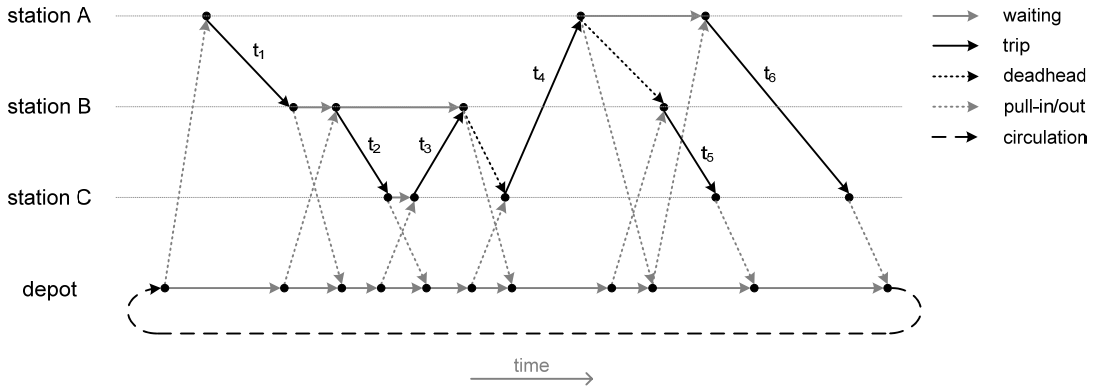

Figure 2.3.: Time-space network with six trips

trips t_3 and t_5 since we assumed in Section 2.1 that a vehicle returns to its depot if the idle time between two consecutive trips is long enough to perform a round-trip to the depot.

So far, a directed acyclic time-space network structure is defined for scheduling vehicles where each path from the first to the last depot node corresponds to a daily schedule for one vehicle. Now, we will turn our attention to scheduling crews. As assumed in Section 2.1 each trip starts and ends with a relief point. As a result, each node in the vehicle scheduling network corresponds to a relief point for drivers and each arc (except waiting arcs in the depot) represents a task. Moreover, each path between two nodes is associated with a piece of work if it satisfies piece feasibility constraints. Additionally, such a path must not contain

waiting arcs in the depot since each stop in the depot terminates the current piece of work. A duty consists of at least one such piece of work (path). A duty is feasible if all duty feasibility constraints are satisfied and, clearly, incurs costs.

To sum up, the integrated vehicle and crew scheduling problem aims at finding a (vehicle) network flow solution and a set of duties such that each each flow unit on the arcs (vehicle activity) of the network flow solution is also covered by exactly one duty while all deadheads not contained in the vehicle schedule are not part of any duty. Furthermore, the sum of vehicle and crew costs is to be minimized.

If there are multiple depots,a network for each depot is set up where each trip is represented by several trip arcs. Each trip arc corresponds to a depot-trip combination. Of course, only one depot-trip combination can be selected for a trip. Next, we discuss the mathematical formulation introduced by [Gintner, 2007] that is based on the time-space network described above.

Let $\mathcal{T} = \{1, 2, \ldots, n\}$ be the set of n timetabled trips and $\mathcal{D} = \{1, 2, \ldots, m\}$ be the set of depots. The set of trips that can be serviced from depot $d \in \mathcal{D}$ is denoted by $\mathcal{T}^d$. For each depot $d \in \mathcal{D}$, $G^d = (N^d, A^d)$ defines an acyclic vehicle scheduling network as described earlier in this section with N^d as the set of nodes and A^d as the set of arcs. $\widetilde{A}^d \subset A^d$ denotes the set of arcs that requires both vehicle and crew activities, i.e., all arcs except waiting arcs of depot timelines. Let $A^d(t) : \mathcal{T} \rightarrow A^d$ be a function that returns the set of trip arcs $(i, j) \in A^d$ for trip $t \in \mathcal{T}$ and depot $d \in \mathcal{D}$. Note that $A^d(t) = \emptyset$ if t cannot be operated from depot d. A vehicle cost c^d_{ij} is associated with each arc $(i, j) \in A^d$ which is typically a function of travel and idle time. Moreover, a fixed (asset) cost for using a vehicle from depot d is put on each circulation arc. The maximum capacity u^d_{ij} of pull-in/out and trip arcs $(i, j) \in A^d, \forall d \in \mathcal{D}$ is set to 1 while all other arcs have a maximum capacity u^d equal to the number of vehicles available in depot $d \in \mathcal{D}$. Finally, two types of decision variables are introduced: flow variables and duty variables. Flow variable y^d_{ij} indicates whether arc $(i, j) \in A^d$ is used and assigned to depot $d \in \mathcal{D}$ or not. Likewise, the binary duty variable $x^d_k, k \in K^d$ with associated cost f^d_k indicates whether duty k is selected for depot $d \in \mathcal{D}$ or not. Furthermore, K^d denotes the set of duties that can be operated from depot $d \in \mathcal{D}$ while $K^d(i, j) \subset K^d$ defines the set of duties covering arc $(i, j) \in \widetilde{A}^d$. The integrated vehicle and crew scheduling problem with multiple depots (MDVCSP) can be stated as follows:

$$\sum_{d \in \mathcal{D}} \sum_{(i,j) \in A^d} y^d_{ij} c^d_{ij} + \sum_{d \in \mathcal{D}} \sum_{k \in K^d} x^d_k f^d_k \rightarrow \min \tag{2.11}$$

$$
\begin{aligned}
s.t. \quad \sum_{d\in\mathcal{D}} \sum_{(i,j)\in A^d(t)} y_{ij}^d &= 1 && \forall t \in \mathcal{T} && (2.12)\\
\sum_{\{j:(j,i)\in A^d\}} y_{ji}^d - \sum_{\{j:(i,j)\in A^d\}} y_{ij}^d &= 0 && \forall d \in \mathcal{D}, \forall i \in N^d && (2.13)\\
\sum_{k\in K^d(i,j)} x_k^d - y_{ij}^d &= 0 && \forall\ d \in \mathcal{D}, \forall (i,j) \in \widetilde{A}^d && (2.14)\\
0 \le y_{ij}^d \le u_{ij}^d, y_{ij}^d &\in \mathbb{N} && \forall d \in \mathcal{D}, \forall (i,j) \in A^d && (2.15)\\
x_k^d &\in \{0,1\} && \forall d \in \mathcal{D}, \forall k \in K^d && (2.16)
\end{aligned}
$$

The objective (2.11) minimizes the sum of vehicle and crew costs. Constraints (2.12)-(2.13) correspond to a multicommodity flow formulation for the vehicle scheduling problem where the set of trip tasks must be partitioned among the depots (2.12) and flow conservation is ensured for each depot (2.13). Constraint set (2.14) establishes the link between vehicle and crew schedule: each arc covered by vehicle(s) must also be covered by the same number of duties assigned to the depot from which the vehicle(s) originate(s). Constraints (2.15) guarantee that the maximum capacity of the flow variables is satisfied.

A feasible solution to MDVCSP consists of a network flow solution and a compatible set of duties. It is important to mention that any feasible solution to the multicommodity flow formulation (2.12)-(2.13) represents several feasible vehicle schedules since only connections between groups of trips are considered in our network. However, a feasible vehicle schedule, that is also compatible to the crew schedule, can always be constructed using a decomposition algorithm (see [Gintner, 2007] for a description of the algorithm).

When the number of variables and constraints in models MDVCSP-H and MDVCSP is compared, we see that in both formulations a flow variable and a constraint are defined for each arc in the underlying vehicle scheduling network. As shown in Table 2.1 the number of arcs is considerably smaller with a time-space network structure. As a result, model MDVCSP is much more promising from a computational point of view since it contains fewer variables and constraints than model MDVCSP-H. Of course, models with smaller dimensions are not necessarily more attractive. However, [Gintner, 2007] shows that MDVCSP is indeed beneficial compared to MDVCSP-H.

2.4. Solution Approach

In this section, we discuss the solution approach to solve model MDVCSP as described in [Gintner, 2007]. Our exposition in this section follows [Gintner, 2007].

The solution method is a combination of column generation and Lagrangian relaxation and has been inspired by [Freling, 1997, Huisman, 2004]. The benefit of combining these methods is described in Section 1.5.3. Basically, the method will be used in the remainder of this thesis. However, in Chapter 3 we will propose new approaches for the column generation pricing problem as well as new methods for finding integer solutions. Furthermore, in Chapter 5 the general approach will be extended to include further requirements from practice. An outline of the approach is shown in Algorithm 5.

Algorithm 5: Solution method for model MDVCSP

(Step 1) **Initialization**
Solve MDVSP and, subsequently, CSP for each depot.
Take CSP solution as initial column set K'.
Set $t = 0$.

(Step 2) **Solve restricted master problem**
Solve a Lagrangian dual problem with the current set of columns K'.
Store lower bound for the current set of columns and dual information.

(Step 3) **Solve pricing problem**
Generate new columns K'' with negative reduced costs.
If $|K''| = 0$ terminate.

(Step 4) **Perform column management**
Add new columns to restricted master problem:
$K' := K' \cup K''$.
Delete columns with high positive reduced costs from K' if $|K'|$ is large.

(Step 5) **Check termination criteria**
Terminate if one of the following criteria is satisfied:
$t \geq t_{max}$
No significant improvement of lower bound.
otherwise set $t = t + 1$ and return to step 2.

(Step 6) **Construct feasible solution**
Use Lagrangian heuristic to construct feasible vehicle and crew schedules.

First, a feasible solution is generated by using the sequential approach where a multiple-depot vehicle scheduling problem (MDVSP) is solved using the solution approach of [Kliewer et al., 2006b] and standard optimization software such as MOPS® (see [Suhl, 2000]) or ILOG CPLEX® (see [ILOG, 2006]). Af-

terwards, a crew scheduling problem (CSP) is solved for each depot based on the vehicle schedule for that depot. The method we used to solve the CSP is described in [Gintner, 2007]. The set of columns obtained by solving the series of crew scheduling problems serves as initial column set for the column generation algorithm.

The main part of the algorithm (Step 2-5) computes a lower bound using a column generation algorithm in combination with Lagrangian relaxation. The Lagrangian relaxation and subgradient method used to obtain dual information and a lower bound on the current set of columns (Step 2) will be described in Section 2.4.1. In order to improve the lower bound the algorithm tries to find new duties with negative reduced costs in the pricing problem (Step 3). The pricing problem is the topic of Section 2.4.2. When the pricing algorithm finds new columns, these columns are added to the restricted master problem and, possibly, columns with high positive reduced costs are deleted in Step 4. The algorithm deletes columns only if the number of columns in the restricted master problem exceeds a given threshold value. The method iterates from Step 2 to 5 as long as new negative reduced cost columns are found, the number of iterations does not exceed t_{max}, and the lower bound improved significantly over the last n iterations. Finally, a feasible solution is computed using a Lagrangian heuristic (Step 6) which we will discuss in Section 2.4.3.

2.4.1. The Master Problem

As described in Section 1.5.3 Lagrangian relaxation can be applied to the extensive formulation (in a column generation context) in order to obtain approximate dual solutions and lower bounds to the restricted master problem. Instead of solving restricted problems with the simplex method to optimality, a subgradient method is used to solve the Lagrangian dual approximately. At the end of the subgradient phase, the Lagrangian multipliers are an approximation of the optimal dual variables for the current restricted master problem. In the following, we discuss the Lagrangian relaxation of model MDVCSP that we will use in the remainder of this thesis.

The linking constraints (2.14) in model MDVCSP require a simultaneous treatment of vehicle and crew scheduling. If these constraints are relaxed in a Lagrangian way, the model decomposes into a multiple-depot vehicle and crew scheduling problem that can be solved separately. Both problems are linked by penalizing incompatible vehicle and crew scheduling solutions in the objective function. While the crew scheduling part of the decomposed problem can be solved efficiently (by pricing out x variables), the remaining vehicle scheduling

part still constitutes an NP-hard problem. Consequently, constraints (2.12) are additionally dualized which results in several small single depot vehicle scheduling problems as vehicle scheduling subproblem. Recall that single depot vehicle scheduling problems can be solved in polynomial time. Additionally, we use the relaxation of model MDVCSP where greater or equal signs replace the equality sign in constraints (2.14). We refer to [Vanderbeck, 1994] for a general discussion about partitioning versus covering formulations.

Next, Lagrangian multipliers μ_{ij}^d and π_t are associated with constraints (2.14) and (2.12), respectively. The objective function (2.11) now reads:

$$\begin{aligned} \min \quad & \sum_{d\in\mathcal{D}} \sum_{(i,j)\in A^d} y_{ij}^d c_{ij}^d + \sum_{d\in\mathcal{D}} \sum_{k\in K^d} x_k^d f_k^d \\ & + \sum_{d\in\mathcal{D}} \sum_{(i,j)\in \widetilde{A}^d} \mu_{ij}^d \left(y_{ij}^d - \sum_{k\in K^d(i,j)} x_k^d \right) \\ & + \sum_{t\in\mathcal{T}} \pi_t \left(1 - \sum_{d\in\mathcal{D}} \sum_{(i,j)\in A^d(t)} y_{ij}^d \right). \end{aligned} \tag{2.17}$$

Furthermore, the Lagrangian subproblem results in:

$$\Phi(\mu,\pi) = \Phi_y(\mu,\pi) + \Phi_x(\mu) + \sum_{t\in\mathcal{T}} \pi_t \tag{2.18}$$

with

$$\begin{aligned} \Phi_y(\mu,\pi) = \Bigg\{ \quad \min \quad & \sum_{d\in\mathcal{D}} \sum_{(i,j)\in A^d} y_{ij}^d \bar{c}_{ij}^d \mid \\ & \sum_{\{j:(j,i)\in A^d\}} y_{ji}^d = \sum_{\{j:(i,j)\in A^d\}} y_{ij}^d, \quad \forall d\in\mathcal{D}, \forall i \in N^d, \\ & 0 \le y_{ij}^d \le u_{ij}^d, \qquad\qquad \forall d\in\mathcal{D}, \forall (i,j)\in A^d \Bigg\} \end{aligned} \tag{2.19}$$

as vehicle scheduling subproblem and

$$\begin{aligned} \Phi_x(\mu) = \Bigg\{ \quad \min \quad & \sum_{d\in\mathcal{D}} \sum_{k\in K^d} x_k^d \bar{f}_k^d \mid \\ & x_k^d \in \{0,1\}, \quad \forall d\in\mathcal{D}, \forall k \in K^d \Bigg\} \end{aligned} \tag{2.20}$$

as crew scheduling subproblem. The reduced cost $\bar{c}_{ij}^d$ on arc $(i,j) \in A^d$ of the

vehicle scheduling network of depot $d \in \mathcal{D}$ is defined as

$$\bar{c}_{ij}^d = \begin{cases} c_{ij}^d + \mu_{ij}^d - \pi_t & \text{for } (i,j) \in \widetilde{A}^d \text{ and } \exists t \in \mathcal{T} : (i,j) \in A^d(t) \\ c_{ij}^d + \mu_{ij}^d & \text{for } (i,j) \in \widetilde{A}^d \text{ and } \nexists t \in \mathcal{T} : (i,j) \in A^d(t) \\ c_{ij}^d & \text{for } (i,j) \notin \widetilde{A}^d \end{cases} \tag{2.21}$$

while

$$\bar{f}_k^d = f_k^d - \sum_{(i,j) \in \widetilde{A}^d(k)} \mu_{ij}^d \tag{2.22}$$

denotes the reduced cost of duty $k \in K^d$ where $\widetilde{A}^d(k) \subseteq \widetilde{A}^d$ corresponds to the set of arcs that is covered by duty $k \in K^d$.

Given multipliers μ and π the vehicle scheduling subproblem $\Phi_y(\mu, \pi)$ results in $|\mathcal{D}|$ minimum cost flow problems (see Section 1.4.1). Note that integrality is not imposed on the flow variables since all bounds are integral and, thus, each solution to the linear program above is integral (see Section 1.4.1). As stated earlier each minimum cost flow problem can be solved in polynomial time. For given multipliers μ, the crew scheduling subproblem $\Phi_x(\mu)$ can easily be solved by setting $x_k^d = 1$ if and only if $\bar{f}_k^d \leq 0$. Notice that both subproblems have the integrality property (see Section 1.5.1).

We obtain a lower bound by approximately solving the Lagrangian dual problem with a subgradient algorithm. However, there are some modifications of the standard subgradient algorithm. In particular, small norm subgradients are constructed as described in [Caprara et al., 1999] and the search direction is calculated similar to [Camerini et al., 1975]. Furthermore, columns in the restricted master problem may have negative reduced costs since the Lagrangian multipliers are not exact. Thus, they should be modified before generating new columns in order to prevent that columns are generated twice. We refer to Section 1.5.3 for a description of the method that is used to adjust the multipliers.

2.4.2. The Column Generation Pricing Problem

After the restricted master problem is solved, the dual information of the solution is used to price out new columns with negative reduced costs, i.e., columns that can improve the objective value of the master. However, the number of feasible pieces of work (and, thus, the number of feasible duties) is vast since vehicle blocks are not known in advance. Therefore, [Freling, 1997, Huisman, 2004] have proposed a two phase pricing procedure for the column generation pricing problem: in the first phase, a *piece generation network* is set up to generate a set of

pieces of work. These pieces serve as input for the second phase where duties are generated. Finally, notice that all work regulations concerning duty feasibility must be considered in the pricing problem.

Generation of Pieces of Work

Recall that a piece of work is defined as a sequence of tasks without a (long) break for which a driver stays with the same vehicle, and that this sequence is only restricted by its duration. The piece generation network $\bar{G}^d = (\bar{N}^d, \widetilde{A}^d)$ is similar to the vehicle scheduling network $G^d = (N^d, A^d)$ from the previous section. More precisely, $\bar{G}^d$ is an acyclic directed time-space network where $\widetilde{A}^d \subset A^d$ specifies the set of trip arcs, deadhead arcs, and waiting arcs outside the depot. Note that we assumed in Section 2.1 that each trip has exactly two relief points: one at the beginning and the other at the end of the trip. Thus, each node in $\bar{N}^d \subset N^d$ corresponds to a relief point. Figure 2.4 shows the piece generation network that is associated with the vehicle scheduling network depicted in Figure 2.3.

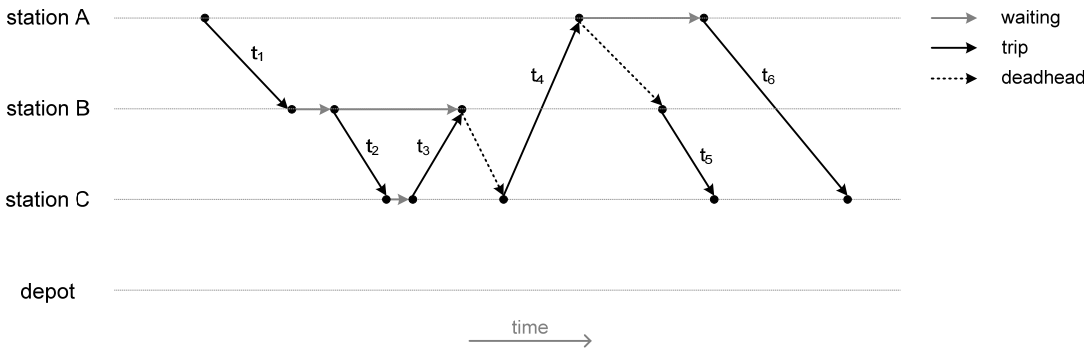

Figure 2.4.: Piece generation network

Let g^d_{ij} be the crew cost associated with arc $(i,j) \in \widetilde{A}^d$. The reduced cost of arc $(i,j) \in \widetilde{A}^d$ is then defined as $\bar{g}^d_{ij} = g^d_{ij} - \mu^d_{ij}$ where μ^d_{ij} are the multipliers associated with linking constraints (2.14) that represent trip, deadhead, or waiting arcs outside the depot. Hence, the reduced cost of a path is equal to the reduced cost of the associated piece of work. Each path between two nodes n_1 and n_2 in network $\bar{G}^d$ is a feasible piece of work if and only if the minimum and maximum duration is satisfied. However, it is not necessary to enumerate all feasible pieces of work since it suffices to prove that no duties with negative reduced costs are left (in order to reach column generation optimality). The sufficient subset of pieces is generated by considering the minimum reduced cost path between each pair of nodes that meet the duration constraints. These paths are generated using an

all-pairs-shortest-path algorithm (e.g. Floyd/Warshall method). Furthermore, three additional pieces of work are considered for each path: a pull-in trip is added at the beginning, a pull-out trip at the end, and both. As shown by [Freling, 1997] it suffices to generate only this subset of pieces to assure column generation optimality. However, there are further constraints concerning piece of work feasibility in practice. In Chapter 5 we will show how further requirements can be considered in the piece generation phase.

Generation of Duties

In the second phase of the pricing algorithm, feasible duties are constructed using the pieces of work generated in the previous phase. A duty is feasible if it satisfies a number of constraints such as minimum/maximum working time or spread time. Furthermore, the number of pieces of work is limited. [Huisman, 2004, Gintner, 2007] consider the case where a duty consists of at most two pieces of work. As a result they simply enumerate all possible combinations of pieces and check duty feasibility of such a combination. Under the assumption of continuous attendance (see Section 2.1) the reduced cost of a duty can be computed by adding up the reduced costs of the pieces. The authors stop enumerating when a specified number of negative reduced cost duties is found or all combinations are checked. However, this approach becomes impractical if more than 2 pieces of work per duty are allowed. In Chapter 5 we will show how duties with more than 2 pieces of work can be efficiently computed using a resource constrained shortest path formulation.

2.4.3. Integer Solutions

The final step of the solution method aims at finding a pair of feasible and compatible vehicle and crew schedules with a Lagrangian heuristic. Only constraints (2.14) are relaxed in a Lagrangian way. Thus, the solution of the vehicle scheduling subproblem gives a feasible vehicle schedule. However, the subproblem corresponds to a MDVSP which is an NP-hard problem. As in the column generation phase, a subgradient algorithm is used to solve the associated Lagrangian dual problem. Since the subgradient method is initialized with good multipliers (from the last iteration of the column generation phase) only few iterations are needed to obtain good multipliers. Finally, for the last k feasible vehicle schedules, the associated CSP is solved for each depot in order to obtain a feasible and compatible crew schedule. Notice that this method always yields solutions with the minimum number of vehicles.

In Section 3.3 we will propose three different ways of obtaining a feasible solution to MDVCSP, namely a modified version of the method just described, different branching approaches in a branch-and-bound framework, and a heuristic branch-and-price approach.

3. New Approaches to Integrated Vehicle and Crew Scheduling

Integrated vehicle and crew scheduling with multiple depots has received increasing attention over the past years. However, large problem instances still require an enormous computational effort to determine adequate solutions. Therefore, we will modify and extend the solution approach for the integrated vehicle and crew scheduling problem with multiple depots that we have described in the previous chapter.

The solution approach is based on Lagrangian relaxation in combination with column generation (see Section 2.4). More precisely, column generation is used to compute a lower bound where Lagrangian relaxation is applied to solve the master problem. In this chapter, we will propose a novel approach for the column generation pricing problem. We will discuss three network models for the column generation pricing problem that are based on the decomposed pricing scheme as described in Section 2.4.2. In particular, we compare a connection-based duty generation network with two novel aggregated time-space networks for duty generation. To the best of our knowledge, subproblem decomposition has not been applied in combination with a time-space network for duty generation before. Then, we will describe methods to solve the resource constrained shortest path problems that appear in the duty generation phase of the decomposed pricing problem. In the third part of the chapter, we propose different methods for finding integer solutions: a modified version of the algorithm described in Section 2.4.3, novel adaptations of branching schemes in a branch-and-bound framework, and a novel heuristic branch-and-price method. Furthermore, we will propose a novel modification of model MDVCSP (see Section 2.3) where we allow drivers to use vehicles from all depots and to change their vehicle whenever possible. Finally, we test our solution approaches on real-world instances and on randomly generated instances from literature.

This chapter is organized as follows. In Section 3.1 we discuss different network formulations for the column generation pricing problem and propose two new formulations for decomposed pricing. We describe our solution method for solving the associated resource constrained shortest path problem with dynamic

programming in Section 3.2. Section 3.3 deals with methods to find feasible solutions. We relax some of our assumptions concerning changeovers in Section 3.4. Finally, we provide extensive computational results on real-world and randomly generated instances in Section 3.5.

3.1. Modeling the Column Generation Pricing Problem

In the previous chapter, we described the basic column generation algorithm to solve the model MDVCSP. For the following discussion we will assume that crews are identical. However, our approach can easily be extended to the case with non-identical crews where we repeatedly solve pricing problems for each crew type/duty type combination. To simplify the exposition we will first recall some definitions and notations.

For integrated vehicle and crew scheduling problems it is impossible to explicitly keep all columns in main memory and, as a result, to solve the master linear program from scratch. Instead, we solve a sequence of *restricted* master programs (RMP) where each problem contains only a small subset of all columns. After the restricted master problem has been solved, the dual information of the solution is used to *price out* new columns that can improve the objective value of the RMP. In other words, the purpose of the *pricing problem* (subproblem) is to find variables with negative reduced costs. In our case, variables correspond to feasible crew duties.

Consider the piece generation network $\bar{G}^d = (\bar{N}^d, \widetilde{A}^d)$ for depot $d \in \mathcal{D}$ and model MDVCSP (2.11)-(2.16) as defined in Section 2.3. Furthermore, recall from Section 2.4 that the reduced cost of duty k from depot $d \in \mathcal{D}$ was defined by

$$\bar{f}_k^d = f_k^d - \sum_{(i,j)\in\widetilde{A}^d(k)} \mu_{ij}^d \tag{3.1}$$

where f_k^d corresponds to the cost of the duty, $\mu_{ij}^d \in \mathbb{R}$ are the Lagrangian multipliers associated with linking constraints (2.14), and $\widetilde{A}^d(k)$ defines the set of vehicle activities that are covered by duty k of depot d. Consequently, for given multipliers μ the pricing problem (subproblem) for each depot $d \in \mathcal{D}$ can be stated as follows:

$$\bar{f}^* = \left\{ \quad \min \quad \bar{f}_k^d x_k^d \quad | \quad x_k^d \in \widetilde{K}^d \right\} \tag{3.2}$$

where $\widetilde{K}^d$ corresponds to the set of feasible duties that can be operated from depot $d \in \mathcal{D}$. Notice that the set $\widetilde{K}^d \supseteq K^d$ defines the set of *all* feasible duties for depot

d while K^d corresponds to the set of duties for depot d that are currently in the RMP. If the pricing problem returns $\bar{f}^* < 0$, the column x_k^d with least negative reduced cost $\bar{f}_k^d \leq 0$ is added to the RMP. The process stops when $\bar{f}^* \geq 0$.

Of course, the structure of the pricing problem is independent of the way the dual information is obtained. In our case, an approximated dual solution is computed by solving a Lagrangian dual problem with a subgradient method. However, a dual solution can also be computed by using a simplex method on the linear relaxation of model MDVCSP where the integrality of variables x_k^d is relaxed.

In the following, we will describe a mathematical formulation for the pricing problem (3.2) in more detail. As stated earlier, a separate pricing problem must be solved for each depot. For notational convenience, however, we describe the pricing problem for a single depot.

Basically, all work regulations concerning duty feasibility must be considered in the pricing problem. Recall that the set of all duties $\widetilde{K}^d$ can be huge. Thus, the subproblem cannot be solved by sorting set $\widetilde{K}^d$ with increasing reduced costs and selecting the least cost column. Instead, the subproblem (3.2) is usually formulated as a *resource constrained shortest path problem* (RCSP, see Section 1.4.1) where each feasible path from the source to the sink represents a feasible duty (see e.g. [Desrochers and Soumis, 1989]). The cost of a path is defined in such a way that it is equal to the reduced cost of the corresponding duty. We will refer to an acyclic network $H = (N, A)$ with source s and sink t that is used to generate negative reduced cost paths as *duty generation network*. We obtain

$$\bar{f}^* = \min \sum_{(i,j)\in A} \bar{f}_{ij} z_{ij} \tag{3.3}$$

$$s.t. \quad \sum_{\{j:(i,j)\in A\}} z_{ij} - \sum_{\{j:(j,i)\in A\}} z_{ji} = \begin{cases} 1 & \text{for } i = s \\ 0 & \text{for } i \in N \setminus \{s,t\} \\ -1 & \text{for } i = t \end{cases} \tag{3.4}$$

$$l^r \leq \sum_{(i,j)\in A} d_{ij}^r z_{ij} \leq u^r \qquad \forall r \in R \tag{3.5}$$

$$z_{ij} \in \{0,1\} \quad \forall (i,j) \in A \tag{3.6}$$

where the binary flow variable z_{ij} indicates whether there is a flow on arc $(i,j) \in A$, $\bar{f}_{ij} = f_{ij} - \mu_{ij}$ is the reduced cost of arc $(i,j) \in A$, and μ_{ij} is the dual variable (Lagrangian multiplier) of arc $(i,j) \in A$ that corresponds to the associated linking constraint (2.14). However, we do not only associate a traversal cost $\bar{f}_{ij}$ with each arc $(i,j) \in A$, we also define a resource consumption $d_{ij}^r \geq 0$ for each resource $r \in R$. Consequently, each path P accumulates $\sum_{(i,j)\in P} d_{ij}^r$ of resource $r \in R$.

We say that a path is *resource feasible* if and only if the resource consumption along the path is greater or equal to lower bound l^r and less or equal to upper bound u^r. As described in Section 1.4.1 the RCSP is NP-hard when there is at least one resource even though pseudo-polynomial algorithms have been proposed in literature.

Typically, it is assumed that the *resource extension function* (REF) is non-decreasing, and reduced cost and resource consumption are separable functions of pieces of work or tasks, respectively. Basically, a REF is associated with an arc and defines how the resources are updated along that arc. Then, reduced cost and resource consumption can be accumulated during path construction. We refer to [Irnich and Desaulniers, 2005,Irnich, 2006] for a thorough description of resource extension functions. Notice that not all constraints arising in practice fit into the structure of the objective function (3.3) and constraints (3.5). Some feasibility constraints can be dealt with when constructing the network (e.g. minimum rest time between two tasks or pieces of work) while others cannot (e.g. maximum working time in any 4-hour period of a duty). Furthermore, public transport companies often apply several duty types that differ in the feasibility constraints imposed. Thus, it might be necessary to solve a separate pricing problem for each duty type, or, if the problem involves multiple depots, for each depot-duty type combination. However, in chapter 5 we will discuss how constraints arising from German regulations can be incorporated into a resource constrained shortest path model.

In the next section, we discuss two modeling approaches for duty generation network H where either tasks or pieces of work serve as network elements. In Section 3.1.2, we describe three different piece-of-work-based network models for a decomposed pricing problem. Furthermore, we give an example how resources can be used to model duty constraints from practice such as minimum/maximum working time or maximum spread time.

3.1.1. Modeling Approaches

A duty is a sequence of pieces of work separated by breaks where a piece of work is a sequence of tasks without a (longer) break on the same vehicle. Furthermore, there are constraints concerning each piece of work and the entire duty. Consequently, we can either use tasks or pieces of work as network elements for the duty generation network H. The choice between tasks and pieces of work involves a trade-off between the size of the network and the resource constraints on the paths. In particular, there are much more feasible pieces of work than tasks. Therefore, piece-based networks are larger, but piece feasibility is checked

in advance (during piece construction). On the other hand, task-based networks are small, but piece feasibility must be checked during duty construction which usually requires additional resources. Additional resources make the RCSP more difficult (and time-consuming) since more constraints must be checked. To sum up, there is a trade-off between memory and time consumption.

Table 3.1 provides several applications of task- and piece-of-work-based network models for crew scheduling both in airline and public transport settings. All authors use the corresponding network model in a column generation context that involves a resource constrained shortest path problem as pricing problem.

	airline	public transport
tasks	[Vance et al., 1997a] [Desaulniers et al., 1999] [Borndörfer et al., 2006] [Sandhu and Klabjan, 2006]	[Friberg and Haase, 1999] [Haase et al., 2001] [Grötschel et al., 2003] [Borndörfer et al., 2004] [Mesquita and Paias, 2006]
pieces of work	[Desaulniers et al., 1997] [Vance et al., 1997a] [Vance et al., 1997b] [Galia and Hjorring, 2004] [Sandhu and Klabjan, 2006]	[Desrochers and Soumis, 1989] [Desrochers et al., 1992] [Carraresi et al., 1995] [Freling, 1997]

Table 3.1.: Network modeling approaches for crew scheduling in literature

[Vance et al., 1997a] test both approaches in an airline setting. However, a direct comparison is not possible since different rule sets are used and the tests are executed on different machines. Even though a direct comparison is not valid the authors observe that the task-based version consumes more time but does not involve prohibitive network dimensions for large problem data as piece-of-work-based networks do. In contrast, [Sandhu and Klabjan, 2006] also describe both modeling approaches, but finally use a piece-of-work-based network in their computational experiments. The authors claim that this network inherently captures more feasibility rules and, thus, requires less resources and less computational time. In conclusion, it remains an open question whether there is a beneficial modeling approach in an airline setting.

In contrast to airline planning there has been no direct comparison between

both modeling approaches in public transport settings. As presented in Table 3.1 some authors use a task-based approach while others prefer a piece-of-work-based network.

However, most approaches of the second category simply enumerate all pieces of work which may lead to a huge number of pieces for integrated vehicle and crew scheduling problems. Hence, there is much room for improvement if not all pieces are enumerated. As shown by [Freling, 1997] the number of pieces can be dramatically reduced when the subproblem is decomposed into a piece and duty generation phase (see Section 2.4.2). In the first phase, a *piece generation network* is used to generate a subset of all pieces of work. These pieces serve as input for the second phase where duties are generated. Recall that this decomposition scheme can only be applied if the resource consumption is equal for all paths between two relief points. Despite this decomposition, up to 96% of the total time in column generation is spent on the pricing problem for large integrated single-depot vehicle and crew scheduling problems. The time for the piece generation phase can be neglected. Recall that [Huisman, 2004, Huisman et al., 2005a] use a procedure similar to [Freling, 1997] for duties with up to two pieces of work, but enumerate all feasible piece combinations during pricing for a multiple-depot integrated vehicle and crew scheduling problem.

When we compare a task-based network with a piece-of-work-based formulation in combination with the decomposed pricing scheme of [Freling, 1997], we see that the total number of feasible paths is higher in the task-based version. In a task-based network for duty generation, there are multiple paths between each pair of relief points while there is exactly one path in a piece-based representation with decomposed pricing (the piece that was generated in the piece generation phase). Consequently, the solution space of the pricing problems is smaller with a piece-based formulation and, thus, appears to be beneficial.

In the following section, we will describe the duty generation network as proposed by [Freling, 1997] for a decomposed pricing problem. Furthermore, we will suggest two novel formulations that have a lower network complexity than the model of [Freling, 1997].

3.1.2. Network Models for a Decomposed Pricing Problem

Let us consider piece generation network $\bar{G} = (\bar{N}, \bar{A})$ as defined in Section 2.4.2, where nodes correspond to relief points. Arcs in $\bar{A}$ represent either deadhead, trip, or non-depot waiting tasks. Recall that $\bar{G}$ is acyclic. The cost associated with each arc $(i, j) \in \bar{A}$ is defined in such a way that the cost of each path equals the reduced costs of the corresponding piece. Since the feasibility of a

piece is only restricted by its duration, we generate the set of pieces by solving a shortest path problem between each pair of nodes in $\bar{N}$ that satisfy the duration constraint. Let ν be the number of relief points, then the number of pieces of work is in $\mathcal{O}(\nu^2)$.

In the following, we will describe the connection-based model of [Freling, 1997], a time-space, and an aggregated time-space duty generation network. To complete the description of the models, we define the resource consumption and resource constraints that are necessary to cover the following constraints: maximum working time, maximum spread time (duty length), minimum start time, maximum end time, minimum break length, and minimum/maximum number of pieces. A piece of work is only restricted by its duration. Notice that the piece of work related constraint has already been checked in the piece generation phase. The same set of duty regulations is used in [Huisman, 2004]. Moreover, we will give the network complexity of each model. Finally, we will compare all network representations.

Connection-based Model

We describe the connection-based duty generation network $H_c^t = (N_c^t, A_c^t)$ for each duty type t as proposed by [Freling, 1997]. Nodes N_c^t correspond to the feasible pieces from the preceding phase. *Source* and *sink* represent the depot. Arcs in A_c^t either represent *breaks* between two pieces or *sign_on*/*sign_off* activities. Furthermore, breaks can be combined with walking, i.e., the driver first takes a break and then walks (or takes a bus) from the arrival station of the first piece to the departure station of the second piece. A break arc represents the connection between two pieces whose connection time is greater or equal the minimum break length plus (if necessary) the additional walking time. A duty starts (ends) with a sign-on (sign-off) arc. Consequently, *sign_on* arcs originate from the *source* while *sign_off* arcs terminate at the *sink*. We only add *sign_on* (*sign_off*) arcs to the network if they are within the minimum start (maximum end) time. Figure 3.1 depicts a connection-based duty generation network with five pieces of work. Notice that each piece of work is shown as a node where the arrival station of the piece is given in the middle of the node. The node of a piece is located on the timeline of the start station at the start time while the end time is not directly shown.

In order to cover the remaining constraints within the duty construction process, the connection-based network requires three resources: number of pieces of work, working time, and spread time. Table 3.2 shows the resource consumption for a connection-based network by arc type with F^t as fixed cost of duty type t,

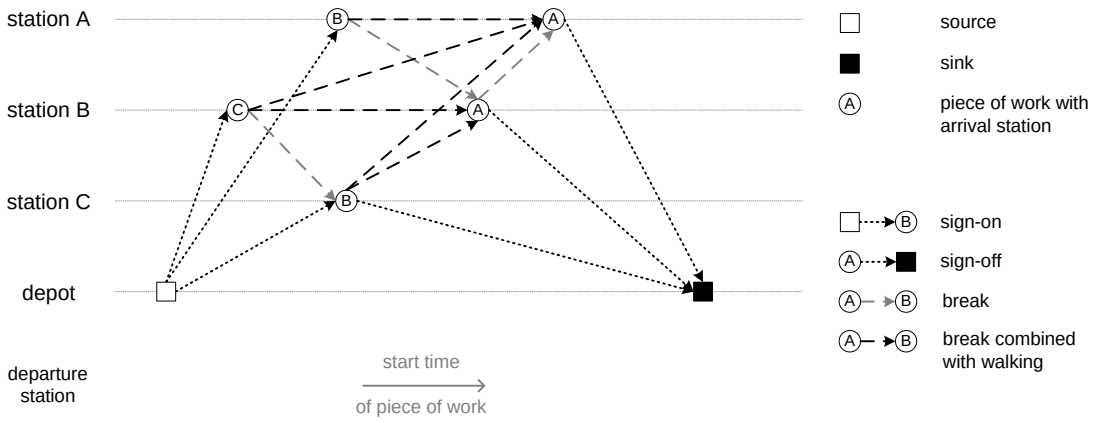

Figure 3.1.: Connection-based duty generation network

v^t as variable costs of duty type t per minute, d_i as length (working time) of piece i in minutes, and l_{ij} as length of arc $(i,j) \in A_c^t$ in minutes. Notice that cost and resource consumption defined on nodes can always be transferred to arcs.

type of arc $(i,j) \in A_c^t$	cost	working time	spread time	number of pieces
sign-on	$F^t + d_j v^t$	d_j	$l_{ij} + d_j$	1
sign-off	0	0	l_{ij}	0
break	$d_j v^t$	d_j	$l_{ij} + d_j$	1
break with walking	$d_j v^t$	d_j	$l_{ij} + d_j$	1

Table 3.2.: Resource consumption for a connection-based network

The connection-based network contains $\mathcal{O}(\nu^2)$ nodes, $\mathcal{O}(\nu^4)$ *break* arcs, and $\mathcal{O}(\nu^2)$ *sign_on*/*sign_off* arcs. As we have seen, we need three resources to cover the rules stated above.

Time-Space Model

Next, we define a time-space duty generation network $H_s^t = (N_s^t, A_s^t)$ for each duty type t where a pair of *piece_start* and *piece_end* nodes is associated with each piece of work. Source and sink represent the depot as before. We have four types of arcs in A_s^t: *sign_on*-, *sign_off*-, *piece*- and *break*-arcs. *Piece*-arcs connect *piece_start* with *piece_end* nodes while *break*-arcs have the opposite direction. Again, breaks can be combined with walking, but must satisfy minimum break time. *Sign_on* arcs originate from *source* and terminate at a *piece_start* node

while *sign_off* arcs emanate from a *piece_end* node and end at *sink*. We only add *sign_on* (*sign_off*) arcs to the network if they are within the minimum start (maximum end) time. Note that there is no direct connection between two *piece*- or *break*-arcs, respectively. The major difference between a connection-based and time-space network is that in a connection-based network pieces are represented by nodes while in a time-space network pieces correspond to arcs. Figure 3.2 shows the corresponding time-space network to the connection-based network in Figure 3.1.

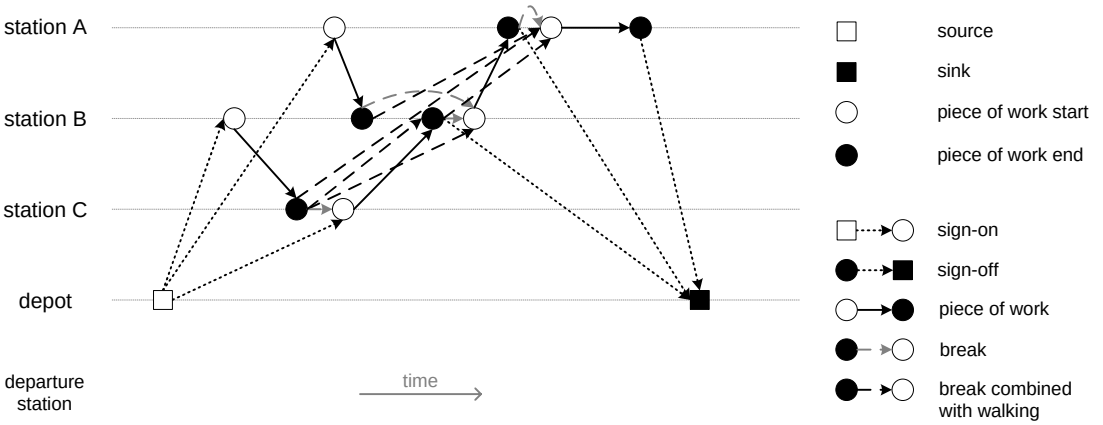

Figure 3.2.: Time-space duty generation network

Similar to the connection-based network, three resources are necessary to validate duties: working time, spread time, and number of pieces. Table 3.3 shows the resource consumption for a time-space network by arc type.

type of arc $(i,j) \in A_s^t$	cost	working time	spread time	number of pieces
sign-on	F^t	0	l_{ij}	0
sign-off	0	0	l_{ij}	0
piece of work	$l_{ij}v^t$	l_{ij}	l_{ij}	1
break	0	0	l_{ij}	0
break with walking	0	0	l_{ij}	0

Table 3.3.: Resource consumption for a time-space network

The time-space representation has $\mathcal{O}(\nu^2)$ nodes, $\mathcal{O}(\nu^2)$ *piece*-, $\mathcal{O}(\nu^4)$ *break*-, and $\mathcal{O}(\nu^2)$ *sign_on*/*sign_off*-arcs. However, the time-space network can be aggregated by exploiting the structure of the underlying piece generation network.

Observe that there are $\mathcal{O}(\nu)$ pieces of work that start at the same relief point. Since a break-arc always follows a piece-arc, we can aggregate all *piece_start* nodes with the same origin and the same time without changing the set of feasible paths. The same reasoning holds for *piece_end* nodes. Due to aggregation the number of nodes reduces to $\mathcal{O}(\nu)$ and, thus, the number of arcs to $\mathcal{O}(\nu^2)$. The overall network size $\mathcal{O}(\nu^2)$ is therefore considerably smaller than the connection-based formulation of [Freling, 1997]. Nevertheless, the time-space network still has $\mathcal{O}(\nu^2)$ break arcs that can be further aggregated to $\mathcal{O}(\nu)$ as we will show in the following.

Aggregated Time-Space Model

We define an aggregated time-space duty generation network $H_a^t = (N_a^t, A_a^t)$ that modifies the time-space network in the following way. We introduce a departure timeline for each station that connects two subsequent *piece_start* nodes of a station by a *waiting* arc. Furthermore, we use a *break* arc to connect a *piece_end* with a *piece_start* node. However, there is at most one *break* arc to connect a *piece_end* with all subsequent *piece_start* nodes of a station as opposed to a time-space network where we have an explicit arc for each compatible pair of *piece_end* and *piece_start* nodes. In an aggregated model, all connections not explicitly present at a station are implicitly included by traversing the timeline. Figure 3.3 depicts an aggregated time-space network with timelines at stations A and B. In order to illustrate timelines, Figure 3.3 does not contain the same set of pieces as the preceding figures.

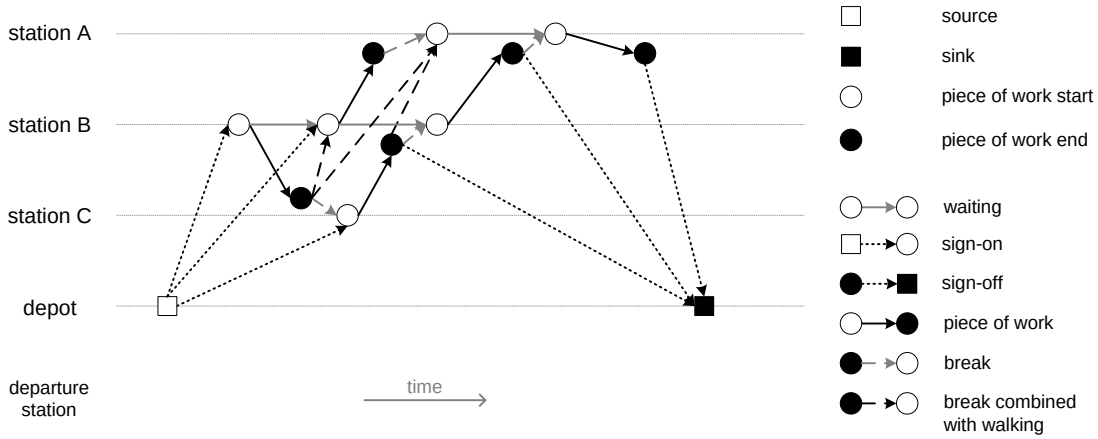

Figure 3.3.: Aggregated time-space duty generation network

Similar to the previous time-space model three resources are necessary to check

duty feasibility: working time, spread time, and number of pieces. Table 3.4 shows the resource consumption for an aggregated time-space network by arc type. However, we need an additional resource if there is a maximum break duration. Furthermore, in many settings duties must not start with a waiting time which cannot be inherently modeled in the network structure. Consequently, we must perform an additional check during duty construction.

type of arc $(i,j) \in A_s^a$	cost	working time	spread time	number of pieces
sign-on	F^t	0	l_{ij}	0
sign-off	0	0	l_{ij}	0
piece of work	$l_{ij}v^t$	l_{ij}	l_{ij}	1
break	0	0	l_{ij}	0
break with walking	0	0	l_{ij}	0
waiting	0	0	l_{ij}	0

Table 3.4.: Resource consumption for an aggregated time-space network

Basically, the aggregated time-space representation has the same network dimensions as the time-space network. However, the number of *break* arcs reduces to $\mathcal{O}(\nu)$ since there is at most one *break* arc from each *piece_end* node to a station. The number of *waiting* arcs grows linearly with the number of relief points.

Comparison

Table 3.5 summarizes the network complexities of the three network representations described earlier. The number of break arcs includes both types of breaks: "pure" breaks and breaks combined with walking. As a result, connection-based networks have the largest number of network elements followed by time-space and aggregated time-space networks. Moreover, all network representations need three resources to validate the feasibility constraints according to [Huisman, 2004].

We will now investigate the actual network dimensions of the formulations for integrated vehicle and crew scheduling problems. We use randomly generated instances available at [Huisman, 2003]. A detailed description of the instances and how they were generated is given by [Huisman, 2004, Huisman et al., 2005a]. The instances have been classified into two classes according to the travel speed where the speed is lower for problems in class B. As a consequence, trips in class B are longer. The instances in class A are considered more demanding than those

		network type		
element	type	connection	time-space	aggr. time-space
arcs	sign-on	$\mathcal{O}(\nu^2)$	$\mathcal{O}(\nu^2)$	$\mathcal{O}(\nu^2)$
	sign-off	$\mathcal{O}(\nu^2)$	$\mathcal{O}(\nu^2)$	$\mathcal{O}(\nu^2)$
	piece of work	–	$\mathcal{O}(\nu^2)$	$\mathcal{O}(\nu^2)$
	break	$\mathcal{O}(\nu^4)$	$\mathcal{O}(\nu^2)$	$\mathcal{O}(\nu)$
	waiting	–	–	$\mathcal{O}(\nu)$
nodes	piece of work	$\mathcal{O}(\nu^2)$	–	–
	piece start/end	–	$\mathcal{O}(\nu)$	$\mathcal{O}(\nu)$
resources	–	3	3	3

Table 3.5.: Network dimensions of different duty generation networks

in class B. We will only report computational results for class A that all involve 4 depots and groups with n trips where $n = 80, 100, 160, 200, 320$ and 400. For each group 10 instances are available. Table 3.6 gives average network dimensions for duty types with 2 pieces of work as described in [Huisman, 2004, Huisman et al., 2005a] including the assumptions stated in Section 2.1. Of course, all network formulations have the same solution space.

As can be seen from Table 3.6 we were not able to set up the network for the connection-based formulation with more than 200 trips due to excessive memory consumption. Furthermore, the total number of arcs and nodes reduces dramatically when a time-space network is used (by 99.66% to 99.94%) instead of the connection-based representation of [Freling, 1997]. The network size can be further reduced by approximately 20% when an aggregated time-space network is used. Notice that the average number of arcs in the aggregated time-space networks can be smaller than the number of pieces since minimum/maximum start and end time constraints are imposed. Due to prohibitive memory consumption we will not consider the connection-based network in the remainder of this thesis.

In Table 3.7 we compare the time-space (*tsn*) with the aggregated time-space (*atsn*) network representation for duty generation. All tests were executed on a Pentium IV 3.40GHz personal computer (2GB RAM) using the algorithm described in Section 2.4 with the following settings. In accordance with [Huisman, 2004, Huisman et al., 2005a] we consider five different types of duties: one tripper type with one piece of work between 30 minutes and 5 hours, and four types consisting of two pieces of work. We use the instances available at [Huisman, 2003] that have also been used and described above. However, we randomly chose three instances out of each group (10 instances). We assign a fixed cost of 1,000 for

trips	elements	network type		
		connection	time-space	aggr. time-space
080	nodes	8,666	405	405
	arcs	3,254,370	10,822	8,652
100	nodes	13,678	507	507
	arcs	7,277,958	16,684	13,493
160	nodes	39,658	812	812
	arcs	67,950,380	47,546	38,158
200	nodes	57,687	1,017	1,017
	arcs	123,414,953	73,383	59,597
320	nodes	–	1,548	1,548
	arcs	–	192,717	157,381
400	nodes	–	1,934	1,934
	arcs	–	290,899	240,993

Table 3.6.: Average network dimensions of different duty generation networks for integrated vehicle and crew scheduling problems

each vehicle and duty and a small variable cost of 1 for each minute a vehicle is outside the depot and 0.1 for each minute a crew is working. In other words, we minimize the total number of vehicles and drivers first and leave operational cost minimization as secondary objective. For all settings, we terminate if the improvement of the lower bound is less than 0.5% in the last 10 iterations or if the computational time is more than 5,400 seconds for the lower bound phase. All data given corresponds to the average over three instances. In order to get a realistic picture of the performance, we applied all preprocessing and acceleration techniques that we will describe in the following sections. Finally, in Table 3.7 we report the number of iterations ($\#iter$), the cpu time in seconds spent on the master (cpu_ma) and pricing problem (cpu_pr), the lower bound (lb), and the total number of vehicles and drivers ($v+d$) of the best feasible solution. Recall that the approach to compute integer solutions (see Section 2.4.3) always yields vehicle schedules with the minimum number of vehicles.

The aggregated version seems to require considerably more column generation iterations (see groups with 160 and 200 trips) and more time to price out new columns. For larger instances the algorithm terminated due to the time limit and returned worse lower bounds with the aggregated version. Therefore, we conclude that (except for the smallest instances) the time-space network outperforms the aggregated time-space formulation in terms of pricing time and solution quality.

network		#trips					
		080	100	160	200	320	400
tsn	#iter	21.3	21.7	28.0	28.0	33.0	28.7
	cpu_ma	172	246	729	941	3,989	4,268
	cpu_pr	12	70	350	361	1,451	1,243
	lb	29,649	37,849	48,331	63,477	77,282	108,366
	v+d	27.3	35.7	46.3	61.3	76.0	105.7
atsn	#iter	18.3	22.3	33.5	36.3	31.0	26.7
	cpu_ma	126	227	956	1,177	4,283	4,623
	cpu_pr	16	42	635	1,381	1,322	1,848
	lb	29,813	37,712	48,208	63,340	78,206	110,865
	v+d	28.0	36.3	46.7	61.3	77.3	107.0

Table 3.7.: Comparison of the time-space and aggregated time-space network representation for duty generation

Notice that the average path length from the source to the sink is longer in the aggregated version. In connection with the dynamic programming algorithm that we use to solve the associated RCSP (see Section 3.2), a longer path length could cause more labels to be generated and evaluated. The more labels the algorithm has to evaluate, the more time is consumed. Due to the worse performance and the reasoning stated above, we will not consider the aggregated time-space network for duty generation in the remainder of this thesis.

In the next section, we will describe methods to solve the resource constrained shortest path problems that appear in the duty generation phase of the decomposed pricing problem.

3.2. Solving the Column Generation Pricing Problem

The methods developed for solving resource constrained shortest path problems can be classified into Lagrangian relaxation, constraint programming, and dynamic programming approaches. However, not all RCSP algorithms are suitable in a column generation context. In column generation pricing, an algorithm does not necessarily need to find the most negative reduced cost column. In order to guarantee convergence of column generation, it suffices that an algorithm returns any negative reduced cost column and returns no column if and only if

there is no negative reduced cost path. Furthermore, algorithms should return multiple paths with negative lengths (*multiple pricing*) since this usually accelerates the convergence of column generation. In the following, we will briefly review methods for the RCSP that satisfy these conditions. For an extensive survey on shortest path problems with resource constraints we refer to [Irnich and Desaulniers, 2005].

Lagrangian relaxation (see Section 1.5.1) methods for RCSP assume that resource consumption is additive along the path and that resource consumption is only constrained as a whole (see [Irnich and Desaulniers, 2005]). In other words, resource constraints cannot vary from node to node and the *resource extension function* (REF) cannot take the structure of a partial path into account. Recall that a REF is associated with an arc and defines how the resources are updated along that arc. [Beasley and Christofides, 1989] and [Grötschel et al., 2003] propose to compute lower bounds for the RCSP with Lagrangian relaxation. In a second step, they exploit these bounds in a tree search procedure. Additional constraints that could not directly be covered in the RCSP can always be considered in the search phase. However, these constraints cannot be covered directly as in a dynamic programming approach.

Constraint programming (see [Marriot and Stuckey, 1998]) approaches allow to tackle a wide range of complex constraints where some cannot be modeled using resources or simple structural constraints: for instance, a crew may not drive more than 5 hours in any 8-hour period. [de Silva, 2001] and [Fahle et al., 2002] use constraint programming to tackle the RCSP as pricing problem. However, it remains an open question if constraint programming is also beneficial when all constraints can be modeled using resources or inherently covered by the duty generation network.

Dynamic programming (see [Ahuja et al., 1993]) is widely used and the most successful approach to solve RCSP in a column generation context. Successful applications include, among others, [Desrochers and Soumis, 1989], [Haase et al., 2001], [Xu et al., 2003], and [Dell'Amico et al., 2006]. As described in [Irnich and Desaulniers, 2005, Irnich, 2006] dynamic programming allows to cover many constraints from practice using specialized (non-decreasing) resource extension functions. This approach appears to be more flexible than Lagrangian methods since the number of applicable REFs is comparatively high.

Since most constraints in public transport that we are aware of can be modeled with resources or inherently covered by the duty generation network described in Section 3.1, we also use a dynamic programming approach (see Section 3.2.1). As stated earlier, in column generation it is only necessary to solve RCSPs to proven optimality to show that no negative reduced cost paths exist. Therefore, it suffices

to approximately (heuristically) solve the RCSP in all but the final iteration and obtain arbitrary negative reduced cost path(s). We will describe preprocessing and further acceleration techniques in Section 3.2.2 and 3.2.3, respectively, that can be heuristically adapted.

3.2.1. Dynamic Programming Algorithms

The basic idea of dynamic programming for the RCSP is to iteratively construct paths starting from source s until sink t is reached. [Joksch, 1966] gives a recursion for the RCSP (3.3)-(3.6) with a single resource $r \in R$, i.e., $|R| = 1$. Let $F_j(t^r)$ be the cost of the shortest path from node s to j in network $H = (N, A)$ where the resource consumption of the path is less than or equal to t^r. We assume that the arcs $(i, j) \in A$ are numbered in such way that $i < j$ holds. Then, the recursion reads

$$F_j(t^r) = \min\left\{ F_j(t^r - 1), \min_{(i,j)\in A | d_{ij}^r \leq t^r} \{F_i(t^r - d_{ij}^r) + c_{ij}\} \right\} \tag{3.7}$$

where we set $F_1(t^r) = 0$ for $0 \leq t^r \leq u^r$ and $F_j(0) = \infty$ for $j = 2, \ldots, |N|$. We compute the minimum cost path from s to t by solving $F_t(u^r)$ where u^r is an upper bound on the consumption of resource r. The running time of the algorithm is in $\mathcal{O}(|A|u^r)$ while the space consumption is in $\mathcal{O}(|N|u^r)$. Hence, $F_j(t^r)$ is a pseudo-polynomial algorithm for RCSP. Furthermore, the dynamic programming recursion can be used to obtain a fully polynomial ϵ-approximation for RCSP by rounding and scaling (see [Hassin, 1992]).

Labeling approaches improve pure dynamic programming methods in such a way that they identify and discard inefficient paths that cannot be part of the optimal solution. Labeling methods use *labels* (*states*) to represent feasible (partial) paths where a label $L_{n_p}^k$ at node $n_p \in N$ corresponding to path $P_{n_p}^k = (s, \ldots, n_{p-1}, n_p)$ is linked with its predecessor label $L_{n_{p-1}}^j$ at node n_{p-1}. Linking allows to reconstruct the path of a label without storing the complete path in each label. Furthermore, each label $L_{n_p}^k = (L_{n_{p-1}}^l, c(P_{n_p}^k), d^1(P_{n_p}^k), \ldots, d^{|R|}(P_{n_p}^k))$ contains the cost $c(P_{n_p}^k)$ and resource consumption $d^r(P_{n_p}^k)$ for each resource $r \in R$. We denote the set of states at node n_p by L_{n_p}. Basically, a pulling dynamic programming algorithm pulls labels from all possible predecessor nodes to a node while updating cost and resource consumptions. A new label l at node j pulled from label k at node i is given by

$$L_j^l = (L_i^k, c(P_i^k) + c_{ij}, d^1(P_i^k) + d_{ij}^1, \ldots, d^{|R|}(P_i^k) + d_{ij}^{|R|}). \tag{3.8}$$

If $d^r(P_i^k) + d_{ij}^r < l^r$ we set $d^r(P_j^l) = l^r$. A new label L_j^l is accepted if it corresponds to a feasible partial path P_j^l, i.e., $d^r(P_j^l) \leq u^r$ for each $r \in R$. Additionally, we

only store labels that are not dominated by any other label at that node. A label L_i^m *dominates* label L_i^n if $c(P_i^m) \leq c(P_i^n)$ and $d^r(P_i^m) \leq d^r(P_i^n)$ for each $r \in R$. Notice that two labels dominate each other if cost and resource consumptions are equal. We call a non-dominated label *efficient* and denote the set of efficient labels at node i by L_i^*. Algorithm 6 describes a (pulling) label setting algorithm where $N^-(j) = \{i : (i,j) \in A\}$ defines the set of predecessors of node $j \in N$.

Algorithm 6: Basic label setting algorithm for the RCSP

(Step 1) **Initialization**
Set $L_s = (\texttt{nil}, 0, \ldots, 0)$.
Set $L_i = \emptyset$ for each $i \in N \backslash \{s\}$.

(Step 2) **Path extensions and dominance checks**
foreach $j \in N \backslash \{s\}$ **do**
  `// loop all predecessor nodes`
  foreach $i \in N^-(j)$ **do**
    `// loop all efficient labels`
    foreach $l \in L_i^*$ **do**
      if $\exists r \in R : d^r(l) + d_{ij}^r > u^r$ **then**
        next l
      `// create new label`
      $L_j^m = (l, c(l) + c_{ij}, d^1(l) + d_{ij}^1, \ldots, d^{|R|}(l) + d_{ij}^{|R|})$
      $L_j = L_j \cup L_j^m$
  `// remove dominated labels`
  $L_j^* = \mathit{Efficient\_Labels}(L_j)$

Procedure *Efficient_Labels* removes dominated states from set L_j. The minimum cost path in L_t^* corresponds to the optimal path. In addition to this generic algorithm, there are label setting and label correcting versions. In the latter case, a label can be corrected several times in the course of the algorithm while in a label setting method all labels are permanent and cannot be changed. Notice that our network is acyclic and that the nodes are treated in topological order and, thus, labels in L_i^* are permanent. However, neither label correcting nor label setting variants improve the pseudo-polynomial worst case complexity of function (3.7).

The running time of the algorithm depends on the implementation of the dominance tests in procedure *Efficient_Labels*. In a naive implementation where each pair of labels is compared, we have a complexity of $\mathcal{O}(|R|(L_{max})^2)$ with $L_{max} \leq \prod_{r \in R}(u^r - l^r)$ as the maximum number of labels at any node (see [Freling, 1997]). The computational effort for dominance tests can be reduced to

$\mathcal{O}(L_{max}(\log L_{max})^{|R|-2})$ with a divide-and-conquer algorithm (see [Kung et al., 1975]).

[Joksch, 1966] already noted that $F_j(t^r)$ is a step function and that it suffices to locate its steps. Furthermore, the author observed that the list of efficient labels are the non-differentiable points of the step function and, thus, only these have to be considered to obtain the optimal solution.

In the following, we will describe the label setting approach of [Desrochers, 1986] which is also described in [Desrosiers et al., 1995]. The method differs from Algorithm 6 in the order how paths are extended. The method has been used by several authors in a column generation context, e.g., [Haase et al., 2001], [Mesquita and Paias, 2006]. The algorithm is a multi-dimensional generalization of a pulling dynamic programming algorithm for the shortest path problem with time-windows (see e.g. [Desrochers and Soumis, 1988]).

The algorithm assumes that each arc $(i, j) \in A$ has positive cost or at least one positive resource consumption: $\exists r \in R : d^r_{ij} > 0$. In our setting, we always have at least one positive resource consumption for each arc (see Section 3.1). Without loss of generality, we order the resources in such a way that $d^1_{ij} \geq \varpi > 0$ holds for each $(i, j) \in A$ where ϖ is a lower bound on the resource consumption of the first resource. For each node $i \in N$, L_i denotes the set of labels and $\bar{L}_i \subseteq L_i$ the subset of permanent labels. Furthermore, set $\bar{L}_i$ is characterized by a variable bound η_i on the consumption of the first resource where $\bar{L}_i$ defines labels such that $l^1 \leq \max_{l \in \bar{L}_i} d^1(l) \leq \eta_i \leq u^1$. Algorithm 7 shows the label setting algorithm of [Desrochers, 1986]. The algorithm chooses unprocessed nodes with a "small" resource consumption first. It guarantees in combination with strictly positive resource consumption $d^1_{ij} > 0$ that all path extensions have a higher resource consumption than the previously created labels. Thus, labels in $\bar{L}_i$ are permanent.

In our computational experiments we found that the dynamic programming method often generated only few new columns, esp. in the final iterations. Therefore, we do not apply dominance tests when pulling labels at the sink node t. Furthermore, the performance of the standard version can be considerably improved by using preprocessing and further acceleration techniques as we will describe in the following sections.

3.2.2. Preprocessing

The purpose of this section is to describe network reductions for duty generation network $H = (N, A)$ in order to improve the performance of the pricing algorithm. In the following, we will discuss generic node and arc reductions introduced by

Algorithm 7: Label setting algorithm of [Desrochers, 1986]

(Step 1) **Initialization**

Set $L_s = \bar{L}_s = (\texttt{nil}, 0, \ldots, 0)$ and $\eta_s = u^1$.
Set $L_i = \bar{L}_i = \emptyset$ for each $i \in N \backslash \{s\}$ and $\eta_i = l^1$.

(Step 2) **Path extensions and dominance checks**

while *true* **do**
- `// select node`
- **if** $\forall i \in N \backslash \{s\} : \eta_i = u^1$ **then**
  - **exit**
- **else**
  - Select $j \in \arg\min_{i \in N \backslash \{s\}} \{\eta_i | \eta_i < u^1\}$.
- `// pull labels at node` j
- **foreach** $i \in N^-(j)$ **do**
  - `// loop all labels`
  - **foreach** $l \in \bar{L}_i$ **do**
    - **if** $\exists r \in R : d^r(l) + d^r_{ij} > u^r$ **then**
      - **next** l
    - **if** ***not*** $\eta_j \leq d^1(l) + d^1_{ij} \leq \min\{u^1, \eta_j + \varpi\}$ **then**
      - **next** l
    - `// create new label`
    - $L^m_j = (l, c(l) + c_{ij}, d^1(l) + d^1_{ij}, \ldots, d^{|R|}(l) + d^{|R|}_{ij})$
    - $L_j = L_j \cup L^m_j$
- `// remove dominated labels`
- $\bar{L}_j = \mathit{Efficient\_Labels}(L_j \cup \bar{L}_j)$
- $\eta_j = \min\{u^1, \eta_j + \varpi\}$

[Aneja et al., 1983]. Furthermore, we propose a novel problem-specific filtering technique that discards arcs based on their reduced costs and does not require to set up network H beforehand.

Generic preprocessing

In [Aneja et al., 1983] the number of arcs and nodes is reduced by computing minimum resource paths from the source to each node in the network and from each node in the network to the sink. The minimum resource paths are computed for each resource and used to identify nodes and arcs that violate the resource limits. Each node and arc that violates resource limits can be discarded. [Freling, 1997] describes essentially the same method but computes minimum and maximum resource paths since there are lower and upper limits on resource consumption.

More formally, let $\underline{P}^r_{ij}$ ($\bar{P}^r_{ij}$) be the path with minimum (maximum) consumption of resource $r \in R$ from node i to j with $i, j \in N$, and denote $\underline{P}^c_{ij}$ as the least cost path from i to j when resource constraints are not considered. Notice that finding the shortest paths from s to each node $i \in N$ requires the same computational effort as finding the shortest path from s to t using a simple $\mathcal{O}(m)$ dynamic programming algorithm. Thus, the overall complexity for computing all shortest paths is $\mathcal{O}(|R|m)$.

Now, we can delete all nodes $i \in N$ from network H that either violate resource limits (3.9) or cannot be part of a negative reduced cost path (3.10):

$$\exists r \in R : d^r(\underline{P}^r_{si}) + d^r(\underline{P}^r_{it}) > u^r \vee d^r(\bar{P}^r_{si}) + d^r(\bar{P}^r_{it}) < l^r \tag{3.9}$$

$$c(\underline{P}^c_{si}) + c(\underline{P}^c_{it}) > 0. \tag{3.10}$$

Likewise, we can remove all arcs $(i, j) \in A$ from network H where at least one of the following conditions is satisfied:

$$\exists r \in R : d^r(\underline{P}^r_{si}) + d^r_{ij} + d^r(\underline{P}^r_{jt}) > u^r \vee d^r(\bar{P}^r_{si}) + d^r_{ij} + d^r(\bar{P}^r_{jt}) < l^r \tag{3.11}$$

$$c(\underline{P}^c_{si}) + c_{ij} + c(\underline{P}^c_{jt}) > 0. \tag{3.12}$$

[Beasley and Christofides, 1989] and, more recently, [Mehlhorn and Ziegelmann, 2000] extend the method of [Aneja et al., 1983] by considering lower and upper bounds which they compute by a Lagrangian relaxation approach. We do not consider these Lagrangian methods here since they require to solve an additional Lagrangian dual. Notice that we would have to solve a separate Lagrangian dual problem for each duty generation network in each column generation iteration. We expect that this method is too time-consuming.

Recently, [Dumitrescu and Boland, 2003] propose a simplification of the method used in [Beasley and Christofides, 1989] and [Mehlhorn and Ziegelmann, 2000].

In particular, the authors use the initial cost instead of performing Lagrangian relaxation and using the resulting reduced costs. Their computational results show that their method is at least as good as the approach of [Beasley and Christofides, 1989] in terms of the degree of reduction.

Problem-specific preprocessing

Table 3.6 showed that between 79-83% of all arcs in a time-space network are piece of work arcs. Therefore, we propose a novel problem-specific reduction technique to discard piece of work arcs that does not require to set up the duty generation network. The arcs are dynamically discarded based on dual information.

Recall that a duty consists of several pieces of work separated by breaks. Furthermore, breaks have a minimum and/or maximum duration. In order to simplify the exposition, we assume that a duty consists of at most two pieces of work. However, our approach can be easily extended to the case with more than two pieces of work.

Let us consider an arbitrary piece of work p^*. Given that a duty may consist of at most two pieces of work, piece p^* can be either at the first or the second position of a duty. Furthermore, it can be connected with other pieces of work that end (start) within the maximum break duration bl_{max} but not within the minimum break length bl_{min} (see Figure 3.4).

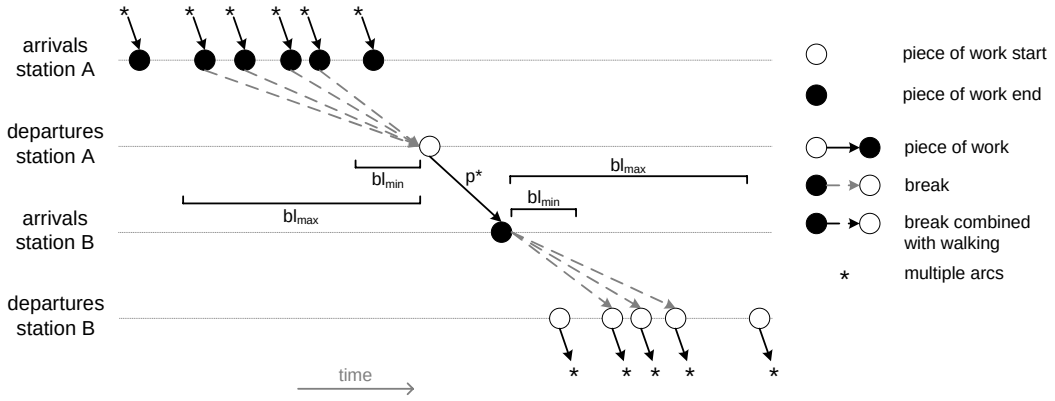

Figure 3.4.: Compatible pieces of work in a time-space duty generation network

Let st_i be the start time of piece i and et_i its end time. Furthermore, we denote by ω_{ij} the walking time between the end location of piece i and the start location of piece j. Two pieces i and j are said to be *compatible* if they can be covered consecutively by the same crew, that is $bl_{min} \leq st_j - et_i - \omega_{ij} \leq bl_{max}$ holds. We

denote by $\mathcal{P}_{p^*}$ the set of pieces that are compatible to piece p^*. Now, we can discard all pieces of work p^* that satisfy the following condition

$$f^t + \min_{p \in \mathcal{P}_{p^*}} \bar{f}_p + \bar{f}_{p^*} \geq 0 \tag{3.13}$$

where f^t denotes the fixed cost of the corresponding duty type and $\bar{f}_p$ the reduced cost of piece p. However, the set of compatible pieces is in $\mathcal{O}(\nu^2)$ for each piece with ν as number of relief points. As a result, the computational effort for checking condition (3.13) can be prohibitively high.

In order to reduce the computational time to check the condition, we define time slots $s \in S$ of length σ for each station (e.g. 15 minutes). Furthermore, for each time slot $s \in S$ we store the arriving piece with minimum reduced cost $\bar{f}^a_s$ and the departing piece with minimum reduced cost $\bar{f}^d_s$ (see Figure 3.5). We

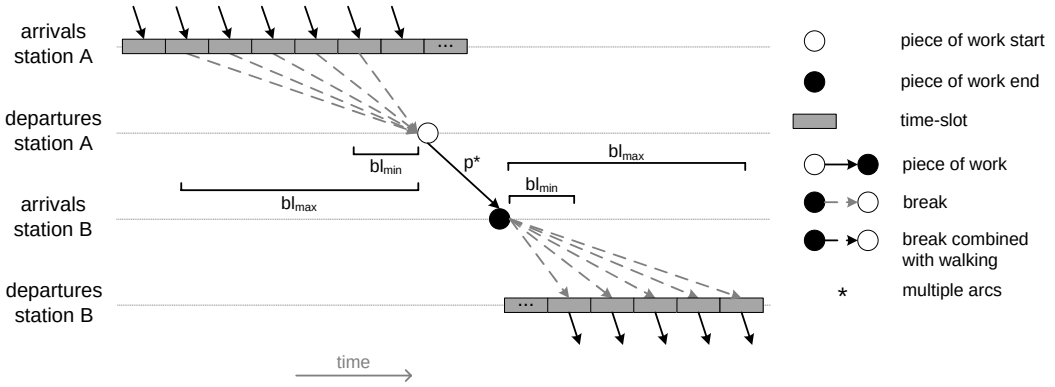

Figure 3.5.: Compatible time slots in a time-space duty generation network

define the set of forward and backward compatible time slots $s \in S$ of piece p^* by $S^f_{p^*}$ and $S^b_{p^*}$, respectively.

$$S^f_{p^*} = \{s \in S | bl_{min} \leq st_{p^*} - st_s - \omega_{sp^*} \vee 0 \leq st_{p^*} - et_s - \omega_{sp^*} \leq bl_{max}\} \tag{3.14}$$

$$S^b_{p^*} = \{s \in S | bl_{min} \leq et_s - et_{p^*} - \omega_{p^*s} \vee 0 \leq st_s - et_{p^*} - \omega_{p^*s} \leq bl_{max}\} \tag{3.15}$$

Similar to condition (3.13) we can discard each piece p^* where

$$f^t + \min_{s \in S^f_{p^*}} \bar{f}^a_s + \bar{f}_{p^*} \geq 0 \quad \wedge \quad f^t + \min_{s \in S^b_{p^*}} \bar{f}^d_s + \bar{f}_{p^*} \geq 0 \tag{3.16}$$

holds. The length of the time slots $\sigma > 0$ can be directly used to control the number forward/backward compatible time slots:

$$1 + \frac{bl_{max} - bl_{min}}{\sigma}. \tag{3.17}$$

It can easily be seen that we discard the same piece arcs with condition (3.13) and (3.16) if we set $\sigma = 1$. Notice that condition (3.16) requires less computational effort than (3.13) and can be straightforwardly adjusted. Furthermore, we see that problem-specific preprocessing with $\sigma = 1$ discards at least as many piece arcs $(i, j) \in A$ representing piece p^* as generic preprocessing (3.12) since $f^t + \min_{p \in \mathcal{P}_{p^*}} \bar{f}_p + \bar{f}_{p^*} \geq c(\underline{P}^c_{si}) + c_{ij} + c(\underline{P}^c_{jt})$ is satisfied.

Computational tests

In the following, we will evaluate whether the time spent on problem reduction actually pays off. We use the test set as described in the Section 3.1.2 which comprises 18 instances.

In Table 3.8 we show the impact of problem-specific (*ts*), and problem-specific plus generic preprocessing (*ts+gen*) on the overall performance. We report the number of iterations (*#iter*), the cpu time in seconds spent on the master (*cpu_ma*) and pricing problem including the time for network reduction (*cpu_pr*), total master and pricing time (*cpu_ma+pr*), the lower bound (*lb*), and the average arc reduction in percent (*arc_red%*) for duties with two pieces of work. All data given corresponds to the average over three instances. Notice that the given lower bound does not necessarily correspond to a valid lower bound. In order to get a realistic picture of the performance, we applied all acceleration techniques that we will describe in the following section except restricted networks. As we can see from Table 3.8, both preprocessing techniques considerably reduce the cpu time for the pricing problem. However, the time and network reduction are more favorable for instances with up to 200 trips. A possible explanation why our reduction performs better on small instances is that cost-based reductions are particularly useful in the final stage of column generation. The final stage is not reached for instances with 320 and 400 trips since column generation terminates due to the time limit.

Figure 3.6 shows the number of arcs (*no of arcs*) in the column generation process for the first depot of instance 320A09 (see [Huisman, 2003]) and three different duty types. Tripper duties consist of exactly one piece of work while split and normal duties have exactly two pieces of work (see [Huisman, 2004] for a complete description of the duty types). We performed 60 iterations and did not use restricted networks as described in the following section. As we can easily see the number of arcs can be almost halved in the course of the column generation process. Furthermore, the results support our claim that the problem-specific (cost-based) reduction works better in the final phase of column generation.

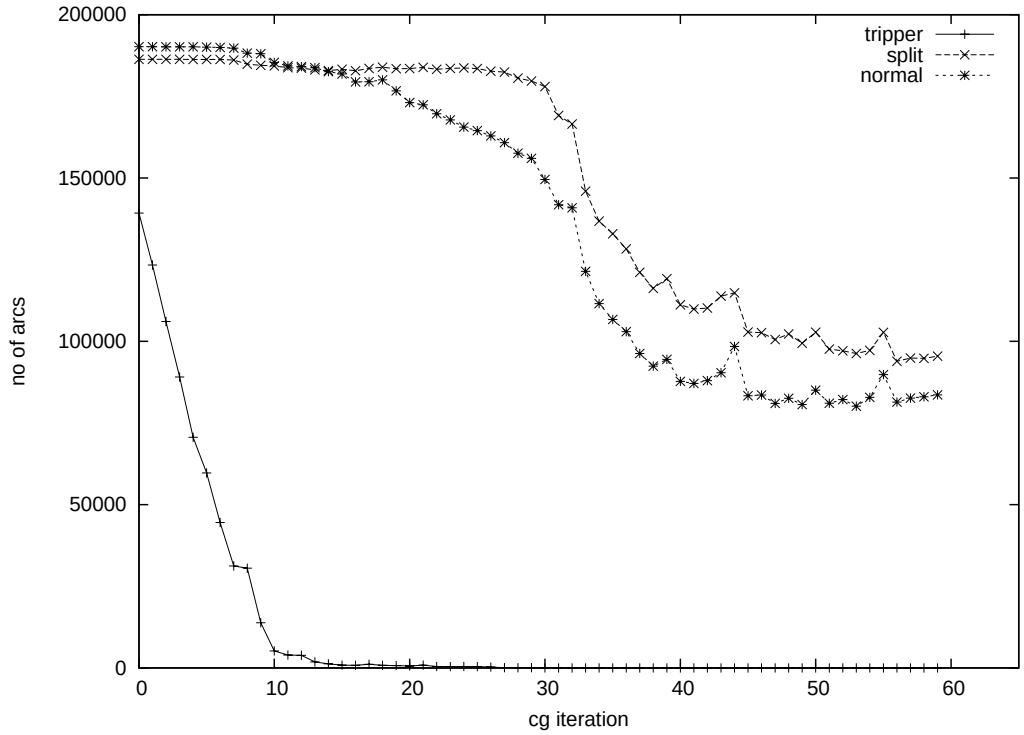

Figure 3.6.: Network reduction in the column generation process for first depot of instance 320A09 and three different duty types

reduction		#trips					
		080	100	160	200	320	400
none	#iter	21.3	26.7	28.3	28.3	29.3	25.0
	cpu_ma	160	282	723	936	3,721	3,708
	cpu_pr	78	342	1,957	3,186	1,647	1,603
	cpu_ma+pr	238	624	2,680	4,122	5,368	5,311
	lb	29,672	37,872	48,314	63,460	78,192	110,849
	arc_red%	0	0	0	0	0	0
ts	#iter	21.3	22.7	29.3	28.3	29.3	26.0
	cpu_ma	173	244	763	973	3,977	4,084
	cpu_pr	35	124	1,364	1,232	1,490	1,319
	cpu_ma+pr	208	368	2,127	2,205	5,467	5,403
	lb	29,647	37,811	48,321	63,456	78,278	110,705
	arc_red%	24	22	20	21	14	14
ts+gen	#iter	20.7	21.3	27.0	28.7	29.7	25.0
	cpu_ma	157	229	707	947	4,022	4,174
	cpu_pr	22	69	531	522	1,772	1,119
	cpu_ma+pr	179	298	1,238	1,496	5,794	5,293
	lb	29,651	37,840	48,320	63,458	78,386	110,670
	arc_red%	42	40	34	36	22	23

Table 3.8.: Impact of different network reduction techniques on the overall performance

3.2.3. Acceleration Techniques

There are numerous acceleration techniques for dynamic programming algorithms applied to column generation pricing in literature. For an excellent survey on master and pricing techniques to accelerate the overall performance we refer to [Desaulniers et al., 2002]. While the results in literature indicate the importance of acceleration methods, specific comparisons demonstrating actual savings are rare. The only attempts we are aware of are [Gamache et al., 1999] for the airline crew rostering problem, [Grönkvist, 2005] for the airline fleet assignment problem, and [Westerlund et al., 2006] for the traveling salesman subtour problem. However, only [Gamache et al., 1999] evaluate the impact of acceleration methods in column generation pricing. The purpose of this section is twofold: first, we describe pricing techniques that were particularly useful in accelerating the overall performance and second, we evaluate the impact of these techniques. Acceleration methods used to improve the performance of the master problem are

discussed in [Gintner, 2007].

Recall that the column generation method does not require the selection of the most negative reduced cost variable (duty). Furthermore, there are many feasible variables with negative reduced cost in the first iterations of the column generation process and only few in the final phase. To reduce the computational effort per iteration, we apply *partial pricing* where we heuristically reduce the search space and gradually increase it if we cannot find any (or enough) columns in the reduced search space. Finally, we perform *full pricing* to prove column generation optimality or to produce a column with negative reduced cost. We define a sequence of pricing heuristics $H_1, \ldots, H_p$ where the search space of H_i is smaller than that of H_{i+1}. We denote the exact pricing algorithm by H_{p+1}. Algorithm 8 describes a generic partial pricing algorithm where procedure *Pricing*(H_i) returns negative reduced cost columns using heuristic H_i. Notice that this algorithm may increase the number of iterations and the overall effect may also be unfavorable.

Algorithm 8: Generic partial pricing algorithm

(Step 1) **Initialization**
Set pricing level $i = 1$.
Define column set $C = \emptyset$.
Define column threshold t.

(Step 2) **Pricing**
while $|C| \leq t \wedge i \leq p + 1$ **do**
  $C =$*Pricing*(H_i)
  $i = i + 1$
return C

Essentially the same idea is used in the pricing step of the simplex algorithm where the reduced costs of the nonbasic variables are computed and one of the negative reduced cost columns (if any) is selected to enter the basis. [Dantzig, 1963] originally proposed the so-called Dantzig rule where all nonbasic variables are checked (full pricing) and the least negative reduced cost column is selected (in case of a minimization problem). [Orchard-Hays, 1968] proposed to check only a part of the nonbasic variables (partial pricing) and select the best variable from this part. The search space is changed when no entering variable with negative reduced cost can be found.

In the following, we will describe the acceleration strategies that we used in our pricing algorithm. We will present a combination of known and novel techniques to further improve the performance of the algorithm. In particular, we

will propose a novel way of generating balanced restricted networks, suggest customized dominance rules, and show how we can strengthen label-pruning bounds previously exposed in literature. Furthermore, we will evaluate the impact on the overall performance using the same assumptions as described in Section 2.1.

Multiple pricing

A well-known strategy to accelerate column generation is to create multiple negative reduced cost columns in each pricing step. With a dynamic programming algorithm we can easily compute multiple paths per iteration since all states at the sink node correspond to feasible paths. Basically, multiple pricing increases the computational effort to solve the master problem, but may decrease the number of iterations.

Similar to the pivot strategy in the simplex algorithm there is probably not a best pricing strategy in column generation. A disadvantage of the common strategy to select columns with minimum reduced costs is the following. If there is a (small) subset of rows with high dual values, it is likely that the pricing algorithm returns many columns covering these rows while only one of these columns can be in the final integer solution. Consequently, we use an adapted version of the *disjoint column method* of [Gamache and Soumis, 1998] to avoid generating similar columns in an iteration. Finally, we refer to [Vanderbeck, 1994] for a thorough discussion on how to select "good" columns. [Freling, 1997] also reviews different strategies for column selection.

In our computational experiments, we found that adding only the least reduced cost column leads to a very poor performance of the overall algorithm. Based on limited computational experience [Vanderbeck, 1994] states that multiple pricing works better if the subproblem solution is computationally expensive (as in our case). Our results seem to support his understanding and, thus, we will not consider adding the least reduced cost column again. Furthermore, our computational results indicate that adding at most 5,000 columns per depot/duty type combination works best for a wide range of problem sizes.

Restricted networks

As stated earlier there are many feasible variables with negative reduced cost in the first iterations of the column generation process and only few in the final phase. To speed up the pricing step, we heuristically reduce the network size in the first iterations by ignoring some arcs and nodes. In particular, we ignore arcs according to the dual information or randomly. In our implementation, we

initially set the maximum number of piece of work arcs to $200|\mathcal{T}|$ where $\mathcal{T}$ corresponds to the set of trips. If the improvement of the lower bound is less than 10% in the last five iterations, we subsequently reintroduce the discarded arcs until we have the complete network in the final iteration(s). We do not reintroduce arcs that cannot be part of a negative reduced cost path after problem-specific preprocessing (see Section 3.2.2). Notice the similarity to traditional scaling algorithms (see [Ahuja et al., 1993]). Examples of restricted networks in column generation can be found in [Barnhart et al., 1995] and [Gamache et al., 1999].

Next, we describe how we actually ignore arcs. Basically, the strategy to simply ignore all piece of work arcs with poor reduced costs can lead to unbalanced duty generation networks. The rationale behind this observation can be understood in the following way. If there is a (small) subset of rows with high dual values, it is very likely that pieces covering these rows will be included in the duty generation network. Thus, the network contains comparatively many arcs covering these rows, and the pricing algorithm is likely to return an unbalanced set of duties. As discussed earlier this may lead to poor convergence of the column generation algorithm. Therefore, we propose to discard pieces in a balanced way using time slots. To the best of our knowledge, this has not been tried before. In this context, we define time slots $s \in S$ of length σ for each station similar to those in Section 3.2.2. Furthermore, we compute the initial acceptance rate by

$$\rho = \min\left\{1.0, \frac{200|\mathcal{T}|}{|\mathcal{P}|}\right\} \tag{3.18}$$

where $\mathcal{P}$ corresponds to the set of pieces of work. Then, we discard $(1-\rho) * 100$ percent of all pieces of a time slot either randomly or according to the dual information. As a result, we obtain a balanced network where the reduction is evenly spread among the complete network. In our computational experiments, we found that discarding pieces randomly outperforms discarding pieces according to the dual information. This was particularly true for large instances. Our interpretation is that randomly restricted networks provide more balanced duty sets.

State space reduction

The basic idea of state space reduction in dynamic programming is to increase dominance between labels (states) and, hence, reduce the solution time. We have tested different ways to increase dominance between labels. However, we will only describe those techniques that proved to work best on a wide range of instances and feasibility constraints.

[Gamache et al., 1999] increase dominance by reducing the range of the units of the labels. In other words, the authors use less precise units of measurement to describe some of the resources. For instance, resources related to time can be measured in hours instead of minutes or can be rounded to the nearest ten minutes. As a consequence, labels that originally had different consumption for some resources are now represented by the same value. This increases the chance that one label dominates another and that fewer labels have to be evaluated. Since the restricted state space does not guarantee column generation optimality, we gradually adapt the unit measurement until the initial resource vectors are used. However, the feasibility of a label is checked at each node always using the initial unit of measurement.

Additionally, dominance can be increased by using stronger dominance rules. In particular, we use four different dominance rules where our dynamic programming algorithm starts with level 1 and ends with level 4 to prove column generation optimality. We increase the dominance level if we have not found enough columns in the current level. The computational effort increases with the level of dominance. Consider the following feasibility constraints for duties as defined in Section 3.1.2: maximum working time, maximum spread time (duty length), minimum start time, maximum end time, minimum break length, and minimum/maximum number of pieces. Furthermore, recall that we need three resources and costs to cover these constraints: working time, spread time, and number of pieces of work. We propose the following dominance rules that are customized for the vehicle and crew scheduling problem.

- *Level 1*: We ignore the spread time (duty length) information in all dominance checks. Thus, we apply the dominance process over three-dimensional vectors instead of four-dimensional ones. The underlying assumptions are that most of the spread time is also working time and that maximum working time is more restrictive than maximum spread time. Therefore, we substitute the spread time information of a label by working time. Moreover, we apply dominance tests for labels that do not satisfy the lower limit for the number of pieces of work.

- *Level 2*: We do not allow dominance tests between labels with a different number of pieces of work. At level 1, we allow to compare two labels that represent two partial paths with a different number of pieces. Furthermore, it is likely that the label with the higher number of pieces also consumes more resources than the label with a smaller number of pieces. Consequently, there is a good chance to discard labels with a higher number of pieces. In other words, we do not prefer labels with a small number of

pieces at level 2. However, we have to check more labels compared to level 1. Again, we ignore the spread time information of the labels.

- *Level 3*: We apply the dominance process over all resources, i.e., a four-dimensional vector, but still ignore lower limits for resource consumption.
- *Level 4*: At the final level, we check all resources considering upper and lower limits for resource consumption. Given a lower limit is a hard constraint, we can consider this limit in two different ways. We either do not apply dominance tests to labels that do not satisfy the lower limit or we introduce a new resource to strictly enforce it. The new resource is the negative of the original resource to enforce the upper limit (see [Gamache et al., 1999]).

Table 3.9 shows the performance of our dynamic programming algorithm in the first column generation iteration. We report the cpu time in seconds to compute the 100 least reduced cost columns (*cpu_pr*) with all other acceleration strategies disabled except network reduction. Furthermore, we give the lower bound (*lb_1*) obtained with a subgradient approach (see Section 2.4.1) after the columns have been added. We use the same test set as described in Section 3.2.2. As we can see from Table 3.9 the cpu time can be dramatically reduced by using inexact dominance tests while there is only a small increase of the lower bound (less than 3%) at level 1 and 2. In other words, it suffices to solve the pricing problem in the initial phase of column generation with inexact dominance rules.

dominance		#trips					
		080	100	160	200	320	400
level 1	cpu_pr	0.1	0.2	0.7	0.9	6.2	7.6
	lb_1	38,931	45,209	68,669	76,191	105,125	144,323
level 2	cpu_pr	0.1	0.2	0.9	1.2	7.9	10.6
	lb_1	38,881	45,206	68,281	75,764	105,440	142,525
level 3	cpu_pr	0.5	0.8	6.6	9.7	183.8.	239.0
	lb_1	38,798	45,197	67,302	75,522	102,452	141,883
level 4	cpu_pr	1.9	4.6	67.3	136.7	3024.1	4399.7
	lb_1	38,798	45,198	67,333	75,522	102,117	142,405

Table 3.9.: Results of dynamic programming algorithm with different dominance tests in the first column generation iteration

Label pruning

Similar to state space reduction, label pruning also relates to the question how the number of evaluated states can be reduced. In this section, however, we consider pruning techniques to reduce the number of labels that do not refer to increasing dominance.

In order to control the number of unprocessed labels, we can set a maximum size $\kappa_{\max}$ of the label list for each node. We only keep the best $\kappa_{\max}$ labels at each node with respect to reduced cost. A disadvantage of this approach may be that the pricing algorithm returns many similar columns (given that we have a small subset of rows with high dual values). Therefore, we use a similar method to [Mesquita and Paias, 2006] where labels are randomly discarded. The authors argue that only a small percentage of the columns generated in earlier iterations will be part of the basis of the current linear restricted master problem. Therefore, they prefer to generate a small set of columns with a greater diversity. The authors show that this method actually reduces cpu time while maintaining the quality of the linear relaxation. Our method differs from the one suggested by [Mesquita and Paias, 2006] in the following way. Let κ_j be the number of labels at node $j \in N$. When pulling labels at node j we can easily estimate the final number of labels $\bar{\kappa}_j$ at that node by

$$\bar{\kappa}_j = \rho \sum_{i \in N^-(j)} |\bar{L}_i| \tag{3.19}$$

where ρ corresponds to the average acceptance rate of a label after the dominance process has been applied. If the estimated number of labels at that node exceeds a given threshold $\kappa_{\max}$ we pull a label from node i with probability $\kappa_{\max}/\bar{\kappa}_j$. With our method, we heuristically reduce the search space only if the number of labels becomes large. For instance, if the number of discarded arcs during preprocessing is large and, hence, the pricing problem is comparatively easy, we still use the exact pricing algorithm.

Furthermore, we use lower bounds to reduce the state space similar to [Lübbecke, 2005]. We first describe the method of [Lübbecke, 2005] and then discuss our adaptations in order to strengthen the lower bounds. Consider label $L_i^k = (L_h^l, c(P_i^k), d^1(P_i^k), \ldots, d^{|R|}(P_i^k))$ at node $i \in N$ that represents path P_i^k. Furthermore, let $A^+(P_i^k) = \{(m,n) \in A | m \geq i\}$ be the set of arcs that are compatible with path P_i^k. Recall that the nodes in the duty generation network are sorted by increasing time. According to [Lübbecke, 2005] a lower bound lb_i^k for label L_i^k can be computed by

$$lb_i^k = c(P_i^k) \quad + \sum_{(m,n) \in A^+(P_i^k)} \bar{f}_{mn}. \tag{3.20}$$

If $lb_i^k \geq 0$ holds, label L_i^k can be discarded. Of course, the lower bound is not very tight and discards only few labels since it does not take the structure of the network and path P_i^k into account. Therefore, we propose a novel way to strengthen bound lb_i^k in the following.

Recall from Section 3.2.2 that we defined time slots $s \in S$ of a specific length for each station and stored the least reduced cost $\bar{f}_s^d$ of all pieces departing within that time slot. Again, we consider the case where a duty consists of at most two pieces of work. However, the approach can be generalized to the n-piece case in a straightforward way. Basically, label elimination is most effective when applied early in the construction of a particular path. Therefore, we consider the cases where (1) the first piece arc and (2) the first break is to be added to path P_i^k of label L_i^k. In the first case, we cannot derive a tighter bound when problem-specific preprocessing has been applied before the network was set-up. In the latter case, however, we will not extend label L_i^k by arc $(i,j) \in A$ if

$$c(P_i^k) + c_{ij} + \min_{s \in S^b_{(i,j)}} \bar{f}_s^d \geq 0 \tag{3.21}$$

where $S^b_{(i,j)}$ denotes the set of backward compatible time slots of break arc (i,j). If we use a time-space duty generation network (see Section 3.1.2), $|S^b_{(i,j)}| = 1$ holds since only the time slot of the piece departure node has to be considered.

Furthermore, we strengthen the pruning conditions in the early iterations in such a way that we only accept labels if the estimated lower bound is significantly below zero.

Computational tests

We use the test set and settings as described in the Section 3.1.2 which comprises 18 instances. In Table 3.10 we show the performance of our dynamic programming algorithm with and without acceleration techniques. We report the number of iterations (*#iter*), the cpu time in seconds spent on the master (*cpu_ma*) and pricing problem (*cpu_pr*), the lower bound (*lb*), and the total number of vehicles and drivers (*v+d*) of the best feasible solution. Notice that the given lower bound does not necessarily correspond to a lower bound for the overall problem (see Section 3.5). All data given corresponds to the average over three instances. With acceleration techniques the time spent on the pricing problem can be dramatically reduced. For instances with more than 160 trips, column generation terminates due to excessive computation time in the pricing problem when no acceleration is used. As we can see for instances with at most 100 trips, acceleration increases the number of iterations, but the cpu time for master and pricing can be reduced by 30%-52%. Time savings for instances with 160 trips are even higher where 80%

of the cpu time to obtain a comparable lower bound can be saved. For larger problems, we can considerably improve the lower bound by using acceleration techniques since more column generation iterations can be performed within the given timeframe.

type		#trips					
		080	100	160	200	320	400
normal	#iter	15.3	15.7	14.3	10.0	6.3	5.0
	cpu_ma	149	207	319	283	14	9
	cpu_pr	111	446	5,103	5,154	5,386	5,391
	lb	29,647	37,832	48,814	66,723	86,682	120,649
	v+d	27.7	35.7	46.3	62.3	79.7	110.0
acceleration	#iter	21.3	21.7	28.0	28.0	33.0	28.7
	cpu_ma	172	246	729	941	3,989	4,268
	cpu_pr	12	70	350	361	1,451	1,243
	lb	29,649	37,849	48,331	63,477	77,282	108,366
	v+d	27.3	35.7	46.3	61.3	76.0	105.7

Table 3.10.: Results of dynamic programming algorithm with and without acceleration techniques

3.3. Integer Solutions

In Section 2.4 we described the solution process for model MDVCSP (see Section 2.3). Basically, the solution approach consists of two stages: a lower bound phase and a final phase where a feasible solution is constructed. In the lower bound phase, we apply column generation in combination with Lagrangian relaxation. In the preceding sections we have described how we modeled and solved the column generation pricing problem while this section is devoted to the final phase where we compute feasible solutions to model MDVCSP.

This section is organized as follows. In Section 3.3.1, we use the approach of [Huisman, 2004] (see Section 2.4.3), but compare different ways of constructing feasible solutions. In Section 3.3.2 we describe different branching rules when feasible solutions are generated with a commercial MIP solver such as ILOG CPLEX [ILOG, 2006] or MOPS [Suhl, 2000]. Finally, in Section 3.3.3, we propose a novel integer procedure which enhances the approach of [Huisman, 2004]. We enhance the approach in the sense that we regenerate columns in the integer

phase and apply depth-first (heuristic) branching in combination with different fixing strategies.

3.3.1. Sequential Approach

In order to ease the exposition, we will describe the Lagrangian heuristic of Huisman (see [Huisman, 2004]) to construct integer solutions in detail. The final step (*integer phase*) of the solution method (see Algorithm 5, Section 2.4) aims at finding a pair of feasible and compatible vehicle and crew schedules with a Lagrangian heuristic. Only the linking constraints (2.14) of model MDVCSP (2.11)-(2.16) are relaxed in a Lagrangian way. The objective function now reads

$$\begin{aligned} \min \quad & \sum_{d\in\mathcal{D}} \sum_{(i,j)\in A^d} y_{ij}^d c_{ij}^d + \sum_{d\in\mathcal{D}} \sum_{k\in K^d} x_k^d f_k^d \\ & + \sum_{d\in\mathcal{D}} \sum_{(i,j)\in \widetilde{A}^d} \mu_{ij}^d \left(y_{ij}^d - \sum_{k\in K^d(i,j)} x_k^d \right) \end{aligned} \tag{3.22}$$

where μ_{ij}^d correspond to the Lagrangian multipliers associated with the linking constraints. Furthermore, the Lagrangian subproblem results in

$$\Phi'(\mu) = \Phi'_y(\mu) + \Phi_x(\mu) \tag{3.23}$$

with

$$\begin{aligned} \Phi'_y(\mu) = \Bigg\{ \quad & \min \quad \sum_{d\in\mathcal{D}} \sum_{(i,j)\in A^d} y_{ij}^d \bar{c}_{ij}^d \mid \\ & \sum_{d\in\mathcal{D}} \sum_{(i,j)\in A^d(t)} y_{ij}^d = 1 \qquad\qquad \forall t \in \mathcal{T} \\ & \sum_{\{j:(j,i)\in A^d\}} y_{ji}^d = \sum_{\{j:(i,j)\in A^d\}} y_{ij}^d, \quad \forall d \in \mathcal{D}, \forall i \in N^d, \\ & 0 \le y_{ij}^d \le u_{ij}^d, y_{ij}^d \in \mathbb{N}, \qquad\qquad \forall d \in \mathcal{D}, \forall (i,j) \in A^d \Bigg\} \end{aligned} \tag{3.24}$$

as vehicle scheduling subproblem and

$$\begin{aligned} \Phi_x(\mu) = \Bigg\{ \quad & \min \quad \sum_{d\in\mathcal{D}} \sum_{k\in K^d} x_k^d \bar{f}_k^d \mid \\ & x_k^d \in \{0,1\}, \quad \forall d \in \mathcal{D}, \forall k \in K^d \Bigg\} \end{aligned} \tag{3.25}$$

as crew scheduling subproblem. The reduced cost $\bar{c}_{ij}^d$ on arc $(i,j) \in A^d$ of the vehicle scheduling network of depot $d \in \mathcal{D}$ is defined as

$$\bar{c}_{ij}^d = \begin{cases} c_{ij}^d + \mu_{ij}^d & \text{for } (i,j) \in \widetilde{A}^d \\ c_{ij}^d & \text{for } (i,j) \notin \widetilde{A}^d \end{cases} \tag{3.26}$$

while

$$\bar{f}_k^d = f_k^d - \sum_{(i,j) \in \widetilde{A}^d(k)} \mu_{ij}^d \tag{3.27}$$

denotes the reduced cost of duty $k \in K^d$ where $\widetilde{A}^d(k) \subseteq \widetilde{A}^d$ corresponds to the set of arcs that is covered by duty $k \in K^d$. In contrast to the lower bound phase, the vehicle scheduling subproblem corresponds to a multiple-depot vehicle scheduling problem that neither has the integrality property nor can be solved in polynomial time. However, the solution of the vehicle scheduling subproblem gives a feasible vehicle schedule. Each feasible vehicle schedule can be used to construct a feasible crew schedule using traditional (sequential) crew scheduling. As in the lower bound phase, we use a subgradient algorithm to solve the associated Lagrangian dual problem.

Unlike [Huisman, 2004] we do not only perform 10 subgradient iterations in the final phase since our computational experiments indicated that better solutions can be found if more iterations are performed. Therefore, we solve the associated CSP for each depot every i-th subgradient iteration after k iterations have been performed in order to obtain a feasible and compatible crew schedule. Similar to the integrated setting, we use column generation in combination with Lagrangian relaxation to solve the associated crew scheduling problems. Furthermore, we apply dynamic programming to solve the pricing problem with the acceleration techniques described earlier. Integer solutions are computed with the commercial MIP solver ILOG CPLEX 10.0 using the columns generated before. Basically, the same approach is used by [Gintner, 2007], but the author uses a different way of pricing new negative reduced cost columns.

In addition to sequential crew scheduling as used by [Huisman, 2004] we can compute feasible crew schedules using a partially or fully integrated approach (see Sections 2.2.2 and 2.2.3, respectively). In particular, we apply the partially integrated method of [Gintner et al., 2006a, Gintner, 2007] to generate feasible solutions. The basic idea is to change a given optimal vehicle schedule without loss of optimality in the crew scheduling phase. The authors set up a time-space network that allows to recombine parts of vehicle blocks in order to disclose additional flexibility in crew scheduling while preserving vehicle schedule optimality.

In other words, the crew scheduling approach does not only consider a single optimal vehicle schedule, but a set of optimal vehicle schedules with minimum fleet size and minimum operational costs. For the remainder of the thesis, we call this crew scheduling approach *adaptive crew scheduling*.

Furthermore, we tested a fully integrated approach where the trip-depot assignment instead of the complete vehicle schedule serves as input. Notice that this results in a (single depot) integrated vehicle and crew scheduling problem for each depot that discloses even more flexibility in the crew scheduling phase than adaptive crew scheduling. However, in our computational experiments we found that this method has a poor performance in terms of solution quality and computational time. Therefore, we will not consider this approach again.

In Table 3.11 we compare the performance of sequential and adaptive crew scheduling to construct integer solutions. We use the same test set (18 problem instances) and settings as described in the preceding sections except that we extend the cpu time for the lower bound phase to 36,000 seconds (10 hours). Furthermore, we try to find an upper bound in a column generation iteration if more than 200 subgradient iterations have been performed. In that case, we solve a multiple-depot vehicle scheduling problem using the best multipliers of that iteration as costs on the arcs. In Table 3.11 we give the number of iterations (*#iter*), the cpu time in seconds spent on the master problem (*cpu_ma*), on pricing problem (*cpu_pr*), on the integer phase (*cpu_ip*), and the total cpu time (*cpu_tot*). The cpu time of the master problem includes the time to find new upper bounds in the lower bound phase. Furthermore, we provide the lower bound (*lb*) and the total number of vehicles and drivers (*v+d*) of the best feasible solution. Notice that the best lower bound obtained is not necessarily a valid lower bound.

The sequential approach to find integer solutions outperforms the adaptive one in terms of solution time, but generates worse solutions. Notice that no integer phase was executed with the adaptive method for the group with 80 trips since the best upper bound obtained in the lower bound phase already had the minimum number of duties. From our point of view, the gain in solution quality outweighs the additional computational effort. Therefore, we will not consider the sequential approach to generate feasible solutions in the remainder of this chapter.

3.3.2. Branch-and-Bound with MIP-Solver

In this section, we apply a branch-and-bound approach (see Section 1.5.4) in the integer phase instead of the method described in the preceding section. First, we

type		#trips					
		080	100	160	200	320	400
sequential	#iter	19.0	19.7	26.0	25.7	34.7	33.0
	cpu_ma	130	188	643	832	5,190	5,501
	cpu_pr	9	35	245	195	5,987	7,958
	cpu_ip	222	473	1,671	2,473	6,140	7,688
	cpu_tot	362	698	2,565	3,506	17,422	21,288
	lb	29,643	37,814	48,304	63,437	77,137	108,001
	v+d	27.3	35.7	46.3	61.7	76.3	106.0
adaptive	#iter	18.0	19.0	23.3	24.7	35.3	32.3
	cpu_ma	153	222	1,170	1,712	7,560	8,318
	cpu_pr	11	32	302	202	5,477	8,120
	cpu_ip	0	578	1,676	2,621	7,635	9,808
	cpu_tot	165	834	3,155	4,541	20,794	26,384
	lb	29,664	37,834	48,382	63,470	77,191	107,922
	v+d	26.7	35.0	45.3	60.7	76.0	105.0

Table 3.11.: Results of sequential and adaptive crew scheduling to find integer solutions

generate a set of promising columns in the lower bound phase (steps 1 to 5 of Algorithm 5). Then, we apply an LP-based branch-and-bound method on model MDVCSP with the restricted set of columns to compute a feasible solution.

Recall that our model MDVCSP has two types of decision variables: flow and duty variables. In the following, we propose three different branching schemes for our model that prioritize either flow or duty variables. Furthermore, we suggest to apply the well-known Ryan-Foster branching rule on our model. To the best of our knowledge, these branching schemes have not been used in combination with model MDVCSP. Furthermore, follow-on branching has not been applied on integrated vehicle and crew scheduling problems. We will conclude this section with computational results comparing the branch-and-bound methods with our approach from the preceding section.

Branching on variables

Several authors have proposed branching rules for integrated vehicle and crew scheduling problems with multiple depots. In the following, we will briefly review these approaches.

The approach of [Borndörfer et al., 2004] relies on model MDVCSP-H (see

Section 2.2.3). They use a solution approach similar to Algorithm 5 since they aim at computing a lower bound first and subsequently generate an integer feasible solution. However, the authors solve the Lagrangian dual problem with an inexact adaptation of a proximal bundle method. The inexact bundle method is embedded in a backtracking procedure to produce integer solution in the second phase. The procedure utilizes the primal information produced by the bundle method to iteratively fix deadhead (flow) variables until the complete vehicle schedule is fixed. In their model fixing a deadhead determines the successor of a service trip, but also implicitly assigns that sequence to a depot.

The approach of [Mesquita et al., 2006] is based upon a model similar to MDVCSP-H that contains fewer constraints. They propose to solve the linear relaxation of a combined multi-commodity flow and mixed set partitioning/covering model with column generation. If the linear relaxation of the root node is not integer, the authors suggest a branch-and-bound and two branch-and-price schemes. The branch-and-bound method branches over the set of feasible duties generated while solving the linear relaxation of the root node. The authors have compared two branching schemes. In one strategy they branch on duty variables first while in the other strategy they first branch on flow variables. Although the authors do not provide computational results for both schemes, they state that the first scheme performs better on their model.

To sum up, [Borndörfer et al., 2004] branch on flow variables while [Mesquita et al., 2006] prefer to branch on duty variables first. Consequently, we propose three branching schemes that prioritize either flow or duty variables of model MDVCSP for branching. Recall that our model is based on a time-space network while those of the authors stated above rely on a connection-based network.

We define the following priority function that returns the branching priority of an arbitrary flow or duty variable

$$\Psi : \{y_{ij}^d | (i,j) \in A^d, d \in \mathcal{D}\} \cup \{x_k^d | k \in K^d, d \in \mathcal{D}\} \rightarrow \mathbb{R}_0^+ \tag{3.28}$$

with the following properties.

1. $\Psi(z) = 0$: Variable z is not selected for branching.

2. $\Psi(z) > 0$: Variable z can be selected for branching if z is not integer in the current solution of the linear relaxation.

3. $\Psi(z_1) > \Psi(z_2)$: If z_1 is not integer, z_1 will be selected for branching before z_2 is chosen. If z_1 is integer, z_2 will be chosen no matter what priority z_1 has.

Furthermore, we denote the set of trip arcs for depot $d \in \mathcal{D}$ by $A_T^d = \bigcup_{t \in \mathcal{T}} A^d(t)$. We define the following branching schemes that first branch on flow variables:

$$ps: \quad \Psi(y_{ij}^d) > \Psi(y_{rs}^d) > \Psi(x_k^d) > 0, \\ \forall (i,j) \in A_T^d, \forall (r,s) \in A^d \backslash A_T^d, \forall x_k^d \in K^d \text{ and } \forall d \in \mathcal{D}, \tag{3.29}$$

$$pv: \quad \Psi(y_{rs}^d) > \Psi(y_{ij}^d) > \Psi(x_k^d) > 0, \\ \forall (i,j) \in A_T^d, \forall (r,s) \in A^d \backslash A_T^d, \forall x_k^d \in K^d \text{ and } \forall d \in \mathcal{D}. \tag{3.30}$$

Branching scheme *ps* first branches on flow variables that correspond to service trips. In other words, we assign trips to depots before we decide about the sequence of trips or about the crew scheduling part. If we assign a trip t_i to depot d_j, we can discard all other trips arcs of t_i that belong to another depot. However, it does not suffice to assign trips to depots to completely define a vehicle schedule. Thus, deadhead connections must be fixed in a second step. In branching rule *pv* we give a higher priority to flow variables that correspond to deadheads. We first decide which trips are operated in sequence before we assign these sequences to a depot. In a time-space network, however, we cannot directly fix a connection between two particular trips since deadhead and waiting activities are aggregated (see Section 2.3). Therefore, we must also branch on (trip) flow variables in order to obtain a complete vehicle schedule.

Finally, we propose branching scheme *pd* that first branches on duty variables. The rationale behind that rule can be understood in the following way. The crew scheduling problem is usually more constrained than the vehicle scheduling problem since many work regulations must be considered. Therefore, it may be beneficial to first decide about the crew schedule and later construct a compatible vehicle schedule.

$$pd: \quad \Psi(x_k^d) > \Psi(y_{ij}^d) = \Psi(y_{rs}^d) > 0, \\ \forall (i,j) \in A_T^d, \forall (r,s) \in A^d \backslash A_T^d, \forall x_k^d \in K^d \text{ and } \forall d \in \mathcal{D} \tag{3.31}$$

A disadvantage of rule *pd* may be that there are many similar duty variables. Thus, forbidding the use of a variable may only have a minor impact since a similar column can be selected at the child node. As a consequence, many nodes must be evaluated until a good solution is found. In the following subsection, we will describe a branching scheme based on the Ryan-Foster rule that overcomes this shortcoming.

Finally, the branching schemes stated above only determine the type of variable that is to be preferred. The schemes do not define which particular variable is chosen. In our computational experiments, however, we found that it performs best if we leave this decision to the MIP solver such as CPLEX.

Branching on follow-ons

Branching on follow-ons relies on a general branching strategy for set partitioning problems that was introduced by [Ryan and Foster, 1981]. The branching scheme is based on the following property. Given a fractional solution to a set partitioning problem, we can identify two rows i and j such that the subset $C(i,j)$ of columns that contain i and j has the property

$$0 < \sum_{c \in C(i,j)} x_c < 1. \tag{3.32}$$

The remaining fraction of cover for each constraint must be provided by columns that do cover both rows at the same time. Thus, an effective constraint branching scheme is to require to cover two rows i and j by the same column on one branch and by different columns on the other. [Vance et al., 1997a] slightly modify the scheme to maintain tractability. They only consider trips (rows) i and j that correspond to trips operated consecutively in a duty (column). Furthermore, the authors show that this modification still constitutes a correct branching scheme. We refer to this strategy as *branching on follow-ons* since we impose which trips can follow trip i in the solution. Moreover, we refer to the trip pair (i,j) as *follow-on*.

In the following we will describe how we adapt branching on follow-ons for the integrated vehicle and crew scheduling problem with multiple depots. Consider two trip arcs $y^d_{ij} \in A^d_T$ and $y^d_{rs} \in A^d_T$ from depot $d \in \mathcal{D}$ and the set of duties $K^d(y^d_{ij}, y^d_{rs})$ where both trips are covered consecutively. Now, we define our branching scheme for two compatible service trips. Two trips must be operated consecutively from depot d on one branch and not consecutively from depot d on the other.

$$\sum_{k \in K^d(y^d_{ij}, y^d_{rs})} x^d_k \;\geq\; 1 \quad \wedge \quad y^d_{ij} = 1 \quad \wedge \quad y^d_{rs} = 1 \qquad \text{1-branch} \tag{3.33}$$

$$\sum_{k \in K^d(y^d_{ij}, y^d_{rs})} x^d_k \;\leq\; 0 \qquad \text{0-branch} \tag{3.34}$$

Notice that arcs y^d_{ij} and y^d_{rs} are not necessarily consecutive in the final vehicle and crew schedule solution since deadhead arcs may be in between.

Given a fractional LP solution there are usually many candidate follow-ons that can be used. Hence, we define the *support* of a follow-on (y^d_{ij}, y^d_{rs}) similar to [Vance et al., 1997a].

$$f(y^d_{ij}, y^d_{rs}) = \sum_{k \in K^d(y^d_{ij}, y^d_{rs})} x^d_k \tag{3.35}$$

Clearly, $0 \leq f(y^d_{ij}, y^d_{rs}) \leq 1$ is satisfied. The support of a follow-on can be interpreted as the probability of including that follow-on in the solution. Furthermore, we define two branching schemes where *fo-flf* selects the least fractional while *fo-fmf* chooses the most fractional follow-on among all candidate follow-ons.

$$\textit{fo-flf}: \quad (y^d_{ij}, y^d_{rs}) = \arg\max f(y^d_{ij}, y^d_{rs}) \tag{3.36}$$

$$\textit{fo-fmf}: \quad (y^d_{ij}, y^d_{rs}) = \arg\min |0.5 - f(y^d_{ij}, y^d_{rs})| \tag{3.37}$$

Branching rule *fo-flf* seems to be particularly useful in combination with a depth-first tree search (see [Vance et al., 1997a]). Notice that selecting a follow-on with $f(y^d_{ij}, y^d_{rs}) = 1$ will not eliminate the current fractional LP solution. Therefore, [Vance et al., 1997a] propose to fix all *perfect* follow-on pairs where $f(y^d_{ij}, y^d_{rs}) = 1$ is satisfied. Of course, additional fixings at a branch-and-bound node are heuristic. In the following section we evaluate both follow-on branching schemes with and without additional fixings.

In contrast to the original branching scheme on set partitioning models, we cannot guarantee that we always find a follow-on for a fractional solution for model MDVCSP. In particular, we cannot find a follow-on if all trips have been assigned to a depot. In such a case, we perform the default branching decision of the MIP solver. However, such a situation never occurred in our computational experiments.

Computational results

In Table 3.12 we compare the performance of our branching rules with the approaches from the preceding sections. We make the same assumptions as before. However, we only report results for five instances with 100 trips from the Huisman test set (see [Huisman, 2003]). For larger instances, we hardly found integer solutions in the set of columns generated in the lower bound phase. In our computational experiments, we tried to generate different numbers of columns, but all settings lead to a very poor performance of the MIP solver. We tested our implementation on a Pentium IV 2.3GHz/2GB RAM personal computer using ILOG CPLEX 10.0. We terminated the branch-and-bound search if the cpu time exceeded 3,600 seconds.

We give results for the sequential (*seq*) and adaptive (*adap*) approach described in Section 3.3.1, using the default setting of CPLEX (*cpxdef*), and the branching schemes from this section. In Table 3.12 we report the total number of vehicles and drivers (*v+d*) of the best feasible, the cpu time in seconds spent on the integer phase (*cpu_ip*) and the number of processed nodes (*nodes*). Additionally, we present the number of problems where the MIP solver found the optimal solution

over the current column set (*#opt*), could improve the best known upper bound (*#impr*), and could not improve the upper bound (*#nimpr*).

type	v+d	cpu_ip	nodes	#opt	#impr	#nimpr
seq	36.0	1,342	–	–	–	–
adap	35.6	1,273	–	–	–	–
cpxdef	36.6	3,096	747	2	2	1
fo-flf_fix	35.4	2,097	454	4	1	0
fo-fmf_fix	35.8	2,414	298	2	3	0
fo-flf	36.0	2,479	575	3	2	0
fo-fmf	36.6	2,698	297	2	3	0
ps	37.6	2,801	394	2	2	1
pd	37.6	3,600	2,873	0	4	1
pv	38.8	2,607	266	2	0	3

Table 3.12.: Results of user-defined branching rules and sequential approaches on model MDVCSP over five instances with 100 trips

The test results indicate that follow-on branching performs much better than using branching priorities. Furthermore, selecting the least fractional follow-on gives better results than choosing the most fractional follow-on for branching. The solution quality can be improved by fixing perfect follow-ons no matter what follow-on scheme is applied. Using CPLEX with its default settings performs better than setting branching priorities, but returns worse results compared to follow-on branching. The adaptive approach gives slightly worse results than the best follow-on variant, but consumes only 60% of its cpu time. Therefore, we conclude that the sequential and adaptive approach are basically more suited to generate integer solutions for model MDVCSP.

3.3.3. Heuristic Branch-and-Price

In this section we propose a novel extension of the sequential approach for the integer phase described in Section 3.3.1. In the sequential approach, we have solved the Lagrangian dual problem where the Lagrangian subproblems correspond to a multiple-depot vehicle scheduling problem and a trivial problem for crew scheduling, respectively. However, in the lower bound phase, we have replaced the equality signs of the linking constraints (2.14) by greater or equal signs. The corresponding Lagrangian multipliers are restricted in sign in the

lower bound phase while they are not in the integer phase. The underlying assumption was that the Lagrangian multipliers that we have found in the lower bound phase are a good approximation of the multipliers required in the integer phase. However, it is an open question whether the columns generated in the lower bound phase are also suitable to obtain good multipliers (and vehicle schedules) in the integer phase. Therefore, we propose to perform multiple iterations in the integer phase where we generate new columns in each iteration. Furthermore, our computational experiments revealed that it is beneficial to fix parts of the vehicle scheduling problem in each iteration of the integer phase.

Our method can be understood as a *heuristic branch-and-price* procedure. In an *exact* branch-and-price approach, the linear relaxation of the root node of the branch-and-bound tree is solved with column generation to optimality. If the solution of the continuous relaxation is not integer, two subproblems are created (branching), one of unprocessed subproblems (nodes) is selected, and the linear relaxation of that node is solved with column generation to optimality. The process iterates until the optimal solution is found or the complete search tree has been investigated. In our method, however, we do not solve the associated linear relaxation of each node to optimality. Instead, we perform only one column generation iteration after a subproblem was selected. Notice that the pricing problem can be solved as described in Sections 2.4.2 and 3.2, respectively. Furthermore, we perform multiple fixings decisions in each node of the search tree in order to (heuristically) reduce the search space. However, if the increase in the lower bound is too large, we reverse fixing decisions, i.e., we perform simple backtracking. The search tree is traversed in a depth-first manner. This type of branch-and-price approach is often referred to as *fix-and-price* procedure. In Algorithm 9 we give an overview of our approach as stated above. If backtracking is performed in step 3, we must guarantee that different fixings are performed in the subsequent branching step to prevent cycling.

In step 4 of our method we fix parts of the vehicle schedule, i.e., a subset of the flow variables of model MDVCSP. Recall that we apply a subgradient algorithm to approximate the values of the dual variables for the current set of columns. Therefore, we do not have primal information in order to separate the current fractional solution. Of course, there are other methods to solve the Lagrangian dual that also provide primal information, e.g., bundle methods (see [Kiwiel, 1995]) or the volume algorithm (see [Barahona and Anbil, 2000]). [Borndörfer et al., 2004] propose to use an inexact adaptation of a proximal bundle method in a fix-and-price (branch-and-generate) framework. We will compare their approach with our method on a set of randomly generated instances in Section 3.5. Furthermore, we tested the volume algorithm. However, early computational ex-

Algorithm 9: Heuristic branch-and-price approach for model MDVCSP

(Step 1) **Initialization**
Select initial column set from lower bound phase N^0.
Define maximum allowed increase of lower bound ϵ.
Set $t = 0$.

(Step 2) **Perform column generation**
Solve Lagrangian dual $l^t = \max_{\mu^t} \Phi'(\mu^t)$ with the current set of columns N^t.
Solve pricing problem $\bar{f}^*(\mu^t)$ and obtain columns $N' \setminus N^t$ with negative reduced costs.
Set $N^{t+1} = (N' \cup N^t)$ and $t = t + 1$.

(Step 3) **Backtracking**
If $l^t - l^{t-1} \geq \epsilon$ reverse fixing decisions of last iteration and return to step 2.

(Step 4) **Branching**
Terminate if all flow variables are fixed.
Fix subset of flow variables.
Set $t = t + 1$ and return to step 2.

periments revealed that the primal information of the volume algorithm lead to worse fixings than the method we describe in the following.

Basically, branching decisions must not destroy the structure of the column generation subproblem. To illustrate this consider a branching scheme based on dichotomy of variables x_k^d. In one branch we fix a duty into the solution ($x_k^d = 1$) while we ban it from the solution ($x_k^d = 0$) in the other branch. Observe that fixing a duty variable into the solution of one depot implies to ban all duties from other depots covering service trips of the fixed duty. However, banning a specific duty from the solution is difficult since we must forbid the specific path from being generated in the pricing problem of the corresponding depot. To prevent "zero" columns from being re-generated significantly complicates the pricing problem (see [Lübbecke and Desrosiers, 2005] and references therein). Branching schemes (fixing decisions) are said to be *compatible* if they do not considerably complicate the pricing problem. In terms of column generation, branching schemes based on the original variables of the compact formulation are compatible (see Section 1.5.2 and [Lübbecke and Desrosiers, 2005]). In our case, the original variables correspond to flow variables y_{ij}^d.

[Holmberg and Yuan, 2000] proposed to perform fixings based on the solution

of the Lagrangian subproblems for a capacitated network design problem. The authors also use a subgradient method to solve the corresponding Lagrangian dual and apply two different fixing schemes denoted by α- and β-fixing, respectively. In the remainder of this section, we propose to use a novel adaptation of α- and β-fixing for model MDVCSP. In particular, we fix flow variables to a depot that either correspond to service trips or connections between service trips (follow-ons). We will conclude this section with computational results concerning our fixing schemes.

Fixing service trips to depots

In this subsection, we will describe how service trips can be fixed to depots. To assign service trips to depots appears to be reasonable since the complexity of the Lagrangian subproblem for vehicle scheduling $\Phi'_y(\mu)$ can be reduced. When all trips are assigned to a depot, the vehicle scheduling subproblem can be solved in polynomial time while its multiple-depot counterpart is NP-hard (see Sections 1.1.1 and 1.4.1). In other words, the more trips are fixed to a depot the less CPU time should be consumed for the corresponding Lagrangian subproblem.

Fixing a service trip $t \in \mathcal{T}$ to a depot $d_i \in \mathcal{D}$ involves the following modifications in the Lagrangian subproblems and the pricing problem, respectively. In the vehicle scheduling subproblem we set $y_{ij}^{d_i} = 1, (i,j) \in A^{d_i}(t)$ or, alternatively, $y_{ij}^{d_i} = 0, (i,j) \in A^d(t), \forall d \in \mathcal{D}\backslash\{d_i\}$. Furthermore, we disable all duties k from the current set of duties $K^d(i,j)$ where $(i,j) \in A^d(t), \forall d \in \mathcal{D}\backslash\{d_i\}$. In the pricing problem, we must prevent duties that cover edge $(i,j) \in A^d(t), \forall d \in \mathcal{D}\backslash\{d_i\}$ from being regenerated. This can easily be done by setting $\mu_{ij}^d = -M$ for $(i,j) \in A^d(t), \forall d \in \mathcal{D}\backslash\{d_i\}$ where M corresponds to a sufficiently high value. Then, all duties covering that edge will not have negative reduced cost according to (3.27). In the following, we describe how we actually decide which trips to fix.

The basic idea of **α-fixing** is to fix those trips that often appear in the solution of the vehicle scheduling subproblem $\Phi'_y(\mu)$. If a flow variable y_{ij}^d is constantly set to one in vehicle scheduling subproblem, it indicates that arc (i,j) is likely to be included in the optimal solution. Likewise, arc (i,j) is probably not included if the Lagrangian subproblem suggests that the value of y_{ij}^d is zero. The straightforward approach would be to fix arcs to one that are part of all subproblem solutions and fix arcs to zero that are not used in any solution. However, the approach is more flexible by introducing parameter $\alpha \in [0, 0.5]$ to allow deviations. In particular, α corresponds to the deviation rate from the straightforward approach sketched

above. Our α-fixing scheme now reads

$$y_{ij}^d = \begin{cases} 1 & \text{if} \quad \sum_{l=1}^{L} y_{ij}^{d,(l)} \geq (1-\alpha) \cdot L \\ 0 & \text{if} \quad \sum_{l=1}^{L} y_{ij}^{d,(l)} \leq \alpha \cdot L \end{cases} \tag{3.38}$$

where $y_{ij}^{d,(l)}$ corresponds to the value of y_{ij}^d in the l-th (subgradient) iteration and L is equal to the number of subgradient iterations performed in that column generation iteration. Obviously, if $\alpha = 0$ we only fix arcs that have the same value in all subgradient iterations.

β-fixing is based on the reduced cost of the flow variables instead of the solutions of the Lagrangian subproblems. Notice that there is a relation between the reduced cost of the flow variables and the solution of the subproblems. Flow variables with high negative reduced costs $\bar{c}_{ij}^d$ are more likely to be selected for the subproblem solution than variables with positive reduced costs. Parameter $\beta \in [0,1]$ defines the ratio of service trips to be fixed to one in one column generation iteration. The number of arcs to be fixed is equal to $\lceil \beta |\mathcal{T}^*| \rceil$ where $\mathcal{T}^* \subseteq \mathcal{T}$ denotes the set of unfixed service trips. Notice that we cannot predict the number of fixed arc when α-fixing is applied. Furthermore, we cumulate the reduced costs of all arcs $(i,j) \in A^d(t), \forall t \in \mathcal{T}, \forall d \in \mathcal{D}$ in the following way:

$$R_{ij}^d = \begin{cases} \bar{c}_{ij}^d & \text{in the first iteration,} \\ \gamma \cdot R_{ij}^d + \bar{c}_{ij}^d & \text{if the lower bound improved.} \end{cases} \tag{3.39}$$

Similar to [Holmberg and Yuan, 2000] we consider reduced costs associated with an improved lower bound to be more reliable and, thus, should have a higher impact (i.e. $\gamma \in [0,1]$). In our implementation, we set $\gamma = 0.5$. Finally, we fix $\lceil \beta |\mathcal{T}^*| \rceil$ arcs that have the smallest (cumulated) reduced costs. However, we only fix a service trip to a particular depot if there is a significant difference to any other depot.

Fixing follow-ons to depots

In the preceding section, we proposed to perform fixing based on information from the vehicle scheduling subproblem while we will concentrate on the crew scheduling subproblem in this section. In particular, we fix variables based on follow-ons (see Section 3.3.2) that appear in the crew scheduling solution of the Lagrangian subproblem. As we will see in Section 3.5 our solutions almost always have the minimum number of vehicles while there is room for improvement concerning the number of duties. Furthermore, it is easier to construct a feasible vehicle schedule when a subset of trips is fixed than to obtain a feasible crew

schedule. Therefore, we hope that variable fixing based on information from crew scheduling solutions will overall improve the solution quality.

Recall that we refer to the pair of trips arcs (y^d_{ij}, y^d_{rs}) with $y^d_{ij}, y^d_{rs} \in A^d_T$ as *follow-on* if the corresponding service trips are operated consecutively. Moreover, $K^d(y^d_{ij}, y^d_{rs})$ denotes the set of duties where both trips are covered consecutively. In the following, we only consider follow-ons that occur on the same vehicle and are serviced by the same crew. In other words, follow-ons must not have a crew break in between since a changeover may occur between two pieces of work. This assumption is necessary to maintain tractability.

Fixing a follow-on $(y^{d_i}_{ij}, y^{d_i}_{rs})$ to a depot $d_i \in \mathcal{D}$ involves the following modifications in the Lagrangian subproblems and the pricing problem, respectively. Clearly, fixing a follow-on to a depot implies to assign each trip of the pair to the depot. Thus, all modifications of the preceding subsection also apply here.

Additionally, we modify the vehicle scheduling subproblem $\Phi'_y(\mu)$ to guarantee that both arcs are connected (operated by the same vehicle). Recall that in a time-space network we only have connections between groups of trips (see Section 2.3). As a consequence, we cannot directly fix a connection between two trips. However, we can enforce a flow between arcs $y^{d_i}_{ij}$ and $y^{d_i}_{rs}$ by modifying the minimum capacity $l^{d_i}_{ij}$ of the shortest path between the arcs. In each subgradient iteration, we compute the shortest path between the end node of $y^{d_i}_{ij}$ and the start node of $y^{d_i}_{rs}$. Then, we increase the minimum capacity of all arcs on the shortest path by one flow unit. As a result, we can always decompose the flow solution of $\Phi'_y(\mu)$ into a set of paths where at least one path covers both trips consecutively. The reasoning still holds if different follow-ons require the same connection arc. In such a case, we simply set the minimum capacity of the connection arc to the number of follow-ons that require the arc. Furthermore, we disable all duties $k \in K^{d_i}$ from the current set of duties with

$$(K^{d_i}(i,j) \cup K^{d_i}(r,s)) \backslash K^{d_i}(y^{d_i}_{ij}, y^{d_i}_{rs}). \tag{3.40}$$

In the column generation pricing problem, we must guarantee that only duties can be constructed for depot d_i that either contain follow-on $(y^{d_i}_{ij}, y^{d_i}_{rs})$ or none of the follow-on arcs. Since we assumed that only follow-ons within a piece of work are considered, it suffices to adapt the piece generation phase (see Section 2.4.2). Pieces of work for depot d_i must operate the trips either consecutively or not at all. Figure 3.7 shows a sample piece generation network to illustrate how trips can be connected. In our sample, we have four service trips denoted by $t_1, \ldots, t_4$ that start at station A, B, and C. Clearly, the shortest path between node i and j is to use trip arcs t_2 and t_4 with cost 30. However, if t_2 is part of another follow-on it must not be connected with trip t_4 and the shortest path

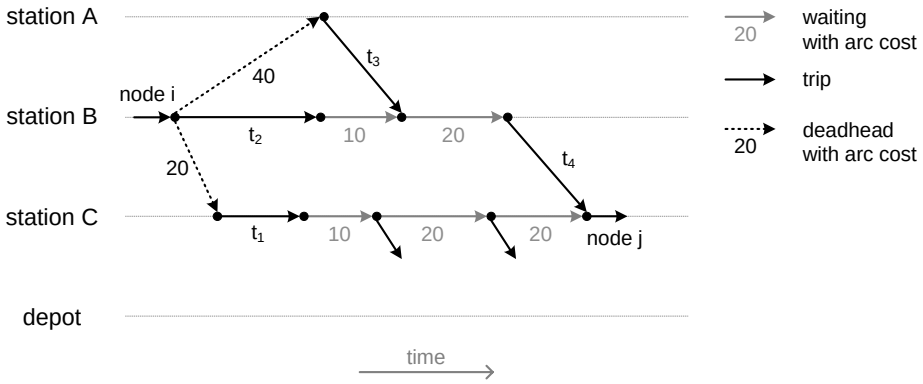

Figure 3.7.: Sample piece generation network for follow-on fixing

then includes t_3 and t_4 with cost 60. As a consequence, it does not suffice to use a simple label correcting algorithm to compute the shortest path between all compatible pairs of nodes where only cost are considered. Similar to the dynamic programming algorithms described in Section 3.2.1, a label must also include the last service trip traversed. Furthermore, a label may only be corrected if the new label has lower cost and the last service trip of the old label equals that of the new one. In our example, we have two labels at the start node of t_4: $(30, t_2)$ and $(60, t_3)$. At that node, we check whether t_2 is a valid predecessor of t_4 and finally chose label $(60, t_3)$ for propagation. Likewise we have two labels at the end node of t_4 for further propagation: $(60, t_4)$ and $(70, t_1)$.

Similar to the preceding subsection, we apply α**-fixing** to decide which follow-ons to fix. We fix follow-on (y^d_{ij}, y^d_{rs}) to 1 if

$$\sum_{l=1}^{L} f^{(l)}(y^d_{ij}, y^d_{rs}) \geq (1 - \alpha) \cdot L \tag{3.41}$$

where $f^{(l)}$ corresponds to the support of the follow-on in the l-th subgradient iteration. We do not fix follow-ons to 0 since this lead to poor results in our experiments. Out of the same reason we also do not perform β**-fixing**.

Computational results

In Table 3.13 we compare the performance of different fixing schemes in our heuristic branch-and-price framework. *FO* refers to follow-on fixing while *ST* denotes service trip fixing schemes with either α- or β-fixing. Finally, we report results when no fixings are made (*IPCG5*). For all settings, we perform five column generation iterations in the integer phase. We use the same test set and

settings as described in Section 3.3.1 and use the adaptive method as *reference*. Furthermore, we try to find an upper bound in the lower bound phase if more than 200 subgradient iterations have been performed in a column generation iteration. In that case, we solve a multiple-depot vehicle scheduling problem using the best multipliers of that iteration as costs on the arcs. We disable backtracking in order to assess the quality of the fixings.

In Table 3.13 we give the cpu time in seconds spent on the integer phase (*cpu_ip*) for each approach. Furthermore, we provide the last lower bound (*lb*) and the total number of vehicles and drivers (*v+d*) of the best feasible solution. Notice that the last lower bound obtained is not necessarily a valid lower bound.

type		#trips					
		080	100	160	200	320	400
reference	cpu_ip	0	578	1,676	2,621	7,635	9,808
	lb	29,664	37,834	48,382	63,470	77,191	107,922
	v+d	26.7	35.0	45.3	60.7	76.0	105.0
IPCG5	cpu_ip	374	442	2,840	5,212	16,802	25,909
	lb	29,605	37,839	48,273	63,485	76,964	107,877
	v+d	26.7	34.3	45.7	60.0	74.3	104.0
$FO_{\alpha=0.01}$	cpu_ip	215	359	2,009	3,658	10,257	18,256
	lb	30,254	39,269	50,892	67,547	82,894	112,359
	v+d	27.0	35.1	45.7	60.8	74.9	105.8
$ST_{\alpha=0.15}$	cpu_ip	286	388	2,073	3,174	16,043	22,413
	lb	29,634	37,962	48,274	63,429	77,071	107,955
	v+d	26.7	34.3	45.7	60.0	74.3	104.0
$ST_{\alpha=0.1}$	cpu_ip	328	432	2,337	4,075	15,298	26,582
	lb	29,578	37,887	48,266	63,573	77,133	107,856
	v+d	27.0	34.7	45.7	60.0	74.3	104.7
$ST_{\beta=0.12}$	cpu_ip	206	299	2,119	3,489	11,580	16,374
	lb	30,448	39,353	51,430	68,046	81,659	113,902
	v+d	27.0	35.0	45.7	60.7	75.0	106.0
$ST_{\beta=0.1}$	cpu_ip	216	335	2,389	3,892	12,050	17,332
	lb	30,198	39,174	50,767	66,440	81,966	113,844
	v+d	27.0	35.0	45.7	60.7	75.0	106.0

Table 3.13.: Results of heuristic branch-and-price algorithms

As expected, the reference (sequential method) requires less computational time than the other approaches since there is no column generation in the integer phase. Notice that no integer phase was performed for the group with 80

trips in the reference method since the best upper bound obtained in the lower bound phase already had the minimum number of duties. Both α-follow-on fixings and β-service trip fixings reduce the computational burden compared to the method without fixings. Unfortunately, there is a remarkable increase in the lower bound as well. Hence, we conclude that those fixing methods are inappropriate for our problem since the total number of vehicles and drivers also increased. Furthermore, the results show that α-service trip fixing does neither worsen the computational time nor the solution quality or the lower bound. Instead, solution times are better than without fixings when $\alpha = 0.15$ is set. To sum up, we infer that column generation in the integer phase improves (with the exception of the 160 trips group) the solution quality but increases the computational burden. Finally, (bold) α-fixings of service trips allow to fix parts of the problem without increasing the lower bound and without worsening the solution quality.

3.4. Integrated Planning with Unrestricted Changeovers

In Section 2.1 we assumed that each crew is assigned to a depot and may only conduct tasks on vehicles from this particular depot. Furthermore, we assumed that a driver may only change his vehicle during a break, i.e., between two pieces of work. In other words, a *changeover* is only allowed between vehicles from the same depot and the driver must take a break after leaving his vehicle. In this section, however, a driver may change between two vehicles of different depots whenever there is a relief point (no matter if he takes a break or not). In the following, we say that changeovers are *unrestricted* while in the original setting changeovers were *restricted*. Similar assumptions were (implicitly) used by [Mesquita and Paias, 2006]. However, in their setting, drivers may additionally walk on deadhead connections that are not part of the vehicle schedule. Of course, this is not very realistic since this assumes that a driver can deadhead by foot within the same time as a driver does by bus. Therefore, in our setting, we will not allow drivers to use deadheads that are not part of the vehicle schedule (except at the beginning and end of a duty - see Section 2.1).

In the following, we propose a novel modification of model MDVCSP that allows unrestricted changeovers as described above. In model MDVCSP, the duty variables and linking constraints are separated by depot. However, if we allow unrestricted changeovers drivers may use vehicles from all depots. As a consequence, duty variables and linking constraints do not need to be separated by depot. In addition to the notation used in Section 2.3 we define K as the

set of duties, $x_k, k \in K$ as binary duty variables, and $\widetilde{A} = \bigcup_{d\in\mathcal{D}} \widetilde{A}^d$ as the set of arcs that require both vehicle and crew activities. Without loss of generality, we assume that the nodes of the $|\mathcal{D}|$ vehicle scheduling networks are numbered in such a way that nodes with the same time and location have the same index in all networks. We consider two arcs to be equal if they have the same start and end index as well as the same type (see Figure 2.3). As a consequence, $|\widetilde{A}| \leq \sum_{d\in\mathcal{D}} |\widetilde{A}^d|$ holds. The model that allows unrestricted changeovers can be stated as follows (MDVCSP-C).

$$\sum_{d\in\mathcal{D}} \sum_{(i,j)\in A^d} y_{ij}^d c_{ij}^d + \sum_{k\in K} x_k f_k \quad \rightarrow \quad \min \tag{3.42}$$

$$s.t. \qquad \sum_{d\in\mathcal{D}} \sum_{(i,j)\in A^d(t)} y_{ij}^d \quad = \quad 1 \qquad \forall t \in \mathcal{T} \tag{3.43}$$

$$\sum_{\{j:(j,i)\in A^d\}} y_{ji}^d - \sum_{\{j:(i,j)\in A^d\}} y_{ij}^d \quad = \quad 0 \qquad \forall d \in \mathcal{D}, \forall i \in N^d \tag{3.44}$$

$$\sum_{k\in K(i,j)} x_k - \sum_{d\in\mathcal{D}} y_{ij}^d \quad = \quad 0 \qquad \forall (i,j) \in \widetilde{A} \tag{3.45}$$

$$0 \leq y_{ij}^d \leq u_{ij}^d, y_{ij}^d \quad \in \quad \mathbb{N} \qquad \forall d \in \mathcal{D}, \forall (i,j) \in A^d \tag{3.46}$$

$$x_k \quad \in \quad \{0,1\} \qquad \forall k \in K \tag{3.47}$$

The objective (3.42) minimizes the sum of vehicle and crew costs. Constraints (3.43)-(3.44) correspond to a multicommodity flow formulation for the vehicle scheduling problem where the set of trip tasks must be partitioned among the depots (3.43) and flow conservation is ensured for each depot (3.44). Constraint set (3.45) establishes the link between vehicle and crew schedule: each arc covered by a vehicle/vehicles must also be covered by the same number of duties. Constraints (3.46) guarantee that the maximum capacity of the flow variables is satisfied. If a driver may walk on deadhead connections that are not part of the vehicle schedule, we replace the equality signs of the linking constraints (3.45) by greater or equal signs. Similar to our solution approach for model MDVCSP we apply column generation in combination with Lagrangian relaxation to solve model MDVCSP-C.

The Master Problem

Basically, we relax the same constraints in a Lagrangian way as stated in Section 2.4.1. We associate Lagrangian multipliers μ_{ij} and π_t with constraints 3.45 and 3.43, respectively. The Lagrangian subproblem

$$\Phi(\mu,\pi) = \Phi_y(\mu,\pi) + \Phi_x(\mu) + \sum_{t\in\mathcal{T}} \pi_t \tag{3.48}$$

remains unchanged except the definition of the reduced cost. Notice that the vehicle scheduling subproblem $\Phi_y(\mu, \pi)$ still constitutes a single depot vehicle scheduling problem for each depot. The reduced cost $\bar{c}^d_{ij}$ for arc $(i,j) \in A^d$ of the vehicle scheduling network of depot $d \in \mathcal{D}$ is defined as

$$\bar{c}^d_{ij} = \begin{cases} c^d_{ij} + \mu_{ij} - \pi_t & \text{for } (i,j) \in \widetilde{A} \text{ and } \exists t \in \mathcal{T} : (i,j) \in A^d(t) \\ c^d_{ij} + \mu_{ij} & \text{for } (i,j) \in \widetilde{A} \text{ and } \nexists t \in \mathcal{T} : (i,j) \in A^d(t) \\ c^d_{ij} & \text{for } (i,j) \notin \widetilde{A} \end{cases} \tag{3.49}$$

while

$$\bar{f}_k = f_k - \sum_{(i,j) \in \widetilde{A}(k)} \mu_{ij} \tag{3.50}$$

denotes the reduced cost of duty $k \in K$ where $\widetilde{A}(k) \subseteq \widetilde{A}$ corresponds to the set of arcs that is covered by duty k.

The Pricing Problem

After the restricted master problem is solved, the dual information of the solution is used to price out new columns with negative reduced costs. As described in Section 2.4.2 we use a two phase pricing procedure. In the first phase, we set up a piece generation network to generate a set of pieces of work. These pieces serve as input for the second phase where duties are generated (see 3.1.2). However, the column generation pricing problem differs from the original version in the way the pieces of work are generated.

Clearly, we no longer have a separate pricing problem for each depot since in model MDVCSP-C the duty variables are not separated by depot. Consequently, we set up a single piece generation network $\bar{G}_c = (\bar{N}_c, \widetilde{A})$. In particular, $\bar{G}_c$ is an acyclic directed time-space network where the set of arcs $\widetilde{A}$ corresponds to all activities that require both vehicle and crew. Note that we assumed that each trip has exactly two relief points: one at the beginning and the other at the end of the trip. Thus, each node in $\bar{N}_c \subset \bigcup_{d \in \mathcal{D}} N^d$ corresponds to a relief point.

Let g_{ij} be the crew cost associated with arc $(i,j) \in \widetilde{A}$. The reduced cost of arc $(i,j) \in \widetilde{A}$ is then defined as $\bar{g}_{ij} = g_{ij} - \mu_{ij}$ where μ_{ij} are the multipliers associated with linking constraints (3.45) that represent trip, deadhead, or waiting arcs outside the depot. Recall that each path corresponds to a piece of work. Hence, the reduced cost of a path is equal to the reduced cost of the associated piece of work. We compute the shortest path between each pair of nodes that meet the (piece) duration constraint. Furthermore, we consider additional pieces for each

path and for each depot: we add a pull-in trip at the beginning, a pull-out trip at the end, and for each depot combination both. Notice, however, that a driver does not necessarily remain on the same vehicle for the duration of the piece of work. Clearly, this requires to adapt the initial definition of a piece of work (see Section 2.1) where the driver stays with the vehicle for the entire piece.

Computational Results

In Table 3.14 we compare our result on model MDVCSP-C with unrestricted changeovers (*unrestricted*) with those of model MDVCSP (*reference*). We use the same test set and settings as described in Section 3.3.1 and use the adaptive method as reference. In Table 3.14 we give the number of iterations (*#iter*), the cpu time in seconds spent on the master problem (*cpu_ma*), on pricing problem (*cpu_pr*), on the integer phase (*cpu_ip*), and the total cpu time (*cpu_tot*). The cpu time of the master problem includes the time to find new upper bounds in the lower bound phase. Furthermore, we provide the best lower bound obtained (*lb*) and the total number of vehicles and drivers (*v+d*) of the best feasible solution. Notice that the best lower bound obtained is not necessarily a valid lower bound.

type		#trips					
		080	100	160	200	320	400
reference	#iter	18.0	19.0	23.3	24.7	35.3	32.3
	cpu_ma	153	222	1,170	1,712	7,560	8,318
	cpu_pr	11	32	302	202	5,477	8,120
	cpu_ip	0	578	1,676	2,621	7,635	9,808
	cpu_tot	165	834	3,155	4,541	20,794	26,384
	lb	29,664	37,834	48,382	63,470	77,191	107,922
	v+d	26.7	35.0	45.3	60.7	76.0	105.0
unrestricted	#iter	25.7	29.3	38.0	37.7	50.7	46.7
	cpu_ma	138	215	596	754	2,039	2,688
	cpu_pr	9	20	161	172	1,370	2,037
	cpu_ip	60	136	482	579	6,659	8,386
	cpu_tot	208	372	1,243	1,513	10,338	13,455
	lb	29,573	37,884	48,195	63,177	76,433	106,889
	v+d	27.0	35.0	45.0	59.7	73.7	103.3

Table 3.14.: Results of integrated planning with restricted and unrestricted changeovers

As we can see in Table 3.14 there is a considerable speed-up and a better solution quality if changeovers are not restricted. As expected, most of the time is saved in the lower bound phase. In fact, for large instances approximately 70% of the time is saved in the lower bound phase. Recall that model MDVCSP-C has fewer constraints and, thus, fewer Lagrangian multipliers. Furthermore, there is only one pricing problem per iteration in the unrestricted case while there is a separate pricing problem for each depot for model MDVCSP. Additionally, we obtained better solutions for instances with more than 80 trips if changeovers are not restricted. In Section 3.5.2, we will give additional results and compare our approach with other methods previously exposed in literature.

3.5. Computational Results

The purpose of this section is to summarize our computational results on real-world problem data from Connexxion and randomly generated instances. All experiments in this section were conducted on a Dell OptiPlex GX620 personal computer running Windows XP with an Intel Pentium IV 3.4 GHz processor and 2 GB of main memory. Our integrated method ICOPT is implemented in C# and has been compiled using the .NET framework version 2.0.50727.

We test our integrated approach on real-world instances from Connexxion which is the largest bus operator in the Netherlands. The instances have been kindly provided by the author of [Huisman, 2004]. The total set involves 1,104 trips and four depots and has been split into 8 smaller instances by [Huisman, 2004, Huisman et al., 2005a]. In the original setting, not all trips were allowed to be driven by a vehicle from every depot. The average number of depots a trip can be operated from was 1.71. However, we will consider a different setting where every trip may be serviced from every depot. Obviously, this makes the problems more difficult since the solution space is expanded.

Moreover, we use the randomly generated instances and settings that have also been used and described in the preceding sections. In order to ease the exposition we recall the basic properties. The instances available at [Huisman, 2003] have been classified into two classes according to the travel speed where the speed is lower for problems in class B. As a consequence, trips in class B are longer. However, we will only report computational results for class A that all involve 4 depots and groups with n trips where $n = 80, 100, 160, 200, 320$ and 400. For each group 10 instances are available. Furthermore, we generated 10 instances with 640 trips and four depots according to [Huisman, 2004, Huisman et al., 2005a]. The instances are available at the web page [Steinzen, 2007]. To the best of

our knowledge, randomly generated instances of that size have not been tackled before.

In accordance with [Huisman, 2004, Huisman et al., 2005a] we consider five different types of duties: one tripper type with one piece of work between 30 minutes and 5 hours, and four types consisting of two pieces of work. Each duty starts with a sign-on and ends with a sign-off. If the first (last) duty activity starts (ends) at the depot we impose a sign-on (sign-off) time of 10 and 5 minutes, respectively. If the duty starts (ends) at another relief point both sign-on and sign-off time increase to 15 minutes plus the deadhead time between the start (end) location and the depot. The time a driver spends on the vehicle is working time. The duty length corresponds to the total duty duration including all activities such as sign-on/off, pieces of work, and breaks. Table 3.15 summarizes the settings we use in this section.

	early		day		late		split	
Type	min	max	min	max	min	max	min	max
Start time			8:00		13:15			
End time		16:30		18:14				19:30
Piece length	0:30	5:00	0:30	5:00	0:30	5:00	0:30	5:00
Break length	0:45		0:45		0:45		1:30	
Duty length		9:45		9:45		9:45		12:00
Working time		9:00		9:00		9:00		9:00

Table 3.15.: Properties of different duty types

The objective is to minimize the total sum of vehicles and drivers. We assign a fixed cost of 1,000 for each vehicle and duty and a small variable cost of 1 for each minute a vehicle is outside the depot and 0.1 for each minute a crew is working. In other words, we minimize the total number of vehicles and drivers first and leave operational cost minimization as secondary objective. Furthermore, we used the following parameter settings.

1. We solved the pricing problem independently for each depot and duty type combination where we generated at most 5,000 duties for each combination.

2. The column generation algorithm is terminated if the improvement of the lower bound is less than 0.2% in the last 10 iterations. Notice that we do not have a valid lower bound unless

$$-\frac{\sum_{d\in\mathcal{D}}\sum_{k\in K_f^d}\bar{f}_k^d}{\max\Phi^f(\mu,\pi)} \leq 0.002 \tag{3.51}$$

is satisfied where K_f^d is the set of duties added in the final column generation iteration and where $\max \Phi^f(\mu, \pi)$ is the best lower bound. Furthermore, all duties with negative reduced must have been added in the final iteration in order to obtain a valid lower bound for the overall problem. However, our computational experiments indicated that terminating according to (3.51) led to higher computational times without improving the final integer solution.

3. The maximum number of iterations in the subgradient algorithm is 1,000 for every column generation iteration.

4. We limit the computational time for the lower bound phase to 21,600 seconds (6 hours).

This section is organized as follows. In Section 3.5.1 we give computational results for the integrated approach on real-world data instances. In Section 3.5.2, we also report results of our method on randomly generated benchmark instances. We present results for the case with restricted and unrestricted changeovers.

3.5.1. Real-world Data Instances

In Table 3.16 we give computational results of our integrated approach on Connexxion data instances. For each of the 8 instances we report the number of trips and the number of depots where each trip can be serviced from every depot. Notice that this setting is different to the results published in [Huisman, 2004, Huisman et al., 2005a] since the solution space is expanded. As a consequence, the results cannot be directly compared. However, the deadhead matrix remains unchanged and is such that some connections to and from depots are not allowed. Furthermore, we give the number of iterations (*#iter*), the CPU time in seconds spent on the master (*cpu_ma*), the pricing problem (*cpu_pr*), the integer phase (*cpu_ip*), and the total time including all initializations (*cpu_tot*). Additionally, we report the final upper bound (*ub*), the number of vehicles (*vehicles*), drivers (*drivers*), total number of vehicles and drivers for integrated (*v+d*) and sequential vehicle and crew scheduling (*v+d seq.*). Finally, we give the total number of duties and drivers obtained by [Huisman et al., 2005a] (*v+d ref.*). Recall that their results cannot be directly compared with ours since, in their setting, not all trips can be serviced from each depot.

The results show that real-world instances with up to 653 trips and 4 depots can be solved. Furthermore, there is an efficiency gain compared to sequential planning. Observe that the computational time does not always increase with the

	instance							
	1	2	3	4	5	6	7	8
#trips	194	210	220	237	304	386	451	653
#depots	4	4	4	4	4	4	4	4
#iter	38	25	37	27	30	52	48	62
cpu_ma	965	572	1,008	741	1,204	2,932	3,468	7,251
cpu_pr	1,358	118	729	157	483	10,936	1,763	17,058
cpu_ip	964	869	634	859	2,062	17,282	8,097	31,648
cpu_tot	3,329	1,566	2,379	1,762	3,763	31,195	13,388	56,097
ub	48,905	87,978	64,803	93,650	111,034	89,459	118,107	189,185
vehicles	19	33	28	34	40	32	47	67
drivers	26	48	32	52	62	51	62	108
v+d	45	81	60	86	102	83	109	175
v+d seq.	52	89	77	95	114	92	138	191
v+d ref.	49	83	68	89	105	90	124	184

Table 3.16.: Results on Connexxion data instances

problem size. The difference can be understood in the following way. The structure of the problems are different since the average trip lengths are comparatively long for instances 2,4, and 5. For these instances, the average number of trips per duty is less than 5 while for all other instances there are at least 6 trips per duty. Therefore, we conclude that our algorithm performs better if the density of the columns is small. Similar results were obtained by [Oukil et al., 2007]. The authors report a strong impact of column density on the computational burden of a column generation algorithm for a multiple-depot vehicle scheduling problem. Finally, recall that for our approach the number of vehicles is always minimal, i.e., equals the number of vehicles when sequential planning is performed.

3.5.2. Randomly Generated Data Instances

In this section, we give computational results of our solution approach for randomly generated data instances. We will first concentrate on the setting where changeovers are restricted. In the second part we focus on the case with unrestricted changeovers.

Restricted Changeovers

In Table 3.17 we report results of our integrated approach on the Huisman data instances type A with 80 to 640 trips and 4 depots. Changeovers are restricted

as stated in Section 2.1. Each trip can be operated from every depot. We use the same parameter settings for each group of 10 instances. The structure of Table 3.17 is similar to Table 3.16.

	#trips						
	080	100	160	200	320	400	640
#iter	20.5	22.0	25.8	28.2	34.7	33.0	46.3
cpu_ma	153	234	528	754	1,825	2,330	5,207
cpu_pr	11	24	228	651	1,593	2,074	9,007
cpu_ip	70	110	819	1,318	10,877	15,772	42,588
cpu_tot	235	369	1,579	2,730	14,325	20,320	57,235
ub	31,338	37,347	52,165	62,568	89,084	109,750	190,488
vehicles	9.2	11.0	14.8	18.4	26.7	32.9	59.9
drivers	19.1	22.7	31.8	38.8	55.8	67.9	120.4
v+d	28.2	33.7	46.6	57.2	82.5	100.8	177.3
v+d seq.	35.3	41.3	53.6	63.9	89.8	108.8	211.3

Table 3.17.: Detailed results on Huisman data instances type A with four depots and restricted changeovers

Similar to our results on real-world problem instances, the total number of vehicles and drivers can be remarkably reduced if integrated planning is performed. For instances with more than 200 trips, the CPU time spent in the integer phase jumps up since we use our heuristic branch-and-price method (see Section 3.3.3). However, we only perform three column generation iterations in the integer phase. We also tested our method with more than three iterations, but the strong increase in solution time did not justify the improvement of the final solution.

In Table 3.18 we compare our results with results from literature. In particular, we summarize the results of our implementation from Table 3.17 (*ICOPT*) and give results of [Gintner et al., 2006b] (*GSS05*), [Borndörfer et al., 2004] (*BLW04*), and [Huisman et al., 2005a] (*HFW05*). For each group of instances, we report the total number of vehicles and drivers (*v+d*) and the total computational time (*cpu_tot*) in seconds. Notice that we do not give the computational time for [Huisman et al., 2005a] since it cannot be compared with the other approaches. Table 3.18 shows that our approach clearly outperforms all other approaches from literature in terms of solution quality and solution time. Furthermore, we have so far tackled the largest instances with 4 or more depots.

[Gintner et al., 2006b] also use model MDVCSP while the other approaches

rely on model MDVCSP-H (see Section 2.2.3). As can be seen from Table 3.18 approaches based on model MDVCSP are beneficial compared to the classic connection-based model MDVCSP-H. As stated earlier our implementation also relies on model MDVCSP, but uses a different pricing scheme and a modified integer phase compared to [Gintner et al., 2006b]. The modifications we proposed in this chapter improve the results of [Gintner et al., 2006b]: the solution quality is better for all groups and the computational time can be reduced.

source		#trips						
		080	100	160	200	320	400	640
ICOPT	cpu_tot[1]	235	369	1,579	2,700	14,325	20,320	57,235
	v+d	28.2	33.7	46.6	57.2	82.5	100.8	177.3
GSS05	cpu_tot[2]	420	660	1,620	3,300	–	–	–
	v+d	29.3	35.0	47.4	58.3	–	–	–
BLW04	cpu_tot[3]	780	1,260	2,640	6,360	19,680	43,200	–
	v+d	29.6	35.7	47.7	59.0	82.8	102.0	–
HFW05	cpu_tot[4]	–	–	–	–	–	–	–
	v+d	29.6	36.2	48.9	60.0	–	–	–

[1,2] on Dell OptiPlex GX620 with Intel Pentium IV 3.4 GHz/2GB
[3] on Dell Precision 650 PC with Intel Dual Xeon 3.0 GHz/4GB
[4] CPU times not comparable

Table 3.18.: Comparison on Huisman data instances type A with four depots and restricted changeovers

Figure 3.8 illustrates that our approach requires between 29 and 73 percent of the computational time of [Borndörfer et al., 2004]. Furthermore, the results indicate that ICOPT is the overall fastest known method for integrated vehicle and crew scheduling problems under the assumptions stated in [Huisman, 2004].

Unrestricted Changeovers

In Table 3.19 we report results of our integrated approach on the Huisman data instances type A with 80 to 640 trips and 4 depots. Changeovers are unrestricted as described in Section 3.4. Each trip can be operated from every depot. We use the same parameter settings for each group of 10 instances. The structure of Table 3.19 is similar to Table 3.17.

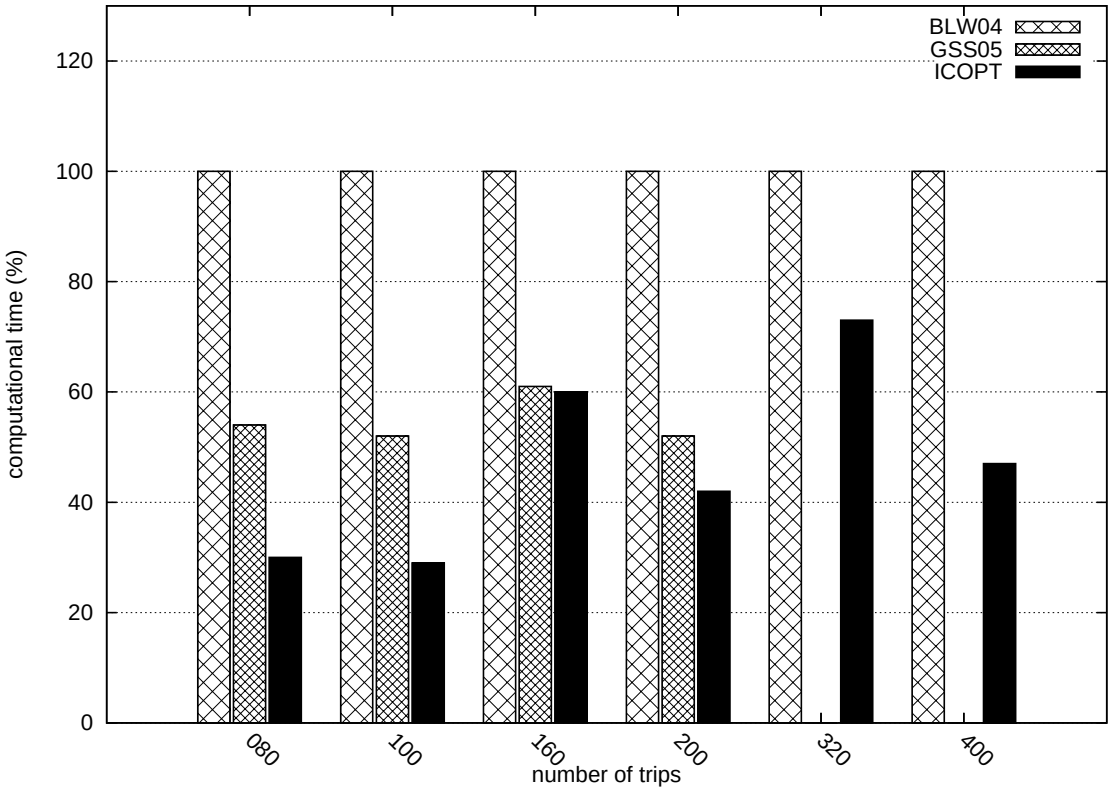

Figure 3.8.: Comparison of computational times in percent on Huisman data instances type A with four depots

	#trips						
	080	100	160	200	320	400	640
#iter	23.2	25.5	36.1	38.9	44.4	47.3	49.3
cpu_ma	126	182	529	789	1,735	2,717	4,116
cpu_pr	9	17	139	195	1,085	2,155	2,805
cpu_ip	62	82	747	996	5,427	6,789	15,724
cpu_tot	197	282	1,422	1,990	8,481	11,947	23,125
ub	31,636	37,347	50,725	61,477	88,394	108,201	190,182
vehicles	9.2	11.0	14.8	18.4	26.7	32.9	56.9
drivers	19.2	22.7	31.3	37.8	55.2	67.6	120.3
v+d	28.4	33.7	46.1	56.2	81.9	100.5	177.2
v+d seq.	34.0	38.9	51.1	61.5	88.0	105.5	184.3

Table 3.19.: Detailed results on Huisman data instances type A with four depots and unrestricted changeovers

Basically, the results show that there is an efficiency gain if vehicle and crew scheduling are treated in an integrated way. Similar to the restricted case, most of the time is spent in the integer phase. Except the 80 trips group the number of vehicles and drivers is smaller compared to our results with restricted changeovers. Thus, we conclude that model MDVCSP-C is computationally more attractive than model MDVCSP. Furthermore, we believe it is worthwhile for planners in practice to allow unrestricted changeovers since the additional flexibility results in efficiency gains.

In Table 3.20 we compare the results of our implementation (*ICOPT_C*) where unrestricted changeovers are allowed with the results of [Mesquita and Paias, 2006] (*MP06*). Recall from Section 2.2.3 that the authors allowed crews to walk on deadhead connections that were not part of the vehicle schedule. Due to the reasons stated in Section 3.4 we did not allow deadheading by foot which obviously reduces the solution space compared to [Mesquita and Paias, 2006]. The structure of Table 3.20 is similar to Table 3.18.

The results show that our approach outperforms the method of [Mesquita and Paias, 2006] in terms of computational time and/or solution quality. Only for the 80 trips group ICOPT_C consumes more time. Furthermore, we solved instances with 640 trips and 4 depots that, to the best of our knowledge, have not been tackled before. Finally, we would like to mention that we can basically compute a valid lower bound with our method while this is not possible for the method of [Mesquita and Paias, 2006] (since the set of tasks is heuristically defined).

source		#trips						
		080	100	160	200	320	400	640
ICOPT_C	cpu_tot[1]	197	282	1,422	1,990	8,481	11,947	23,125
	v+d	28.4	33.7	46.1	56.2	81.9	100.5	177.2
MP06	cpu_tot[2]	72	428	2,436	3,064	11,023	13,453	–
	v+d	28.7	35.5	46.1	56.9	82.5	101.8	–

[1] on Dell OptiPlex GX620 with Intel Pentium IV 3.4 GHz/2GB
[2] on PC with Intel Pentium IV 3.2 GHz

Table 3.20.: Comparison on Huisman data instances type A with four depots and unrestricted changeovers

3.6. Summary

In this chapter, we discussed solution approaches for the integrated multiple-depot vehicle and crew scheduling problem. More precisely, we addressed the column generation pricing problem and methods to construct integer solutions. Moreover, we proposed a novel model variation where drivers can use vehicles from all depots and can change their vehicles whenever possible.

The column generation pricing problem corresponds to a resource constrained shortest path problem. We proposed two novel network formulations for a decomposed pricing problem. Moreover, we showed that the network complexity of our models is superior to models previously exposed in literature. We applied dynamic programming to solve the resource constrained shortest path problems. We suggested novel, problem-specific reduction techniques that considerably sped up the solution process. Furthermore, we presented a combination of known and novel techniques to further improve the performance of the algorithm.

We discussed three methods to compute integer solutions: a Lagrangian heuristic (sequential approach), a branch-and-bound approach with novel branching schemes, and a novel heuristic branch-and-price algorithm (fix-and-optimize). Our computational tests revealed that the latter approach generates high-quality solutions while the sequential method finds good solutions in a short timeframe. Furthermore, we found the branch-and-bound method inappropriate to find good quality solutions for medium-sized instances in a reasonable timeframe.

We proposed a novel model variation where drivers are not tied to vehicles of a single depot and where vehicle changes may be performed during a piece of work.

We concluded this chapter with an extensive computational study on real-world and randomly generated benchmark instances. Our results indicate that

medium-sized with about 640 trips and 4 depots could be solved efficiently. In fact, our method outperformed other approaches previously exposed in literature in terms of solution quality and computational time. For well-known benchmark instances, we presented previously unknown solutions and were able to tackle the largest instances so far.

4. A Hybrid Evolutionary Algorithm

In this chapter, we present a novel hybrid evolutionary algorithm for the multiple-depot integrated vehicle and crew scheduling problem that combines mathematical programming techniques with an evolutionary algorithm. The algorithm is novel since an evolutionary algorithm has not been applied to integrated vehicle and crew scheduling problems before. This section is partly based on [Steinzen et al., 2007a].

Evolutionary algorithms (EA) are adaptive heuristic search methods that are based on the idea of evolutionary processes in nature. In evolutionary processes, populations evolve in accordance with the principle of natural selection or, in other words, the "survival of the fittest". Individuals that are successful in adapting to their environment have a better chance of surviving and reproducing than individuals with a worse fitness. As a result, genes from highly successful individuals will spread across the population from generation to generation. The combination of good genes from different individuals can yield even more fit offspring.

An EA simulates this processes by creating an initial population of *individuals* and applying genetic operators in each generation/reproduction. Each individual is represented by a string or *chromosome* and corresponds to a possible solution to the (combinatorial) optimization problem. The fitness of an individual represents the value of the objective function. Furthermore, individuals with a high fitness get the opportunity to reproduce among each other by exchanging genetic information. Algorithm 10 shows the basic steps of a simple evolutionary algorithm. For an extensive survey on evolutionary algorithms the reader is referred to [Bäck, 1996].

This chapter is organized as follows. In Section 4.1 we discuss a decomposition approach of model MDVCSP (see Section 2.3). The decomposition approach provides the basis for the hybrid evolutionary algorithm that we describe in Section 4.2. Finally, we compare the computational results of our evolutionary algorithm with other integrated approaches in Section 4.3.

Algorithm 10: Basic Evolutionary Algorithm

(Step 1) **Initialization**
Generate initial population and evaluate fitness of each individual.

(Step 2) **Evolutionary process**
repeat
  Select parents from population.
  Recombine genes of parents to produce children.
  Evaluate fitness of children.
  Replace (parts of the) population by children.
until *sufficiently good solution is found*

4.1. Problem Decomposition

Our solution approach decomposes the MDVCSP (see Section 2.3) into different subproblems as shown in Figure 4.1. First, we assign each trip $t \in \mathcal{T}$ to a depot $d \in \mathcal{D}$. Thus, we obtain a *trip-depot vector* $\theta \in \{1, \ldots, |\mathcal{D}|\}^{|\mathcal{T}|}$ where each trip is assigned to exactly one depot. In a second phase, we compute the optimal solution of each single depot vehicle scheduling problem and, afterwards, we solve a crew scheduling problem for each depot given the vehicle schedule for that depot. The main advantage of this decomposition is that the vehicle

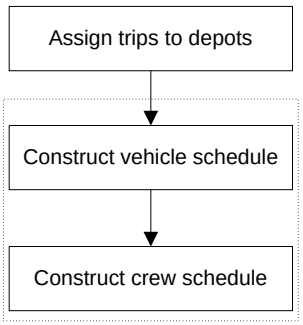

Figure 4.1.: Problem decomposition for evolutionary algorithm

scheduling problem with multiple depots is NP-hard unlike the single depot case that appears in our second phase.

In the second phase, we schedule vehicles independently of crews which corresponds to a traditional (sequential) approach to vehicle and crew scheduling. However, we also consider a partial integration similar to [Gintner et al., 2006a] where we allow to recombine parts of vehicle blocks in order to disclose addi-

tional flexibility in crew scheduling while preserving vehicle schedule optimality. The additional flexibility often results in vehicle schedules that allow better crew schedules (with less duties) compared to the sequential approach.

Furthermore, vehicle and crew scheduling can be considered in an integrated way for a given trip-depot vector. That is, we solve $|\mathcal{D}|$ integrated vehicle and crew scheduling problems with a single depot. Typically, the additional freedom in scheduling vehicles dependent on crews (and vice versa) leads to better solutions compared to the sequential or partially integrated method.

In summary, there are three different methods to construct a feasible vehicle and crew schedule for a given trip-depot vector: sequential, partially integrated, and fully integrated. Of course, there is a strong relationship between the level of integration and the computational time needed to solve the corresponding problems. The computational time increases with the level of integration. The evolutionary algorithm we propose in the next section is based on this decomposition approach where we first make a trip-depot assignment.

4.2. Components of Evolutionary Algorithm

Our evolutionary algorithm (EA) is based on a non-binary representation that is equal to the trip-depot vector θ from the previous section. A chromosome is a string of length equal to the number of trips where the i-th entry contains (the index of) the depot the i-th trip is assigned to. We use an evolutionary algorithm to find a good trip-depot assignment where the fitness of a chromosome (individual) is evaluated using mathematical programming techniques. In particular, we use the all-purpose MIP solver CPLEX [ILOG, 2006] and column generation in combination with Lagrangian relaxation.

The size of the EA search space with the non-binary representation is $|T|^{|D|}$. Notice that a feasible vehicle schedule can always be constructed from a given trip-depot vector since each trip can always be covered by a short vehicle block: pull-out trip - service trip - pull-in trip. Furthermore, duty constraints in practice are such that almost all vehicle blocks can be covered by a feasible crew duty. As a consequence, virtually all chromosomes represent feasible trip-depot assignments and, thus, we do not use a repair mechanism to transform infeasible to feasible solutions. However, if a trip-depot assignment θ does not yield both feasible vehicle and crew schedules, the individual will be discarded after the evaluation of the fitness function. Moreover, the search space can be reduced if the number of trips assigned to a depot must be greater or equal a lower limit $\rho < |T|$. Assigning a small number of trips to a depot often leads to inefficient vehicle and

crew schedules since many deadheads and/or vehicles are needed to cover trips that are long way/time away from each other.

In the following, we will describe the essential components of our evolutionary algorithm. After discussing how the initial population is set up, we describe three different methods to calculate the fitness of an individual. Finally, we define the genetic operators and the termination criteria we use in our algorithm.

4.2.1. Initialization

In the first step of the algorithm an initial population is generated to serve as seed for the evolutionary process. We create our solutions (1) in areas where good solutions are likely to be found and (2) randomly in order to cover a wide range of the solution space. We apply four heuristics of the first category that analyze the geographical structure of the problem, i.e., the start and end location of the service trips. The first three heuristics have been proposed by [de Groot and Huisman, 2004] in combination with a heuristic to split large problem instances such that each split problem can be solved with an integrated approach.

- Assign a service trip to the depot closest to its start location,
- Assign a service trip to the depot closest to the combination of start and end location,
- Solve the multiple-depot vehicle scheduling problem and assign a service trip to the depot where it is assigned to in the optimal solution,
- Assign a service trip to the depot either closest to its start or end location.

The rationale behind these heuristics can be understood in the following way. If the trips assigned to the same depot are operated in the same geographical area, it is likely that many of these trips can be covered without extensive deadheading. Furthermore, few deadheads result in a low unproductive overhead since a vehicle and driver outside of the depot spend most of the time on transporting passengers. As stated earlier, we require that at least $\rho = 10$ trips are assigned to a depot. However, if one of the heuristics above leads to an assignment violating the minimum assignment, we randomly shift these trips to other depots.

4.2.2. Fitness Calculation

As described in Section 4.1 there are three different methods of constructing vehicle and crew schedules for a given trip-depot assignment: sequential, par-

tially integrated, and fully integrated. We apply mathematical programming techniques in order to assess the quality of a particular trip-depot assignment.

In the sequential and partially integrated method, we solve a single depot vehicle scheduling problem for each depot with the network simplex implementation of CPLEX. The solution for each depot $d \in \mathcal{D}$ consists of a set of arcs $\bar{A}^d \subset A^d$ and the corresponding flow values. Thus, the total fitness f_θ^v of the vehicle schedule of an individual θ reads:

$$f_\theta^v := \sum_{d \in \mathcal{D}} \sum_{(i,j) \in \bar{A}^d} y_{ij}^d c_{ij}^d. \tag{4.1}$$

Now, we construct a crew schedule based on the vehicle schedule either sequentially or partially integrated. Although we omit the mathematical details, in both cases model MDVCSP reduces to a separate, generalized set partitioning problem (SPP) for each depot. If the vehicle schedule is given (the y variables are fixed) only constraints (2.14) and (2.16) remain where the right-hand side of (2.14) is a constant p_{ij}^d equal to the flow value of y_{ij}^d. Our computational experiments indicate that it suffices to compute a lower bound on the SPP instead of constructing a feasible crew schedule for each individual.

Since the number of duty variables can be vast even for small-sized problems, we apply a column generation algorithm to compute a lower bound of the SPP. Traditionally, column generation (see Section 1.5.2) is a method to solve linear programs that involve a large number of columns. Instead of solving a large problem with all feasible columns (duties), a sequence of *restricted* master linear programs (RMP) is solved where each problem contains only a small subset of all columns. As described in Section 2.4 for the integrated case we found it very promising to solve the master problem with Lagrangian relaxation (with constraints (2.14) relaxed) instead of the linear relaxation. That is, we solve the Lagrangian dual with a subgradient method in order to obtain an approximate dual solution and a lower bound for the current RMP. The dual information $\pi_{ij}^d \geq 0, d \in \mathcal{D}, (i,j) \in \bar{A}^d$ is used to price out new columns that are added to the RMP. To solve the pricing problem we set up a duty generation network and solve an associated resource constrained shortest path problem with a dynamic programming algorithm (see Sections 3.1 and 3.2). The column generation process iterates until no new columns can be found. Finally, we end up with a $|D|$ separate sets of columns and an approximate solution to the Lagrangian dual (i.e.

the lower bound) for each depot $d \in \mathcal{D}$:

$$z^d := \max_{\pi}\{\min \sum_{k \in K^d} x_k^d (f_k^d - \sum_{(i,j)\in \bar{A}^d(k)} \pi_{ij}^d) + \sum_{(i,j)\in \bar{A}^d} \pi_{ij} p_{ij}^d | \quad x_k^d \in \{0,1\}\}. \tag{4.2}$$

Furthermore, the overall fitness f_θ for an assignment θ is defined as the sum of vehicle fitness and crew fitness for each depot:

$$f_\theta := f_\theta^v + \sum_{d\in\mathcal{D}} z^d. \tag{4.3}$$

Notice that the set of columns that we obtain while solving the Lagrangian dual for each depot can be used to construct a feasible integer solution (feasible crew schedule). We use the branch-and-bound implementation of CPLEX to generate integer solutions. However, it can be quite time consuming to construct an integer solution for each individual. Therefore, we only compute an integer solution in the final phase of our EA for individuals with a good overall fitness. The integrality gap between the lower bound and the final integer solution was always less than 2%.

Finally, we would like to mention that both sequential and partially integrated fitness calculation do not require an elaborate solution method for integrated problems (such as [Borndörfer et al., 2004] or [Huisman, 2004]). Instead, we only need a sequential vehicle and crew scheduling algorithm that is used in most commercial software packages for public transport companies. The partially integrated evaluation only differs in the way duties are generated in the column generation pricing problem (but is essentially a set partitioning model as in the sequential crew scheduling problem). Of course, a sequential approach is much easier to implement than a fully integrated one.

So far, we have defined the fitness for a sequential and partially integrated evaluation of a trip-depot assignment. Now, we will specify a fully integrated evaluation. For a given trip-depot assignment model MDVCSP reduces to a minimum cost flow problem in combination with a set partitioning problem. Although this is an NP-hard problem, the vehicle scheduling subproblem can be solved in polynomial time. Again, we first compute a lower bound in order to assess the quality of trip-depot assignment.

We relax constraints (2.12) and (2.14) in a Lagrangian way and use a column generation algorithm similar to the one described earlier. The Lagrangian subproblem constitutes a single depot vehicle scheduling problem (with flow conservation constraints (2.13)) that has to be solved in each iteration of the subgradient

method. Notice that the Lagrangian subproblem in the previous subsection was a trivial selection problem since no constraints remained after relaxing (2.14). Again, we end up with $|D|$ separate sets of columns and an approximate solution to the Lagrangian dual (i.e. the lower bound) for each depot $d \in \mathcal{D}$. As described earlier the sets of columns can be used to compute an integer feasible solution.

However, it turned out that the computational time to generate an integer solution with a fully integrated model can be prohibitive. On the other hand, the lower bound derived from a fully integrated evaluation gives a better indication on the quality of an assignment than the methods described earlier. This is due to the fact that in this approach vehicles are scheduled dependent on crews (and vice versa).

Therefore, we use the fully integrated fitness calculation only in the first phase of our algorithm. That is, after creating the initial population we use the fully integrated method for fitness assignment in the first iterations of the EA. In our computational experiments we found that using the full integrated evaluation in the first $\lceil |\mathcal{T}|/3 \rceil$ iterations provides robust results. Later we recalculate the fitness of each individual and iterate using one of the other methods. Our tests indicate that this improves the overall quality of the population.

4.2.3. Genetic Operators

The parents are selected based on the tournament selection discussed in [Beasley and Chu, 1996]. In tournament selection two pools of randomly selected individuals from the population are constructed. In a second step, we choose the individual with the best fitness from each pool. Our computational tests indicate that forming two pools with 2 individuals each performs best (binary tournament selection).

The recombination performed is based on the fusion operator proposed by [Beasley and Chu, 1996]. The fusion operator produces a single child and takes both the structure of the parents and their fitness into account. The basic idea is to copy an assignment (gene) to the child if both parents assign the trip to the same depot. If the gene values are different in both parents, it is more likely to inherit the gene from the parent with the better fitness. We compare two parents θ_1 and θ_2 and apply the following rules to obtain child θ':

$$\theta_1[i] = \theta_2[i] \text{ then } \theta'[i] := \theta_1[i] = \theta_2[i] \tag{4.4}$$

$$\theta_1[i] \neq \theta_2[i] \text{ then } \theta'[i] := \begin{cases} \theta_1[i] \text{ with prob. } p := \frac{f_{\theta_2}}{f_{\theta_1}+f_{\theta_2}} \\ \theta_2[i] \text{ with prob. } 1-p. \end{cases} \tag{4.5}$$

Our computational test showed that the quality of the final solution is very

sensitive to the mutation rate. In particular, a high mutation rate almost always led to a bad solution quality. Therefore, we use mutation only to eliminate duplicate individuals.

The evaluation of the fitness of an individual is the most time consuming step in our algorithm. Thus, we propose a tabu list in order to store individuals that have been evaluated before. If we generate a child in the recombination phase that has been constructed before, we randomly shift trips between non-empty depots until a new individual is generated. Tests showed that in most cases 2 trip shift suffice to create a new individual.

4.2.4. Termination

We have two different termination criteria. The most obvious is to terminate when a given time limit is exceeded. Furthermore, we terminate whenever there has been no significant improvement of the fitness of the currently best individual within the last $3|\mathcal{T}|$ iterations.

4.3. Computational Results

In this section, we report computational results for the hybrid evolutionary algorithm described in this chapter. All experiments in this section were conducted on a Dell OptiPlex GX620 personal computer running Windows XP with an Intel Pentium IV 3.4 GHz processor and 2 GB of main memory. Our algorithm is implemented in C# and has been compiled using the .NET framework version 2.0.50727.

We use the randomly generated instances and settings that have also been used and described in the preceding chapter. In order to ease the exposition we recall the basic properties. The instances available at [Huisman, 2003] have been classified into two classes according to the travel speed where the speed is lower for problems in class B. As a consequence, trips in class B are longer. However, we will only report computational results for class A that all involve 4 depots and groups with n trips where $n = 80, 100, 160$ and 200. For each group 10 instances are available. In accordance with [Huisman, 2004, Huisman et al., 2005a] we consider five different types of duties: one tripper type with one piece of work between 30 minutes and 5 hours, and four types consisting of two pieces of work. A detailed description of the duty types is given in Section 3.5.

Furthermore, we used the following parameter settings. In all tests we set the population size to 20 and the number of children per iteration to 2. We also tested other settings, but the configuration above appears to be very insensitive

to the problem size. The CPU time limit is set to 30 minutes for instances with 80 to 100 trips, to 60 minutes for 160 trips, and 120 minutes for 200 trips. In the following, all results given for our evolutionary algorithms correspond to the average of five runs.

First, we compare our EA methods with the traditional sequential approach to vehicle and crew scheduling. However, we do not use the fully integrated evaluation in the first $\lceil |\mathcal{T}|/3 \rceil$ iterations of the EA. Consequently, we do not require an integrated solution method (such as [Huisman et al., 2005a]). We rather apply a method similar to the traditional sequential approach to compute the fitness of the individuals. Table 4.1 reports the average solution values for each problem class for the sequential approach (vehicle first - crew second), the EA with the sequential fitness calculation (*EA-S*), and the EA with partially integrated fitness calculation (*EA-PI*). For each solution approach the number of vehicles, drivers, and the sum of vehicles and drivers (*v+d*) is given. Furthermore, we present the relative deviation of the EA solutions from the sequential approach.

	#trips							
Method	80		100		160		200	
Seq.								
vehicles	9.2		11.0		14.8		18.4	
drivers	25.8		29.9		38.8		47.1	
v+d	35.0		40.9		53.6		65.5	
EA-S								
vehicles	9.3	+1.0%	11.3	+2.7%	15.2	+2.7%	18.6	+1.1%
drivers	23.2	-10.0%	27.6	-7.6%	35.5	-8.5%	42.5	-9.8%
v+d	32.5	-7.1%	38.9	-4.8%	50.7	-5.4%	61.1	-6.7%
EA-PI								
vehicles	9.5	+3.2%	11.5	+4.5%	15.3	+3.4%	18.6	+1.1%
drivers	23.2	-10.0%	27.4	-8.4%	36.8	-5.1%	42.5	-9.8%
v+d	32.7	-6.6%	38.9	-4.9%	52.1	-2.8%	61.1	-6.7%

Table 4.1.: Comparison of sequential vehicle and crew scheduling and evolutionary algorithms on Huisman data instances type A

It is easy to see that both evolutionary approaches significantly reduce the number of duties between 5.1% and 10.0%. Although the number of vehicles is slightly higher in EA-S and EA-PI, the total number of vehicles and drivers can be decreased between 2.8% and 7.1%. Furthermore, crew costs generally dominate vehicle costs in practice (see [Bodin et al., 1983]) and, thus, the total savings will be even higher than the numbers indicate. Finally, EA-S appears to perform

better than EA-PI. One reason may be that the number of evaluated individuals is much higher in EA-S. To sum up, the EA approaches use an essentially sequential approach for fitness evaluation, but outperform a stand-alone sequential approach.

Next, we compare EA-S with fully integrated approaches from literature. In Table 4.2 we report the average results of the evolutionary algorithm EA-S* where we use the fully integrated evaluation in the first $\lceil |\mathcal{T}|/3 \rceil$ iterations, [Huisman, 2004, Huisman et al., 2005a] (*HFW05*), and [Borndörfer et al., 2004] (*BLW04*). We do not consider the results of [Mesquita and Paias, 2006] here since they use different assumptions (see Section 2.2.3). We give the total number of vehicles and drivers (*v+d*), the average computation time (*cpu_tot*), and, for the EA, the average standard deviation (*avgsdev*) of the sum of vehicles and drivers. Notice that CPU times cannot be directly compared.

		#trips			
		80	100	160	200
EA-S*	v+d	32.1	37.4	49.2	60.8
	avgsdev	0.38	0.36	0.35	0.36
	cpu_tot[1]	1,800	1,800	3,600	3,600
ICOPT	v+d	28.3	33.7	46.8	57.2
	cpu_tot[2]	235	369	1,740	2,700
BLW04	v+d	29.6	35.7	47.7	59.0
	cpu_tot[3]	780	1,260	2,640	6,360
HFW05	v+d	29.6	36.2	49.5	60.4
	cpu_tot[4]	–	–	–	–

[1,2] on Dell OptiPlex GX620 with Intel Pentium IV 3.4 GHz/2GB
[3] on Dell Precision 650 PC with Intel Dual Xeon 3.0 GHz/4GB
[4] CPU times not comparable

Table 4.2.: Comparison of evolutionary algorithm EA-S* with other approaches from literature on Huisman data instances type A

It can be seen that our algorithm performs worse than the best known fully integrated algorithm ICOPT, but the solution quality of EA-S* increases with the problem size. Furthermore, we can conclude that EA-S* is competitive with the integrated approach of HFW05 for instances with 160 and 200 trips, respectively. In particular, EA-S* was able to find better solutions than HFW05 for the 160 trips class.

4.4. Summary

We have suggested a hybrid evolutionary algorithm to tackle multiple-depot integrated vehicle and crew scheduling problems in public transport. The evolutionary algorithm uses Lagrangian heuristics based on column generation to compute the fitness of the individuals. The algorithm is novel since an evolutionary algorithm has not been applied to integrated vehicle and crew scheduling problems before. The algorithm is based on a problem decomposition that first assigns trips to depots and, thus, reduces the multiple-depot integrated problem to several integrated problems with a single depot. Unlike the multiple-depot case the single depot case has a vehicle scheduling subproblem that can be solved in polynomial time.

The results reported in the previous section indicate that medium-sized problem instances with multiple depots can be solved by using the proposed evolutionary algorithm. Furthermore, our approach discloses significant savings compared to the traditional sequential approach without requiring a fully integrated solution method. Although our algorithm performs worse than the best known integrated algorithm, it proved to be competitive with other integrated approaches from literature especially for medium-sized instances.

In addition to partitioning trips among the depots, trips must be assigned to vehicle blocks and crew duties. A current limitation of our approach is that we do not take this assignment into account. Further research will focus on how to partition the trips assigned to a depot among vehicles and drivers with a local search heuristic.

5. Practical Extensions

In the preceding chapters we considered rules and regulations for duty generation that were relatively simple. We had at most two pieces of work in a duty with a break in between. Furthermore, the break had to be taken outside of a vehicle and, thus, there was a changeover after each break. A piece of work was only restricted by its duration. Finally, the following constraints were imposed: maximum working time, maximum spread time (duty length), minimum start time, maximum end time, minimum break length, and minimum/maximum number of pieces. In this chapter, however, we will extend and change some of the assumptions in order to apply our approach on practical scenarios arising in Germany. However, regulations differ from company to company, and it is almost impossible to list all rules that may occur. Therefore, we will consider only those regulations that are either based on federal regulations or are widely used. In particular, we will consider two different types of break rules: block rules and ratio rules. Moreover, a piece of work is not only restricted by its duration, but also by its driving time. We use an external black-box verifier that discards all duties which do not comply with the complete set of requirements for a company. Our approach of Chapter 3 and the modifications that we propose in this chapter are being integrated in the commercial software tool interplan® of the PTV AG (see [PTV AG, 2007]).

This chapter is organized as follows. In Section 5.1 we give a description of rules and regulations arising in Germany. We discuss the extensions and modifications of our modeling and solution approach to cover these rules in Section 5.2. In Section 5.3 we give an overview of how our implementation is being integrated in the commercial software tool interplan®. We conclude this chapter (Section 5.4) with computational results and a case study on the local public transport company in Paderborn (see [PaderSprinter, 2007]).

5.1. Rules and Regulations in Germany

In this section, we will describe common rules and regulations arising in Germany. The basic regulations are defined in the federal regulations for drivers of trucks

and buses (in German: Fahrpersonalverordnung (*FPersV*), see [Bundesministerium für Verkehr, Bau- und Wohnungswesen, 2005]). In the following, we will introduce new definitions and redefine those that are different to the definitions in Chapter 3.

Layover time Layover time is the time a driver is waiting with a vehicle between two service trips. The driver does not steer the vehicle.

Driving time The driving time includes at least all activities of a driver where the driver is scheduled to steer a vehicle. These activities include but are not limited to service trips, deadheads, and turnarounds. However, the term driving time is not clearly defined in FPersV.

Continuous driving time A sequence of activities is counted as continuous driving time if there is no layover time of a specified minimum length therein. A layover that is longer than the specified minimum length is called *long layover*.

Working time The working time involves at least all activities of a driver that the driver spends on the vehicle either driving or waiting. However, only waiting tasks up to a given length are counted.

Each duty must satisfy at least one break rule if the driving time of that duty exceeds 270 minutes. There are two groups of break rules: block and ratio break rules. It is often subject to in-house agreements which rules may actually be applied.

Block break rules A duty satisfies a block break rule if a duty either contains 1 break of at least 30 minutes, 2 breaks of at least 20 minutes, or 3 breaks of at least 15 minutes. In the following, we denote these rules by *block30*, *block20*, and *block15*, respectively.

Ratio break rules A duty satisfies a ratio break rule if the total layover time of a duty amounts to at least $\frac{1}{6}$ ($\frac{1}{5}$) of the total driving time where layovers of less than 10 (8) minutes do not count. In the following, we denote these rules by *ratio6* and *ratio5*, respectively.

A duty is said to be *connected* if it complies with a block or ratio break rule and the longest break/layover does not exceed a given threshold. If the threshold is exceeded, we have a *split* duty that consist of two parts: one in the early morning and another in the late afternoon with a long break in the middle. If the total driving time of such a duty part exceeds 270 minutes, this part must satisfy a

break rule. If each part has more than 270 minutes of driving time, the parts may satisfy two different break rules. In addition to the break rules we consider the following constraints for the construction of a feasible duty where the actual parameter values may differ by duty type.

- minimum/maximum working time
- minimum/maximum spread time (duty length)
- maximum continuous driving time
- maximum total driving time
- earliest/latest start time of duty
- earliest/latest end time of duty
- minimum/maximum number of pieces of work
- minimum working time before first break
- minimum working time after last break
- sign-on/off time inside/outside of depot

In practice, not only the construction of a single duty is constrained but also specific requirements for groups of duties must be met. In particular, companies often limit the (minimum/maximum) number or ratio of duties of a particular type in the final crew scheduling solution. For instance, the ratio of split duties is often restricted or the average working time of duties is limited. Rules concerning groups of duties or the crew schedule as a whole are called *global constraints* while regulations for a single duty are said to be *local constraints*.

5.2. Extensions of Modeling and Solution Approach

In this section, we will discuss the extensions and modifications of our modeling and solution approach to cover the rules and regulations described in the preceding section. We will first describe modifications due to local constraints and then discuss our extensions for global constraints. Recall that the following constraints have already been considered in the preceding chapter: minimum/maximum working time, minimum/maximum spread time (duty length), earliest/latest start time of duty, earliest/latest end time of duty, minimum/maximum number of pieces of work, and sign-on/off time inside/outside of depot.

5.2.1. Driving Time Constraints

There are two types of driving time constraints: maximum total and maximum continuous driving time. Recall that we apply a two-phase pricing scheme (see Section 2.4.2) where we set up a *piece generation network* in the first phase to generate a set of pieces of work. These pieces serve as input for the second phase where duties are generated. In Section 3.1.2, we discussed three network formulations for the duty generation phase where the maximum total driving can easily be covered by introducing another resource (total driving time). Furthermore, we compute the total driving time of each piece generated in the first phase. All arcs associated with a piece of work consume the driving time of the corresponding piece. However, notice that the introduction of driving time destroys our assumption that a piece of work is only restricted by its duration. Clearly, not all paths between two relief points necessarily consume the same amount of driving time since a path may contain waiting activities. In fact, it no longer suffices to compute the least reduced cost piece of work for each pair of relief points in order to prove column generation optimality. Instead, we must compute the set of non-dominated (see Section 3.2.1) pieces of work where both driving time and reduced cost are considered. The set of non-dominated pieces can be generated using a simple dynamic programming algorithm.

In so far as the maximum continuous driving time is concerned, the piece generation phase needs further modifications. In the preceding chapter, we used the piece generation network $\bar{G}^d = (\bar{N}^d, \bar{A}^d)$ for each depot $d \in \mathcal{D}$ as defined in Section 2.4.2 where nodes $\bar{N}^d$ correspond to relief points. For a given node $n_i \in \bar{N}^d$, we computed the shortest path from node n_i to each node $n_j \in \bar{N}^d$ that satisfied the duration constraint. In particular, $\delta_{\min} \leq t_{n_j} - t_{n_i} \leq \delta_{\max}$ holds with $\delta_{\min}$ ($\delta_{\max}$) as the minimum (maximum) duration of a piece of work and t_{n_i} as the time of node n_i. Thus, the set of (destination) nodes $\bar{N}_f(n_i) \subset \bar{N}$ that we consider for node n_i read

$$\bar{N}_f^d(n_i) = \{n_j \in \bar{N}^d \quad | \quad \delta_{\min} \leq t_{n_j} - t_{n_i} \leq \delta_{\max}\}. \tag{5.1}$$

As described in the preceding section, a piece of work is not restricted by its duration, it is rather limited by the maximum continuous driving time. As a consequence, we have to modify the definition of set $\bar{N}_f^d(n_i)$. Before we start with column generation, we compute the shortest path from node n_i to all nodes n_j with $t_{n_j} > t_{n_i}$. The costs on the arcs are defined in such a way that the costs equal the driving times of the corresponding activity. Finally, the set $\bar{N}_f^d(n_i)$ consists of all nodes n_j where the length of shortest path is less or equal to the maximum continuous driving time. For column generation, we redefine the costs on the arcs

such that they correspond to the reduced cost. Furthermore, it does not suffice to compute the shortest paths between node $n_i \in \bar{N}^d$ and nodes $\bar{N}_f^d(n_i)$ in a column generation iteration since the maximum continuous driving time may be violated. Thus, the construction of pieces relies on a k-shortest-path enumeration where we only accept non-dominated paths that satisfy the maximum continuous driving time. The total number of pieces can be prohibitive for duty generation since there can be multiple paths between each pair of relief points (instead of a single one). However, the number of pieces can be reduced by applying state space reduction (see Section 3.2.3). A simple heuristic is to construct only a single piece for each pair of relief points: the first feasible piece returned by the k-shortest-path algorithm.

Clearly, not all activities in the piece generation network correspond to activities where the crew actually drives the vehicle. As a consequence, the set $\bar{N}_f^d(n_i)$ may contain more destination nodes than necessary since the shortest paths can contain long layovers that reset the maximum continuous driving time. However, there may be other paths without long layovers between these nodes that constitute the shortest path in a subsequent column generation iteration.

5.2.2. Block and Ratio Break Rules

The purpose of this section is to describe the modifications necessary to cover block and ratio break rules. We will first discuss changes due to block break rules and concentrate on ratio break rules afterwards. Clearly, the modifications only involve the duty generation network. Thus, we will discuss the changes of the three network formulations for decomposed pricing described in Section 3.1.2, namely the connection-based, time-space, and aggregated time-space model.

Recall that a duty satisfies a block break rule if a duty either contains 1 break of at least 30 minutes (*block30*), 2 breaks of at least 20 minutes (*block20*), or 3 breaks of at least 15 minutes (*block15*). Furthermore, drivers must take their breaks at given break locations. We assume that a transfer matrix is given where the walking time from all start/end locations of service trips to all break locations is defined. For all three network models, we modify the definition of *break* arcs. In particular, we allow only those break arcs where the minimum break length (30, 20, or 15 minutes) combined with walking to/from the break location is satisfied. Furthermore, we introduce a new type of arcs that connects two pieces of work: *layover* arcs. In a connection-based network, *layover* arcs originate and terminate at a *piece* node while they connect a *piece_end* with a *piece_start* node in a time-space network. As opposed to *break* arcs, *layover* arcs must correspond to a long layover (see Section 5.1), but must not be longer than the minimum

break length. Additionally, traversing a *layover* arc does not increase the number of pieces of work. In this context, the resource that counts the number of pieces is used to count the number of pieces that succeed a valid break. For all network models, the minimum number of pieces is 2 for *block30*, 3 for *block20*, and 4 for *block15*.

For ratio break rules, we must track both driving and layover time. Recall that the total layover time of a duty must amount to at least $\frac{1}{6}$ ($\frac{1}{5}$) of the total driving time where layovers of less than 10 (8) minutes do not count. Thus, we introduce two additional resources: total layover and total driving time. Notice that we have already defined a resource to count the total driving time if the maximum total driving time is restricted. The pricing algorithm must check the ratio of layover to driving time at the sink node.

5.2.3. Break Positions

In this section, we consider the situation where a minimum working time ϖ_b before the first and a minimum working time ϖ_a after the last break is imposed. In the following, we will describe the modifications when a decomposed pricing scheme is used. As stated in the preceding section, the modifications only involve the duty generation network. Thus, we will again discuss the changes of the three network formulations.

Recall that a duty starts (ends) with a sign-on (sign-off) activity. In the *connection-based model*, we have sign-on (sign-off) arcs from the source to piece nodes (from piece nodes to the sink). Let P^d denote the set of pieces of work of depot $d \in \mathcal{D}$. Furthermore, recall that the set of pieces corresponds to the set of nodes in the duty generation network. We compute the working time $\check{w}_i^d$ of a piece of work $i \in P^d$ before the driver takes his first break (block break rules) or has a long layover (ratio break rules). Similarly, we calculate the working time $\hat{w}_i^d$ of a piece $i \in P^d$ after the driver took his last break or had the last long layover, respectively. A piece of work node $i \in P^d, d \in \mathcal{D}$ may only be connected with the sink if $\hat{w}_i^d \geq \varpi_a$ holds while it may be connected with the source only if $\check{w}_i^d \geq \varpi_b$ is satisfied. Clearly, the network complexity is still $\mathcal{O}(\nu^4)$.

For the *(aggregated) time-space model*, the network formulation is modified in a different way. Recall that multiple piece of work arcs originate from a piece start node and terminate at a piece end node. As a consequence, we cannot simply modify the definition of *sign_on*/*sign_off* arcs as we have done for the connection-based model. Instead, we modify the (aggregated) time-space model in the following way. We introduce a *duty_start* (*duty_end*) node for each *piece_start* (*piece_end*) node. Furthermore, *sign_on* arcs originate from *source*

and terminate at a *duty_start* node while *sign_off* arcs emanate from a *duty_end* node and end at *sink*. In addition to the *piece* arcs that connect *piece_start* with *piece_end* nodes, we introduce further piece arcs. If $\check{w}_i^d \geq \varpi_b$ is satisfied for piece $i \in P^d$, there is piece arc from the corresponding *duty_start* to the *piece_end* node. Likewise, we introduce another piece arc for piece $i \in P^d$ from the associated *piece_start* node to the *duty_end* node if $\hat{w}_i^d \geq \varpi_a$ holds. Figure 5.1 depicts a time-space duty generation network with 5 pieces of work where a minimum working time before the first and after the last break is imposed. In our example, the first piece can only be at the beginning of a duty while the next two can be both at the beginning and end of a duty. The last piece must not be at the beginning of a duty. Despite the additional piece arcs, the network complexity of both models remains $\mathcal{O}(\nu^2)$.

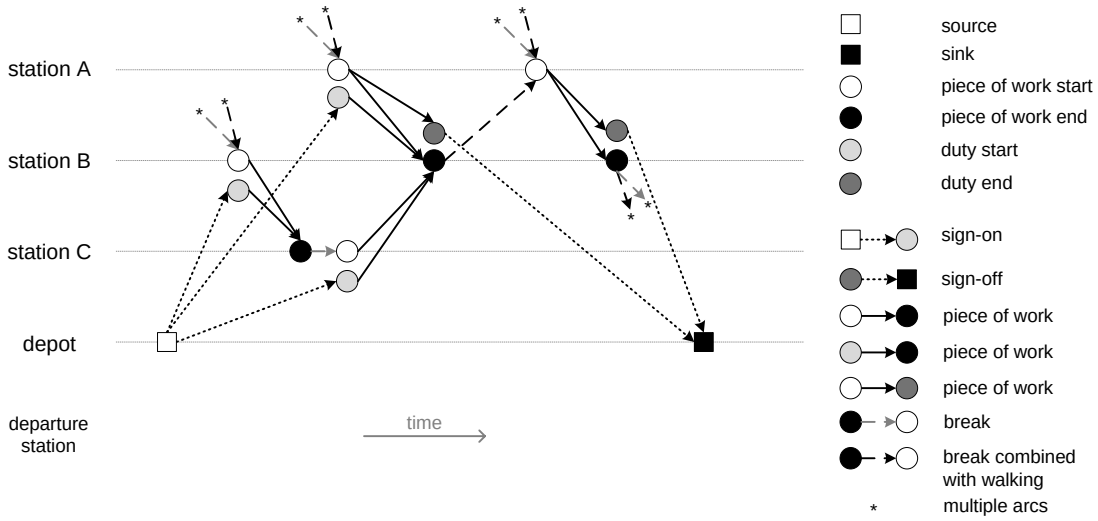

Figure 5.1.: Modified time-space duty generation network to consider specific break positions

5.2.4. Duty Mix

As stated earlier global constraints deal with groups of duties at once while local constraints define the feasibility of a single duty. In this section, we discuss duty type constraints that belong to global constraints. In particular, we limit the (minimum/maximum) number or ratio of duties of a particular type in the final crew scheduling solution (*duty mix*).

We extend model MDVCSP (see Section 2.3) to impose a maximum number

$t^d_{\max}$ of duties of a specific type $t \in T$ for each depot $d \in \mathcal{D}$ as follows.

$$\sum_{k \in K^d} t^d_k x^d_k \leq t^d_{\max} \qquad \forall d \in \mathcal{D} \tag{5.2}$$

t^d_k is equal to 1 if duty k is of type t and 0 otherwise. Notice that the same constraints can be used to restrict the ratio ν^t of duties of type $t \in \mathcal{T}$ in the final solution. We set $t^d_k = 1 - \nu^t$ if duty $k \in K^d$ has type t and $t^d_k = -\nu^t$ otherwise. Furthermore, we define $t^d_{\max} = 0, \forall d \in \mathcal{D}$.

As described in Section 2.4.1 we use Lagrangian relaxation in combination with column generation to compute a lower bound. Therefore, we associate Lagrangian multipliers $\delta^d \leq 0, \forall d \in \mathcal{D}$ with constraints (5.2) and dualize the constraints. The Lagrangian subproblem now reads

$$\Phi^m(\mu, \pi, \delta) = \Phi_y(\mu, \pi) + \Phi^m_x(\mu, \delta) + \sum_{t \in \mathcal{T}} \pi_t + \sum_{d \in \mathcal{D}} \delta^d t^d_{\max}. \tag{5.3}$$

The vehicle scheduling subproblem remains unchanged (see Equation (2.20)) while the crew scheduling subproblem now involves the multipliers associated with the duty mix constraints:

$$\Phi^m_x(\mu, \delta) = \left\{ \min \sum_{d \in \mathcal{D}} \sum_{k \in K^d} x^d_k \bar{f}^d_k \mid x^d_k \in \{0,1\}, \quad \forall d \in \mathcal{D}, \forall k \in K^d \right\}. \tag{5.4}$$

The reduced cost of duty $k \in K^d$ is denoted by

$$\bar{f}^d_k = f^d_k - \delta^d t^d_{\max} - \sum_{(i,j) \in \widetilde{A}^d(k)} \mu^d_{ij} \tag{5.5}$$

where $\widetilde{A}^d(k) \subseteq \widetilde{A}^d$ corresponds to the set of arcs that is covered by duty $k \in K^d$.

When the sequential approach is used (see Sections 2.4.3 and 3.3.1) to compute feasible solutions we only relax constraints (2.14) in a Lagrangian way. Thus, the solution of the vehicle scheduling subproblem gives a feasible vehicle schedule. Each feasible vehicle schedule can be used to construct a feasible crew schedule using sequential crew scheduling for each depot. However, the duty mix constraints must be considered in the crew scheduling problem. In our implementation, we use Lagrangian relaxation in combination with column generation to generate a promising set of columns. Finally, we use the branch-and-bound implementation of ILOG CPLEX to find integer solutions.

Clearly, equation (5.2) can be adjusted if the duty mix is restricted for all depots at once. Notice that this requires to perform sequential crew scheduling in the integer phase for all depots at once.

5.3. System Overview

Even though we can handle most duty construction rules directly, some regulations are too complex to be covered efficiently. As a consequence, we ignore the rule in our pricing algorithm and check duties with an external *black-box verifier*. The verifier either accepts or rejects a duty, but does not expose details of the evaluation process to the rest of the system. In case that the duty is accepted the verifier also computes accurate planned (operational) cost. Accurate cost calculation can be a very challenging task since it is often subject to in-house regulations and may strongly differ from company to company. The duty costs in our pricing algorithm are defined in such a way that we never overestimate the (operational) cost of a duty. If we would overestimate the cost of a duty in our pricing algorithm, the duty might be discarded by our pricing scheme although the duty is valuable (has negative reduced cost). However, the planned cost returned by the verifier can turn a negative reduced cost duty into a non-negative reduced cost duty. In such a case, we discard the duty. Similar approaches are used by [Borndörfer et al., 2006] and [Galia and Hjorring, 2004] for airline crew scheduling. However, the authors use a *rule verification oracle* where only pairing (duty) feasibility is checked.

Furthermore, we allow the verifier to propose duties for addition to the restricted master problem (RMP). More precisely, the verifier is regularly called to propose duties that are added to a *column pool*. For example, we can add feasible columns that are part of the current solution from practice or similar to duties of the current solution. Basically, a column pool contains a set of feasible duties that are *explicitly* kept in computer memory, but that are not part of the current restricted master problem. Recall that in the original version of our pricing algorithm, columns are only *implicitly* available unless they are added to the RMP. Columns in the column pool can easily be priced in subsequent iterations and are added to the RMP if they have negative reduced cost. Clearly, columns added to the RMP are deleted from the pool. Furthermore, we add columns to the pool that were deleted from the RMP (due to high positive reduced cost). To sum up, we use a combination of implicit and explicit column generation.

The system described in this section is being integrated in the commercial software tool interplan® of the PTV AG (see [PTV AG, 2007]). Figure 5.2 depicts an overview of the system with our optimization system ICOPT. In addition to the interaction described above, our optimization system transfers feasible solution(s) to interplan. Interplan displays the solution(s) on a graphical user interface. In this context, the planner can decide whether he would like to accept a solution as final solution or whether further optimization is required. In other

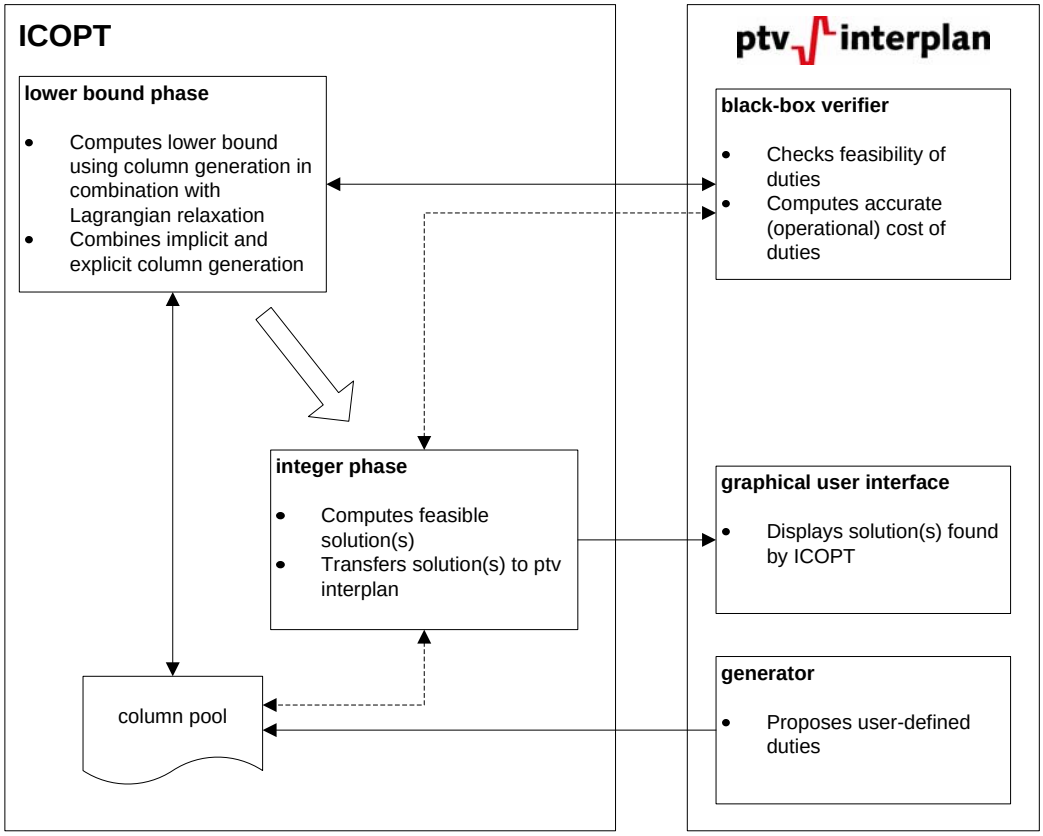

Figure 5.2.: Integration of ICOPT with PTV interplan®

words, the purpose of ICOPT is not to provide a single final solution, it should rather generate several solutions. As a consequence, the planner can select that solution among the set of generated solutions that meets his requirements in terms of costs and other soft (social) factors.

5.4. Computational Results

The purpose of this section is to summarize our computational results on real-world problem data from Connexxion and randomly generated instances. All experiments in this section were conducted on a Dell OptiPlex GX620 personal computer running Windows XP with an Intel Pentium IV 3.4 GHz processor and 2 GB of main memory. Our integrated method ICOPT is implemented in C# and has been compiled using the .NET framework version 2.0.50727. For a description of the problem instances we refer to Section 3.5.

We use the same parameter settings and cost function as described in Section 3.5. Recall that we assigned a fixed cost of 1,000 for each vehicle and duty and a small variable cost of 1 for each minute a vehicle is outside the depot and 0.1 for each minute a crew is working. However, we consider a different set of duty types and work regulations. More precisely, we consider three block break rules with at least two pieces of work and one tripper type with a single piece. The tripper duty type does not require a break, but the total driving time is limited to 04:30 hours. Table 5.1 summarizes the duty types that require at least one break. All duty types arise from federal regulations in Germany. Notice the difference between total and continuous driving time: a sequence of vehicle activities is counted as continuous driving time if there is no long layover therein while the total driving time is additive over the whole duty. In other words, the continuous driving time is reset to zero if the driver has a long layover.

In Table 5.2 we give computational results of our integrated approach on Connexxion data instances. For each of the 8 instances we report the number of trips and the number of depots where each trip can be serviced from every depot. We give the number of iterations (*#iter*), the CPU time in seconds spent on the master (*cpu_ma*), the pricing problem (*cpu_pr*), the integer phase (*cpu_ip*), and the total time including all initializations (*cpu_tot*). Moreover, we report the final upper bound (*ub*), the number of vehicles (*vehicles*), drivers (*drivers*), total number of vehicles and drivers for integrated (*v+d*) and sequential vehicle and crew scheduling (*v+d seq.*). Finally, we give the total number of duties and drivers that we obtained with the comparatively easy duty types in Section 3.5.1 (*v+d ref.*).

	block30		block20		block15		split	
Type	min	max	min	max	min	max	min	max
Break length	00:30	01:00	00:20	01:00	00:15	01:00	01:00	–
No. of breaks	1	–	2	–	3	–	1	–
Driving time	–	09:30	–	09:30	–	09:30	–	09:30
Driving time (continuous)	–	04:30	–	04:30	–	04:30	–	04:30
Long layover time	00:10	–	00:10	–	00:10	–	00:10	–
Duty length	07:00	12:00	07:00	12:00	07:00	12:00	07:00	14:00
Working time	–	09:00	–	09:00	–	09:00	–	09:00
Sign-on	00:08	00:08	00:08	00:08	00:08	00:08	00:08	00:08
Sign-off	00:06	00:06	00:06	00:06	00:06	00:06	00:06	00:06
Sign-on after break	–	–	–	–	–	–	00:05	00:05
Sign-off before break	–	–	–	–	–	–	00:05	00:05

Table 5.1.: Properties of different duty types

	instance							
	1	2	3	4	5	6	7	8
#trips	194	210	220	237	304	386	451	653
#depots	4	4	4	4	4	4	4	4
#iter	26	23	31	24	25	35	28	31
cpu_ma	794	641	1,003	729	1,138	2,233	2,605	4,676
cpu_pr	2,260	584	1,750	567	1,261	10,383	8,661	16,521
cpu_ip	10,097	1,300	1,537	4,656	8,377	5,292	5,982	20,107
cpu_tot	13,163	2,535	4,306	5,959	10,792	18,346	17,719	41,954
ub	48,834	82,824	65,610	87,640	105,976	86,397	120,898	183,017
vehicles	19	33	28	34	40	32	47	67
drivers	26	43	33	46	57	48	65	103
v+d	45	76	61	80	97	80	112	170
v+d seq.	49	85	70	91	106	85	120	175
v+d ref.	45	81	60	86	102	85	109	174

Table 5.2.: Results on Connexxion data instances with practical extensions

As can be seen from Table 5.2 we could solve problem instances with up to 653 trips and four depots using rather complex duty types. Furthermore, the results show that the total number of vehicles and drivers can be significantly reduced when an integrated approach is used. As expected the time spent on the pricing problem increases compared to the relatively easy duty types in Chapter 3 (see Table 3.16). Except for problem 7 the number of duties is equal or smaller than the reference where the easy duty types were used. Thus, we conclude that our method in combination with the modifications described in this chapter can also efficiently cover complex duty types. However, a thorough investigation would require to compute valid lower bounds for both duty sets and compare these bounds with the best corresponding upper bounds.

5.5. Summary

In this chapter we considered practical rules and regulations arising in public transport companies in Germany. We suggested extensions and modifications of our modeling and solution approach from Chapter 3 to cover these practical extensions. The enhancements included driving time constraints, complex break rules where many pieces of work are allowed, break positions, and duty mix constraints. Furthermore, we gave an overview of how our implementation is being integrated in the commercial software tool interplan®. We tested the applicabil-

ity of the proposed techniques using real-life data instances with up to 653 trips and four depots. The results indicate that our approach can efficiently cover duty types with many pieces of work and complex feasibility rules.

6. Ex-Urban Vehicle and Crew Scheduling with Irregular Timetables

In the preceding chapters, we focused on how to conduct operations of a given timetable at minimum cost. In this chapter, however, we will consider another aspect which is related to the quality of crew schedules.

We will discuss the case where timetables consist of many trips serviced every day together with some exceptions that do not repeat daily. In particular, service trips to schools, production facilities, or public swimming baths are often subject to change, e.g. trips may be operated on every day except Sunday or on Monday only. Unless specifically imposed, traditional methods for vehicle and crew scheduling usually produce schedules that contain irregularities which are not desired in practice. A crew schedule is called irregular if it cannot be repeated many times. Similar to airline crew scheduling (see [Klabjan et al., 2001]), *regularity* is an important aspect for crew schedules in public transport since regular solutions can improve operational reliability and can reduce training costs (see [Dallaire et al., 2004]). Furthermore, regular solutions are less error-prone and crews often prefer to repeat itineraries. In current practice, companies often try to increase regularity of crew scheduling solutions by one of the following heuristic two-phase procedures:

- *All first - irregular second*: First, the planner solves a crew scheduling problem for a particular period with both regular and irregular trips. In a second step, he or she fixes the subset of crew duties that can be operated the whole period and reoptimizes all unfixed trips. Notice that the second problem can also contain some regular trips.

- *Regular first - irregular second*: The set of service trips is divided into regular and irregular trips. First, a crew scheduling problem for the set of regular trips is solved while the irregular trips are left for subsequent optimization.

In both cases, the second problem has a sparse schedule and, thus, likely requires extensive deadheading and even its optimal solution yields a high costs. On the other hand, if the second problem contains many trips, the corresponding solution has low cost but low regularity as well.

As stated earlier, we are concerned with the regularity of crew schedules and not with the regularity of vehicle schedules. In fact, vehicles are rather insensitive to the quality of their schedules as opposed to drivers. In order to test our approaches, we will concentrate on scenarios where crew scheduling plays the major role. This holds particularly for ex-urban scenarios as we will see in the following section.

To the best of our knowledge, solution approaches to improve the regularity of crew schedules in public bus transport have not been described in literature before. In this chapter, we propose two approaches that capture both costs and regularity of crew scheduling solutions. In particular, we propose a novel combination of local branching and follow-on branching that improves the regularity of crew schedules while cost optimality is maintained. Furthermore, we compare four bi-objective metaheuristics that include both cost and regularity as objective functions. The latter approach can be used to get a quick estimate of the solution quality obtained with the first approach.

This chapter is organized as follows. In Section 6.1, we give a formal problem definition for the ex-urban vehicle and crew scheduling problem with irregular timetables. We discuss other approaches related to public (bus) transport from literature in Section 6.2. In the next section, we describe how local branching and user-defined branching rules can be used to steer the solution method to regular crew scheduling solutions. In the same section, we describe four bi-criteria metaheuristics from literature that try to approximate the set of Pareto optimal solutions. Finally, we provide computational results on real-world and randomly generated instances in Section 6.5. The chapter is concluded with a short summary (Section 6.6).

6.1. Problem Definition

In this section, we will first describe the ex-urban vehicle and crew scheduling problem and discuss regularity of crew schedules afterwards.

Public transport scenarios can be categorized according to the structure of the underlying transportation network. *Urban* service provides connections within the city while *ex-urban* (*regional*) service connects the city with the suburbs and minor towns in the region of the city. Of course, many companies offer

a mixture of both categories. Many regional scenarios have in common that the line network is star-shaped around the depots with only few relief points. Furthermore, distances between relief points are such that drivers are virtually tied to their vehicle in order to reach the relief points. In other words, pieces of work often correspond to vehicle blocks. When traditional vehicle and crew scheduling (vehicles first - crew second) is applied in an ex-urban setting, vehicle blocks are likely to be too long to meet break requirements, or drivers cannot return to their home depot. Conclusively, crews must be scheduled at the same time as vehicles or before vehicles in order to guarantee the feasibility of the crew schedule. In the remainder of this chapter, we will assume that drivers may only change their vehicles in depots (ex-urban scenario).

Crews can easily be scheduled before vehicles if there is a single depot and vehicle changes outside the depot are not allowed (or drivers can walk from all relief opportunities to the depot). In such a case, we first solve an *independent crew scheduling problem* (ICSP) that we define as follows. Given the traveling times between all pairs of locations and a set of tasks which corresponds to the set of service trips[1], find a minimum cost set of duties such that all tasks are covered by feasible duties (see also [Huisman, 2004]). Since each duty starts and ends at the depot, the vehicle rotations that result from the crew scheduling solution can be put together to form a feasible vehicle schedule (using a vehicle scheduling method). The approach to schedule crews before vehicles is also referred to as *partial integration* (see [Borndörfer et al., 2004]). However, the number of vehicles is not necessarily minimal in contrast to a fully integrated approach (see Section 2.3). Notice that a feasible vehicle schedule can also be constructed when there are multiple depots and duties that start and end at the same depot. If continuous attendance is required (see Section 2.1), and a driver must not stay on his (idle) vehicle during a break, each piece of work must start and end at the same depot. As a result, drivers spend their breaks in a depot and take the same or a different vehicle for the consecutive piece of work.

We will now formally define the vehicle and crew scheduling problem with irregular timetables. Let F be a timetable with tasks $f_1, \ldots, f_n$ where task f_i starts earlier than f_{i+1}. Furthermore, a reference crew schedule $R = \{R_1, \ldots, R_u\}$ with duties $R_i = \{f_{i1}, \ldots, f_{ip}\}$ that is compatible to timetable F is given. The *integrated vehicle and crew scheduling problem with irregular timetables* (VCSP-IT) for timetable $F' \neq F$ and given depots, relief points, and a reference crew schedule R can be stated as follows: find minimum cost sets of vehicle blocks and crew duties such that both vehicle and crew schedule are feasible and mu-

[1] In order to stay consistent with the terminology of the preceding chapters, we modify the definition of a task: a task no longer needs to start and end with a relief point.

tually compatible. Furthermore, crew schedule $D = \{D_1, \ldots, D_v\}$ should have a small *distance* to reference schedule R. A crew schedule with a small distance to reference R is called *similar* or *regular*. However, minimizing costs remains the primary objective.

The perception of distance between two crew schedules can differ from company to company. A very simple *distance measure* is to count the number of duties in the new crew schedule that could not be preserved from the reference crew schedule. In the following, we will describe a more elaborate distance measure that basically counts the number task sequences not preserved from the reference. Let $Q = F \cap F'$ be the set of *regular* tasks that are part of both timetables. A *regular pair* $S \subseteq Q$ is an ordered pair of regular tasks (f_i, f_{i+k}) that are operated consecutively in both reference R and new crew schedule D. We denote by S^1 the first task of regular pair S while S^2 corresponds to the second task. Notice that an irregular trip may be operated between f_i and f_{i+k}, but no regular trip. Clearly, a regular trip to cannot be at the first (second) position of more than one regular pair. However, it may be at the first position in one pair and at the second in another pair. Furthermore, a *regular chain* $T = (S_1, \ldots, S_j) = ((S_1^1, S_1^2), \ldots, (S_j^1, S_j^2))$ with $j \geq 1$ and $S_i^2 = S_{i+1}^1, 1 \leq i < j-1$ is an ordered sequence of interconnected regular pairs. $\widetilde{T}$ denotes the number of regular tasks of regular chain T. Furthermore, let $\bar{S}$ and $\bar{T}$ denote the set of all regular pairs and chains, respectively. We define distance measure $\sigma^p(\sigma^c)$ that corresponds to the number of regular tasks that are not part of a regular pair (chain).

$$\sigma^p = |Q| - 2|\bar{S}| \tag{6.1}$$

$$\sigma^c = |Q| - \sum_{T \in \bar{T}} \widetilde{T} \tag{6.2}$$

Of course, there are numerous other distance measures possible. However, we believe that our measures give an intuitive approach to regularity of crew schedules. Therefore, we will focus on σ_p and σ_c in the remainder of this chapter. However, our approaches also work with other distance measures.

6.2. Literature Review

In this section, we review state-of-the-art models and solution methods for crew scheduling with irregular timetables from both public transport (bus and railway) and airline perspectives. Since we are concerned about the regularity of crew schedules, we do not consider vehicle scheduling in our literature review. As we will see, the literature on irregular timetables in public bus transport is virtually non-existent. Therefore, we include railway and airline settings in our review.

Solution approaches can mainly be categorized into *regularity* and *rescheduling approaches*. Regularity approaches build a solution from scratch for a given (long) period where the solution should inherently contain as many regular patterns as possible. In rescheduling methods, a reference schedule is given and a new solution for a (short) period is constructed where the new solution should be as similar as possible to the reference. In the following, we will review models and solution methods based on both approaches.

Regularity Approaches

[Tajima and Misono, 1997] describe an airline crew scheduling problem with many irregular flights. The authors seek to find a set of pairings (duties) that cover all flights in the planning period (one month) where essentially the total number of *man-days* is minimized. The number of man-days of a pairing is equal to the number of days it lasts. The secondary objective is to minimize costs. Furthermore, a large portion (between 9% and 54%) of all flights is not flown on every day of the planning period. The authors propose a heuristic that systematically merges irregular flights into pairings that only consist of regular flights. Their computational tests involve two real-world data instances with 8,876 and 9,504 flights where the ratio of irregular flights was 54% and 9%, respectively. Their experiments revealed that the instances could be solved in 41 and 92 minutes on an IBM RS/6000 model 900. Moreover, their method could find better solutions than manual planning by experienced engineers. Although the primary objective was to minimize the number of man-days, the approach manages to produce regular crew schedules. For the first instance, 81% of the pairings were regular while 92% of the pairings were flown every day for the second one. However, the authors do not report the impact on operational costs since regular pairings may contain a lot of (paid) waiting time.

[Klabjan et al., 2001] introduce the weekly airline crew scheduling model with regularity. The model captures the trade-off between regularity and costs in a weekly schedule. The set of flights is partitioned into groups in such a way that regularity is easily obtainable in each group. A g-regular group for $g = 4, \ldots, 7$ contains flights that can be repeated g consecutive days of the week. By definition, regular flights i from a g-regular group have $g_i \geq g$. Each g-regular group is subsequently partitioned by g-regular pairings. All flights not assigned to a g-regular group, g=4,. . . ,7, are called irregular flights and must be assigned to irregular pairings. In their model, the authors assign penalty costs to irregular flights. Penalty costs decrease with increasing regularity. However, the complete regularity model is intractable and, thus, the authors resort to

an approximate model and solution methodology. In particular, pairings are produced in decreasing order of regularity. 7-regular pairings are produced first and an appropriate subset is computed to form 7-regular pairings in the final weekly solution. The flight schedule is reduced by all flights already covered by 7-regular pairings. In the next stage, the remaining flights can only be covered by 6-regular pairings. The process iterates until irregular pairings are generated and the complete flight schedule is partitioned. Computational results with three real-world data instances show that problems with at most 492 flights can be solved in 47 hours computational time. The tests were performed on two clusters: one consisting of 16 machines each with Quad Pentium Pro 200MHz/256 MB main memory and the other comprised of 48 machines each with Dual Pentium II 300MHz/512 MB main memory. The solutions reported improve on existing solutions used by the airline both in terms of regularity and costs.

Rescheduling Approaches

We distinguish between *unplanned* and *planned* rescheduling. Unplanned rescheduling of crews is necessary when the planned crew schedule cannot be executed due to irregular operations or disruptions. Planners usually aim to determine new crew assignments that make as few changes to the original schedule as possible. In other words, planners like to find a new solution with a small distance to the original (reference) solution. Unplanned crew rescheduling is also referred to as crew recovery. Typically, the underlying flight schedule may be changed in crew recovery problems, i.e., flights may be delayed or even canceled, if no feasible recovery scheme is found in a given timeframe. Notice that the underlying timetable must not be altered in the problem stated in the preceding section. Furthermore, typical scenarios for crew recovery include local disruptions while irregular trips are often spread over the complete timetable. In conclusion, solution approaches for crew recovery do not seem to be well suited for our problem stated in Section 6.1. However, recent approaches to airline crew rescheduling (recovery) include, among others, [Lettovsky et al., 2000], [Guo et al., 2005], [Nissen and Haase, 2006], and [Medard and Sawhney, 2007]. A recent survey can be found in [Clausen et al., 2005].

In planned crew rescheduling the changes in the underlying timetable are typically known in advance. [Huisman, 2007] describes the planned crew rescheduling problem in a railway setting at NS which is the largest passenger railway operator in the Netherlands. At NS crew scheduling is performed in two stages. First, solutions for an annual plan are constructed, i.e., for a general Monday, Tuesday, and so on. In a second phase, the general days are adapted to individual days

where specific changes in the timetable for those days are considered. The author states that the changes in the timetable are mainly due to track maintenance or extra service trips that are both usually known in advance. He suggests a set covering formulation where original duties are replaced by new (similar) duties such that all tasks of the modified timetable are covered and total costs of the new duties are minimized. He used a heuristic based on column generation in combination with Lagrangian relaxation and an elaborate set covering heuristic to compute integer solutions. The computational experiments involved two real-world scenarios and were performed on personal computer with a Pentium IV 3.0 GHz processor/512 MB main memory. The instances with 5,683 and 7,740 tasks had 355 (6.2%) and 827 (10.6%) expired tasks, respectively. For the first instance, only 12.6% of the original duties needed modifications while the ratio increased to 29.5% for the second instance. The author could solve the first instance in approximately 9 hours and the second one in less than 16 hours.

The only approach for public bus transport we are aware of is described in [Dallaire et al., 2004]. However, the authors do not provide any details on their approach which is part of the commercial software package HASTUS/CrewOpt (see [GIRO Inc., 2007]). They rather emphasize the practical importance of generating efficient solutions that are similar to a reference crew schedule (when the underlying timetable is changed).

6.3. Mathematical Formulation

In this section, we will give the mathematical formulation that we will use in the remainder of this chapter. Recall that we assumed that drivers may only change their vehicles in depots (ex-urban scenario). Therefore, we propose to solve the independent crew scheduling problem (ICSP - see Section 6.1) first and, then, put the vehicle rotations from the crew scheduling solution together such that the vehicle schedule is feasible (see Section 2.1). In Section 6.4 we will seek to improve the regularity of crew schedules for the independent crew scheduling problem.

Let $\mathcal{T}$ be the set of tasks. Furthermore, we define K as the set of all feasible duties and $K(t), t \in \mathcal{T}$ as the set of duties that cover task t. The cost of duty $k \in K$ is denoted by c_k. Finally, decision variables x_k indicate whether duty k is selected in the solution or not. The ICSP can be formulated as set partitioning

problem:

$$\sum_{k \in K} c_k x_k \rightarrow \min \tag{6.3}$$

$$s.t. \quad \sum_{k \in K(t)} x_k = 1 \qquad \forall t \in \mathcal{T}, \tag{6.4}$$

$$x_k \in \{0,1\}. \tag{6.5}$$

The objective (6.3) is to minimize the total costs of the selected duties, and constraints (6.4) assure that each task will be covered by exactly one duty. When the equality sign in constraints (6.4) is replaced by a greater or equal sign "$\geq$", we obtain a set covering formulation. Then, tasks may be assigned to more than one driver where the additional drivers are passengers. The set covering formulation is computationally more attractive than the set partitioning formulation (see [Vanderbeck, 1994]). In the remainder of this chapter, we will consider a set covering formulation.

6.4. Solution Approaches

The purpose of this section is to present two solution approaches that improve the regularity of crew schedules compared to traditional crew scheduling. For both approaches we use model (6.3)-(6.5) and apply a column generation algorithm in combination with Lagrangian relaxation. We solve the corresponding Lagrangian dual with a subgradient algorithm (see Section 1.5.1) to obtain approximate dual values. The column generation pricing problem corresponds to a resource constrained shortest path problem (see Section 3.1) and is solved with a dynamic programming algorithm (see Section 3.2).

The columns generated in the column generation phase serve as input to the second phase where an appropriate integer solution is sought. In the following, we suggest two methods for the second phase that take the trade-off between costs and regularity into account. In particular, we propose a novel combination of local branching and follow-on branching in Section 6.4.1 while we discuss four bi-objective metaheuristics in Section 6.4.2.

6.4.1. Local Branching and Branching Rules

Our solution approach is based on the observation that (independent) crew scheduling problems have thousands of optimal solutions. This is mainly due to degeneracy.

In Table 6.1 we give the average number of optimal solutions for independent crew scheduling problems with 80,100, and 160 trips (tasks). We used the same instances and duty type definitions as in chapters 3 and 4. However, we enumerated at most 2,500 different optimal solutions per instance with the branch-and-bound implementation of ILOG CPLEX 9.1.3. The root node of the branch-and-bound tree was solved with a column generation algorithm, i.e., we did not regenerate columns during tree search. As we can see in Table 6.1, the average number of different optimal solutions can be very high in independent crew scheduling problems. Furthermore, the number of optimal solutions increases if a mere 0.01% deviation to the optimal solution value is allowed.

#trips	#instances	opt. tolerance	
	solved	0.00%	0.01%
80	10	1,052	1,115
100	9	723	945
160	9	1,807	2,046

Table 6.1.: Average number of optimal solutions for independent crew scheduling on Huisman data instances type A

The basic idea of our solution method is to systematically search an optimal solution among all optimal solutions that is as similar as possible to a given reference solution. In particular, we use *local branching cuts* to select suitable solution subspaces and explore these subspaces with an adapted version of *follow-on branching*. The exposition in this section is partly based on [Steinzen et al., 2007b].

Local Branching to Find Regular Crew Schedules

Local branching (see [Fischetti and Lodi, 2003]) is an exact solution method for general mixed integer programs. The basic idea of local branching is to define suitable solution subspaces that are efficiently explored with a generic MIP solver. In other words, *local branching cuts* are added to *strategically* define subspaces that are *tactically* explored with a black-box solver. The procedure can be viewed as a two-level branching scheme that aims at finding good incumbent solutions at early stages of the computation. The underlying assumption is that small instances of a problem can be efficiently solved with a generic solver while large instances cannot.

Given a feasible *start solution* $\bar{x} \in \{0,1\}^{|K|}$ of ICSP we define the Hamming distance

$$\Delta(x, \bar{x}) = \sum_{k \in L_0} (1 - x_k) + \sum_{k \in K \setminus L_0} x_k \tag{6.6}$$

where $L_0 = \{k \in K : \bar{x}_k = 1\}$ denotes the *support* of $\bar{x}$. The distance $\Delta(x, \bar{x})$ counts the number of variables in x that flip their values with respect to $\bar{x}$ (either from 1 to 0 or from 0 to 1). For a given neighborhood parameter $\kappa \in \mathbb{N}^+$, the solution space can be partitioned with local branching cuts:

$$\Delta(x, \bar{x}) \leq \kappa \qquad \text{(left branch)}, \tag{6.7}$$

$$\Delta(x, \bar{x}) \geq \kappa + 1 \qquad \text{(right branch)}. \tag{6.8}$$

For an appropriate value κ, subspace $\Delta(x, \bar{x}) \leq \kappa$ can be efficiently explored with a generic MIP solver. If the subspace contains a new incumbent $\bar{x}^2$, the scheme is reapplied to the right branch where two new subspaces are constructed: $\Delta(x, \bar{x}^2) \leq \kappa$ and $\Delta(x, \bar{x}^2) \geq \kappa + 1$. On the other hand, if subspace $\Delta(x, \bar{x}) \leq \kappa$ does not contain a new incumbent, the remaining (large) subspace $\Delta(x, \bar{x}) \geq \kappa + 1$ has to be explored with a MIP solver. Notice that the concept of local branching is quite different to standard branching: the solution method first explores promising solution subspaces instead of cutting fractional solutions. For further details on local branching we refer to [Fischetti and Lodi, 2003].

For independent crew scheduling, we use a local branching scheme to first explore regions of the solution space that contain solutions similar to a given reference crew schedule R. Similar to equation (6.1) let σ_k^p be the number of tasks of duty k that are not part of a regular pair. Then, we solve ICSP (possibly to optimality) with a modified objective function to obtain a start solution $\bar{x}$ as basis for local branching. The start solution should be similar to the reference crew schedule and should have sufficiently low costs. Therefore, we replace the original cost c_k of column k by $\hat{c}_k = c_k + \alpha\sigma_k^p$ and define α in such a way that σ_k^p dominates the modified cost. Finally, we restore the objective function and use $\bar{x}$ to define the initial neighborhood for local branching.

We would like to mention that the choice of parameter α is crucial for the performance of the solution procedure. If parameter α is too small, we get a start solution with low costs and low similarity. As a consequence, it is difficult to improve the similarity with local branching. On the other hand, if parameter α is too large, the computational burden to find a minimum cost solution can be very high. In our computational experiments we found that $\alpha \in [150, 400]$ is a robust parameter setting.

Follow-On Branching to Find Regular Crew Schedules

In order to simplify the exposition, we will briefly recall the basic idea of follow-on branching. Branching on follow-ons relies on a general branching strategy for set partitioning problems that was introduced by [Ryan and Foster, 1981]. The branching scheme is based on the following property. Given a fractional solution to a set partitioning problem, we can identify two rows (tasks) $t_i \in \mathcal{T}$ and $t_j \in \mathcal{T}$ such that the subset $K(t_i, t_j)$ of columns that contain t_i and t_j has the property

$$0 < \sum_{k \in K(t_i,t_j)} x_k < 1. \tag{6.9}$$

The remaining fraction of cover for each constraint must be provided by columns that do cover both rows at the same time. Thus, an effective constraint branching scheme is to require to cover two rows t_i and t_j by the same column on one branch and by different columns on the other. [Vance et al., 1997a] slightly modify the scheme to maintain tractability. They only consider trips (tasks/rows) t_i and t_j that correspond to trips operated consecutively in a duty (column). Furthermore, the authors show that this modification still constitutes a correct branching scheme. We refer to this strategy as *branching on follow-ons* since we impose which task can follow task t_i in the solution. Moreover, we refer to the task pair (t_i, t_j) as *follow-on*. Notice that each regular pair $S_i \in \bar{S}$ is also a follow-on. In the following, we will describe how follow-on branching is used to construct regular crew schedules.

A regular crew schedule contains as many regular pairs and chains as possible. We modify the follow-on branching scheme in such a way that an (cost) optimal solution has a high regularity as well. In the following, we will propose three novel adaptations of follow-on branching: branching on regular pairs (*fo-r1*), regular chains (*fo-r2*), and pieces of work (*fo-r3*).

Similar to our definition in Section 3.3.2 we define the support of a regular pair $(t_i, t_j) \in \bar{S}$:

$$g(t_i, t_j) = \sum_{k \in K(t_i,t_j)} x_k. \tag{6.10}$$

Since we aim at generating regular crew schedules we branch on a candidate regular pair $(t_i, t_j) \in \bar{S}$ where $0 < g(t_i, t_j) < 1$ is satisfied. Branching scheme *fo-r1* selects the regular pair with the best support among all regular pairs.

$$\textit{fo-r1}: \qquad (t_i, t_j) = \arg \max_{(t_i,t_j) \in \bar{S}} g(t_i, t_j) \tag{6.11}$$

However, if $\bar{S} = \emptyset$ we choose the follow-on with $t_i, t_j \in \mathcal{T}$ and $\max g(t_i, t_j)$.

Branching scheme *fo-r2* does not rely on the support of single regular pairs, but tries to fix regular chains of maximum length. Recall that $\bar{T}$ is associated with the set of regular chains. Furthermore, we associate $K(T_i)$ with the set of duties that cover regular chain T_i. The set of candidate regular chains $\bar{T}_c$ contains all regular chains $T_i \in \bar{T}$ where $0 < g(T_i) < 1$ with $g(T_i) = \sum_{k \in K(T_i)} x_k$ is satisfied. Algorithm 11 depicts branching scheme *fo-r2* where we try to branch on a regular chain of maximum length if there are candidate chains.

Algorithm 11: Branching on regular chains (*fo-r2*)

Find candidates
Compute set of candidate regular chains $\bar{T}_c = \{T_i : 0 < g(T_i) < 1\}$.

Branching
if $\bar{T}_c \neq \emptyset$ **then**
  Branch on follow-on $t_i, t_j \in \mathcal{T}$ with $\max g(t_i, t_j)$
else
  Initialize $\bar{T}_c^{\max} = \{T_i \in \bar{T}_c : |T_i| = \max_{T_j \in \bar{T}_c} |T_j|\}$
  Branch on regular chain $T_i \in \bar{T}_c^{\max}$ with $\max g(T_i)$

Notice that scheme *fo-r2* corresponds to the latter scheme *fo-r1* if the set of candidate regular chains $\bar{T}_c$ only consists of chains of length two.

Finally, we propose branching scheme *fo-r3* where we branch on a piece of work whenever that piece of work forms a regular chain. If several pieces correspond to candidate regular chains, we select the piece with the maximum number of tasks. Algorithm 12 presents how branching on regular pieces of work is performed.

Local and Follow-On Branching to Find Regular Crew Schedules

Local branching and follow-on branching can be combined. In particular, we embed follow-on schemes *fo-r1* to *fo-r3* into local branching to explore neighborhoods $\Delta(x, \bar{x}) \leq \kappa$. We hope to explore neighborhoods $\Delta(x, \bar{x}) \leq \kappa$ in such a way that (1) an new incumbent is found fast and (2) the new incumbent has a smaller distance than other solutions in the neighborhood. If the reference solution is of high quality, a valuable follow-on might be selected first and might reduce the computational time to explore the neighborhood. To sum up, we *strategically* define subspaces with local branching and *tactically* explore them with follow-on branching.

Algorithm 12: Branching on regular pieces of work (*fo-r3*)

Find candidates
Compute set of candidate regular chains $\bar{T}_c = \{T_i : 0 < g(T_i) < 1\}$.

Branching
if $\bar{T}_c \neq \emptyset$ **then**
  Branch on follow-on $t_i, t_j \in \mathcal{T}$ with $\max g(t_i, t_j)$
else
  if $\exists T_i \in \bar{T}_c : T_i$ *is piece of work* **then**
    Initialize $\bar{T}_{cp} = \{T_i \in \bar{T}_c : T_i$ *is piece of work*$\}$
    Branch on regular chain $T_i \in \bar{T}_{cp}$ with $|T_i| = \max_{T_j \in \bar{T}_{cp}} |T_j|$ and $\max g(T_i)$
  else
    Initialize $\bar{T}_c^{\max} = \{T_i \in \bar{T}_c : |T_i| = \max_{T_j \in \bar{T}_c} |T_j|\}$
    Branch on regular chain $T_i \in \bar{T}_c^{\max}$ with $\max g(T_i)$

6.4.2. Bi-Objective Metaheuristics

In this section, we explicitly consider regularity as second objective function in the integer phase instead of implicitly seeking regular crew schedules as in the preceding section. In particular, we extend model ICSP by a second objective function. The exposition in this section is partly based on [Suhl et al., 2007].

The consideration of both cost and regularity as objective functions leads to the following bi-objective set partitioning problem (2ICSP):

$$\sum_{k \in K} c_k x_k \quad \rightarrow \quad \min \tag{6.12}$$

$$\sum_{k \in K} \sigma_k x_k \quad \rightarrow \quad \min \tag{6.13}$$

$$s.t. \qquad \sum_{k \in K(t)} x_k \quad = \quad 1 \qquad \forall t \in \mathcal{T}, \tag{6.14}$$

$$x_k \quad \in \quad \{0, 1\}, \tag{6.15}$$

where σ_k denotes the distance of duty k to reference schedule R. Model 2ICSP corresponds to model ICSP except the additional objective function (6.13) where we minimize the distance to the reference solution.

In *multicriteria (multiobjective) optimization* (see [Ehrgott, 2005]), we look for *Pareto optimal* solutions instead of seeking optimal solutions as in the single objective case. We call a solution $\widetilde{x}$ Pareto optimal, if there is no other solution that is at least as good as $\widetilde{x}$ with respect to both objective functions and strictly better

with respect to one objective. If $\widetilde{x}$ is Pareto optimal, $\widetilde{z} = (\sum_{k \in K} c_k \widetilde{x}_k, \sum_{k \in K} \sigma_k \widetilde{x}_k)$ is said to be *efficient*. In other words, we want to identify a set of efficient crew schedules where for each crew schedule a reduction in one objective would necessarily lead to an increase in the other objective. Optimization problems with multiple objectives usually have many efficient solutions while our approach from the preceding section returns a single (at least cost-effective) crew schedule.

We now have to choose an appropriate solution approach to solve our bi-objective problem. Multiobjective metaheuristics have been successfully applied to multiobjective optimization problems in general (see [Gandibleux et al., 2004]) and multiobjective crew scheduling problems in particular (see [Lourenço et al., 2001]). In the following, we will briefly describe four well-known metaheuristics from literature and adapt the methods to the bi-objective set covering problem stated above. All methods approximate the set of efficient solutions. We discuss an evolutionary algorithm, a tabu search method, a simulated annealing approach, and an ant colony optimization algorithm. First and foremost, the purpose of our study is to compare the performance of different multiobjective metaheuristics on our particular problem. For a recent survey on heuristic methods for multiobjective optimization the reader is referred to [Ehrgott and Gandibleux, 2004].

Strength Pareto Evolutionary Algorithm (SPEA2) The improved Strength Pareto Evolutionary Algorithm (SPEA2) proposed by [Zitzler et al., 2002] is an evolutionary algorithm (see Chapter 4) to approximate the set of Pareto optimal solutions. Their method is an enhancement of [Zitzler and Thiele, 1999]. The basic idea of their approach is to use the dominance criterion for fitness calculation and selection of solutions. Furthermore, non-dominated solutions are stored in an external archive, i.e., independent from the current population. The authors present promising results of their algorithm as compared with other evolutionary approaches. In our implementation, we use a binary representation of solutions and apply the genetic operators proposed by [Beasley and Chu, 1996].

Multiobjective Tabu Search Tabu search (see [Glover and Laguna, 1993]) is a local search method where a selective history of the search states is stored. In its simplest form a tabu list is used to prevent the search method to get stuck in local optima. Our implementation is based on [Lourenço et al., 2001] who suggested a multiobjective tabu search method for the crew scheduling problem. Basically, they use a weighted scalarizing function to represent multiple objectives. Furthermore, the authors propose an optimized intensification strategy and insert as well as remove tabu lists. In

addition to tabu lists, we apply the greedy heuristic of [Caprara et al., 1999] to construct new feasible solutions. The construction heuristic requires to initially solve a Lagrangian dual problem with a subgradient method (see Section 1.5.1).

Multiobjective Simulated Annealing Simulated Annealing (see [Dowsland, 1993]) is a local search method that explores the neighborhood of an incumbent solution and allows a worse solution to be accepted as starting point for further exploration. The acceptance rate is based on the quality of the solution and the computational time spent. Accepting worse solutions allows a simulated annealing method to backtrack from local optima. The first multiobjective version of simulated annealing was introduced by [Serafini, 1992] where the author considered several multiobjective acceptance rules. In our implementation, we considered acceptance rule M which is a mixture of the Chebyshev and product rule. Furthermore, we apply the heuristic of [Caprara et al., 1999] to construct neighboring solutions.

Multiobjective Ant Colony Optimization Ant Colony Optimization (see [Dorigo and Stützle, 2004]) is a metaheuristic that is inspired by the behavior of real ants. In nature, ants indirectly communicate by means of pheromone trails in order to find the shortest path between their ant hill and the food source. The shorter the path between nest and food source is, the more ants can use it in a given timeframe, and, as a consequence, the stronger the pheromone trail will be. Our implementation is based on the population-based ant colony optimization approach of [Guntsch and Middendorf, 2003]. However, our implementation differs from theirs in the following way. We alternately construct new solutions using cover costs (see [Marchiori and Steenbeek, 2003]) and the greedy heuristic of [Caprara et al., 1999].

In Section 6.5 we provide computational results concerning the quality of the approximated Pareto fronts of the heuristics stated above. In the following, we illustrate how the Pareto front can be used to support planners while assessing the trade-off between cost and regularity.

A Basic Decision Support System

In Figure 6.1 we present a basic interface of a decision support system to evaluate the trade-off between costs and regularity. The major purpose of the system is to illustrate the trade-off in a straightforward way. In the upper right part of the system, the solutions of the current Pareto front are shown where the

distance measure is shown on the vertical axis and costs are displayed on the horizontal axis. Notice that a high regularity corresponds to a small distance, i.e., in our example low cost solutions have a low regularity as well. Furthermore, the approximated Pareto front allows the planner to estimate the actual trade-off between costs and regularity while the branching approach described in Section 6.4.1 basically provides a single (at least cost-effective) solution.

In addition to the front shown in the upper right part of the interface, the parameters of the metaheuristics can be altered in the upper left part. The current status is displayed in the lower part when the system is running. During runtime the current front is regularly updated. Moreover, if the heuristic uses a scalarizing weighted objective function, the planner may interactively direct the search of the metaheuristics by changing the relation of cost and regularity/distance (see Figure 6.2).

6.5. Computational Results

We test our approaches on real-world and randomly generated data instances. We consider two real-world and eight randomly generated data instances. The artificial instances were generated as described in [Huisman, 2004]. However, all instances have a single depot and drivers may only change their vehicle in that depot. We make these assumptions in order to reflect a typical ex-urban scenario (see Section 6.1). Furthermore, we assume that a reference crew schedule is known for each data instance.

In Table 6.2 we give details on the data instances that result from solving the linear relaxation of the ICSP with a column generation algorithm. The last two instances correspond to real world problems while the others were randomly generated. We report the ratio of irregular trips in percent ($\%irr$), the number of rows ($\#rows$), columns ($\#cols$), and non-zeros ($\#nnz$). For each data instance the ratio of irregular trips refers to number of new trips, i.e. trips that are not in the reference schedule, compared to the total number of trips. In the second part of the table we give details on the column generation phase: the number of iterations ($\#iter$), and the computational time spend on master (cpu_ma) and pricing problem (cpu_pr). To maintain comparability between both approaches, we used operating costs as single objective in the column generation phase.

Notice that a direct comparison between both approaches is not possible. Our branching scheme provides a single solution while the bi-objective metaheuristics return a set of Pareto optimal solutions. Therefore, we first give results on our branching scheme and then on the bi-objective metaheuristics.

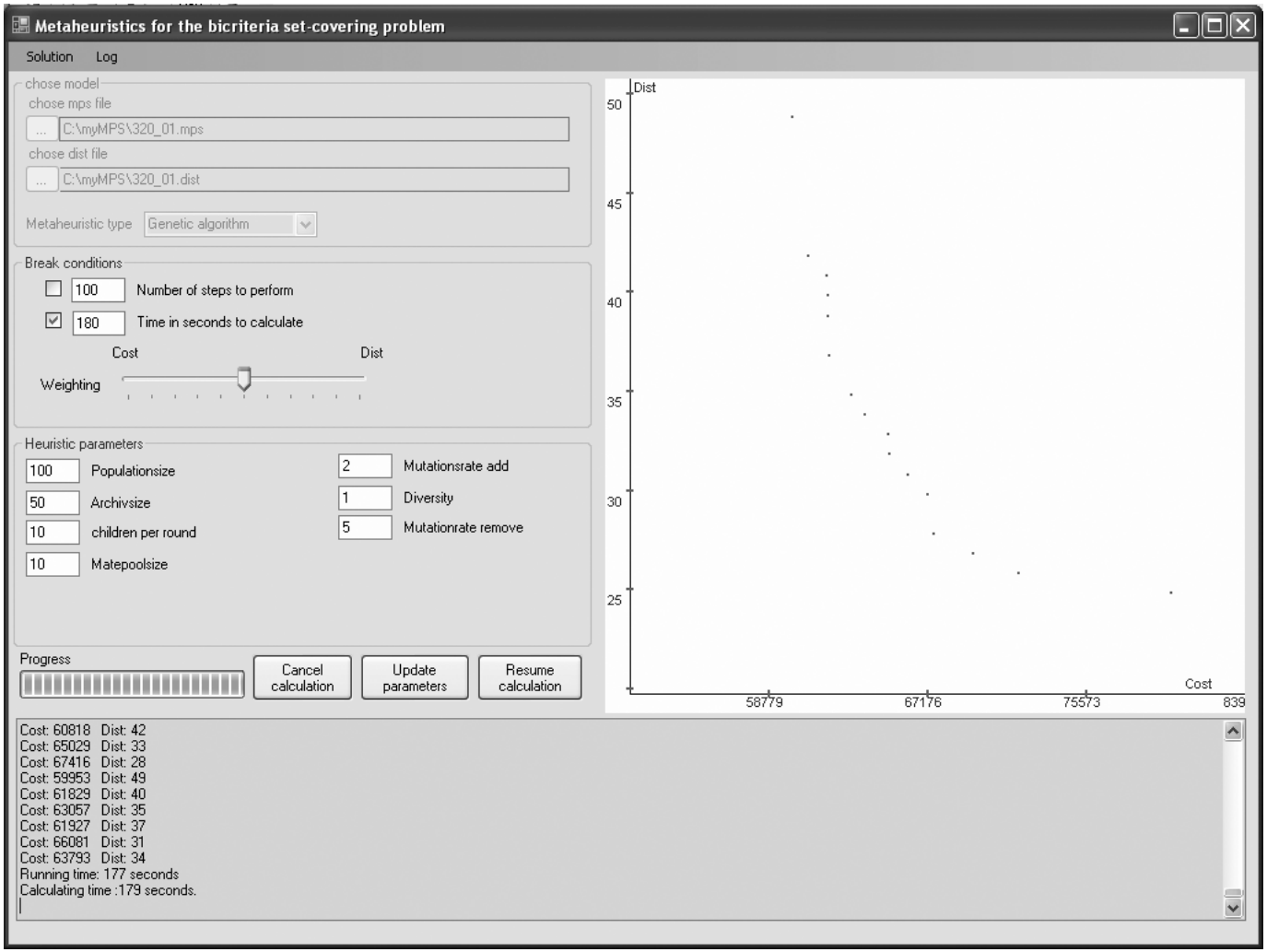

Figure 6.1.: Basic decision support system to estimate the trade-off between costs and regularity

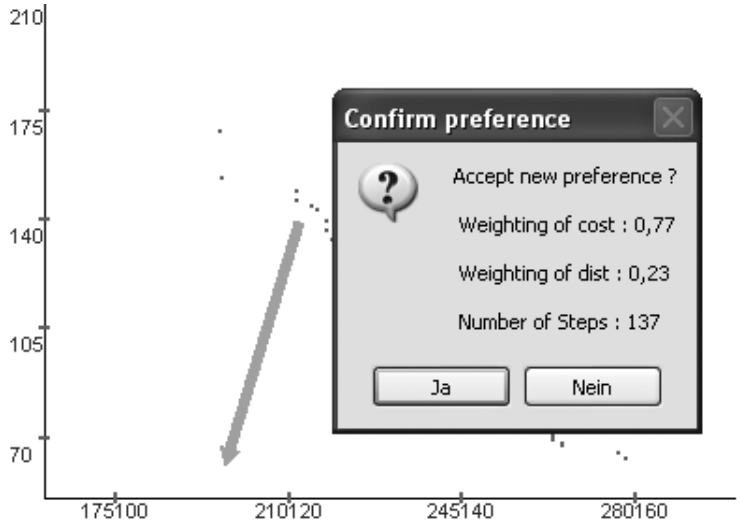

Figure 6.2.: The search direction of the metaheuristic can be interactively changed if a scalarizing weighted objective function is used

Local Branching and Branching Rules

In addition to the assumptions stated above we apply the following parameter settings for our branching approach:

- The computational time to find an integer solution is limited to 2 hours (7,200 seconds).
- In our local branching implementation, at most 20% of the variables of the incumbent may flip their values. Furthermore, the computational time to explore subspaces $\Delta(x, \bar{x}^i) \leq \kappa$ (left branches) is limited to 15 minutes (900 seconds). If the time limit is reached and no new incumbent is found, we reduce the size of the subspace by 50% to speed-up its exploration. For further details we refer to [Fischetti and Lodi, 2003].

All computational experiments with the branching schemes were performed on a personal computer running Windows XP with an Intel Pentium IV 2.2 GHz processor and 2 GB of main memory.

In Table 6.3 we show results on the regularity of crew schedules when we apply local branching (*locbr*) and follow-on branching (*fo-r1, fo-r2, fo-r3*) as described in Section 6.4.1. Furthermore, we compare our method with the default branch-and-bound implementation of ILOG CPLEX 9.1.3 (*cpx-def*) and local branching in combination with default branching of CPLEX (*locbr_cpx-def*). For each method we give the average over the ten instances described in Table 6.2.

instance	%irr	#rows	#cols	#nnz	#iter	cpu_ma	cpu_pr
art320_1	5.0	320	100,944	857,215	31	245	140
art320_2	5.0	320	60,128	384,478	21	143	85
art400_1	5.0	400	72,673	459,906	22	125	122
art400_2	5.0	400	57,769	352,592	21	130	77
art640_1	5.0	640	156,044	1,227,320	41	1,006	1,673
art640_2	5.0	640	104,595	643,113	28	572	695
art800_1	5.0	800	135,572	852,337	37	1,060	2,054
art800_2	5.0	800	162,209	1,158,539	39	1,773	2,887
real430	4.4	430	98,710	1,204,084	31	391	297
real433	4.8	433	103,516	1,236,954	31	411	257

Table 6.2.: Description of data instances

In Table 6.3 we report the computational time in seconds spent in the second (integer) phase (*cpu_ip*), the optimality gap in percent (%gap) and three regularity measures. The regularity measures are defined as follows. The percentage of preserved duties (*%prd*) refers to the percentage of duties in the new crew schedule that could be (exactly) kept from the reference crew schedule. Likewise we define the percentage of preserved regular pairs (*%prp*). The average regular chain length of a crew schedule corresponds to the average number of regular tasks in a duty. In this context, the percentage of the average chain length (*%avgcl*) refers to the average regular chain length of the new crew schedule compared with average regular chain length of the reference crew schedule. For example, if the reference schedule has on the average 8 regular tasks per duty, and the average regular chain length in the new crew schedule is 4 tasks, then $avgcl = \frac{4}{8} = 50\%$.

As can be seen from Table 6.3 branching scheme *fo-r1* provides the best results in terms of solution time and solution quality. Recall that objective function and, thus, solution quality refer to operational costs. On the other hand, local branching considerably improves the regularity of the new crew schedules, e.g., the number duties that can be kept from the reference. Basically, we generally observe an increase of solution time and decrease of solution quality if local branching is used. However, local branching in combination with scheme *fo-r1* gives a better solution quality than the default version of CPLEX. To sum up, we conclude that local branching effectively improves the regularity while follow-on branching scheme *fo-r1* is well suited to improve solution quality and time. The combination of both methods leads to improved solutions in terms of

			regularity measures		
method	cpu_ip	%gap	%prd	%prp	%avgcl
cpx-def	2,437	1.93	6.3	53.5	31.0
fo-r1	2,095	0.42	7.7	54.4	31.2
fo-r2	3,649	2.20	8.2	56.8	33.7
fo-r3	4,247	2.81	6.6	55.0	32.5
locbr_cpx-def	6,420	2.60	27.4	79.0	50.1
locbr_fo-r1	5,492	1.55	28.0	80.2	51.2
locbr_fo-r2	5,806	3.81	32.3	81.1	54.5
locbr_fo-r3	6,270	3.70	25.6	80.0	51.2

Table 6.3.: Results on regularity of crew schedules for branching approaches

both cost and regularity compared to a traditional approach with CPLEX.

Bi-Objective Metaheuristics

Typically, performance of optimization algorithms is assessed on both computational time consumed and solution quality. For the single objective case, it is common practice to monitor the computational time and to define quality by means of the value of the objective function. As to the computational effort, multiobjective optimization algorithms can be evaluated in the same way as single objective methods. However, if there are multiple objectives, an algorithm returns a set of non-dominated solutions. Obviously, we cannot compare two sets of solutions in the same straightforward way as two single solutions: solutions in either set may be dominated by solutions in the other set and other solutions may be incomparable.

The *hypervolume measure* (see [Zitzler and Thiele, 1998] and [Fleischer, 2003] for a multidimensional generalization) is one of the most commonly applied measures to compare the results of multiobjective optimization algorithms. If there are two minimizing objective functions and upper bounds for both objectives given, the hypervolume measures the area covered by the non-dominated solutions. In Figure 6.3 we illustrate the hypervolume for five non-dominated solutions and two objective functions with upper bounds u_1 and u_2. The hypervolume indicator allows to infer that an approximate set is not worse than another, but it does not provide an indication how much better the approximation actually is (see [Zitzler et al., 2003]). In the following, we will compare the bi-objective metaheuristics described in Section 6.4.2 using the hypervolume

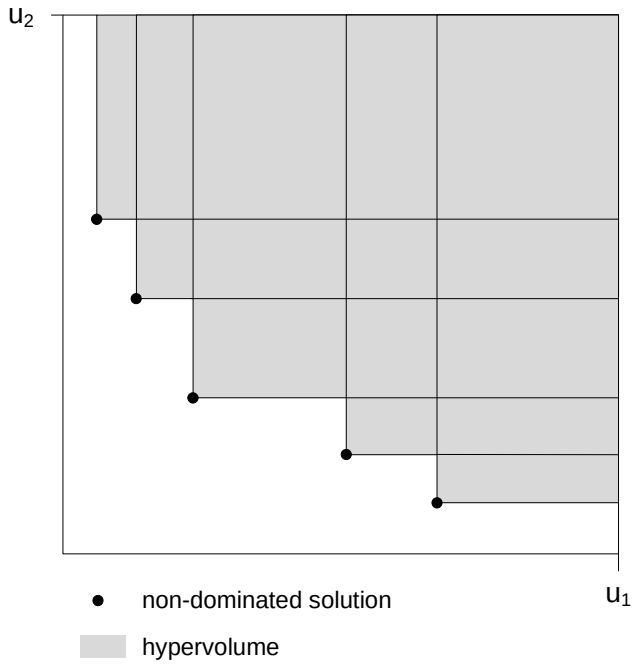

Figure 6.3.: Example of hypervolume measure for five non-dominated solutions and two objectives

measure. All computational experiments were performed on a personal computer running Windows XP with an Intel Pentium IV 3.0 GHz processor and 2 GB of main memory. The computational time for each run was limited to 10 minutes since our primary goal is to provide a quick estimate for the trade-off.

In order to restrict the impact of random effects, we repeated our experiments ten times for each test problem and algorithm. The hypervolume of a particular algorithm corresponds to the average over all ten runs. We build a list for each instance ranking the algorithms where we put the algorithm with the largest volume on the first position, the second largest volume on the second position, and so on. In Table 6.4 we report how often a metaheuristic held a particular position in the hypervolume ranking. In particular, we give results for simulated annealing (SA), tabu search (TS), evolutionary algorithm (EA), and ant colony optimization (ACO).

While there is a clear performance gap between the EA and ACO as well as ACO and TS/SA, the fronts achieved by SA and TS are rather close together.

In Figures 6.4 to 6.7 we present an approximate Pareto front generated by the metaheuristics for each of four instances. Furthermore, we give the solutions obtained with CPLEX (*cpx-def*) and local branching in combination with follow-on branching version 1 (*locbr_fo-r1*) as described in the preceding section. Finally,

algorithm	#hv positions			
	1st	2nd	3rd	4th
SA	–	–	5	5
TS	–	–	5	5
EA	10	–	–	–
ACO	–	10	–	–

Table 6.4.: Number of positions held in hypervolume ranking by bi-objective metaheuristics on test set (10 instances)

we give grey lines for the minimum distance and operational cost if the corresponding second objective is omitted. Notice that all figures are differently scaled and that GA corresponds to the results of the evolutionary algorithm SPEA2.

Approximate Pareto fronts computed with the EA contain many high-quality solutions that are well-spread. The fronts of the ACO approach comprises many solutions that are usually of average quality but better spread than those of the EA. The simulated annealing algorithm produces many low cost solutions. As a consequence, the approximated Pareto front is primarily located in low cost areas. The tabu search method generates approximated fronts with few low-cost solutions that are very close to each other. We conclude that the evolutionary algorithm SPEA2 is well-suited to provide an estimate for the trade-off between costs and regularity in a short timeframe. Finally, the figures show that the default version of CPLEX and local branching in combination with follow-on branching provide (almost) optimal solutions concerning costs. However, especially SPEA2 could always find solutions with lower distance (but higher costs). As a consequence, we believe that the bi-objective metaheuristics can provide reasonable, additional information for the planner to assess the quality of a solution concerning regularity. This additional information can be generated in a short timeframe.

6.6. Summary

In this chapter, we discussed the ex-urban vehicle and crew scheduling problem with a single depot and irregular timetables. Unless specifically imposed, traditional vehicle and crew scheduling usually produces irregular crew schedules which are undesired in practice. We presented two solution approaches that improve the regularity of crew schedules compared to traditional crew scheduling.

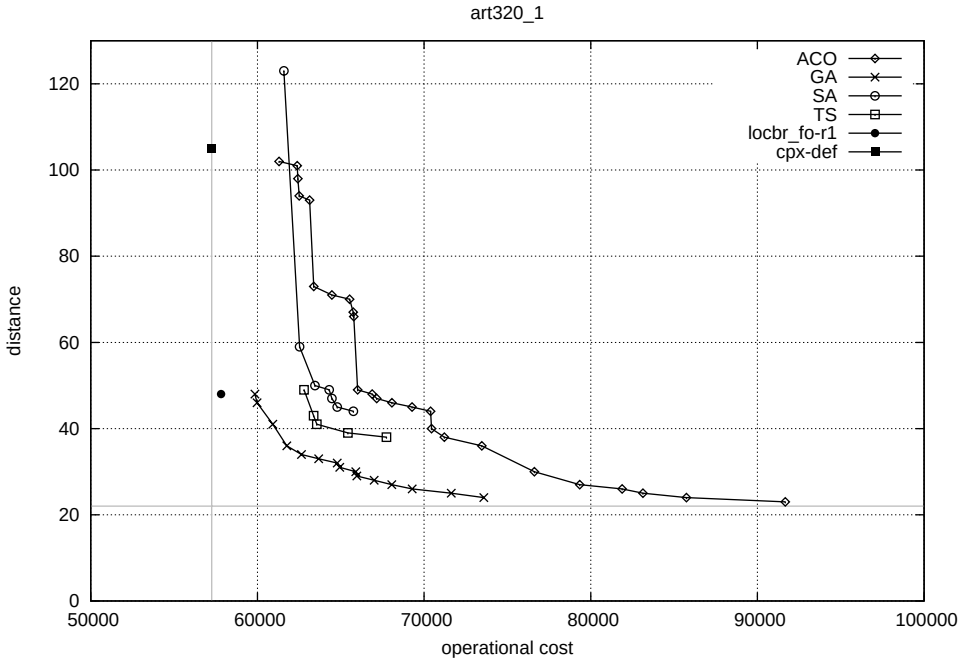

Figure 6.4.: Approximate Pareto fronts for instance art320_1 generated by metaheuristics compared to branching schemes

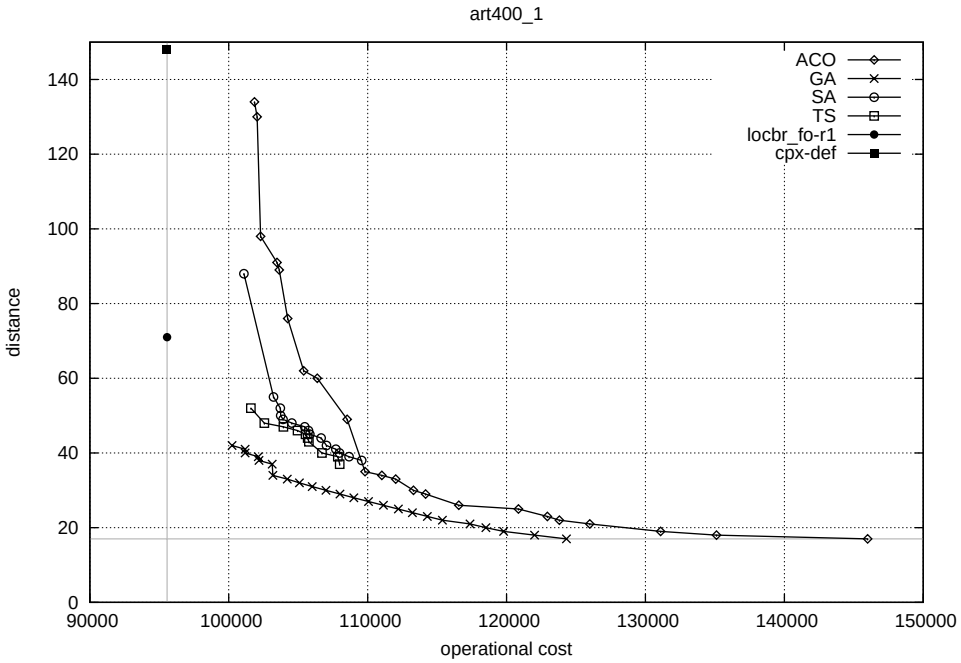

Figure 6.5.: Approximate Pareto fronts for instance art400_1 generated by metaheuristics compared to branching schemes

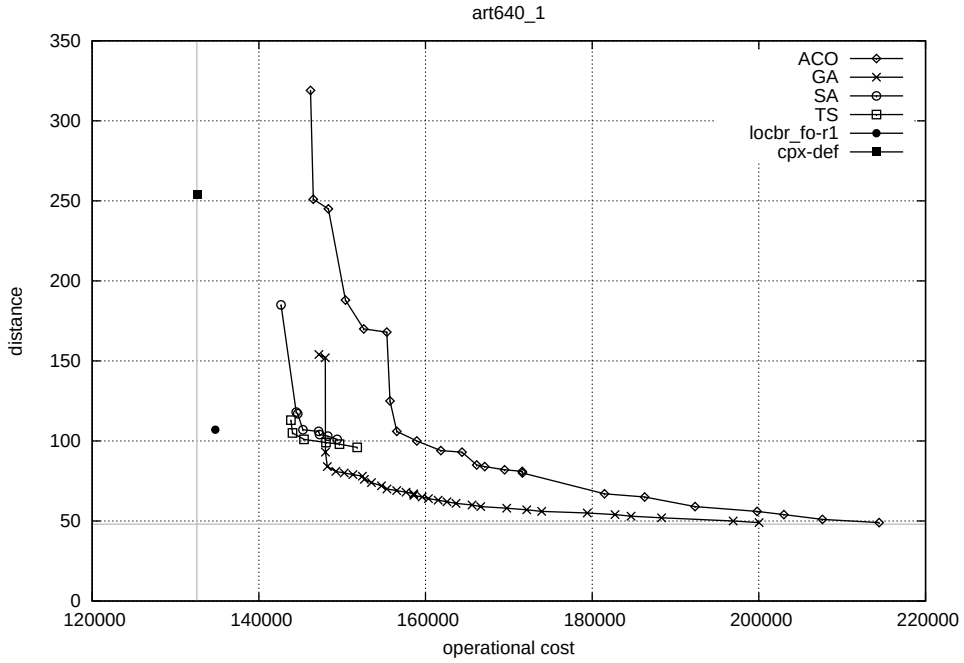

Figure 6.6.: Approximate Pareto fronts for instance art640_1 generated by metaheuristics compared to branching schemes

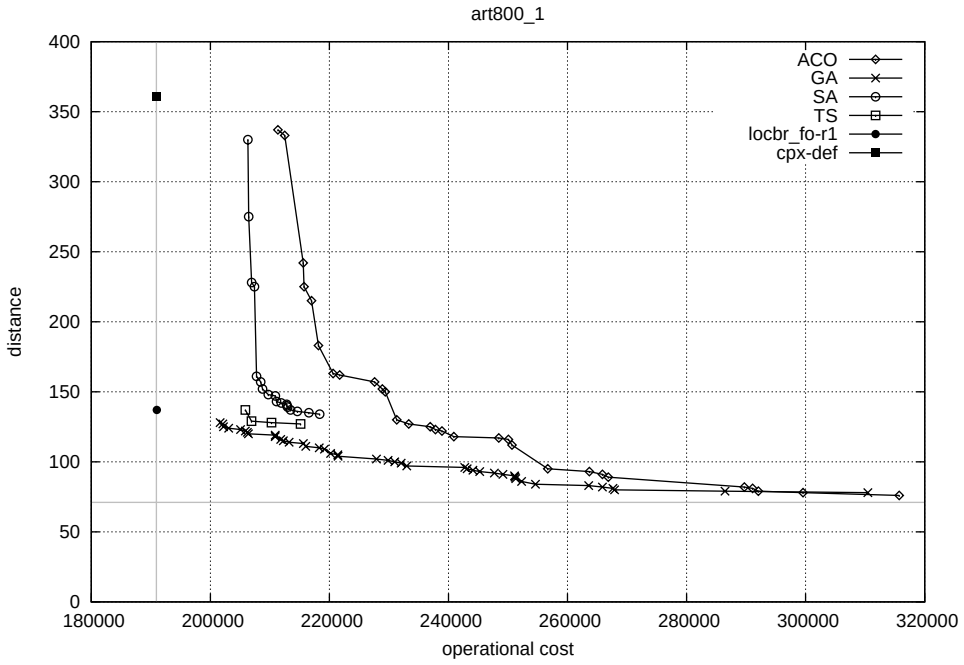

Figure 6.7.: Approximate Pareto fronts for instance art800_1 generated by metaheuristics compared to branching schemes

In particular, we proposed a novel combination of local branching and follow-on branching. Furthermore, we showed how bi-objective metaheuristics can be used to quickly estimate the quality of the solution generated with the latter approach.

Finally, a computational study that involved randomly generated and real-life data showed the applicability of the proposed techniques. In fact, our branching scheme lead to improved solutions in terms of both cost and regularity compared to a traditional approach with CPLEX.

7. Summary and Concluding Remarks

In this thesis we addressed the integrated multiple-depot vehicle and crew scheduling problem in public bus transport. Vehicle and crew scheduling are two major planning problems that basically aim at assigning scheduled trips of a given timetable to vehicle and crew itineraries. For several years now, Operations Research has been successful for solving both planning problems. Traditionally, both planning steps have been approached sequentially where vehicle schedules are determined before crew schedules. However, the integrated consideration of vehicle and crew scheduling has received considerable attention over the past years. Several authors have shown that the integrated treatment of both planning steps discloses additional flexibility that can lead to gains in efficiency compared to sequential planning.

In Chapter 1 we introduced the vehicle and crew scheduling problem and provided the necessary background for combinatorial optimization problems and solution techniques. In the following Chapter 2 we defined the integrated multiple-depot vehicle and crew scheduling problem in public transport. We reviewed models and solutions techniques that are used in literature for sequential, partially integrated, and fully integrated vehicle and crew scheduling. Furthermore, we thoroughly described the modeling approach, mathematical formulation, and solution approach that provided the starting point for the following chapters of the thesis. The two-phase solution approach is based on column generation in combination with Lagrangian relaxation. In the first phase a lower bound is computed while feasible solutions are constructed in the second phase.

The main contributions of this thesis for the integrated vehicle and crew scheduling problem with multiple depots were described in Chapter 3. More specifically, we proposed an approach for the column generation pricing problem that involved two novel network formulations for a decomposed pricing problem. We showed that the network complexity of our approach is beneficial compared to other approaches previously exposed in literature. We applied a dynamic programming method to solve the pricing problem. In this context, we discussed known as well as novel adaptations of preprocessing and acceleration

techniques that were essential to solve large problem instances. Furthermore, we discussed three solution methods to construct integer solutions, namely a Lagrangian heuristic, a branch-and-bound method, and a novel heuristic branch-and-price method. Basically, the Lagrangian heuristic generated good quality solutions in a short timeframe while the branch-and-price heuristic provided high quality solutions at high computational costs. The branch-and-bound method appeared to be inappropriate for solving large instances. Finally, we presented a new model variation where drivers are not tied to vehicles from a single depot and can change their vehicle whenever there is a relief point (unrestricted changeovers). Our computational study involved real-world and randomly generated benchmark instances with up to 653 trips and four depots. The experiments showed the effectiveness of our approach. In this context, we presented previously unknown solutions for the widely used benchmark instances of [Huisman, 2003]. In fact, the results indicated that our method outperformed other approaches from literature in terms of computational time and solution quality. Furthermore, we solved benchmark instances with 640 trips and four depots. To the best of our knowledge, randomly generated instances of that type and size have not been tackled before. We obtained similar results for the model variation with unrestricted changeovers.

In Chapter 4 we dealt with a novel hybrid evolutionary algorithm to tackle integrated vehicle and crew scheduling problems. Our method combined mathematical programming techniques with an evolutionary algorithm. We applied an evolutionary algorithm to find a good trip-depot assignment where the fitness of an individual is evaluated using column generation in combination with Lagrangian relaxation. The computational experiments were performed with the randomly generated benchmark instances that have been used in the preceding chapter. We compared different versions of the evolutionary algorithms with each other, with the traditional sequential approach, and with an integrated treatment of both planning steps. The results indicated that medium-sized problem instances with multiple depots can be solved by using the evolutionary algorithm. Furthermore, our approach disclosed significant savings compared to the traditional sequential approach without requiring a fully integrated solution method. Although our algorithm performed worse than the best known integrated algorithm, it proved to be competitive with other integrated approaches from literature especially for medium-sized instances.

In Chapter 5 we considered practical rules and regulations arising in public transport companies in Germany. We suggested extensions and modifications of our modeling and solution approach from Chapter 3 to cover these practical extensions. The enhancements included driving time constraints, complex break

rules where many pieces of work are allowed, break positions, and duty mix constraints. Furthermore, we gave an overview of how our implementation is being integrated in the commercial software tool interplan®. We tested the applicability of the proposed techniques using real-life data instances. The results on instances with up to 653 trips and four depots indicate that our approach can efficiently cover duty types with many pieces of work and complex feasibility rules.

In Chapter 6 we did not only focus on how to conduct operations at minimum cost but also on another aspect which is related to the quality of crew schedules. In practice, timetables consist of many trips serviced every day and some exceptions that do not repeat daily. In other words, timetables in practice are irregular and, unless specifically imposed, traditional vehicle and crew scheduling usually produces irregular crew schedules which are undesired in practice. Therefore, we addressed the ex-urban vehicle and crew scheduling problem with irregular timetables. We proposed two approaches that capture both costs and regularity of crew scheduling solutions. More specifically, we suggested a novel combination of local branching and follow-on branching that improves the regularity of crew schedules while cost optimality is maintained. Furthermore, we compare four bi-objective metaheuristics that take both cost and regularity as objective functions. The latter approach can be used to get a quick estimate of the solution quality obtained with the first approach. Our computational study with real-world and artificial instances showed that the branching approach led to improved solutions in terms of both cost and regularity compared to a traditional approach.

At the beginning of this thesis (see Section 1.6) we stated three research objectives. In short, our objectives were (1) to develop models and techniques for the integration of vehicle and crew scheduling that allow to tackle large problem instances, (2) to develop models and techniques to increase the regularity of crew schedules when timetables are irregular, and (3) to test the applicability of the proposed techniques in practice. From our perspective these objectives have been achieved. In Chapters 3 and 5 we approached the first objective. We obtained promising results concerning the effectiveness of our methods for large problem instances. Furthermore, we modeled complex duty feasibility rules in Chapter 5. With respect to the second objective, we suggested models and techniques to increase the regularity of crew scheduling solutions in Chapter 6. Our computational results for the partially integrated (ex-urban) vehicle and crew scheduling indicate that the regularity can be improved while maintaining cost optimality. However, we left a fully integrated consideration for future research. We devoted Chapter 5 and in part Chapter 3 to achieve the last objective. We tested our approaches on real-world instances and showed their effectiveness. Furthermore,

our methods are being integrated in the commercial software package interplan® for public transport companies.

Finally, we would like to make some suggestions for future research in the field of vehicle and crew scheduling. Although some progress has been made over the past years, we are not aware of an approach that could deal with 1,000s or even 10,000s of trips. However, problem instances of such size with many depots are common in big cities such as the German towns of Munich, Hamburg, or Berlin. Therefore, we suggest to pursue further research on faster solution procedures for integrated problems. Moreover, the partial integration of vehicle scheduling and timetabling results in gains in efficiency (see [Kliewer et al., 2006a]). Hence, we deem it worthwhile to include timetable considerations into the integrated treatment of multiple-depot vehicle and crew scheduling. Finally, we suggest to continue research on aspects related to the quality of vehicle and crew schedules such as robustness or quality of work conditions.

A. Definitions and Abbreviations

In this appendix we summarize definitions and abbreviations that we have introduced throughout this thesis.

Definitions

depot	maintenance and storage facility where buses may be parked and serviced when not in use
(service) trip	vehicle activity with passengers and defined by start and end locations and times
deadhead (trip)	vehicle activity without carrying passengers such as movements or idle times outside the depot (or both)
pull-in trip	moves a vehicle from the depot to the start location of the first trip of a vehicle block
pull-out trip	moves a vehicle from the end location of the last trip of a vehicle block to the depot
compatible trips	two trips that can be covered consecutively by the same vehicle
vehicle block	sequence of compatible trips that can be executed by a single vehicle, starts with a pull-in and ends with a pull-out trip
relief point	defines a location and time where a driver may change his vehicle
task	elementary portion of work between two relief points that can be assigned to a driver
piece (of work)	sequence of tasks without a (long) break for which a driver stays with the same vehicle
duty	sequence of pieces of work that can be assigned to an anonymous driver and satisfies a wide variety of regulations
changeover	change of a vehicle of a driver

continuous attendance	a driver is required to be present if a bus is outside of a depot

Abbreviations

ACO	ant colony optimization
CSP	crew scheduling problem
EA	evolutionary algorithm
IP	integer program
LDP	Lagrangian dual problem
LP	linear program
MCFP	minimum cost flow problem
MDVSP	multiple-depot vehicle scheduling problem
MDVCSP	integrated multiple-depot vehicle and crew scheduling problem (formulation (2.11)-(2.16))
MDVCSP-H	integrated multiple-depot vehicle and crew scheduling problem (formulation (2.1)-(2.10))
MDVCSP-C	integrated multiple-depot vehicle and crew scheduling problem with unrestricted changeovers (formulation (3.42)-(3.47))
MFP	multicommodity flow problem
MP	master problem
RCSP	resource constrained shortest path problem
REF	resource extension function
RMP	restricted master problem
SA	simulated annealing
SCP	set covering problem
SDVSP	single-depot vehicle scheduling problem
SP	shortest path problem
SPP	set partitioning problem
TS	tabu search
TSN	time-space network
VCSP	integrated vehicle and crew scheduling problem

List of Figures

List of Tables

List of Algorithms

Bibliography

[Achterberg et al., 2005] Achterberg, T., Koch, T., and Martin, A. (2005). Branching rules revisited. *Operations Research Letters*, 33:42–54.

[Ahuja et al., 1993] Ahuja, R. K., Magnanti, T. L., and Orlin, J. B. (1993). *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, New Jersey.

[Aneja et al., 1983] Aneja, Y., Aggarwal, V., and Nair, K. (1983). Shortest chain subject to side constraints. *Networks*, 13(2):295–302.

[Balas and Padberg, 1976] Balas, E. and Padberg, M. (1976). Set partitioning: A survey. *SIAM Review*, 18(4):710–760.

[Ball et al., 1983] Ball, M., Bodin, L., and Dial, R. (1983). A matching based heuristic for scheduling mass transit crews and vehicle. *Transportation Science*, 17(1):4–31.

[Barahona and Anbil, 2000] Barahona, F. and Anbil, R. (2000). The volume algorithm: producing primal solutions with a subgradient method. *Mathematical Programming*, 87(3):385–399.

[Barahona and Jensen, 1998] Barahona, F. and Jensen, D. (1998). Plant location with minimum inventory. *Mathematical Programming*, 83:101–111.

[Barnhart et al., 2003] Barnhart, C., Cohn, A. M., Johnson, E. L., Klabjan, D., Nemhauser, G. L., and Vance, P. H. (2003). Airline crew scheduling. In Hall, R. W., editor, *Handbook of Transportation Science*, volume 56 of *International Series in Operations Research & Management Science*, pages 517–560. Springer, New York.

[Barnhart et al., 1995] Barnhart, C., Hatay, L., and Johnson, E. L. (1995). Deadhead selection for the long-haul crew pairing problem. *Operations Research*, 43(3):491–499.

[Barnhart et al., 1998] Barnhart, C., Johnson, E. L., Nemhauser, G. L., Savelsbergh, M. W., and Vance, P. H. (1998). Branch-and-price: Column generation for solving huge integer programs. *Operations Research*, 46(3):316–329.

[Bäck, 1996] Bäck, T. (1996). *Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms*. Oxford University Press, Oxford, UK.

[Beasley and Christofides, 1989] Beasley, J. and Christofides, N. (1989). An algorithm for the resource constrained shortest path problem. *Networks*, 19:379–394.

[Beasley and Chu, 1996] Beasley, J. and Chu, P. (1996). A genetic algorithm for the set covering problem. *European Journal of Operational Research*, 94(2):392–404.

[Ben Amor et al., 2006] Ben Amor, H., Desrosiers, J., and de Carvalho, J. V. (2006). Dual-optimal inequalities for stabilized column generation. *Operations Research*, 54(3):454–463.

[Bertossi et al., 1987] Bertossi, A. A., Carraresi, P., and Gallo, G. (1987). On some matching problems arising in vehicle scheduling models. *Networks*, 17:271–281.

[Bertsekas and Castañon, 1992] Bertsekas, D. P. and Castañon, D. A. (1992). A forward/reverse auction algorithm for asymmetric assignment problems. *Computational Optimization and Applications*, 1(3):277–297.

[Bianco et al., 1994] Bianco, L., Mingozzi, A., and Ricciardelli, S. (1994). A set partitioning approach to the multiple depot vehicle scheduling problem. *Optimization Methods and Software*, 3:163–194.

[Bixby et al., 1992] Bixby, R. E., Gregory, J. W., Lustig, I. J., Marsten, R. E., and Shanno, D. (1992). Very large-scale linear programming: A case study in combining interior point and simplex methods. *Operations Research*, 40(5):885–897.

[Bodin et al., 1983] Bodin, L., Golden, B., Assad, A., and Ball, M. (1983). Routing and scheduling of vehicles and crews: the state of the art. *Computers & Operations Research*, 10(2):63–211.

[Borndörfer et al., 2001] Borndörfer, R., Grötschel, M., and Löbel, A. (2001). Scheduling duties by adaptive column generation. Technical Report ZIR 01-02, Konrad Zuse Zentrum für Informationstechnik, Berlin.

[Borndörfer et al., 2002] Borndörfer, R., Löbel, A., and Weider, S. (2002). Integrierte Umlauf- und Dienstplanung im Nahverkehr. Technical Report ZIB Report 02-10, Konrad-Zuse Zentrum, Berlin. in German.

[Borndörfer et al., 2004] Borndörfer, R., Löbel, A., and Weider, S. (2004). A bundle method for integrated multi-depot vehicle and duty scheduling in public transit. Technical Report ZR 04-14, Konrad-Zuse Zentrum für Informationstechnik, Berlin, Germany.

[Borndörfer et al., 2006] Borndörfer, R., Schelten, U., Schlechte, T., and Weider, S. (2006). A column generation approach to airline crew scheduling. In Haasis, H.-D., Kopfer, H., and Schönberger, J., editors, *Operations Research Proceedings 2005*, pages 343–348, Berlin. Springer.

[Bundesministerium für Verkehr, Bau- und Wohnungswesen, 2005] Bundesministerium für Verkehr, Bau- und Wohnungswesen (2005). Verordnung zur Durchführung des Fahrpersonalgesetzes (Fahrpersonalverordnung - FPersV). *Bundesgesetzblatt Teil I*, 40:1882–1933. (in German).

[Bunte and Kliewer, 2006] Bunte, S. and Kliewer, N. (2006). An overview on vehicle scheduling models. Technical Report 11/2006, University of Paderborn, DS&OR Lab. Presented at 10th International Conference on Computer-Aided Scheduling of Public Transport (CASPT2006), Leeds, UK, June 21-23, 2006.

[Camerini et al., 1975] Camerini, P., Fratta, L., and Maffioli, F. (1975). On improving relaxation methods by modified gradient techniques. *Mathematical Programming Study*, 3:26–34.

[Caprara et al., 1999] Caprara, A., Fischetti, M., and Toth, P. (1999). A heuristic method for the set covering problem. *Operations Research*, 47(5):730–743.

[Carpaneto et al., 1989] Carpaneto, G., Dell'Amico, M., Fischetti, M., and Toth, P. (1989). A branch and bound algorithm for the multiple depot vehicle scheduling problem. *Networks*, 19:531–548.

[Carraresi and Gallo, 1984] Carraresi, P. and Gallo, G. (1984). Network models for vehicle and crew scheduling. *European Journal of Operational Research*, 16(2):139–151.

[Carraresi et al., 1995] Carraresi, P., Nonato, M., and Girardi, L. (1995). Network models, lagrangean relaxation and subgradients bundle approach in crew scheduling problems. In *Computer-Aided Transit Scheduling, Proceedings of the Sixth International Workshop*, volume 430 of *Lecture Notes in Economics and Mathematical Systems*, pages 188–212, Berlin. Springer.

[Cavique et al., 1999] Cavique, L., Rego, C., and Themido, I. (1999). Subgraph ejection chains and tabu search for the crew scheduling problem. *Journal of the Operational Research Society*, 50(6):608–616.

[Clausen et al., 2005] Clausen, J., Larsen, A., and Larsen, J. (2005). Disruption management in the airline industry - concepts, models and methods. Technical Report 2005-01, Informatics and Mathematical Modelling, Technical University of Denmark, Lyngby, Denmark.

[Daduna and Paixão, 1995] Daduna, J. R. and Paixão, J. M. P. (1995). Vehicle scheduling for public mass transit - an overview. In Daduna, J., Branco, I., and Paixão, J., editors, *Proceedings of the Sixth International Workshop on Computer-Aided Scheduling of Public Transport*, volume 430 of *Lecture Notes in Economics and Mathematical Systems*, pages 76–90. Springer, Heidelberg.

[Dallaire et al., 2004] Dallaire, A., Fleurent, C., and Rousseau, J.-M. (2004). Dynamic constraint generation in crewopt, a column generation approach for transit crew scheduling. Technical report, GIRO Inc., Montréal, Canada. (submitted to 9th International Conference on Computer-Aided Scheduling of Public Transport (CASPT), San Diego).

[Danna and Le Pape, 2005] Danna, E. and Le Pape, C. (2005). Branch-and-price heuristics: A case study on the vehicle routing problem with time windows. In Desaulniers, G., Desrosiers, J., and Solomon, M., editors, *Column Generation*, chapter 4, pages 99–129. Springer, New York.

[Dantzig, 1963] Dantzig, G. (1963). *Linear Programming and Extensions*. Princeton University Press, Princeton.

[Dantzig and Wolfe, 1960] Dantzig, G. B. and Wolfe, P. (1960). Decomposition principle for linear programs. *Operations Research*, 8(1):101–111.

[Darby-Dowman et al., 1988] Darby-Dowman, K., Jachnik, J. K., Lewis, R. L., and Mitra, G. (1988). Integrated decision support systems for urban transport scheduling: Discussion of implementation and experience. In Daduna, J. R. and Wren, A., editors, *Proceedings of the Fourth International Workshop on Computer-Aided Transit Scheduling*, Lecture Notes in Economics and Mathematical Systems, pages 226–239, Berlin. Springer.

[de Groot and Huisman, 2004] de Groot, S. W. and Huisman, D. (2004). Vehicle and crew scheduling: Solving large real-world instances with an integrated approach. Technical Report EI2004-13, Erasmus University Rotterdam, San Diego.

[de Silva, 2001] de Silva, A. (2001). Combining constraint programming and linear programming on an example of bus driver scheduling. *Annals of Operations Research*, 108:277–291.

[Dell'Amico et al., 1993] Dell'Amico, M., Fischetti, M., and Toth, P. (1993). Heuristic algorithms for the multiple depot vehicle scheduling problem. *Management Science*, 39(1):115–125.

[Dell'Amico et al., 2006] Dell'Amico, M., Righini, G., and Salani, M. (2006). A branch-and-price approach to the vehicle routing problem with simultaneous distribution and collection. *Transportation Science*, 40(2):235–247.

[Desaulniers et al., 1997] Desaulniers, G., Desrosiers, J., Dumas, Y., Marc, S., Rioux, B., Solomon, M. M., and Soumis, F. (1997). Crew pairing at air france. *European Journal of Operational Research*, 97:245–259.

[Desaulniers et al., 1999] Desaulniers, G., Desrosiers, J., Lasry, A., and Solomon, M. M. (1999). Crew pairing for a regional carrier. In Wilson, N. H., editor, *Computer-Aided Transit Scheduling*, volume 471 of *Lecture Notes in Economics and Mathematical Systems*, pages 19–41, Berlin. Springer.

[Desaulniers et al., 2002] Desaulniers, G., Desrosiers, J., and Solomon, M. M. (2002). Accelerating strategies in column generation methods for vehicle routing and crew scheduling problems. In Ribeiro, C. and Hansen, P., editors, *Essays and Surveys in Metaheuristics*, pages 309–324. Kluwer, Boston.

[Desaulniers et al., 2005] Desaulniers, G., Desrosiers, J., and Solomon, M. M., editors (2005). *Column Generation*. Springer, New York.

[Desaulniers and Hickman, 2006] Desaulniers, G. and Hickman, M. D. (2006). Public transit. In Barnhart, C. and Laporte, G., editors, *Transportation*, Handbooks in Operations Research and Management Science, pages 69–127. North Holland, The Netherlands.

[Desrochers, 1986] Desrochers, M. (1986). *La fabrication d'horaires de travail pour les conducteurs d'autobus par une méthode de génération de colonnes*. PhD thesis, Université de Montréal, Montréal, Canada. (in French).

[Desrochers et al., 1992] Desrochers, M., Gilbert, J., Sauvé, M., and Soumis, F. (1992). Crew-opt: Subproblem modeling in a column generation approach to urban crew scheduling. In Desrochers, M. and Rousseau, J., editors, *Computer-Aided Scheduling*, volume 386 of *Lecture Notes in Economics and Mathematical Systems*, pages 395–406, Berlin. Springer.

[Desrochers and Soumis, 1988] Desrochers, M. and Soumis, F. (1988). A generalized permanent labelling algorithm for the shortest path problem with time windows. *INFOR*, 26(3):191–212.

[Desrochers and Soumis, 1989] Desrochers, M. and Soumis, F. (1989). A column generation approach to the urban transit crew scheduling problem. *Transportation Science*, 23(1):1–13.

[Desrosiers et al., 1995] Desrosiers, J., Dumas, Y., Solomon, M. M., and Soumis, F. (1995). *Time Constrained Routing and Scheduling*, volume 8 of *Handbooks in Operations Research and Management Science*, chapter 2, pages 35–139. Elsevier Science, Amsterdam.

[Dorigo and Stützle, 2004] Dorigo, M. and Stützle, T. (2004). *Ant Colony Optimization*. MIT Press (Bradford Books).

[Dowsland, 1993] Dowsland, K. A. (1993). *Simulated Annealing*, chapter 2, pages 20–69. Blackwell, London.

[Du Merle et al., 1999] Du Merle, O., Villeneuve, D., Desrosiers, J., and Hansen, P. (1999). Stabilized column generation. *Discrete Mathematics*, 194:229–237.

[Dumitrescu and Boland, 2003] Dumitrescu, I. and Boland, N. (2003). Improved preprocessing, labeling and scaling algorithms for the weight-constrained shortest path problem. *Networks*, 42(3):135–153.

[Ehrgott, 2005] Ehrgott, M. (2005). *Multicriteria Optimization*, volume 491 of *Lecture Notes in Economics and Mathematical Systems*. Springer, Berlin, 2nd edition.

[Ehrgott and Gandibleux, 2004] Ehrgott, M. and Gandibleux, X. (2004). Approximative solution methods for multiobjective combinatorial optimization. *TOP*, 12(1):1–63.

[Fahle, 2002] Fahle, T. (2002). *Integrating Concepts from Constraint Programming and Operations Research Algorithms*. PhD thesis, University of Paderborn.

[Fahle et al., 2002] Fahle, T., Junker, U., Karisch, S. E., Kohl, N., Sellmann, M., and Vaaben, B. (2002). Constraint programming based column generation for crew assignment. *Journal of Heuristics*, 8:59–81.

[Falkner and Ryan, 1992] Falkner, J. and Ryan, D. M. (1992). EXPRESS: Set partitioning for bus crew scheduling in Christchurch. In Desrochers, M. and Rousseau, J., editors, *Computer-Aided Scheduling*, volume 386 of *Lecture Notes in Economics and Mathematical Systems*, pages 359–378, Berlin. Springer.

[Fischetti and Lodi, 2003] Fischetti, M. and Lodi, A. (2003). Local branching. *Mathematical Programming*, 84(1):23–47.

[Fischetti et al., 1987] Fischetti, M., Lodi, A., Martello, S., and Toth, P. (1987). The fixed job schedule problem with spread-time constraints. *Operations Research*, 35(6):849–858.

[Fischetti et al., 1989] Fischetti, M., Lodi, A., Martello, S., and Toth, P. (1989). The fixed job schedule problem with working-time constraints. *Operations Research*, 37(3):395–403.

[Fischetti et al., 2001] Fischetti, M., Lodi, A., Martello, S., and Toth, P. (2001). A polyhedral approach to simplified crew scheduling and vehicle scheduling problems. *Management Science*, 47(6):833–850.

[Fischetti et al., 1999] Fischetti, M., Lodi, A., and Toth, P. (1999). A branch-and-cut algorithm for the multi depot vehicle scheduling problem. Technical report.

[Fischetti and Toth, 1989] Fischetti, M. and Toth, P. (1989). An additive bounding procedure for combinatorial optimization problems. *Operations Research*, 37(2):319–328.

[Fisher, 1981] Fisher, M. L. (1981). The lagrangian relaxation method for solving integer programming problems. *Management Science*, 27(1):1–19.

[Fleischer, 2003] Fleischer, M. (2003). The measure of pareto optima. In Fonseca, C. M., Fleming, P. J., Zitzler, E., Deb, K., and Thiele, L., editors, *Evolutionary Multi-Criterion Optimization: Proceedings of the Second International Conference on Evolutionary Multi-Criterion Optimization (EMO2003)*, volume 2632 of *Lecture Notes in Computer Science*, pages 519–533, Berlin. Springer.

[Forbes et al., 1994] Forbes, M., Holt, J. N., and Watts, A. M. (1994). An exact algorithm for multiple depot bus scheduling. *European Journal of Operational Research*, 72:115–124.

[Fores et al., 2002] Fores, S., Proll, L. G., and Wren, A. (2002). Tracs ii: a hybrid ip/heuristic driver scheduling system for public transport. *Journal of the Operational Research Society*, 53(10):1093–1100.

[Freling, 1997] Freling, R. (1997). *Models and Techniques for Integrating Vehicle and Crew Scheduling*. PhD thesis, Erasmus University of Rotterdam.

[Freling et al., 2001a] Freling, R., Huisman, D., and Wagelmans, A. P. (2001a). Applying an integrated approach to vehicle and crew scheduling in practice. In Voß, S. and Daduna, J., editors, *Computer-Aided Scheduling of Public Transport*, volume 505 of *Lecture Notes in Economics and Mathematical Systems*, pages 73–90, Berlin. Springer.

[Freling et al., 2003] Freling, R., Huisman, D., and Wagelmans, A. P. (2003). Models and algorithms for integration of vehicle and crew scheduling. *Journal of Scheduling*, 6:63–85.

[Freling et al., 2001b] Freling, R., Wagelmans, A. P., and Paixão, J. M. P. (2001b). Models and algorithms for single-depot vehicle scheduling. *Transportation Science*, 35(2):165–180.

[Friberg and Haase, 1999] Friberg, C. and Haase, K. (1999). An exact branch and cut algorithm for the vehicle and crew scheduling problem. In Wilson, N. H., editor, *Computer-Aided Transit Scheduling*, volume 471 of *Lecture Notes in Economics and Mathematical Systems*, pages 63–80, Berlin. Springer.

[Gaffi and Nonato, 1999] Gaffi, A. and Nonato, M. (1999). An integrated approach to ex-urban crew and vehicle scheduling. In Wilson, N. H., editor, *Computer-Aided Transit Scheduling*, volume 471 of *Lecture Notes in Economics and Mathematical Systems*, pages 103–128, Berlin. Springer.

[Galia and Hjorring, 2004] Galia, R. and Hjorring, C. (2004). Modelling of complex costs and rules in a crew pairing column generator. In Ahr, D., Fahrion, R., Oswald, M., and Reinelt, G., editors, *Operations Research Proceedings 2003 - Selected Papers of the International Conference on Operations Research (OR 2003)*, pages 133–140, Berlin. Springer.

[Gamache and Soumis, 1998] Gamache, M. and Soumis, F. (1998). A method for optimally solving the rostering problem. In Yu, G., editor, *Operations Research in the Airline Industry*, volume 9 of *International Series in Operations Research & Management Science*, pages 124–157. Kluwer Academic Publishers, Boston.

[Gamache et al., 1999] Gamache, M., Soumis, F., Marquis, G., and Desrosiers, J. (1999). A column generation approach for large-scale aircrew rostering problems. *Operations Research*, 47(2):247–263.

[Gandibleux et al., 2004] Gandibleux, X., Sevaux, M., Sörensen, K., and T'kindt, V., editors (2004). *Metaheuristics for Multiobjective Optimisation*, volume 535 of *Lecture Notes in Economics and Mathematical Systems*. Springer, Heidelberg, Germany.

[Garey and Johnson, 1979] Garey, M. and Johnson, D. (1979). *Computer and Intractability: A Guide to NP-Completeness*. Freeman, San Francisco.

[Gavish and Shlifer, 1978] Gavish, B. and Shlifer, E. (1978). An approach for solving a class of transportation scheduling problems. *European Journal of Operational Research*, 3:122–134.

[Geoffrion, 1974] Geoffrion, A. (1974). Lagrangean relaxation for integer programming. *Mathematical Programming Study*, 2:82–114.

[Gintner, 2007] Gintner, V. (2007). *Integrierte Umlauf- und Dienstplanung im ÖPNV*. PhD thesis, University of Paderborn, Germany (to be submitted).

[Gintner et al., 2004] Gintner, V., Kliewer, N., and Suhl, L. (2004). A crew scheduling approach for public transit enhanced with aspects from vehicle scheduling. Technical Report WP0407, University of Paderborn, Decision Support & Operations Research Lab, Paderborn.

[Gintner et al., 2005] Gintner, V., Kliewer, N., and Suhl, L. (2005). Solving large multiple-depot multiple-vehicle-type bus scheduling problems in practice. *OR Spectrum*, 27(4):507–523.

[Gintner et al., 2006a] Gintner, V., Kramkowski, S., Steinzen, I., and Suhl, L. (2006a). Adaptive Dienst- und Umlaufplanung im ÖPNV. In Haasis, H.-D., Kopfer, H., and Schönberger, J., editors, *Operations Research Proceedings 2005*, pages 55–60, Berlin. Springer.

[Gintner et al., 2006b] Gintner, V., Steinzen, I., and Suhl, L. (2006b). A time-space network based approach for integrated vehicle and crew scheduling in public transport. In Binetti, M., Civitella, F., Liddo, E. D., Dell'Orco, M., and Ottomanelli, M., editors, *Proceedings of the EWGT2006 Joint Conferences*, pages 371–377, Bari, Italy.

[GIRO Inc., 2007] GIRO Inc. (2007). Hastus – transit scheduling and operations. available at `http://www.giro.ca/en/products/hastus/index.htm` (last access on July 9^{th}, 2007).

[Glover and Laguna, 1993] Glover, F. and Laguna, M. (1993). *Tabu Search*, chapter 3, pages 70–150. Blackwell, London.

[Gopalakrishnan and Johnson, 2005] Gopalakrishnan, B. and Johnson, E. L. (2005). Airline crew scheduling: State-of-the-art. *Annals of Operations Research*, 140(1):305–337.

[Grönkvist, 2005] Grönkvist, M. (2005). *The Tail Assignment Problem*. PhD thesis, Chalmers University of Technology and Göteborg University.

[Grötschel et al., 2003] Grötschel, M., Borndörfer, R., and Löbel, A. (2003). Duty scheduling in public transit. In Jäger, W., editor, *Mathematics - key technologies for the future*, pages 653–674. Springer, Berlin.

[Guntsch and Middendorf, 2003] Guntsch, M. and Middendorf, M. (2003). Solving multi-criteria optimization problems with population-based ACO. In Goos, G., Hartmanis, J., and van Leeuwen, J., editors, *Proceedings of Second International Conference on Evolutionary Multi-Criterion Optimization (EMO2003)*, volume 2632 of *Lecture Notes in Computer Science*, pages 464–478, Berlin. Springer.

[Guo et al., 2005] Guo, Y., Suhl, L., and Thiel, M. P. (2005). Solving the airline crew recovery problem by a genetic algorithm with local improvement. *Operational Research – An International Journal*, 5(2).

[Haase et al., 2001] Haase, K., Desaulniers, G., and Desrosiers, J. (2001). Simultaneous vehicle and crew scheduling in urban mass transit systems. *Transportation Science*, 35(3):286–303.

[Hadjar et al., 2006] Hadjar, A., Marcotte, O., and Soumis, F. (2006). A branch-and-cut algorithm for the multiple depot vehicle scheduling problem. *Operations Research*, 54(1):130–149.

[Hassin, 1992] Hassin, R. (1992). Approximation schemes for the restricted shortest path problem. *Mathematics of Operations Research*, 17(1):36–42.

[Held and Karp, 1971] Held, M. and Karp, R. M. (1971). The travelling salesman problem and minimum spanning trees: Part ii. *Mathematical Programming*, 1:6–25.

[Held et al., 1974] Held, M., Wolfe, P., and Crowder, H. (1974). Validation of subgradient optimization. *Mathematical Programming*, 6:62–88.

[Hoffman and Padberg, 1993] Hoffman, K. L. and Padberg, M. (1993). Solving airline crew scheduling problems by branch-and-cut. *Management Science*, 39(6):657–682.

[Hollis et al., 2006] Hollis, B., Forbes, M., and Douglas, B. (2006). Vehicle routing and crew scheduling for metropolitan mail distribution at australia post. *European Journal of Operational Research*, 173:133–150.

[Holmberg and Yuan, 2000] Holmberg, K. and Yuan, D. (2000). A lagrangean heuristic based branch-and-bound approach for the capacitated network design problem. *Operations Research*, 48(3):461–481.

[Huisman, 2003] Huisman, D. (2003). Random data instances for multiple-depot vehicle and crew scheduling. available at `http://people.few.eur.nl/huisman/instances.htm` (last access on April 11^{th}, 2007).

[Huisman, 2004] Huisman, D. (2004). *Integrated and Dynamic Vehicle and Crew Scheduling*. PhD thesis, Erasmus University of Rotterdam.

[Huisman, 2007] Huisman, D. (2007). A column generation approach to solve the crew re-scheduling problem. *European Journal of Operational Research*, 180(1):163–173.

[Huisman et al., 2005a] Huisman, D., Freling, R., and Wagelmans, A. P. (2005a). Multiple-depot integrated vehicle and crew scheduling. *Transportation Science*, 39(4):491–502.

[Huisman et al., 2005b] Huisman, D., Jans, R., Peeters, M., and Wagelmans, A. P. (2005b). Combining column generation and lagrangian relaxation. In Desaulniers, G., Desrosiers, J., and Solomon, M., editors, *Column Generation*, chapter 9, pages 247–270. Springer, New York.

[ILOG, 2006] ILOG (2006). *CPLEX 10.0.1 User's Manual*. ILOG, Gentilly Cedex, France.

[Irnich, 2006] Irnich, S. (2006). Resource extension functions: Properties, inversion, and generalization to segments. Technical Report 2006-01, Deutsche Post Endowed Chair of Optimization of Distribution Networks, RWTH Aachen University, Aachen, Germany.

[Irnich and Desaulniers, 2005] Irnich, S. and Desaulniers, G. (2005). Shortest path problems with resource constraints. In Desaulniers, G., Desrosiers, J., and

Solomon, M., editors, *Column Generation*, chapter 2, pages 33–65. Springer, New York.

[Jans and Degraeve, 2004] Jans, R. and Degraeve, Z. (2004). An industrial extension of the discrete lot-sizing and scheduling problem. *IIE Transactions*, 36(1):47–58.

[Joksch, 1966] Joksch, H. (1966). The shortest route problem with constraints. *Journal of Mathematical Analysis and Applications*, 14:191–197.

[Kiwiel, 1995] Kiwiel, K. (1995). Approximations in proximal bundle methods and decomposition of convex programs. *Journal of Optimization Theory and Applications*, 84(3):529–548.

[Klabjan et al., 2001] Klabjan, D., Johnson, E. L., Nemhauser, G. L., Gelman, E., and Ramaswamy, S. (2001). Airline crew scheduling with regularity. *Transportation Science*, 35(4):359–374.

[Klabjan et al., 2002] Klabjan, D., Johnson, E. L., Nemhauser, G. L., Gelman, E., and Ramaswamy, S. (2002). Airline crew scheduling with time windows and plane-count constraints. *Transportation Science*, 36(3):337–448.

[Kliewer, 2005] Kliewer, N. (2005). *Optimierung des Fahrzeugeinsatzes im öffentlichen Personennahverkehr*. PhD thesis, University of Paderborn, Germany.

[Kliewer et al., 2006a] Kliewer, N., Bunte, S., and Suhl, L. (2006a). Time windows for scheduled trips in multiple depot vehicle scheduling. In Binetti, M., Civitella, F., Liddo, E. D., Dell'Orco, M., and Ottomanelli, M., editors, *Proceedings of the EURO Working Group on Transportation (EWGT) Joint Conferences 2006*, pages 340–346, Bari, Italy.

[Kliewer et al., 2006b] Kliewer, N., Mellouli, T., and Suhl, L. (2006b). A time-space network based exact optimization model for multi-depot bus scheduling. *European Journal of Operational Research*, 175(3):1616–1627.

[Kung et al., 1975] Kung, H., Luccio, F., and Preparata, F. (1975). On finding the maxima of a set of vectors. *Journal of the Association for Computing Machinery*, 22(4):469–476.

[Kwan et al., 1999] Kwan, A., Kwan, R. S., and Wren, A. (1999). Driver scheduling using genetic algorithms with embedded combinatorial traits. In Wilson,

N. H., editor, *Computer-Aided Transit Scheduling*, volume 471 of *Lecture Notes in Economics and Mathematical Systems*, pages 81–102, Berlin. Springer.

[Kwan et al., 2001] Kwan, R. S., Kwan, A., and Wren, A. (2001). Evolutionary driver scheduling with relief chains. *Evolutionary Computation*, 9:445–460.

[Lavoie et al., 1988] Lavoie, S., Minoux, M., and Odier, E. (1988). A new approach for crew pairing problems by column generation with an application to air transportation. *European Journal of Operational Research*, 35(1):45–58.

[Lübbecke, 2005] Lübbecke, M. (2005). Dual variable based fathoming in dynamic programs for column generation. *European Journal of Operational Research*, 162:122–125.

[Lübbecke and Desrosiers, 2005] Lübbecke, M. and Desrosiers, J. (2005). Selected topics in column generation. *Operations Research*, 53(6):1007–1023.

[Löbel, 1996] Löbel, A. (1996). Solving large-scale real-world minimum-cost flow problems by a network simplex method. Technical Report SC96-7, Konrad-Zuse Zentrum für Informationstechnik (ZIB), Berlin.

[Löbel, 1997] Löbel, A. (1997). *Optimal vehicle scheduling in public transit*. PhD thesis, Technical University Berlin, Germany.

[Löbel, 1998] Löbel, A. (1998). Vehicle scheduling in public transit and lagrangean pricing. *Management Science*, 44(12):1637–1650.

[Lettovsky et al., 2000] Lettovsky, L., Johnson, E. L., and Nemhauser, G. L. (2000). Airline crew recovery. *Transportation Science*, 34(4):337–348.

[Leuthardt, 1998] Leuthardt, H. (1998). Kostenstrukturen von Stadt-, Überland- und Reisebussen. *Der Nahverkehr*, 6:19–23. (in German).

[Li and Kwan, 2005] Li, J. and Kwan, R. S. (2005). A self-adjusting algorithm for driver scheduling. *Journal of Heuristics*, 11(4):351–367.

[Linderoth and Savelsbergh, 1999] Linderoth, J. T. and Savelsbergh, M. W. (1999). A computational study of search strategies for mixed integer programming. *INFORMS Journal on Computing*, 11(2):173–187.

[Lourenço et al., 2001] Lourenço, H. R., Paixão, J. M. P., and Portugal, R. (2001). Multiobjective metaheuristics for the bus driver scheduling problem. *Transportation Science*, 35(3):331–343.

[Marchiori and Steenbeek, 2003] Marchiori, E. and Steenbeek, A. (2003). An evolutionary algorithm for large scale set covering problems with application to airline crew scheduling. In et al., S. C., editor, *Real-World Applications of Evolutionary Computing: EvoWorkshops 2000: EvoIASP, EvoSCONDI, EvoTEL, EvoSTIM, EvoRob, and EvoFlight, Edinburgh, Scotland, UK*, volume 1803 of *Lecture Notes in Computer Science*, pages 367–381, Berlin. Springer.

[Marriot and Stuckey, 1998] Marriot, K. and Stuckey, P. (1998). *Programming with Constraints*. MIT Press, Cambridge, Massachusetts.

[Marsten et al., 1975] Marsten, R. E., Hogan, W., and Blankenship, J. (1975). The boxstep method for large-scale optimization. *Operations Research*, 23(3):389–405.

[Medard and Sawhney, 2007] Medard, C. P. and Sawhney, N. (2007). Airline crew scheduling: From planning to operations. *European Journal of Operational Research*, 183(3):1013–1027.

[Mehlhorn and Ziegelmann, 2000] Mehlhorn, K. and Ziegelmann, M. (2000). Resource constrained shortest paths. In Paterson, M., editor, *Proc. 8th Annual European Symposium on Algorithms (ESA2000)*, volume 1972 of *Lecture Notes in Computer Science*, pages 326–337, Berlin. Springer.

[Mesquita and Paias, 2006] Mesquita, M. and Paias, A. (2006). Set partitioning/covering-based approaches for the integrated vehicle and crew scheduling problem. *Computers & Operations Research*, in press.

[Mesquita et al., 2006] Mesquita, M., Paias, A., and Respício, A. (2006). Branching approaches for the integrated vehicle and crew scheduling. Technical Report 9/2006, Operations Research Center at the University of Lisbon (CIO), Lisbon, Portugal.

[Mesquita and Paixão, 1992] Mesquita, M. and Paixão, J. M. P. (1992). Multiple depot vehicle scheduling problem: A new heuristic based on quasi-assignment algorithms. In Desrochers, M. and Rousseau, J.-M., editors, *Proceedings of the Fifth International Workshop on Computer-Aided Scheduling of Public Transport (CASPT)*, volume 386 of *Lecture Notes in Economics and Mathematical Systems*, pages 167–180, Berlin. Springer.

[Mesquita and Paixão, 1999] Mesquita, M. and Paixão, J. M. P. (1999). Exact algorithms for the multi-depot vehicle scheduling problem based on multicommodity network flow type formulations. In Wilson, N. H., editor, *Computer-*

Aided Transit Scheduling, volume 471 of *Lecture Notes in Economics and Mathematical Systems*, pages 221–243, Berlin. Springer.

[Nemhauser and Wolsey, 1988] Nemhauser, G. and Wolsey, L., editors (1988). *Integer and Combinatorial Optimization*. Wiley, New York.

[Nissen and Haase, 2006] Nissen, R. and Haase, K. (2006). Duty-period-based network model for crew rescheduling in european airlines. *Journal of Scheduling*, 9(3):255–278.

[Orchard-Hays, 1968] Orchard-Hays, W. (1968). *Advanced Linear-Programming Computing Techniques*. McGraw-Hill, New York.

[Orloff, 1976] Orloff, C. S. (1976). Route constrained fleet scheduling. *Transportation Science*, 10(2):149–168.

[Osman and Laporte, 1996] Osman, I. H. and Laporte, G. (1996). Metaheuristics: A bibliography. *Annals of Operations Research*, 63(5):513–623.

[Oukil et al., 2007] Oukil, A., Amor, H. B., Desrosiers, J., and Gueddari, H. E. (2007). Stabilized column generation for highly degenerate multiple-depot vehicle scheduling problems. *Computers & Operations Research*, 34:817–834.

[PaderSprinter, 2007] PaderSprinter (2007). PaderSprinter - public transport company in Paderborn. available at `http://www.padersprinter.de/` (last access on June 13^{th}, 2007).

[Paixão and Branco, 1987] Paixão, J. M. P. and Branco, I. (1987). A quasi-assignment algorithm for bus scheduling. *Networks*, 17:249–269.

[Patrikalakis and Xerocostas, 1992] Patrikalakis, I. and Xerocostas, D. (1992). A new decomposition scheme of the urban public transport scheduling problem. In Desrochers, M. and Rousseau, J.-M., editors, *Proceedings of the Fifth International Workshop on Computer-aided Scheduling of Public Transport*, volume 386 of *Lecture Notes in Economics and Mathematical Systems*, pages 407–425, Berlin. Springer.

[Pepin et al., 2006] Pepin, A.-S., Desaulniers, G., Hertz, A., and Huisman, D. (2006). Comparison of heuristic approaches for the multiple depot vehicle scheduling problem. Technical Report EI2006-34, Econometric Institute, Erasmus University Rotterdam.

[Polyak, 1967] Polyak, B. (1967). A general method of solving extremum problems. *Soviet Mathematics Doklady*, 8:593–597.

[PTV AG, 2007] PTV AG (2007). ptv interplan - advanced operational planning for public transport companies. available at `http://www.english.ptv.de/cgi-bin/traffic/traf_ip.pl` (last access on June 13^{th}, 2007).

[Reeves, 1993] Reeves, C. R., editor (1993). *Modern heuristic techniques for combinatorial problems*. Blackwell, London.

[Ribeiro and Soumis, 1994] Ribeiro, C. and Soumis, F. (1994). A column generation approach to the multiple-depot vehicle scheduling problem. *Operations Research*, 42(1):41–52.

[Rodrigues et al., 2006] Rodrigues, M. M., de Souza, C., and Moura, A. (2006). Vehicle and crew scheduling for urban bus lines. *European Journal of Operational Research*, 170:844–862.

[Ryan and Foster, 1981] Ryan, D. M. and Foster, B. (1981). An integer programming approach to scheduling. In Wren, A., editor, *Computer Scheduling of Public Transport: Urban Passenger Vehicle and Crew Scheduling*, pages 269–280, Amsterdam. North-Holland.

[Sandhu and Klabjan, 2006] Sandhu, R. and Klabjan, D. (2006). Integrated airline fleeting and crew pairing decisions. *Operations Research*, to appear.

[Scott, 1985] Scott, D. (1985). A large scale linear programming approach to the public transport scheduling and costing problem. In Rousseau, J.-M., editor, *Computer Scheduling of Public Transport 2*, pages 473–491. Elsevier Science Publishers, Amterdam.

[Serafini, 1992] Serafini, P. (1992). Simulated annealing for multiple objective optimization problems. In *Proceedings of the 10th International Conference on Multiple Criteria Decision Making*, pages 87–96, Taipeh, Taiwan.

[Shen and Kwan, 2001] Shen, Y. and Kwan, R. S. (2001). Tabu search for driver scheduling. In Voß, S. and Daduna, J., editors, *Computer-Aided Scheduling of Public Transport*, volume 505 of *Lecture Notes in Economics and Mathematical Systems*, pages 121–135, Berlin. Springer.

[Silva et al., 1999] Silva, G. P., Wren, A., Kwan, R. S., and Gualda, N. D. F. (1999). Bus scheduling based on an arc generation - network flow approach. Technical report, University of Leeds - School of Computer Studies.

[Steinzen, 2007] Steinzen, I. (2007). Instances for multiple-depot vehicle and crew scheduling. available at `http://dsor.upb.de/~isteinzen/` (last access on April 11^{th}, 2007).

[Steinzen et al., 2007a] Steinzen, I., Becker, M., and Suhl, L. (2007a). A hybrid evolutionary algorithm for the vehicle and crew scheduling problem in public transit. In *Procs. of the IEEE Congress on Evolutionary Computation (CEC2007)*, Singapore. IEEE Press. (accepted for publication).

[Steinzen et al., 2007b] Steinzen, I., Gintner, V., and Suhl, L. (2007b). Local Branching und Branching-Strategien für Umlauf- und Dienstplanung im Regionalverkehr mit unregelmäßigen Fahrplänen. In Günter, H.-O., Mattfeld, D., and Suhl, L., editors, *Management logistischer Netzwerke: Entscheidungsunterstützung, Informationssysteme und OR-Tools*, pages 407–424. Physica-Verlag, Heidelberg.

[Suhl et al., 2007] Suhl, L., Kliewer, N., and Steinzen, I. (2007). Optimierungssysteme für die Dienstplanung im ÖPNV. In Oberweis, A., Weinhardt, C., Gimpel, H., Koschmieder, A., Pankratius, V., and Schnitzler, B., editors, *eOrganisation: Service-, Prozess, Marketengineering - 8. Internationale Tagung Wirtschaftsinformatik 2007*, pages 447–464, Karlsruhe, Germany. Universitätsverlag Karlsruhe.

[Suhl, 2000] Suhl, U. (2000). MOPS - Mathematical Optimization System. *OR News*, 8:11–16.

[Tajima and Misono, 1997] Tajima, A. and Misono, S. (1997). Airline crew-scheduling with many irregular flights. In Leong, H. W., Imai, H., and Jain, S., editors, *Proceedings of the 8th International Symposium on Algorithms and Computation - ISAAC97*, volume 1350 of *Lecture Notes in Computer Science*, pages 2–11, Heidelberg. Springer.

[Tosini and Vercellis, 1988] Tosini, E. and Vercellis, C. (1988). An interactive system for extra-urban vehicle and crew scheduling problems. In Daduna, J. R. and Wren, A., editors, *Proceedings of the Fourth International Workshop on Computer-Aided Transit Scheduling*, Lecture Notes in Economics and Mathematical Systems, pages 41–53, Berlin. Springer.

[Valouxis and Housos, 2002] Valouxis, C. and Housos, E. (2002). Combined bus and driver scheduling. *Computers & Operations Research*, 29(3):243–259.

[Vance et al., 1997a] Vance, P. H., Atamtürk, A., Barnhart, C., Gelman, E., Johnson, E. L., Krishna, A., Mahidhara, D., and Rebello, R. (1997a). A heuristic branch-and-price approach for the airline crew pairing problem. Technical Report LEC-97-06, Georgia Institute of Technology, Atlanta, USA.

[Vance et al., 1997b] Vance, P. H., Barnhart, C., Johnson, E. L., and Nemhauser, G. L. (1997b). Airline crew scheduling: A new formulation and decomposition algorithm. *Operations Research*, 45(2):188–200.

[Vanderbeck, 1994] Vanderbeck, F. (1994). *Decomposition and Column Generation for Integer Programs*. PhD thesis, Université Catholique de Louvain.

[Villeneuve et al., 2005] Villeneuve, D., Desrosiers, J., Lübbecke, M., and Soumis, F. (2005). On compact formulations for integer programs solved by column generation. *Annals of Operations Research*, 139(1):375–388.

[Wedelin, 1995] Wedelin, D. (1995). An algorithm for large scale 0-1 integer programming with application to airline crew scheduling. *Annals of Operations Research*, 57:283–301.

[Westerlund et al., 2006] Westerlund, A., Göthe-Lundgren, M., and Larsson, T. (2006). A stabilized column generation scheme for the traveling salesman subtour problem. *Discrete Applied Mathematics*, 154(15):2212–2238.

[Wolsey, 1998] Wolsey, L. A. (1998). *Integer Programming*. Wiley Interscience, New York.

[Wren and Rousseau, 1995] Wren, A. and Rousseau, J.-M. (1995). Bus driver scheduling - an overview. In Daduna, J., Brance, I., and Paixao, J., editors, *Computer-Aided Transit Scheduling*, volume 430 of *Lecture Notes in Economics and Mathematical Systems*, pages 173–187, Berlin. Springer.

[Wren and Wren, 1995] Wren, A. and Wren, D. O. (1995). A genetic algorithm for public transport driver scheduling. *Computers & Operations Research*, 22(1):101–110.

[Xu et al., 2003] Xu, H., Chen, Z.-L., Rajagopal, S., and Arunapuram, S. (2003). Solving a practical pickup and delivery problem. *Transportation Science*, 37(3):374–364.

[Yunes et al., 2005] Yunes, T., Moura, A., and de Souza, C. (2005). Hybrid column generation approaches for urban transit crew management problems. *Transportation Science*, 39(2):273–288.

[Zitzler et al., 2002] Zitzler, E., Laumanns, M., and Thiele, L. (2002). Spea2: Improving the strength pareto evolutionary algorithm for multiobjective optimization. In Giannakoglou, K., Tsahalis, D., Périaux, J., Papailiou, K., and Fogarty, T., editors, *Evolutionary Methods for Design, Optimisation and Control*, pages 95–100, Barcelona, Spain. CIMNE.

[Zitzler and Thiele, 1998] Zitzler, E. and Thiele, L. (1998). Multiobjective optimization using evolutionary algorithms - a comparative case study. In Eiben, A., Bäck, T., Schönauer, M., and Schwefel, H., editors, *Parallel Problem Solving from Nature — PPSN V*, volume 1498 of *Lecture Notes in Computer Science*, pages 292–301, Berlin, Germany. Springer.

[Zitzler and Thiele, 1999] Zitzler, E. and Thiele, L. (1999). Multiobjective evolutionary algorithms: a comparative case study and the strength pareto approach. *IEEE Transactions on Evolutionary Computation*, 3(4):257–271.

[Zitzler et al., 2003] Zitzler, E., Thiele, L., Laumanns, M., Fonseca, C., and da Fonseca, V. G. (2003). Performance assessment of multiobjective optimizers: An analysis and review. *IEEE Transactions on Evolutionary Computation*, 7(2):257–271.