Constraint Programming-Based Heuristics for the Multi-Depot Vehicle Routing Problem with a Rolling Planning Horizon

Von der Mercator School of Management, Fakultät für Betriebswirtschaftslehre, der

Universität Duisburg-Essen

zur Erlangung des akademischen Grades

eines Doktors der Wirtschaftswissenschaft (Dr. rer. oec.)

genehmigte Dissertation

von

Christoph Reiners

aus

Duisburg.

Referent:	Prof. Dr. Alf Kimms
Korreferent:	Prof. Dr. Peter Chamoni
Tag der mündlichen Prüfung:	16. Dezember 2015

Vorwort

Diese Arbeit ist während meiner Beschäftigung als wissenschaftlicher Mitarbeiter am Lehrstuhl für Logistik und Operations Research an der Mercator School of Management, Fakulät für Betriebswirtschaftslehre, der Universität Duisburg-Essen entstanden.

Zuerst bin ich Herrn Prof. Dr. Alf Kimms zum Dank für sein Vertrauen und Verständnis verpflichtet. Er empfahl mir das Themengebiet für diese Arbeit und ließ mir große Freiheiten bei der Bearbeitung. Herrn Prof. Dr. Peter Chamoni gilt mein Dank für die Begutachtung dieser Arbeit, aber auch für seine Unterstützung bei der Überwindung der vielen Stufen, die zwischen dem ersten Semester und dem heutigen Tag lagen.

Für die gute Zusammenarbeit, viele Diskussionen und das angenehme Arbeitsumfeld am Lehrstuhl danke ich Christina Ackermann, Sarah Bretschneider, Demet Çetiner, Andreas Elias, Susanne Fronhoffs, Michaela Graf, Waldemar Grauberger, Nicole Jaschiniski, Igor Kozeletskyi, Klaus-Christian Maassen, Marc Maiwald und Kerstin Seekircher. Miriam Schwermer und Sophia Zorell gilt mein Dank für die Unterstützung bei der Literaturverwaltung und -recherche, Lakshita Chandana Wasalatantri für seine Hilfe bei der Implementierung der Algorithmen und der Datenauswertung. Ebenso gilt mein Dank dem Team des Lehrstuhls für Wirtschaftsinformatik und Business Intelligence, insbesondere Tanja Bley, Jens Kaufmann, Andrea Kiausch, Stefan Krebs und Caroline van der Sluijs hatten immer ein offenes Ohr und gute Ratschläge bei allen Problemen und Sorgen, aber auch viele fröhliche Momente mit ihnen werden mir in guter Erinnerung bleiben. Auch Nina Fiederling, Eva-Maria Schäfer und Annika Wienke sorgten für viele Lichtblicke im Uni-Alltag.

Bedanken möchte ich mich auch bei allen Freunden, die mir beim Sport und anderswo Ablenkung und damit einen klaren Blick auf die Dinge ermöglichten. Ganz besonders danke ich Sonja und Sebastian Radau für ihre endlose Geduld.

Diese Arbeit wäre nie ohne die Unterstützung meiner ganzen Familie vollendet worden, die mir auch in den schwersten Stunden Rückhalt und Mut gab. Lieben Dank an Euch alle dafür. Besonders hervorheben möchte ich meine Eltern Brigitte und Franz-Josef Reiners und meinem Bruder Philipp, ebenso Elisabeth Bölt und Christin Loskamp. Vielen Dank für all die kleinen und großen Dinge, die Ihr für mich getan habt! Euch widme ich diese Arbeit.

Aachen, im Dezember 2015 Christoph Reiners

Contents

List of Figures

List of Tables

List of Acronyms

AC	Arc Consistency
ACO	Ant Colony Optimization
AI	Artificial Intelligence
ALNS	Adaptive Large Neighborhood Search
B&B	Branch&Bound
B&P	Branch&Price
BJ	Backjumping
BT	Naive Backtracking
CG	Column Generation
CLP	Constraint Logic Programming
COP	Constraint Optimization Problem
CP	Constraint Programming
CSP	Constraint Satisfaction Problem
FC	Forward Checking
GA	Genetic Algorithm
LDS	Limited Discrepancy Search
LNS	Large Neighborhood Search
LP	Linear Program
LS	Local Search
MAC	Maintaining Arc Consistency
MC-*k*	Maintaining Strong *k*-consistency
MDPVRP	Multi-Depot Periodic Vehicle Routing Problem
MDVRP	Multi-Depot Vehicle Routing Problem
MDVRPHF	Multi-Depot Vehicle Routing Problem with Heterogeneous Fleet
MDVRPTW	Multi-Depot Vehicle Routing Problem with Time Windows
MDVRPTW-RH	Multi-Depot Vehicle Routing Problem with Time Windows and a Rolling Planning Horizon
MDVSP	Multi-Depot Vehicle Scheduling Problem
MILP	Mixed Integer Linear Program
OR	Operations Research
PC	Path Consistency
PRL	Path Relinking
PVRP	Periodic Vehicle Routing Problem
SA	Simulated Annealing

SDVRP	Split Delivery Vehicle Routing Problem
SS	Scatter Search
TS	Tabu Search
TSP	Traveling Salesman Problem
TSPTW	Traveling Salesman Problem with Time Windows
VNS	Variable Neighborhood Search
VRP	Vehicle Routing Problem
VRPTW	Vehicle Routing Problem with Time Windows

Chapter 1

Introduction

1.1 Motivation and Research Question

The transportation sector—and freight carriers in particular—underlies a permanent stress of competition that forces players to permanently adapt their processes to the current situation. Currently, carriers face two megatrends that drive this pressure: *Globalization* and *Environmental Aspects*.

Globalization describes the ongoing process of international integration which leads to supraterritorial connections of more and more people (Lechner, 2009, pp. 15–18). Its effects on the transportation sector and especially on carriers are twofold. On one hand, it incurs a growth in trade and hereby in transportation. The German Federal Ministry of Transport (BMVBS , 2007, p. 10) expects a raise of 71% in total until 2025 compared to 2004, the increment will be highest in road transportation (84%). Likewise, another recent development—the rise of *E-commerce* (Laudon, 2015)—causes a growth in the amount of goods that must be delivered to customers. That increases the number of local, national, and international delivery tours enormously. On the other hand, the globalization process incurs a situation with more intense competition. This may be tightened by the fact that competitors underlie fewer regulations and/or originate in countries with lower wage levels. In such a highly competitive environment according to Porter (2004, Ch. 2) there are two main strategies to gain competitive advantages: *Cost advantage* and *Differentiation* (The third introduced strategy *Focus* will be neglected here). Differentiation means that a carrier offers a service that is important for the customer. In return, the customer pays a surplus on the competitor's price. An example for such an advantage would be a reliable and accurate service or quick response times for new or changed orders. A cost advantage results from economies of scale. Carrier's costs decrease with a growing amount of transported goods. Because of advantageous cost structures the carrier is able to set lower prices than its competitors. The profit per unit is lower than in the case of diversification but the total profit results from a larger basis.

Discussion about the greenhouse effect draws social attention to **environmental aspects**, especially climate protection and green road freight transportation (Demir et al., 2003). For carriers this means that CO_2 reduction is a sales argument that can attract customers. While reduction can be achieved by replacing old trucks with new ones with lower emission levels in the long term, in the short run increasing utilized capacity can be an attractive option.

Therefore, the number of empty truck tours must be reduced as well as the unused capacity in less-than-truckload trips. This can be achieved by bundling demand from several customers that can be combined subject to load requirements.

Summing up, one can see that a high degree of capacity utilization is fundamental for success in the transportation market. This is most obvious for the strategy that seeks cost advantages: Per unit costs decrease with a higher tonnage. But even a carrier that follows a diversification strategy can gain profits from better capacity utilization. For, a faster service goes along with a better utilization of the vehicles, e.g. if additional demand is integrated in tour plans while a vehicle operates the tour. Of course, for both strategies reducing the empty truck tours results in a lower CO_2 emission and lower fuel costs. Currently, according to *Eurostat* (*ec.europa.eu/eurostat/*) 38% of all road transportation vehicle trips in the European Union have not carried any load in 2013 (667,897,000 journeys without freight in total). Christie and James (2006) claim that computer-generated tours can save up to 40% of fuel and greenhouse gas emissions.

An additional aspect that must be considered in this context is the rapid development in modern communication technologies that enables dispatchers to communicate easily with the drivers and allow real-time positioning (Ghiani et al., 2003).

All of these aspects can be treated by *Dynamic Planning*. It can be applied in situations where not all data are known at the moment the planning process is carried out for the first time. Demand and/or customers are hidden at the beginning of the planning process and will occur step by step. We will use a rolling horizon planning process to address these issues and describe its characteristics, problems and ways to handle them. For instance, there are not only new customers that must be inserted into the existing tours subject to restrictions on load and time, the planner must also take into account the customers that already have been visited as they influence capacities and temporal dependencies.

Beside the economic focus there is emphasis on Constraint Programming (CP) as an optimization technique. It originates from the fields of Logic Programming and Artificial Intelligence (AI) and has its strengths in solving combinatorial optimization problems. In the past years a lot of research has been made on the possibilities to integrate CP and Operations Research (OR) and to combine the advantages of both fields (Chandru and Hooker, 1999; Hooker, 2002, 2012). Although CP-based techniques perform well on some combinatorial problems there is only little research on the Traveling Salesman Problem (TSP) and the Vehicle Routing Problem (VRP). By developing some heuristics that make use of CP principles we want to overcome this issue. We limit ourselves to heuristics because an optimal solution of one step would not be carried out in total due to the permanent reoptimization. Therefore it is sufficient to compute feasible solutions that meet the customers' requirements. From the carrier's perspective the implementation of a rolling-horizon planning process produces better results compared to a period-to-period planning scheme. Therefore, developing general frameworks—that can be extended to compute optimal solutions in the future—may be sufficient at the moment.

1.2 Outline

This thesis is organized as follows: Chapter 2 defines the Multi-Depot Vehicle Routing Problem with Time Windows and a Rolling Planning Horizon (MDVRPTW-RH). A description is given in Section 2.1 and Section 2.2 introduces a Mixed Integer Linear Program (MILP) to model the problem. Section 2.3 sketches potential areas of applications. In the remainder of the chapter we survey variants from the literature and give an overview of solution methods.

The foundations of CP are presented in Chapter 3. After a brief summary of the origins and historic developments (Section 3.1), we introduce some definitions that are substantial for CP in Section 3.3. Afterwards, we will describe the solution process for CP models which is twofold: The core concept is *propagation* and Section 3.4 explains its idea and its importance within the solution process. The second component, search engines, is described in Section 3.6. Section 3.5 explains global constraints that simplify modeling and guide the solution process. The concepts of the entire chapter are illustrated on a running example of a TSP that is introduced in Section 3.2.

The focus of this thesis is on two algorithms to solve the problem described in Chapter 2. Both methods have in common that they incorporate techniques from CP in mathematical programming approaches. First, in Chapter 4 a Genetic Algorithm (GA) is introduced to solve the MDVRPTW-RH in a *cluster first, route second* manner. While the former part is done by a GA, the latter is solved by a CP model. We will describe the algorithm's operations in detail (Section 4.3) and its extension to deal with rolling horizon planning (Section 4.4). Furthermore, we will evaluate the solution by test instances that are well-known in literature in Section 4.5.

Chapter 5 introduces a Branch&Price (B&P) approach to solve the problem: Section 5.2 describes the components of the Column Generation (CG) process. Because solutions found by this process must not necessarily be integer, we embed it into a B&P algorithm in Section 5.3. The rolling horizon setting will be introduced in Section 5.4. We will use the same test data as in Chapter 4 to evaluate the algorithm and compare the performance of both algorithms.

The final Chapter 6 draws some conclusions. The key findings of this thesis are summarized and we suggest some directions and open questions for future research.

Chapter 2

The Multi-Depot Vehicle Routing Problem with Rolling Horizon Planning

The static Multi-Depot Vehicle Routing Problem (MDVRP) is a generalization of the VRP that allows vehicles to start from a specific number of different depots. The VRP describes the problem to service a set of customers under restrictions that limit the number of customers a single vehicle can visit. Therefore, multiple vehicles are required. We will extend the static MDVRP in this thesis to a dynamic problem: At specific points in time a reoptimization process is executed to update the tours and include additional demand that was unknown when routing decisions had been made formerly. Because there exists a vast amount of literature on the VRP we will focus on the specified problem in this chapter and refer to Toth and Vigo (2002b) for the foundations of this problem. Reviews can be found in Laporte (1992), Golden et al. (2008), Eksioglu et al. (2009), and Laporte (2009).

In this chapter we will start by introducing assumptions for the dynamic MDVRP (Section 2.1) that build the underlying characteristics for the heuristics in Chapters 4 and 5. To define the problem properly a MILP formulation of the MDVRP is given in Section 2.2. Afterwards, we will review literature relevant to this topic in Section 2.4. Here, the literature relevant to dynamic planning (Section 2.4.1) will be covered before we summarize the research on the MDVRP (Section 2.4.2).

2.1 Assumptions

The basic characteristics and underlying assumptions that apply to the MDVRPTW-RH in this thesis are described here:

Assumption 1. Every customer must be visited exactly once: The triangular inequality is valid, i.e. it is never cheaper to pass a third customer to arrive at another one than directly traversing to it. On the other hand, it is not possible to skip a customer in the planning process.

Assumption 2. Vehicles start at several depots: There is a set of depots and these depots can have different coordinates. Every customer can be serviced by any depot.

Assumption 3. Every vehicle has a maximum capacity that cannot be exceeded: The considered problem is a pick-up problem. Each vehicle starts empty at its depot and collects the demand at the customers. If the capacity is exhausted, the vehicle returns to the depot to discharge.

Assumption 4. Split-loads are not allowed: A vehicle must accommodate the total demand of each customer on its tour. If the remaining capacity is not sufficient to service a customer, the customer cannot be part of the tour.

Assumption 5. Time windows must be considered: While it is possible to arrive earlier at a customer and to wait for the ready time, it is strictly forbidden to arrive later than the due date. A service time may occur in a node, that means the vehicle must stay in a node for this time span. The service cannot start before the ready time.

Assumption 6. The tour length is limited: The maximum duration is subject to an upper bound. These restrictions can originate from technical issues (e.g. limitations on fuel, service requirements of a vehicle) or regulations (e.g. European Working Time Directive, labor agreement on driving hours).

Assumption 7. Every vehicle can service any customer: As mentioned above, we consider a pick-up problem. Vehicles collect goods from customers and carry them to the depot.

Assumption 8. The fleet size is limited to a number of vehicles: At every depot there is a fixed number of vehicles. It is mandatory that the number of tours originating at a specific depot is smaller or equals the number of vehicles in this depot. The number of vehicles is identical in all depots. Vehicles at every single depot are assumed to be homogeneous throughout this thesis.

Assumption 9. Every vehicle returns to its start depot: The last node in every tour must be a depot , i.e. every vehicle must return to its start depot. Therefore, the number of vehicles in every depot at the beginning equals the number at the end of the planning horizon.

Assumption 10. The total travel time should be minimized: The total travel time is the sum of time that is required to move between all customers and depots. However, waiting times at customers are excluded. The objective function neglects service times, too. But including them into the objective function would add a constant term because the number of customers as well as the service times are known and are present in every solution. However, the duration of the service is considered in the time window constraints to get feasible solutions.

Assumption 11. The tours are updated regularly: Planning takes place in a rolling horizon framework: There is a finite total planning horizon (e.g. a working day) that is divided into a fixed number of subhorizons (e.g. hours of a working day). At specified points in time (e.g. the begin of every working hour) additional customers (whose demand occurs while the vehicles are on their journeys) are integrated into the current routes where new orders must be serviced in the following subhorizons but not in the current one.

Assumption 11 is required because customer demand is not known for the full planning horizon in advance but becomes known successively. To handle this situation we introduce

a rolling planning horizon: The total planning horizon is divided into a specified number of smaller subhorizons which are optimized consecutively. At the beginning only an initial set of customers is known. Tours are planned on the basis of these information and vehicles start their trips based on this knowledge. At the beginning of any later subhorizon a new planning step takes place: The dispatcher incorporates the current position of the vehicles and the customers each vehicle has visited already. These customers are called *fixed customers*. Due to the fact that they have been visited in the past it is neither possible to change their position in a tour nor to adapt the point in time the customers are visited at. Additionally, there emerge new customer demands within the sub-horizon which is considered. Taking into account all current, new, and fixed nodes the algorithm reoptimizes the tours. Between two reoptimization steps no adaption of the plans is carried out.

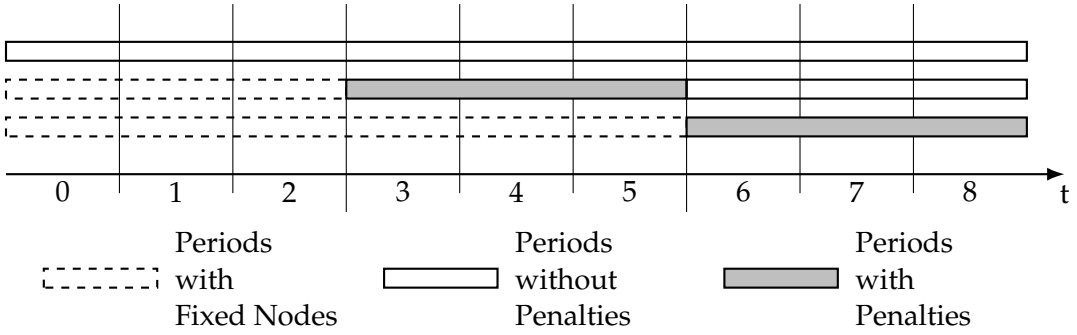
However, this planning procedure may result in some kind of *nervousness* (Kimms, 1998; Schönberger, 2011, p. 123). *Nervousness* regards the fact that a time is announced to a customer it should be serviced at but due to reoptimization this date is subject to change. For, to minimize the travel distances the algorithm rearranges nodes and therefore changes the arrival times. To mitigate this effect we introduce a penalty term to the objective function: Deviations from announced times are penalized by a term depending on the first announced and the new arrival time found in recomputation.

Figure 2.1 illustrates the procedure for a total planning horizon consisting of 9 periods which is subdivided into three subhorizons of three periods each, e.g. a nine-hour working day where announcements are made on a minute basis. In iteration 0 planning is done for the current customer set. There are no penalties because no arrival times were announced in the past. The resulting plan is fixed for the first subhorizon (i.e. the first three hours). In the next planning step (at the end of period 2) the process "rolls" to the next subhorizon that consists of working hours 3 to 5. Here, all nodes in the past are fixed but the plans for future subhorizons can still be changed. Therefore, from now on penalty terms are active. All customers whose announced arrival times falls into the third to fifth hour (grey bar) cause a penalty if the new arrival time differs from the announcement. Customers with announced arrival times after the end of period 5 (white bar with solid lines) will not get penalized as we assume that there is sufficient time to react to the new prediction. After planning for the second subhorizon was completed the parts of the tours that fall into this time span are fixed. Finally, in iteration 2 the plan for the third subhorizon is fixed. This time the penalty will apply to all customers that have been scheduled in the former planning step. Because the end of this subhorizon equals the end of the total planning horizon there are no further planning steps or subhorizons.

This idea of penalty terms is motivated by the fact that in practice it may be more problematic for a customer to react to rescheduling that deals with immediate dates than reacting to events in the distant future. In the former case he is not able to adapt his plans. In the worst case additional storing costs arise for the customers. Otherwise, if the rescheduled time is earlier than before, problems may occur because the cargo is not ready for transport. However, if the rescheduling event concerns the distant future, he can update his production plans himself and take the new situation into account.

Even if we deal with a rolling planning horizon we assume that vehicles that return to the depot due to the capacity limit (Assumption 3) cannot be rescheduled in the total planning

Figure 2.1: Penalties for Delays in the Rolling Horizon Setting



horizon. The idea is that we want to balance the workload over all vehicles in the total planning horizon and avoid that one vehicle services a large number of customers while others are idle.

Assumption 12. Each vehicle starts as late as possible: The vehicles do not start immediately at the beginning of the subhorizon but at the point in time that is the latest one to arrive at the customer in time. For instance, consider a situation where the initial planning step at 10:00 schedules a vehicle to arrive at a customer node at 12:00. The travel time between the depot and the customer is ten miuntes and there is only this customer in the tour. Thus, the vehicle starts at the depot at 11:50 (and not at 10:00). If the first reoptimization takes place at 11:00, the trip can still be canceled if the customers are rescheduled to other tours and no other customers are added to the tour.

We stress Assumption 12 because the following situation can arise: In the solutions of some steps more vehicles may be scheduled than in succeeding steps or even the final solution. However, the solutions with fewer trips are not incomplete: One or more trips may simply have been canceled before the vehicle started and its nodes are part of other tours now.

These assumptions will be recalled later in this thesis to justify parts of the algorithms and frameworks. In a first step we will describe a MILP that covers all these properties in the following section.

2.2 Mathematical Model

To give an exact definition of the problem we will introduce a MILP model: The problem can be stated as an undirected graph $G = (N, A)$, where N is a set of nodes representing the set of customers C and the set of depots D, that is $N = C \cup D$. The arcs A describe connections between nodes and are weighted with the travel time between nodes i and j: c_{ij}, where $i, j \in N$. Travel times are symmetric, i.e. $c_{ij} = c_{ji}$. Nevertheless, the total travel time can deviate from objective function value because of waiting times at customers (Assumption 10).

Routing decisions are made as follows: Consider a set of o depots $D = \{0, \ldots, o-1\}$. For every depot we distinguish between a start depot and an end depot. The set $C = \{o, \ldots, o+n\}$ contains (all) n customer nodes. The set $N_d = \{d, o, \ldots, o+n, o+n+d\}$ aggregates all customer and the depot nodes for tours that start from a specific depot d, where node d is the start depot and node $o+n+d$ is the end depot. Furthermore, we use $C_d^+ = C \cup \{o+n+d\}$ to represent

the customer set and the incoming depot $d \in D$ resp. $C_d^- = C \cup \{d\}$ to abbreviate the customer set and the start depot d. Every depot d is the starting point for a fleet of m homogeneous vehicles $F_d = \{0, \ldots, m-1\}$.

We will use the following notation to model the problem:

Parameters:

D	Set of depot nodes
C	Set of customer nodes (in the current iteration)
C'	Set of all customers of the previous iteration
$N = C \cup D$	Set of all nodes (in the current iteration)
N_d	Set of start and end depot for depot $d \in D$ and all customer nodes in C
F_d	Set of vehicles in depot $d \in D$
c_{ij}	Travel time between node i and j, where $i, j \in N$
a_{di}	Demand of node $i \in N_d$ for vehicles from depot $d \in D$
e_i	Ready time of node $i \in N$
l_i	Due date of node $i \in N$
v_d	Number of homogeneous vehicles in depot $d \in D$, i.e. $v_d = \lvert F_d \rvert$
Q_d	Capacity of a single vehicle from depot $d \in D$
w_i	Tolerated delay at customer $i \in C'$
p_i	Penalty for delayed time units at customer $i \in C'$
s_{di}	Service time for node $i \in N$ for vehicles from depot $d \in D$
t'_i	Arrival time computed in the last iteration for all postponed nodes $i \in C'$
M	Sufficient large number, e.g. $M = \max_{i \in D} l_i$

It may be confusing that demand a_{di} and service time s_{di} depend on a depot but this generalization is necessary for the model to cope with the characteristics of the rolling horizon framework in Chapter 5. In general, we assume that both values are equal for all depots.

Variables:

x_{dkij}	$\begin{cases} = 1, & \text{if vehicle } k \text{ of depot } d \text{ goes directly from customer } i \text{ to } j \\ = 0, & \text{else} \end{cases}$
t_{dki}	Arrival time of vehicle k of depot d at node i
f_i	Time lag (too early) at node i compared to former iteration (in time units)
g_i	Time lag (delay) at node i compared to former iteration (in time units)
b_i	$\begin{cases} = 1, & \text{if penalty is active in node } i \\ = 0, & \text{else} \end{cases}$

$$\textbf{MDVRPTW-RH: } \min \sum_{d \in D} \sum_{k \in F_d} \sum_{i \in C_d^-} \sum_{j \in C_d^+} c_{ij} x_{dkij} + \sum_{i \in C'} p_i b_i \tag{2.1}$$

s. t.

$$\sum_{j \in C_d^+} x_{dkdj} = 1 \qquad d \in D, k \in F_d \tag{2.2}$$

$$\sum_{i \in C_d^-} x_{dki(o+n+d)} = 1 \qquad d \in D, k \in F_d \tag{2.3}$$

$$\sum_{k \in F_d} \sum_{j \in C_d^+} x_{dkdj} = v_d \qquad d \in D \tag{2.4}$$

$$\sum_{d \in D} \sum_{k \in F_d} \sum_{i \in C_d^-} x_{dkij} = 1 \qquad j \in C \tag{2.5}$$

$$\sum_{d\in D}\sum_{k\in F_d}\sum_{j\in C_d^+} x_{dkij} = 1 \qquad i \in C \tag{2.6}$$

$$\sum_{i\in C_d^-} x_{dkij} - \sum_{i\in C_d^+} x_{dkji} = 0 \qquad d \in D, k \in F_d, j \in C \tag{2.7}$$

$$t_{dki} + c_{ij} + s_{di} - (1 - x_{dkij})M \leq t_{dkj} \qquad d \in D, k \in F_d, i, j \in N_d \tag{2.8}$$

$$t_{dki} \geq e_i \qquad d \in D, k \in F_d, i \in N_d \tag{2.9}$$

$$t_{dki} \leq l_i \qquad d \in D, k \in F_d, i \in N_d \tag{2.10}$$

$$\sum_{i\in C_d^-}\sum_{j\in C} a_{di} x_{dkij} \leq Q_d \qquad d \in D, k \in F_d \tag{2.11}$$

$$t_{dki} + f_i \geq t'_i - (1 - \sum_{j\in C_d^-} x_{dkji})M \qquad d \in D, k \in F_d, i \in C' \tag{2.12}$$

$$t_{dki} - g_i \leq t'_i + (1 - \sum_{j\in C_d^-} x_{dkji})M \qquad d \in D, k \in F_d, i \in C' \tag{2.13}$$

$$f_i + g_i \leq w_i + b_i M \qquad i \in C' \tag{2.14}$$

$$f_i \geq 0 \qquad i \in C' \tag{2.15}$$

$$g_i \geq 0 \qquad i \in C' \tag{2.16}$$

$$t_{dki} \geq 0 \qquad d \in D, k \in F_d, i \in C \tag{2.17}$$

$$u_i \geq 0 \qquad i \in C \tag{2.18}$$

$$b_i \in \{0, 1\} \qquad i \in C' \tag{2.19}$$

$$x_{dkij} \in \{0, 1\} \qquad d \in D, k \in F_d, i, j \in N_d \tag{2.20}$$

The objective function (2.1) minimizes the sum of the pure travel time of all vehicles and the penalties for postponement. These penalties occur from the fact that some nodes in the set of N' may be visited earlier or later than announced. Constraints (2.2) and (2.3) state that a vehicle must leave its start depot and arrive at the end depot (Assumption 2). Constraint (2.4) limits the number of vehicles that start from one depot to its fleet size (Assumption 8).

Constraint (2.5) states that every customer must be serviced exactly once, i.e. exactly one vehicle must arrive at each customer node (Assumptions 1 and 4). Constraint (2.6) forces one vehicle to leave each node. Flow conservation is ensured by Constraint (2.7): If there is a vehicle arriving at a customer node from any other node, it must proceed to any other node. It also ensures that the same vehicle enters and leaves a node as by Constraint (2.6) only *any* vehicle is considered. Because of the structure of (2.5)–(2.7) either (2.5) or (2.6) could be relaxed without changing the solution of the model.

Constraint (2.8) states that the arrival time at node j must be later than the arrival time at the preceding node i plus the service in node i and travel time between the two nodes c_{ij} if j is the direct successor of i: $x_{ij} = 1$. Otherwise, the constant term M is subtracted from the right-hand side to guarantee that the inequality is feasible. The arrival time has to be equal to or later than the ready time (Constraint (2.9)) but not later than the due date (Constraint (2.10)) to hold Assumption 5. Note that it is possible to "arrive" earlier at a node and wait until the customer is ready for service. Therefore, *arrival time* refers to the point in time when the service starts. Nevertheless, as the term is common in literature we will use it throughout this thesis.

Furthermore, we have chosen to use an explicit formulation to consider the service time of a node although it is common in literature to include it in the travel time towards this node. This formulation is advantageous as we only want to evaluate the pure travel time in the objective function (2.1) (Assumption 10).

Constraint (2.11) limits the load of a vehicle to its capacity (Assumption 3). The sum of demand over all customers on the vehicle's tour starting at depot d cannot exceed the amount of Q_d.

The remaining constraints compute the penalties of the announced arrival time (Assumption 11): Constraint (2.12) computes the lead time f_i (a node is visited earlier) between the announced arrival time and the arrival time in the current iteration in node i. Constraint (2.13) does the same for the lag time g_i (node is visited later) in node i. Note that at most one of these variables will take a value larger than 0 in an optimal solution. Therefore, Constraint (2.14) sums up the values and checks if the deviation is larger than the allowed tolerance w_i in node i. If so, the binary variable b_i will be set to 1 and the penalty will be active in the objective function (2.1). We use the big M formulation again because only one vehicle arrives at a particular node and the other vehicles may not cause penalties. This means, that a fixed penalty will be added for every postponed node. Constraints (2.15)–(2.18) restrict the values of the concerned variables to be non-negative, Constraints (2.19)–(2.20) define the binary variables.

Because subtours are eliminated by the time window constraints we do not need any additional constraints to avoid them: A subtour would result in a situation where it would not be possible to set a unique arrival time for the nodes in it.

An additional constraint considering the maximum length of a tour (that includes waiting times) could be

$$t_{dk(o+n+d)} - t_{dkd} \leq dur \qquad d \in D, k \in F_d$$

to limit the duration of every tours to dur.

The model (2.1)–(2.20) is actually redundant: The decision *which* vehicle from a specific depot services a customer does not influence the quality of the solution. However, this information is required in the rolling horizon setting where we include any constraints on fixed nodes: To provide a complete model we introduce the set Fix that is updated within the planning framework but outside the model and contains four-tuples (d, k, i, j) of a depot d, a vehicle k and the nodes i and j that should be fixed in Constraint (2.21):

$$x_{dkij} = 1 \qquad (d, k, i, j) \in Fix \tag{2.21}$$

Furthermore, a set $FixTimes$ is used to keep track of the arrival times at the fixed nodes t_i^{fix} that must be assigned to their value from the former iteration in Constraint (2.22):

$$t_{dki} = t_i^{\text{fix}} \qquad (d, k, i) \in FixTimes \tag{2.22}$$

We will show a different way to handle fixed nodes in Chapter 5 of this thesis that does not necessarily require any additional constraints in the rolling horizon framework. This approach would allow us to skip the decision for a specific vehicle in the variables x_{dkij} and t_{dki}.

Next, we will consider the complexity of this problem. First of all, the MDVRPTW-RH can be reduced to the MDVRP by the following assumptions:

- The time windows are negligible, i.e. $e_i = 0, i \in N$ and $l_i = \infty, i \in N$.
- The penalty factor is set to 0, i.e. $p_i = 0, i \in N$.
- Sets C' and $N'$are empty. There are no nodes whose postponement must be considered.

As we have stated above, the MDVRP is a generalization of the VRP if the number of depots is set to 1, i.e. $D = \{0\}$ and $|D| = 1$. It has been proven that the VRP is $\mathcal{NP}$-hard, unless $\mathcal{P} = \mathcal{NP}$. The proof can be found in Karp (1972), Bertossi et al. (1987), and Toth and Vigo (2002a); we refer to Lenstra and Kan (1981) for a broader overview on complexity results regarding other types of routing problems. A general introduction to complexity can be found in Garey and Johnson (1979).

Besides the unique penalty term there are further ideas to consider postponed nodes:

- **Number of deviations from announced appointments:** This indicator counts how often all customers are rescheduled (over all periods). The result is the number of rescheduling events during the planning horizon. Because an absolute number may not be significant, the ratio of rescheduled customers to the total number may be used. To soften the constraints a buffer may be introduced: Only if the absolute difference $|\hat{t}_i - \bar{t}_i|$ of rescheduled arrival time $\hat{t}_i$ and announced arrival time $\bar{t}_i$ undercuts or exceeds a threshold, customer i is marked as rescheduled.
- **Total deviation from announced appointments:** The former indicator does not consider the total deviation. However, in a company it may make a difference if a transport is only one minute later than the announced time (considering the buffer) or the carrier delays for some days. Therefore a more valid indicator catches the total deviation using a unit of time at all customers $i \in N$: $dev_{time} = \sum_{i \in N} |\hat{t}_i - \bar{t}_i|$. The penalty term in the objective function (2.1) would be replaced by a summand $dev_{time} \cdot b$, where b would be a global penalty factor per time unit.

2.3 Managerial Impact

In this section we want to give a brief description to some applications and sketch the economic advantages that can result from rolling horizon planning. The area of application can be divided into three stages according to the length of the time horizon in (a) short-haul planning, (b) medium-haul planning, and (c) long-haul planning.

Short-Haul Planning

Short-haul planning has a very short planning horizon that consists of a single day or even a few hours. The focus within this area is on service operators. Short latency on customer demands is a fundamental principle in this sector as it heavily influences customer satisfaction and services are substitutable in many cases. Unsatisfied customers tend to choose a different operator.

An area of application could be a parcel service that delivers and collect packages. Customers announce pick-up orders during the day. All vehicles start in the morning delivering parcels (and picking up already known ones) and receive additional requests online during their tours. Then, the parcel can be collected on the same day. Examples can be found in Ichoua et al. (2000) and Ninikas and Minis (2014). In special situations a parcel could be delivered immediately on the same tour, i.e. by a bike courier or in express mail (Gendreau et al., 2006).

Ferrucci et al. (2013) describe a situation that focuses on a pure delivery situation: In press delivery some subscribers may not receive their newspaper due to thievery or a delivery failure. To improve customer satisfaction a client should get a new copy of its paper on the same day. Therefore, it is necessary to update the delivery tours immediately.

Another field of interest may be the traveling repairman problem, where a repairman has to service customers (repair their machines) and to minimize the total waiting time (latency) of all machines (García et al., 2002; Luo et al., 2014). It may be efficient to insert new tasks into the route, especially if the profits depend on the waiting time (Dewilde et al., 2013). Of course, including additional tasks into a route is limited by the (free) time to fulfill it. Therefore it may be easier to set up a rolling horizon planning method for a parcel service (that requires only little capacity relative to the vehicle's total capacity) than to update routes of maintainers where tasks can consume a large time of the working day. On the other hand, maintenance tasks can be urgent which stresses the importance of flexible planning approaches. Krumke et al. (2003) introduce an online planning method for this problem.

Further applications can be found in distribution of heating oil (Chen and Xu, 2006) and waste collection (Pang and Muyldermans, 2012) that can be time-critical in some cases.

Medium-Haul Planning

Medium-haul planning can be applied in road networks. It deals with planning horizons that are longer than one day. Examples for this class of problems are Dynamic Multi-Period Vehicle Routing Problems. In these problems, demand occurs over time but must not necessarily serviced on the current day but in one of the following periods. The decision process determines on which day a customer will be serviced and computes tours for the different vehicles on specific days (Mourgaya and Vanderbeck, 2006). However, every vehicle must start and finish in a depot on every day. Customer demand emerges over time and might have a due date, i.e. it must be fulfilled on the same day or in the following days. Examples can be found in Angelelli et al. (2007) and Angelelli et al. (2009, 2010).

Economic benefits in this situation result from short latency as described above and a better utilization of the vehicles' capacity. For, the dispatcher gains more flexibility in the planning process. Even if capacity is not considered, flexibility could be advantageous to balance tour duration et cetera. Furthermore, it is easier to cluster nodes to avoid detours. Nodes can be added to a cluster on the same day which results in the avoidance of extra tours if this cluster will not be serviced in the following days but the order is due before the next trip to this region.

Another aspect is to include demand from online freight marketplaces (Song and Regan, 2001). Carriers may bid on load to avoid empty trucks that return to the depot after they

have fulfilled their delivery task. As the demand must not necessarily be known when the vehicle starts the tour must be updated. This can be for full truckload or less-than-truckload transportation. Klundert and Otten (2011) describe this setting.

Long-Haul Planning

An application for long-haul planning (Ghiani et al., 2004, Ch. 6) may occur in ship scheduling. Especially two types of the common ship services are promising for rolling horizon planning: *tramp shipping* and *operating own ships*. Tramp ships act like cabs and wait for orders from customers. It can be efficient for the ship owner to update the plans regularly to pick up additional demand in harbors that are on this route. However, the decision maker has to ponder the elongated travel time and the additional revenue. The same is true for companies that use their own ships. Additional load from third parties could be transported and the utilization of capacity could be improved. A review on ship routing and scheduling can be found in Meng et al. (2014). Illustrative examples for ship scheduling under uncertainties can be found in Agra et al. (2013) and Tirado et al. (2013).

Another area of application for long term planning is long-haul transportation in road networks. The same characteristics as mentioned for the ship sector can be applied considering a very long planning horizon, e.g. trucks that cross Europe (Pankratz, 2005).

As can be seen from the three stages rolling horizon planning offers chances to gain competitive advantages resulting from more satisfied customers and higher capacity utilization. Furthermore, rolling horizon planning in the traffic sector is not limited to a single domain but can be applied universally. To keep the description general we will use the terms *vehicle*, *depot*, *order*, and *customer* in the remainder of this thesis.

2.4 Literature Review

In this section we will review literature that is relevant to the planning problem that is considered in this thesis. We will start in Section 2.4.1 with the field of dynamic planning for VRPs. The main part is Section 2.4.2, where we review the research on the MDVRP.

2.4.1 Dynamic Vehicle Routing Problems

According to Psaraftis (1995) there are two dimensions regarding dynamism in VRPs. The *evolution of information* describes whether a dispatcher's information basis is subject to change during the operation of routes, e.g. new customer demand arises. If so, these problems are called *dynamic*. If all information remain unchanged over time, the problem is *static*. The second characteristic is *quality of information*, i.e. the uncertainty in the problem. Problems can be classified to be *deterministic* or *stochastic*. In general, there are two reasons for dynamism: customer demand and travel times. In this thesis it will be assumed that customer demand becomes known over time, but does not change once it has appeared. The travel times between all customers and depots are given in advance and remain constant. Thus, we deal with

a dynamic and deterministic problem. More information about the other ideas can be found in Ghiani et al. (2003) and Pillac et al. (2013). Basics and characteristics of Dynamic VRPs are introduced in Psaraftis (1991).

For dynamic and deterministic routing problems there are two major methods for planning: *Periodic optimization* and *continuous reoptimization*. *Periodic optimization* solves a static problem at the beginning of the planning horizon that includes all known data. After a fixed interval of time (or every time new demand arrives), new customer demand will be added to the instance and the static problem will be solved again on the current status. This procedure is repeated until the end of the planning horizon. The approach is advantageous because (well-known) solution methods for static problems can be used, while solving the optimization problem repeatedly incurs longer computation and therefore waiting time. This procedure is called rolling horizon planning in other fields of OR; we refer to Sahin et al. (2013) for a review on rolling horizon planning in Supply Chain Management.

The first paper that uses this planning procedure is Psaraftis (1980) who solve a new dial-a-ride problem to every time an additional customer request arrive by dynamic programming. Yang et al. (2004) apply the same reoptimization strategy for the Pickup&Delivery Problem but solve a Linear Program (LP) instead. Pankratz (2005) provide a GA that is executed every time new customer demand was announced to the dispatcher.

The study of Kilby et al. (1998) describes a framework for the VRP that updates routes only after a certain interval and not immediately on changes in the demand pool. The framework plans the tours at the beginning of a day with all known nodes and updates the routes with an algorithm that adds new customers at the position which increases total costs least. Montemanni et al. (2005) solve the same problem setting by Ant Colony Optimization (ACO). A CG procedure is presented by Chen and Xu (2006). They use existing columns from the last decision point in time to derive new columns for the current optimization period.

The second method for planning in dynamic and deterministic problems is *continuous reoptimization*. Instead of computing fundamentally new solutions, this approach loads good solutions from memory and fits them to the new situation.

Gendreau et al. (1999) use a Parallel Tabu Search to solve a series of static VRPs. Every time a new request arrives the execution of the algorithm selects a good solution from a pool and uses it as an initial solution for the Tabu Search (TS) on the extended problem instance. Bent and van Hentenryck (2004) present an approach that permanently maintains a pool of feasible solutions. These solutions are adapted to the current events (mainly vehicles arriving at customer nodes and new customer demand). Another heuristic, a GA, is presented by Barkaoui and Gendreau (2013). It is able to add customer demand to the data set during the runtime of the algorithm. The problem setting is that each vehicle is informed about the next customer it shall head to not before it arrives at the current customer. This assumption allows long runtime and permanent reoptimization. The work of Rousseau et al. (2013) describes a procedure that inserts new customer demand in existing tours at the moment of its occurrence and applies a Local Search (LS) algorithm in between.

A drawback of continuous reoptimization is that the arrival times tend to be volatile as it only gets known when the vehicle starts its trip to that customer. Because we want to limit

the degree of nervousness and provide reliable arrival times we will prefer rolling horizon planning throughout this thesis.

Another interesting aspect is the combination of dynamism and collaboration in VRPs: In Wang and Kopfer (2014) combinatorial auctions (Abrache et al., 2007) are used in a collaborative tour planning problem with less-than-truckload requests. In this scenario carriers can keep their sensitive information unexposed. The authors extend their approach to deal with rolling horizon planning with a fixed interval length in Wang and Kopfer (2013) and to deal with a second planning scheme that uses different triggers in Wang and Kopfer (2015). Wang et al. (2014) broaden the focus and allow forwarding requests to third-party carriers besides exchanging them within the collaboration. Kimms and Kozeletskyi (2015) investigate how the cost in a cooperation between different salesmen should be allocated from a game theoretic point-of-view. They use an Approximate Dynamic Programming approach to handle stochastic (future) demand. In Kimms and Kozeletskyi (2013) the game-theoretic concept *core* is used to obtain an allocation between carriers.

2.4.2 Multi-Depot Vehicle Routing Problem

In general, there is a vast amount of literature on the VRP. Nowadays, many authors intend to apply the problem to real-life situations and current research deals with problems like electric vehicles (Schneider et al., 2014), E-commerce (Yanik et al., 2014), and green vehicles routing problems (Lin et al., 2014). Because these situations are complex many VRP variants are combined. We aim to focus the review on the MDVRP in this section but for the reason mentioned before the characteristics are mixed up. For detailed information on the problem variants we refer to Bräysy and Gendreau (2005a,b), Kallehauge et al. (2005), and Kallehauge (2008) for the Vehicle Routing Problem with Time Windows (VRPTW), to Archetti and Speranza (2012) for the Split Delivery Vehicle Routing Problem (SDVRP), and to Mourgaya and Vanderbeck (2006) for the Periodic Vehicle Routing Problem (PVRP). Vidal et al. (2013a, 2014) try to unify the solutions methods for the numerous variants of the VRP and define the Multi-Attribute VRP. These attributes are the specific constraints that add special characteristics to a problem.

To organize the literature review that deals with the MDVRP we will use the classification scheme of Bodin and Golden (1981) that is given in Table 2.1 on page 17. This taxonomy presents some characteristics that describe vehicle routing and scheduling problems. However, it is a very general framework. Because we limit the review to the MDVRP we will only consider parts of some characteristics. That are problems with time windows (A,2) or unspecified times (A,3) and multi-depot problems (B,2) with more than one vehicle (C,2).

Before we start with the overview, we want to clarify the difference between the MDVRP and the Multi-Depot Vehicle Scheduling Problem (MDVSP). The first one builds routes, which are sequences of customers that a vehicle must traverse in order. All tours start and end in the vehicles' depots. In a *schedule* points in time of arrival and departure are added to each node. If these times are predetermined, the problem is a MDVSP (Bodin and Golden, 1981). Therefore, the MDVSP belongs to (A,1) in the taxonomy and is not considered here. There exist various papers that deal with heuristic and exact solution approaches, e.g. Mesquita and Paixão (1992), Desaulniers et al. (1998), Kliewer et al. (2006), Hadjar et al. (2006), Pepin et al. (2009), and

Groiez et al. (2013). For the sake of completeness we will sketch the field of the Multi-Depot Multiple TSP (Kara and Bektas, 2006; Benavent and Martínez, 2013): A set of tours is built for the salesmen—that start from and finish at different depots—to service all nodes at minimal costs.

Another, more detailed taxonomy is suggested by Eksioglu et al. (2009). Several characteristics have been added and the scheme is finer grained. However, this classification is too detailed to provide a broad overview and we limit our scope to the more general taxonomy of Bodin and Golden (1981).

Table 2.2 gives an overview on the classification, where the notation is the same as in Table 2.1. Note that only *additional* constraints are considered in this table. All problems have the following constraints in common (if no contradictory constraint is posted): (1) All customers must be served, (2) Each customer must be visited by exactly one vehicle, (3) All tours must start and end in the same depot, and (4) There are no subtours in a solution.

Solution methods for the MDVRP can be divided into two subclasses: Those that simultaneously assign customers to depots and build the routes and those that do the assignment first and compute routes in the second step.

In the first papers published on this topic the savings heuristic (Clarke and Wright, 1964) is an important part of the solution algorithms: The original algorithm joins routes whose savings value $s_{ij} = c_{0i} + c_{0j} - c_{ij}$ is largest, where 0 is the (single) depot.

The MDVRP is presented first by Tillman (1969). In that early stage the author use the term *multi terminal delivery problem* but a terminal corresponds to a depot in this work. The author assume that the demand is probabilistic and further restrictions are present, i.e. a maximum travel distance per vehicle. A customer node is assigned to its closest depot node and a vehicle is assigned to every customer node, initially. The author claim to present the first solution method for the MDVRP, which is a modified savings heuristic. To include multiple depots, one needs to compute a modified distance between every depot and each nodes using the formula $\tilde{c}^i_j = \min_{d \in D} c_{dj} - (c_{ij} - \min_{d \in D} c_{jd})$, where the superscript denotes the depot and the subscript is a customer node. The result is largest for the combination of node and depot that are closest to each other and is negative for large distances. Then, the savings are calculated by $s^i_{jk} = \tilde{c}^i_j + \tilde{c}^i_k - c_{jk}$. The objective is to maximize the sum of savings, whereby only tours are combined that start/end at the same depot. Negative values of $\tilde{c}^i_j$ reduce the savings and a closer depot is selected instead. This approach is extended in the work of Tillman and Cain (1972) who integrate a Branch&Bound (B&B) search in that a row reduction matrix is applied in every node of the B&B tree. The objective is to maximize the savings again. Starting with a solution where every node is assigned to a single tour the upper bound is 0 because no tours have been joined yet and thus there are no savings. The upper bound is obtained by a process similar to the Hungarian method (König, 1916; Kuhn, 1955). The branching procedure fixes and forbids specific connections between nodes. Golden et al. (1977) improve the algorithm of Tillman and Cain (1972) in terms of memory management. These changes enable them to solve larger instances. Perl and Daskin (1985) solve the MDVRP within their three-step solution heuristic for a warehouse location problem. A solution is obtained by using a savings-based heuristic, too. Another savings-based approach is introduced by Benton (1986). The algorithm

Table 2.1: Taxonomy of the VRP (Bodin and Golden, 1981)

Characteristics			
A	Time to Service a Particular Node or Arc	1	Time Specified and Fixed in Advance
		2	Time Windows
		3	Time Unspecified
B	Number of Depots	1	One Depot
		2	More Depots
C	Size of Vehicle Fleet Available	1	One Vehicle
		2	More than One Vehicle
D	Type of Fleet Available	1	Homogeneous Vehicles
		2	Heterogeneous Vehicles
E	Nature of Demand	1	Deterministic
		2	Stochastic
F	Location of Demands	1	At Nodes
		2	On Arcs
		3	Mixed
G	Underlying Network	1	Undirected
		2	Directed
		3	Mixed
H	Vehicle Capacity Constraints	1	Imposed: All the Same
		2	Imposed: Not All the Same
		3	Not Imposed
I	Maximum Vehicle Route-times	1	Imposed : All the Same
		2	Imposed : Not All the Same
		3	Not Imposed
J	Costs	1	Variable or Routing Costs
		2	Fixed Operating or Vehicle Acquisition Costs
K	Operations	1	Pick-Up Only
		2	Delivery Only
		3	Mixed
L	Objective	1	Minimize Routing Costs Incurred
		2	Minimize Sum of Fixed and Variable Costs
		3	Minimize Number of Vehicles Required
M	Other (Problem-specific) Constraints		

works in a cluster first, route second manner, where groups of customers are built first and the routing decision is made on single clusters in a second step. The procedure computes initial solutions based on a variant of the savings heuristic first. Afterwards, the single tours obtained in the first step are optimized by an adapted version of the B&B algorithm presented by Little et al. (1963) that is capable to solve TSPs. Chao et al. (1993) use a two-phase approach: They build an initial solution by assigning each customer to its closest depot and carry out the modified savings heuristic described in Golden et al. (1977). Improvement is done by a *one-point movement* that deletes a node from one tour and adds it to another tour (at the same or another depot) if the total tour length decreases. Additionally, a *clean up step* and *2-opt* improvement (Lin, 1965) are carried out to improve the solution. Sumichras and Markham (1995) deal with a MDVRP variant where a fleet of trucks must supply a number of plants with raw materials from a number of sources. Each source offers only a single kind of raw material and may has different prices for it. On a tour vehicles pick up raw material, deliver it to the plants and pick up the next raw material, deliver it and so on. A solution is obtained by a modification of the savings algorithm.

Besides these papers on the savings algorithm there is another idea for a heuristic solution method: Customer are iteratively added to a tour in a position that increases cost least (*cheapest insertion heuristic*). Wren and Holliday (1972) introduce an algorithm that follow this idea and construct several tours simultaneously. It is able to deal with restrictions on load, tour length and a maximum number of vehicles per depot. The algorithm is divided into two phases, the first one constructs initial solutions while the second one executes so-called *refinement steps* to improve the current solution by small changes. To construct an initial solution all customers are serialized and added to a vehicle whose increase in travel distance is least (regarding tour length and capacity constraints). Seven refinement procedures are executed iteratively until no more improvements can be achieved. Another heuristic is presented by Gillett and Johnson (1976). Their cluster first, route second algorithm is based on the idea of decomposing the total set of nodes into smaller sets of customer nodes and one depot each. After this decomposition it is possible to compute near-optimal solutions by a modified sweep heuristic (Gillett and Miller, 1974). However, the quality of the solution depends strongly on the composition of the clusters. To obtain good clusters the authors suggest a cost-based insertion heuristic that assigns customer nodes to the depot where the extra travel distance is minimized. If a node is equidistant from two depots, a combination of the actual and the closest unassigned node is used to make the insertion decision. After the initial solution (based on the sweep algorithm for single tours) was computed, a reassignment procedure tries to improve the solution by transferring some nodes to other tours.

Ball et al. (1983) suggest two *route first, cluster second* algorithms that are based on insertion techniques. They rely on the idea of Beasley (1983) to relax constraints of the subproblem to find a giant tour of all nodes and divide this tour into smaller ones that are feasible in terms of duration and load. A third algorithm is a savings-based insertion heuristic. It turns out that the insertion heuristic produces better results for instances with a small number of tours while the route first, cluster second algorithms outperforms the former heuristic on data sets with a large number of tours.

Salhi and Nagy (1999) introduce insertion operators for the MDVRP with Backhauls. One

or more backhauls are inserted gradually in a planned route for line-haul customers. The insertion decision is made on the basis of cost information. Yang and Chu (2000) develope a simple heuristic for the Multi-Depot Periodic Vehicle Routing Problem (MDPVRP) that adds the customers—sorted according to their periodic demand—to existing routes one after another. If no tour maintains the restrictions, a further vehicle is chosen to start a tour.

The work of Giosa et al. (2002) neither fits exactly in the class of saving algorithms nor the insertion heuristics, but works on a similar principle. The authors introduce a *cluster first, route second* approach, too. They present algorithms that rely on three different measures: (1) *Assignment through urgencies* computes a score based on the distance of a customer to another/all other depots. The customer with the greatest score is assigned next. (2) *Cyclic assignment* iteratively adds nodes that are close to a node that is currently at the end of a tour to the same depot. (3) *Assignment by clusters* tries to build compact clusters according to the distance. It is obvious that no assignment procedure uses the time window information to get good assignments but only checks for feasibility. Tansini and Viera (2006) address this issue and introduce new proximity measures for the assignment phase that take care of time windows and travel times between nodes. Lim and Wang (2005) present a single-step approach to solve the MDVRP: They try to insert every node in all vehicle routes and estimate the tour length. A customer is added to the tour (and simultaneously to the depot) whose increase in distance is minimal over all tours and where sufficient capacity is available. This process is repeated until all customers are assigned.

A third stream in the MDVRP literature is about metaheuristics: Cassidy and Bennett (1972) presents *TRAMP*, a program to improve delivery routes in a real-world problem of servicing school canteens. Its overall goal is to minimize the delivery time. *TRAMP* is able to generate an initial solution itself or to import an existing solution. These solutions are improved by exchanging the position of a delivery node (school) in a tour or assigning tours to new depots (kitchens) which could be seen as a kind of neighborhood search. The program has an option to include time windows at delivery nodes or an upper bound for finishing all tours and is able to handle vehicles of different sizes. Renaud et al. (1996) provide the first "real" metaheuristic for the MDVRP. The TS algorithm starts with a set of initial tours that is found by assigning customers to depots and solved a VRP afterwards. The TS itself applies three basic operators that are based on the concept of changing the position of nodes in a tour or exchanging them between tours in different phases. Cordeau et al. (1997) present another TS to solve the MDVRP. It used a subroutine *GENI* (Gendreau et al., 1992) to insert and remove customers depending on the distance to their closest neighbors. In the search process a node that is removed from a current tour and inserted into another one and cannot be part of its original tour for a given number of iterations (except it improves the objective). Cordeau et al. (2001) present a further TS that removes customers from routes and inserts them in other routes in the position where the objective value is increased least. The objective function captures the distance and penalties that depended on maximum tour duration, maximum load and waiting time at customers. Again, reinsertion of a customer in its "old" route is forbidden for a given number of iterations.

The paper of Hadjiconstantinou and Baldacci (1998) deals with the MDPVRP and introduces a heuristic that includes four steps: (1) Assign customers to the nearest depot. (2) Assign visit combinations (customer and day) for every depot based on approximate cheapest inser-

tion information. (3) Solve a VRP for every day using TS. (4) Interchange subsets of customers from single tours. Tarantilis and Kiranoudis (2002) introduce a list-based threshold accepting algorithm for dispatching in a real-world problem. The algorithm is similar to Simulated Annealing (SA) with node exchange but uses a list that manages the acceptance of worse solutions and is regularly updated by the algorithm itself.

Crevier et al. (2007) deal with a MDVRP with inter-depot routes. In their model depots can act as intermediate replenishment facilities for vehicles. The solution procedure is a TS heuristic based on *GENI* (Gendreau et al., 1992). The tabu list contains customer-route combinations that are forbidden for a number of iterations. However, they are revoked if they improve the solution. A Variable Neighborhood Search (VNS) variant is presented by Polacek et al. (2004). They use a *shaking* procedure that exchanges parts of the routes (*CROSS-exchange* (Taillard et al., 1997)) to get new neighborhood structures. Solutions obtained by shaking undergo a local search phase with a *3-opt* operator with special characteristics to escape from local optima (sequences cannot be inversed to keep solutions feasible with respect to time windows, limits on the length of exchange sequence to three to guarantee short computation times). Some improvement procedures to solve the Multi-Depot Vehicle Routing Problem with Pickup and Deliveries that manipulate customers' and depots' positions in routes are presented in Nagy and Salhi (2005). The heuristic framework works with two different levels of infeasibility: (1) Solutions are called weakly feasible if all tours' lengths does not exceed the maximum duration and neither the *total* pick up nor *total* delivery demand of any tour is larger than the capacity of any vehicle. (2) Strong feasible tours have a duration that is shorter than the upper bound and the vehicles' load is smaller than the capacity on *every arc*. Search starts with weak feasible solutions and is guided to strong feasible solutions.

Pisinger and Ropke (2007) transform the MDVRP (and other variants of the VRP) to a Rich Pickup and Delivery Problem with Time Windows by introducing an artificial depot. After this transition the Adaptive Large Neighborhood Search (ALNS) framework by Ropke and Pisinger (2006) can be used to solve the problem. The variables that build neighborhoods are the customer requests. These requests are removed from tours and added to a *request bank* by a set of operators that is chosen randomly. Additionally, there are a number of parallel and sequential insertion operators to repair the solutions.

Polacek et al. (2008) present another VNS that solves the Multi-Depot Vehicle Routing Problem with Time Windows (MDVRPTW); Neighborhoods are created by exchanging parts of different tours and *3-opt* is applied to single tours afterwards for refinement. Furthermore, two ways to compute bounds are presented and the authors describe two variants to parallelize the solution process: A master process handles the best solution found so far while worker processes discover the solution space. The variants mainly differ in the frequency of communication between the master and worker processes.

Liu et al. (2010) describe a MDVRP with full truckload and carrier collaboration. However, the last mentioned aspect is limited to the consideration by a joint pool of demand and vehicles and does not influence the solution process. The algorithm itself consists of three parts: A greedy heuristic that constructs cycles, a second heuristic that builds tours based on the cycles, and a LS (k-opt) approach for improvement.

Aras et al. (2011) introduce the Selective Multi-Depot Vehicle Routing Problem with Pricing. The problem originates from the remanufacturing industry and there is no need to service all customer nodes, i.e. there is a reservation price for every node and the decision maker has to set an acquisition price for each customer. Only if the (global) acquisition price is higher than the reservation price, the remanufacturer will earn a given revenue. In the objective the difference of revenues minus acquisition price and travel costs shall be maximized. Thus, there is a decision about the routing and the acquisition price. The solution procedure is a TS algorithm again. Kuo and Wang (2012) use a VNS to solve this problem. This heuristic exchanges nodes and arcs in and between different tours.

Rahimi-Vahed et al. (2013) present a Path Relinking (PRL) algorithm for the MDPVRP. This solution approach does not rely on randomness but searches the solution space systematically. The problem is decomposed into a MDVRP and a PVRP which are solved separately. The solutions of these problems are added to a reference list. This list is used by the PRL algorithm that follows a search trajectory towards a better solution. The search process combines solutions from the list and performes a neighborhood search that shall result in a solution that includes the best characteristics of both input solutions. Subramanian et al. (2013) present a hybrid multi-start algorithm that is able to deal with a large variety of VRP problems. A Granular Tabu Search is proposed by Escobar et al. (2014): This search method excludes moves that do not lead to a high-quality solution and considers only potentially promising operators in any iteration (Toth and Vigo, 2003). The proposed TS includes different diversification operators that are problem-specific. Levin and Yovel (2014) present a graph-based LS algorithm. Finally, Salhi et al. (2014) propose a heuristic based on a VNS for the Multi-Depot Vehicle Routing Problem with Heterogeneous Fleet (MDVRPHF), a generalization of MDVRP that allows different characteristics on the vehicles in the fleet. The algorithm assigns customers to their closest depots (*borderline customers*—that cannot be assigned to a depot clearly—are inserted in their best position) and all routes are put together to build a large set of routes. Afterwards, VNS with different operators is applied to this set of routes and a diversification and an optimization operator are executed on single tours. Afterwards, the algorithm applies VNS again. This will be done iteratively until the termination criterion is met.

Salhi et al. (1998) provide an Adaptive Genetic Algorithm that consists of two phases. Clustering is done by a GA and afterwards for every tour a TSP is solved by an insertion heuristic. The clusters are built using circles that are placed on the "map" of customers. A customer is assigned to the depot whose circle covers it. Because node assignment depends on the position and the radius of theses circles the GA alters exactly these values. Building routes within a cluster is done by a cheapest insertion heuristic and the results are heuristically improved afterwards. Thangiah and Salhi (2001) extend this approach in terms of adding a LS to build tours out of the clusters found by the GA. For every new tour position refinements (exchanging nodes between tours) are made before the fitness value is computed. Jeon et al. (2007) solve the MDVRP with two depots where every customer has to be visited twice with an extended GA that emphasizes the meaning of the initial population and includes node exchange mechanisms. Chen and Xu (2008) present a hybrid algorithm that is a combination of a GA and SA. It accepts solutions which are obtained from a GA with a *substring exchange crossover* and *inversion mutation operator* and worsen the objective function value with a probability that de-

creases over time. Better solutions are always accepted. Ho et al. (2008) integrate inter- and intra-tour swap procedures (2-*opt*) into a GA where the initial solution is obtained at random or by a savings heuristic. Lau et al. (2010) introduce a GA that is improved by the application of fuzzy logic. It controls the crossover and mutation rate. A Hybrid GA that performs well on the MDVRP, MDPVRP, and the PVRP is presented by Vidal et al. (2012). The algorithm uses neighborhood-based route improvement operators in its education phase and introduces a new population management that improves population diversity. In Vidal et al. (2013b) the authors extend their algorithm to deal with time windows but did not focus on the MDVRP.

Further metaheuristics are introduced by some authors: An ACO algorithm to deal with the MDVRP with a heterogeneous fleet is suggested by Benslimane and Benadada (2013). Zhang et al. (2011) consider an extension of the MDVRP with weight-related costs in the objective function. That means that vehicles with a higher amount of load would cause higher costs. They use a Scatter Search (SS) algorithm to solve the problem. Salhi and Sari (1997) consider the MDVRPHF again, i.e. some vehicles had different capacities (and costs). Furthermore, the problems of clustering, routing, and assigning truck sizes are handled simultaneously. The authors uses a composite heuristic algorithm that carries out a sequence of optimization steps. Some of these heuristics improve single tours while others try to find better solutions considering all depots. Two reduction tests are presented to cut down computation time.

Additionally, there are some matheuristics: Parthanadee and Logendran (2006) present a MDVRP model for a real-life application. A food company distributes different products from different depots. The MILP is divided into two parts: The first part deals with decisions on product allocation while the second part contains the routing constraints. Three variants of TS are introduced to solve the problem: The first one is the basic heuristic that saves tabu candidates in a short-term memory. The second heuristic includes a diversification process that guides search to new regions of the solution space. Finally, the third variant extends the basic version by an intensification process if a good solution is found. Dondo and Cerdá (2007) introduce a three-stage algorithm. In the first step few clusters are built up on distance, load, and time window information. Next, these clusters are assigned to vehicles and tours are built out of these clusters. Due to the reduced problem size (clusters instead of nodes) this can be done by solving a MILP. Finally, the third step makes the routing decision within clusters and schedules the arrival times. This is done by solving a TSP for every vehicle. Gulczynski et al. (2011) presents a matheuristic for the Multi-Depot Split Delivery Vehicle Routing Problem. They use the heuristic by Golden et al. (1977) to assign customers to depots and solve the SDVRP for each depot separately. In a post-processing step the algorithm tries to improve the solution. However, this step does not create new splits but makes routing decisions.

Of course, there exists a number of *exact methods* to solve the MDVRP: Laporte et al. (1988) use a graph representation for an asymmetric MDVRP variant. The problem is reduced to a TSP which is solved by using an adaption of the B&B search specialized to solve TSPs introduced by Carpaneto and Toth (1980). Lei et al. (2011) present a Distributed Constraint Optimization Problem to solve a variant of the MDVRP. A further exact solution approach can be found in the work of Bettinelli et al. (2011). They use a B&P framework where *2-path* inequalities are added to strengthen the upper bound. Furthermore, two kinds of branching rules are introduced. To solve the subproblem in the underlying CG process they suggest three different routines: Exact

Dynamic Programming, Heuristic Dynamic Programming, and a Greedy Pricing Algorithm. Muter et al. (2014) present a one-phase and a two-phase algorithm. The first one uses a classical approach with a set-covering formulation and a subproblem that is a variant of the shortest path problem and is solved by a label-correcting algorithm. The second approach first enumerates all (non-dominated) routes that are required to solve the subproblem. In a second step a *depot graph* is used to construct a solution from the routes. Contardo and Martinelli (2014) suggest further exact methods to solve the MDVRP: They give a set partioning and a vehicle flow formulation of the problem. To strengthen these formulations they introduce a new class of valid inequalities. The proposed solution method is CG, where the subproblem is solved by exact and heuristic dynamic programming approaches. Chan et al. (2001) describe solution procedures to solve a stochastic depot location and vehicle routing problem. At first, they present a method based on an extension of Bender's Decomposition that is called *Stochastic Decomposition* and proceeds in two steps: A random-sampling step that allocates load to vehicles and a second step that constructs routes using classical mathematical programming methods. A special variant of a MDVRP with a single hub is described by Irnich (2000). The specific structure of the problem (narrow time windows, short routes) allows the complete enumeration of all tours. Out of these tours a set covering or set partitioning problem is solved to find the cheapest combination of tours. A model to replenish petroleum from multiple depots to gas stations is provided by Cornillier et al. (2012). To solve the problem initially feasible tours are generated and selection is done by a MILP.

Table 2.2: Taxonomy of the MDVRP

	A		D		E		F		G		H			I			J			K			L		
Paper	2	3	1	2	1	2	1	2	1	2	1	2	3	1	2	3	1	2	1	2	3	1	2	3
Tillman (1969)		X	X			X	X		X		X			X			X		X			X		
Tillman and Cain (1972)		X	X		X		X		X		X			X			X		X			X		
Cassidy and Bennett (1972)	X			X	X		X		X			X				X	X			X				X
Wren and Holliday (1972)		X	X		X		X		X		X			X			X		X			X		X
Gillett and Johnson (1976)		X		X	X		X		X		X			X			X			X		X		
Ball et al. (1983)		X	X		X		X			X			X	X				X			X			X
Perl and Daskin (1985)		X	X		X		X		X		X			X			X		X				X	
Benton (1986)		X	X		X		X		X				X	X			X			X			X	
Laporte et al. (1988)		X	X		X		X		X		X					X	X	X		X				X
Chao et al. (1993)		X	X		X		X		X		X					X	X			X		X		
Sumichras and Markham (1995)	X		X		X		X		X				X	X			X	X			X	X		
Renaud et al. (1996)		X	X		X		X		X		X			X			X			X		X		
Cordeau et al. (1997)		X	X		X		X		X		X			X			X			X		X		
Salhi and Sari (1997)		X		X	X		X		X			X		X			X	X		X				X
Hadjiconstantinou and Baldacci (1998)	X		X		X		X		X				X	X			X		X			X		
Salhi et al. (1998)		X	X		X		X		X		X					X	X			X			X	
Salhi and Nagy (1999)		X	X		X		X		X		X					X	X				X	X		
Irnich (2000)	X			X	X		X		X			X			X		X				X	X		
Yang and Chu (2000)		X	X		X		X		X					X			X			X				X
Chan et al. (2001)		X	X			X	X		X		X					X	X	X		X			X	
Cordeau et al. (2001)	X		X		X		X		X			X			X		X		X			X		
Thangiah and Salhi (2001)		X	X		X		X		X		X					X	X			X		X		X
Giosa et al. (2002)	X		X		X		X		X		X					X	X			X		X		
Tarantilis and Kiranoudis (2002)		X	X		X		X		X		X					X	X			X		X		
Polacek et al. (2004)	X		X		X		X		X					X			X			X		X		
Lim and Wang (2005)		X	X		X		X		X		X					X	X			X		X		
Nagy and Salhi (2005)		X	X		X		X		X		X			X			X				X	X		
Parthanadee and Logendran (2006)		X	X			X	X		X			X				X	X			X		X		
Tansini and Viera (2006)	X		X		X		X		X		X					X	X			X		X		

Table 2.2: Taxonomy of the MDVRP (ctd.)

	A		D		E		F		G		H			I			J			K			L		
Paper	2	3	1	2	1	2	1	2	1	2	1	2	3	1	2	3	1	2	1	2	3	1	2	3
Crevier et al. (2007)		X	X		X		X		X		X			X			X			X		X		
Dondo and Cerdá (2007)	X			X	X		X		X			X		X			X		X			X		
Jeon et al. (2007)		X	X		X		X		X		X					X	X			X		X		
Pisinger and Ropke (2007)	X		X		X		X			X	X					X	X				X	X		
Chen and Xu (2008)		X	X		X		X		X		X					X	X			X		X		
Ho et al. (2008)		X	X		X		X		X		X					X	X			X			X	
Polacek et al. (2008)	X			X	X		X		X			X			X		X			X		X		
Lau et al. (2010)		X	X		X		X		X		X			X					X		X	X		
Liu et al. (2010)		X	X		X		X		X		X			X			X				X	X		
Aras et al. (2011)		X	X		X		X		X							X	X	X	X				X	
Bettinelli et al. (2011)	X			X	X		X			X	X			X			X	X		X			X	
Gulczynski et al. (2011)		X	X		X		X		X							X	X		X			X		
Lei et al. (2011)	X		X		X		X						X			X	X			X				X
Zhang et al. (2011)		X	X		X		X		X			X					X	X		X			X	
Kuo and Wang (2012)		X	X		X		X		X							X	X			X		X		
Cornillier et al. (2012)	X			X	X		X			X	X			X			X			X		X		
Vidal et al. (2012)		X	X		X		X		X			X					X			X		X		
Benslimane and Benadada (2013)		X		X	X		X		X		X			X			X	X			X		X	
Rahimi-Vahed et al. (2013)		X	X		X		X		X			X					X			X		X		
Subramanian et al. (2013)		X	X		X		X		X			X					X	X		X		X		
Contardo and Martinelli (2014)		X	X		X		X		X			X					X			X		X		
Escobar et al. (2014)		X	X		X		X		X			X					X			X		X		
Levin and Yovel (2014)		X	X		X		X						X			X	X			X		X		
Muter et al. (2014)		X	X		X		X		X			X					X			X		X		
Salhi et al. (2014)		X		X	X		X				X					X	X	X		X			X	
Xiong and Wang (2014)	X			X	X		X		X		X					X	X			X	X			

In summary, we can see from Table 2.2 that research on the MDVRP deals mainly with static planning scenarios, uncertainty is mainly incorporated by stochastic demand (column *E*). Furthermore, in most of the papers limits in total load and/or travel time are present (columns *H* and *I*), while time windows are seldom taken into account. All but one studies consider routing costs, only some include fixed costs (per vehicle) in the decision model. The objective that is used most often is to minimize the routing costs (column *L*). It can be observed that there are no arc routing problems in this domain (column *F*) and that most papers model problems with undirected graphs (column *G*). Finally, the fleet is assumed to be homogeneous in the majority of the publications (column *D*).

A summary of the solution methods for the MDVRP and its variants is presented in Table 2.3 and gives some categories to specify the solution process. The categories are taken from the second classification scheme by Bodin and Golden (1981), the most-right column gives the shorthand of the metaheuristic that is used, if applicable. The meaning of the categories was explained within the description, additionally an overview on heuristics for the VRP can be found in Cordeau et al. (2002).

Table 2.3 shows a chronological development: In the early days, most of the papers use saving-based approaches. Later on, the focus is on exchange mechanisms that are widely used in metaheuristics, too. Within the field of metaheuristics the focus is on LS techniques and its variants. The comparatively small number of exact approaches towards the variety of heuristics and decomposition schemes emphasizes the difficulties to solve the problem optimally.

Table 2.3: Solution Methods for the MDVRP

Paper	Cluster First, Route Second	Route First, Cluster Second	Savings/Insertions	Improvement/Exchange	Mathematical Programming-Based	Interactive Procedures	Exact Procedures	Remarks
Tillman (1969)	X		X					
Tillman and Cain (1972)	X						X	
Cassidy and Bennett (1972)	X			X		X		
Wren and Holliday (1972)	X		X					
Gillett and Johnson (1976)	X		X					
Ball et al. (1983)		X	X					
Perl and Daskin (1985)	X		X					
Benton (1986)	X		X				X	
Laporte et al. (1988)							X	
Chao et al. (1993)	X		X	X				

Table 2.3: Solution Methods for the MDVRP (ctd.)

Paper	Cluster First, Route Second	Route First, Cluster Second	Savings/Insertions	Improvement/Exchange	Mathematical Programming-Based	Interactive Procedures	Exact Procedures	Remarks
Sumichras and Markham (1995)			X	X				
Renaud et al. (1996)				X				TS
Cordeau et al. (1997)								TS
Salhi and Sari (1997)				X				
Hadjiconstantinou and Baldacci (1998)				X				TS
Salhi et al. (1998)			X	X				GA
Salhi and Nagy (1999)		X	X					
Irnich (2000)							X	
Yang and Chu (2000)			X					
Chan et al. (2001)			X					
Cordeau et al. (2001)				X				TS
Thangiah and Salhi (2001)				X				GA, LS
Giosa et al. (2002)	X		X					
Tarantilis and Kiranoudis (2002)				X				
Polacek et al. (2004)				X				VNS
Lim and Wang (2005)			X		X			
Nagy and Salhi (2005)		X		X				
Parthanadee and Logendran (2006)	X			X				
Tansini and Viera (2006)	X							
Crevier et al. (2007)				X				TS
Dondo and Cerdá (2007)	X				X			
Jeon et al. (2007)				X				GA
Pisinger and Ropke (2007)								ALNS
Chen and Xu (2008)				X				GA, SA
Ho et al. (2008)				X				GA
Polacek et al. (2008)				X				VNS
Liu et al. (2010)	X			X				LS
Aras et al. (2011)				X				TS
Bettinelli et al. (2011)							X	
Gulczynski et al. (2011)		X			X			
Lei et al. (2011)							X	
Zhang et al. (2011)				X				SS
Cornillier et al. (2012)					X			SS
Kuo and Wang (2012)				X				VNS

Table 2.3: Solution Methods for the MDVRP (ctd.)

Paper	Cluster First, Route Second	Route First, Cluster Second	Savings/Insertions	Improvement/Exchange	Mathematical Programming-Based	Interactive Procedures	Exact Procedures	Remarks
Vidal et al. (2012)				X				GA
Benslimane and Benadada (2013)				X				ACO
Rahimi-Vahed et al. (2013)				X				PRL
Subramanian et al. (2013)			X	X				
Contardo and Martinelli (2014)							X	
Escobar et al. (2014)				X				TS
Levin and Yovel (2014)				X				LS
Muter et al. (2014)							X	
Salhi et al. (2014)				X				VNS
Xiong and Wang (2014)				X				GA

Chapter 3

Foundations of Constraint Programming

In this chapter we will introduce the basic concept of Constraint Programming to allow the reader a deeper understanding of the hybrid solution procedures presented below. A TSP example presented in Section 3.2 will be used to illustrate most of the aspects concretely.

The first part of this chapter (Section 3.1) sketches the historical development of CP from its beginnings in the 1960s to nowadays. Afterwards a short overview is given to introduce the main concepts of the language and solution concepts in Constraint Programming. On the basis of this knowledge the following sections deepen the knowledge on (1) modeling (Sections 3.3 and 3.5), (2) constraint propagation (Section 3.4) and (3) search (Section 3.6). Finally, an overview on the previous work in CP on the TSP and VRP is presented in Section 3.7.

Because this chapter summarizes only basic concepts that are necessary to understand the general ideas of CP we refer the reader to the more detailed textbooks by Frühwirth and Abdennadher (1997), Marriott and Stuckey (1999), Apt (2009), Dechter (2009), and Ghédira (2013) for comprehensive details on the topics. Brief introductions can be found in Barták (1999), Kumar (1992), and Pape (2010). An overview on CP tools can be found in Michel et al. (2007) and Schulte and Carlsson (2006).

3.1 Historic Background

In the following we will give a brief introduction to the origins of CP to clarify its connection to AI. A more detailed review can be found in Barták (2011) that was the basic paper for this section. More details on the development of logic-based methods can be found in Hooker (1994, pp. 6–7), Freuder and Mackworth (2006, pp. 13–16) and Apt (2009, p. 5).

The basic work in the field of AI was Ivan Sutherlands "Sketchpad: A man-machine graphical communication system" (Sutherland, 1963). Therefore, Barták (2011) states that the birth year of CP was 1963.

According to Freuder and Mackworth (2006) two streams in the field of AI can be identified that have influenced the development of CP: a *language stream* and an *algorithmic stream*.

While the latter uses the paradigm of AI as search, the first one is heavily influenced by logic programming. It deals with the development of novel programming languages and constraint systems. Both streams acted independently in the first decades.

The most important languages from the beginnings of CP were *CONSTRAINTS* (Sussman and Steele, 1980), *REF-ARF* (Fikes, 1970), *Alice* (Lauriere, 1978), *Absys* (Foster and Elcock, 1969), *MOLGEN* (Stefik, 1981) and *PLANNER* (Hewitt, 1968). A first milestone in the development of CP was *PROLOG* (*Programmation en Logic*) in the early 1970s. While *PROLOG* only dealt with equality constraints, *PROLOG II* handled inequality constraints as well. The first real Constraint Logic Programming (CLP) language was the advancement to *PROLOG III*. An introduction to *PROLOG* can be found in Bratko (2001).

In the algorithm stream the first important contribution was Waltz (1975). In this paper a pruning algorithm was presented (cf. Algorithm 2 in Section 3.4) as a starting point for constraint consistency techniques. In the following, Montanari (1974) introduced the terms of Arc Consistency and Path Consistency. Furthermore, Mackworth (1977) made fundamental contributions to pruning algorithms as we will explain in Section 3.4.

Both streams were merged in the 1970s by three teams of researchers (Jaffar and Lassez, 1987; Jaffar et al., 1992; Hentenryck, 1989) who developed independent Constraint Logic Programming systems. The last mentioned work by Hentenryck (1989) dealt with *CHIP* (*Constraint Handling in PROLOG*) and was the real fusion of the language and algorithm stream. It was able to solve finite domain problems and combined the backtracking search and consistency techniques. For a very extensive survey on CLP we refer to Jaffar and Maher (1994). Furthermore, the convergence of both streams moved the focus from specific application areas to more general fields of application. Over the years a lot of new application areas were identified, as electric circuit analysis, chemical reasoning, diagnosis problems, scheduling problems etc. As we will show later, not all current solvers are based on logic programming techniques (Milano and Trick, 2004). Therefore, we will use the term Constraint Programming (CP) instead of CLP in the remaining chapters to avoid confusion.

We shift the description of further developments to the concrete sections on constraint propagation (Section 3.4) and search (Section 3.6) as the concepts are discussed more detailed there. The referred sections chronologically show further milestones in the evolution of what is understood as CP in this thesis.

The aim of CP community is to go on towards a true declarative programming language. In general, declarative programming language emphasize the description of a problem (*what*) that should be computed, the way the program works (*how*) is not of interest. Freuder (1997, p. 57) tries to connect this paradigm with the idea behind CP: ”Constraint programming represents one of the closest approaches computer science has yet made to the Holy Grail of programming: the user states the problem, the computer solves it.” This quotation emphasizes the general goal and the origins with its two streams but focuses on the language stream and its modeling operators for users. Hentenryck and Michel (2005, p. xvi) reduce this idea to the simple formula *”Constraint Programming = Constraints + Search”*.

3.2 Running Example: Traveling Salesman Problem

To illustrate the theoretic aspects of CP we use the TSP as a running example. A Traveling Salesman Problem deals with the situation where a salesman should visit each city exactly once. The objective is to minimize the total travel distance between cities in this route. The salesman must start and end his tour in his "home town", i.e. he must return to the city where he started his journey. We will refer to this city as *depot* in the remainder. The problem can be modeled as an directed graph $G = (N, A)$, where $N = \{0, \ldots, n-1\}$ is the set of n nodes (that correspond to cities) and the arc set A includes all arcs that connect the nodes with each other. Every arc is weighted with the travel distance $cost_{ij}$ between nodes i and j. We refer the reader to Applegate (2006) for more details on the TSP. The binary decision variable x_{ij} is $x_{ij} = 1$ if node j is the direct successor of node i, $x_{ij} = 0$ otherwise.

A mathematical model for the problem can be stated as follows:

$$\min \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} cost_{ij} x_{ij} \tag{3.1}$$

s. t.

$$\sum_{i=0}^{n-1} x_{ij} = 1 \qquad j \in 0, \ldots, (n-1) \tag{3.2}$$

$$\sum_{j=0}^{n-1} x_{ij} = 1 \qquad i \in 0, \ldots, (n-1) \tag{3.3}$$

$$\sum_{i \in Q} \sum_{j \in Q \setminus V} x_{ij} \geq 1 \qquad 2 \leq |Q| < \left\lfloor \frac{n}{2} \right\rfloor \tag{3.4}$$

$$x_{ij} \in \{0, 1\} \qquad i, j \in 0, \ldots, (n-1) \tag{3.5}$$

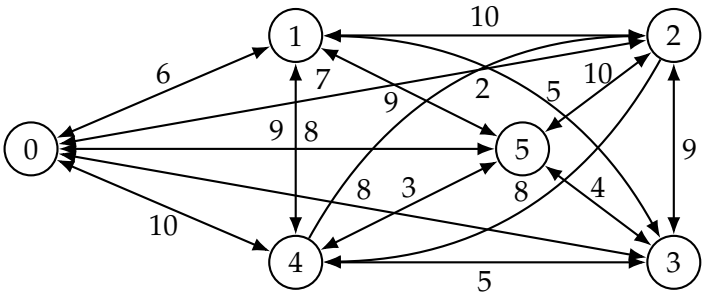
The objective function (3.1) minimizes the total costs of the tour. Constraints (3.2) and (3.3) state that every node must have exactly one successor and predecessor, i.e. every node is visited exactly once. However, a solution that fulfills these conditions does not necessarily build a complete cycle, it can include subtours. An additional constraint is required to forbid these subtours. This is done by Constraint (3.4): It divides the set N into two subsets Q and V. Set Q includes $2 \leq |Q| < \lfloor \frac{n}{2} \rfloor$ nodes, the second set V the remaining nodes. The constraint states that there must be a connection between the two sets and if this is valid for all decompositions of set N, the solution cannot contain any subtour. The explicit formulation of the TSP has $2n + \sum_{l=2}^{\lfloor n/2 \rfloor} \binom{n}{l}$ constraints.

We will not only make use of the model but also refer to an instance to illustrate the main ideas in CP. Therefore, the travel costs $cost_{ij}$ between two nodes i and j are given in Table 3.1 and Figure 3.1 presents a graphical illustration of the data set. It is obvious that this is an asymmetric problem, i.e. $cost_{24} \neq cost_{42}$.

Table 3.1: Cost Matrix for the Running Example

$cost_{ij}$	0	1	2	3	4	5
0	—	6	7	8	10	9
1	6	—	10	5	8	9
2	7	10	—	9	8	10
3	8	5	9	—	5	4
4	10	8	2	5	—	3
5	9	9	10	4	3	—

Figure 3.1: Graphical Illustration of the TSP Instance



3.3 Fundamental Principles

This section introduces the key concepts of CP. It starts with some basic elements that are necessary for modeling and presents some variants of CP problems. The section is a composition of basic knowledge in CP from different textbooks (Marriott and Stuckey, 1999; Apt, 2009; Dechter, 2009; Ghédira, 2013).

A ***variable*** is an object that has a name and can take different values. In CP a variable (named) x must take a value from its domain $dom(x)$, a finite set of m integers $dom(x) = \{a_0, \ldots a_{m-1}\}$. The initial domain is denoted by $dom^{init}(x)$. A value a is valid for a variable iff it is contained in $dom(x)$. Because domains are subject to change in the solution process, valid values can become invalid over time. For the sake of simplicity, we assume in the remainder that the values within a domain are in a strict order, i.e. values are sorted increasingly: $a_0 < a_1 < \ldots < a_{m-2} < a_{m-1}$.

A variable is denoted as *fixed* if its domain contains only one value ($|dom(x)| = 1$). If there are two or more elements in the domain, the variable is *unfixed* ($|dom(x)| > 1$). A variable can be fixed implicitly or explicitly. If a variable x is explicitly set to a single value $a \in dom(x)$ the variable is said to be *instantiated* or *assigned*. Otherwise, the variable is *uninstantiated* or *unassigned*. Implicitly fixed variables result from deduction in the search process: Relations between variables can declare some values to be invalid for a variable. We will illustrate this mechanism more detailed later.

Most CP models contain more than one variable. We use $\mathcal{X}$ to denote the set of all n variables, that is $\mathcal{X} = \{x_0, \ldots, x_{n-1}\}$. Furthermore, $\mathcal{D}$ is the set of domains of all variables in $\mathcal{X}$: $\mathcal{D} = \{dom(x_0), \ldots, dom(x_{n-1})\}$.

In the model of our running example we build connections between nodes. Every node is associated with a variable that takes the number of its successor node and is named x_i for node

$i \in N$. Thus, we have $n = 6$ variables, that is $\mathcal{X} = \{x_0, \ldots, x_5\}$. The initial domains contain all nodes ($m = n = 6$) in the problem: $dom^{init}(x_i) = \{0, \ldots, 5\}, i \in N$. If—for some reason—the successor of node 0 shall be 2, we would assign $x_0 = 2$ and variable x_0 would be (explicitly) fixed.

In this thesis we limit the scope to finite domain problems. Therefore all variables have an initial domain with a finite number of elements and the values are discrete. Note that our definition includes Boolean variables that only can take the values *true* or *false*. We refer to Benhamou and Granvilliers (2006) for details on continuous and interval problems.

A ***constraint*** c_i on $\mathcal{X}$ is a relation R_i defined on the subset of variables that is included in c_i. The relation R_i includes all values that are allowed for a simultaneous assignment. It is defined by a subset of the domains of the k variables included in c_i, that is $R_i \subseteq dom(x_{i_0}) \times \ldots \times dom(x_{i_{(k-1)}})$. In the remainder, we will use the shorthand R_{x_0} to refer to all permitted values for a constraint that includes variable x_0, R_{x_0,x_1} for the permitted tuples for constraints with variables x_0 and x_1 and so forth. By the *scope(c_i)* of a constraint c_i we denote the set of k variables the constraint c_i is defined on, that is $scope(c_i) = \{x_{i_0}, \ldots x_{i_{(k-1)}}\}$. Thus, a constraint is a restriction on the combination of values that the variables $\{x_h | h \in scope(c_i)\}$ can take simultaneously. If $k = 1$, we call the constraint *unary* and if $k = 2$, it is a *binary* constraint. Constraints with more than two variables are called *non-binary*. The *degree* $deg(c_i)$ of a constraint is equal to the number of variables it incurs, i.e. $deg(c_i) = |scope(c_i)|$. The set $\mathcal{C}$ includes all constraints.

In our running example it is required that each node has a successor. To build a cycle that contains all nodes it is necessary that no node succeeds itself. For the depot node 0 we can enforce this by the unary constraint

$$x_0 > 0. \tag{3.6}$$

Constraint (3.6) enforces the successor of node 0 to be greater than 0 and fulfills the requirement. However, this constraint is not applicable to the other variables $x_1, \ldots, x_n - 1$. Therefore, we post the more general Constraint (3.7):

$$x_i \neq i \qquad\qquad i \in N \tag{3.7}$$

Furthermore, we know that every node can only be serviced exactly once. Therefore, the variables' values must be pairwise distinct. That can be stated by the following binary Constraint (3.8):

$$x_i \neq x_j \qquad\qquad i, j \in N : i \neq j \tag{3.8}$$

These constraints forbid that two customer nodes have the same successor which would be an infeasible solution.

The constraints described so far were *non-decomposable* (or *atomic*) constraints, they cannot be expressed by a set of other constraints. Besides these atomic constraints there exists another class of constraints: ***Global constraints*** are patterns that capture precise relational semantics and can be applied over an arbitrary number of variables. A global constraint can be composed of atomic constraints and/or other global constraints. In summary, it can be said that the presence of global constraints is one of the main aspects in CP in the purpose of supporting users with modeling and to become a declarative programming language.

An example for a global constraint is *ALLDIFFERENT($\mathcal{X}$)* that enforces all variables in its scope to take different values. Thus, *ALLDIFFERENT($\mathcal{X}$)* consists of Constraint (3.8) which can be replaced by the global constraint. However, global constraints do not only simplify modeling, they also speed up the solution process. Choosing a global constraint signals to the solver that a specific structure is present in the problem. The solution process will use a specialized algorithm for deducing; in the case of *ALLDIFFERENT($\mathcal{X}$)* it is an algorithm that is used for bipartite matching. A more detailed description on this and other global constraints will be given in Section 3.5.

To get one or more feasible solutions for our running example we have to add a second global constraint to the problem that forbids subtours. This constraint is called *TOUR($\mathcal{X}$)* and enforces all nodes of a graph to build a cycle, i.e. to build a Hamiltonian cycle (Jungnickel, 2013, Sec. 1.4). Because our variables correspond to nodes, we can state *TOUR($\mathcal{X}$)*. Then, the cycle starts and ends in the depot and includes all nodes. Thus, one global constraint is sufficient to model the running TSP example which is easily understandable, too. That impressively demonstrates the advantages of CP in modeling.

A ***Constraint Satisfaction Problem*** P is a triple $P = \langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$ where $\mathcal{C}$ is a tuple of constraints each on a subset of $\mathcal{X}$ with domains $\mathcal{D}$. A *solution* to the CSP P $sol(P)$ is an n-tuple $A = \langle a_0, a_2, \ldots, a_{n-1} \rangle$ where $a_i \in \mathcal{D}_i$ and each $c_j, j \in \mathcal{C}$ is satisfied. The consistence of a CSP depends on the existence of a solution: It is called *consistent*, if it has at least one solution whereas a CSP with no solution is *inconsistent*.

Depending on the given task it may be required to

- find the set of all solutions $sol(P)$ of the CSP.
- discover if any solution exists. If $sol(P = \emptyset)$, the CSP is *unsatisfiable* or *inconsistent*. Otherwise the CSP is *satisfiable* or *consistent*.
- find a single solution.

Our example CSP consists of the successor variables $\mathcal{X}$ and the associated domains $\mathcal{D}$ and the Constraints (3.6), (3.7) (or *ALLDIFFERENT($\mathcal{X}$)*) and *TOUR($\mathcal{X}$)* at the moment. We can use the model to find a set of all feasible tours. The example's solution is a set of 120 feasible tours. From an economic point of view it may be preferable to evaluate these solutions with respect to their costs.

This can be done by introducing an objective function that maps a solution to a real number: $obj : sol(P) \rightarrow \mathbb{R}$. This objective function should be optimized, i.e. maximized or minimized. Then, a ***Constraint Optimization Problem*** is a combination of a CSP P and an objective function obj. In the remainder we will assume that we handle minimization problems unless specified otherwise.

The objective function of our problem is the sum of the travel costs between all nodes and their successors that has to be minimized:

$$\min \sum_{i=0}^{|N|-1} cost_{i,x_i} \tag{3.9}$$

The objective function (3.9) points out a characteristic that may be surprising for members of the OR community because in CP variables can take variables as their indices (cf. Section 3.5). The best solution has travel costs of 27 while the feasible route with the highest cost has an objective function value of 54.

Beside the CSP and the COP there are further types of CP problems: *Partial CSPs* (Freuder and Wallace, 1992) do not require a complete satisfaction of all constraints but relax some variables or constraints of the original CSP. These techniques are a way to solve overconstrained CSPs, that are problems where no solution can be found where all constraints are satisfied. Therefore, *soft constraints* are introduced, whose satisfaction is desirable but not necessarily required. A variant of the Partial CSP is the *Maximum CSP* (Freuder and Wallace, 1992) that seeks for a complete instantiation where the number of satisfied constraint is maximized (or the number of unsatisfied constraints is minimized). The *Minimum CSP* (Li et al., 2012) works analogously and tries to minimize the number of satisfied constraints. Some works try to assign a penalty value to every constraint that evaluates the importance of a constraint. The solution of such a weighted CSP minimizes the sum of the penalties from the violated constraints (Ansótegui et al., 2013). Schiex et al. (1995) introduce a general framework called *Valued CSP* to handle this class of problems.

Finally, the concept of *Dynamic CSP* allows adding and relaxing constraint to react to real-world problems. A Dynamic CSP is a sequence $P_0, \ldots, P_i, P_{i+1}, \ldots, P_k$, where each CSP P_{i+1} results from an elementary modification (addition or removal of constraint(s)) to CSP P_i (Dechter and Dechter, 1988). Note that these modifications question all operations on the domains executed so far and the solution itself.

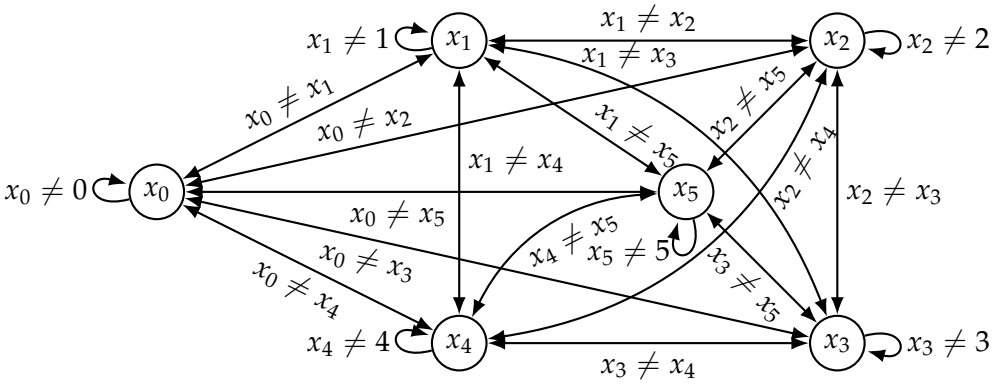
3.4 Constraint Propagation

Constraint propagation (*propagation* for short) is a fundamental concept of CP. It embeds reasoning in the solution process. As a result some values or combinations of values for one or more variables are explicitly forbidden (*pruned*) because at least one of the constraints cannot be satisfied otherwise. A value that is forbidden because of reasoning is called *nogood*. Forbidding a value results in the reduction of the number of values in the variable's domain. Furthermore, reasoning can detect failures (or inconsistencies) during search earlier.

Before we start with further explanations of propagation algorithms, we will first introduce the concept of the ***constraint graph*** $G = (V, E)$ that will be used in the following definitions on unary and binary constraints. These graphs can be used to capture the relations within a CSP. Nodes V represent the variables X and the edges E connect all nodes that are included in the constraints of the problem. A missing edge between two nodes expresses that there is no constraint that has both variables in its scope. Constraints with more than two variables can be illustrated in constraint hypergraphs where hyperedges build "regions" that include all nodes of the constraints. Lecoutre (2009) provides detailed information on this topic.

Figure 3.2 illustrates the constraint graph for the Constraints (3.7) and (3.8) of our example. Because of Constraint (3.8), all variables are connected by an edge. Additionally, every node has a loop because there is the unary Constraint (3.7) (that includes Constraint (3.6)).

Figure 3.2: Constraint Graph for the TSP Example



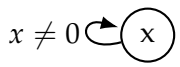
A number of synonyms for *constraint propagation* exists, e.g. *constraint relaxation*, *filtering algorithms*, *constraint inference*, and *local consistency enforcing* (Bessière, 2006). For simplicity, we will only use the term constraint propagation in the remainder.

There are different levels of consistency that depend on the number of variables in the scope of the constraints. In the next sections we will give definitions about different consistency levels and explain the propagation algorithms.

3.4.1 Node Consistency

Node Consistency (Mackworth, 1977) deals with unary constraints, which are constraints with a single variable. A node (variable) is ***node consistent*** iff every value in the domain $dom(x_i)$ of x_i coincides with every unary constraint of $\mathcal{C}$ related to x_i. A CSP is node consistent iff all of its nodes are node consistent. The absence of unary constraints means that the problem is always node consistent. An example of a constraint graph is given in Figure 3.3.

Figure 3.3: Constraint Network with Unary Constraint



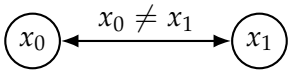
By definition, propagation for unary constraints does not require any interaction with other variables. Therefore node consistency can be achieved by simply passing all feasible values in the domain of a variable and excluding infeasible instantiation from the domains that are caused by unary constraints (Mackworth, 1977). It is sufficient to apply this algorithm once since there are no relations to other variables that could influence the result (Apt, 2009, p. 138). Therefore the algorithm obtains node consistency in st modz $\mathcal{O}(nd)$, where n is the number of nodes and d is the maximum domain size (Ghédira, 2013, p. 31).

To illustrate node consistency, consider Constraint (3.6) again ($x_0 > 0$). We get the node consistent problem by removing the value 0 from the domain of x_0. Since there are no further unary constraints no additional pruning steps are required.

3.4.2 Arc Consistency

Arc Consistency deals with binary constraints that connect two variables of a CSP. A binary constraint c that consists of the variable pair $(x_i, x_j) \in \mathcal{X}$, is ***arc consistent*** iff for any value $a_i \in dom(x_i)$ that satisfies all unary constraints there is a value $a_j \in dom(x_j)$ that is compatible with a_i. A CSP is *arc consistent* iff all of its binary constraints are arc consistent. The absence of binary constraints means that the problem is arc consistent by definition. Throughout this chapter we assume all problems to be node consistent and skip this step in all algorithms. The maximum number of binary constraints in a problem is $n(n-1)/2$: Each of the n variables can be related to at most $n-1$ binary constraints and symmetric constraints must be removed. Figure 3.4 is an example for a constraint graph for a binary constraint.

Figure 3.4: Constraint Network with Binary Constraint



Over the years propagation algorithms evolved and lead to more efficient procedures. This is true for all kinds of consistency levels but we will exemplary describe this development in detail for arc consistency propagation only to demonstrate efforts in efficiency. We chose this way because arc consistency can be understood easily compared to higher levels of consistency and the evolution is clear. The following algorithms are denoted by *AC-i* emphasizing the fact that they enforce arc consistency and a consecutive number, which is not strictly based on chronology.

The basis for the first algorithms *AC1–AC3* is a simple function REVISE(i,j). It is based on the observation by Fikes (1970) that values of $dom(x_i)$ which are not compatible with any value in $dom(x_j)$ (line 3) can be pruned from $dom(x_i)$ (line 4), where R_{ij} is the set of permitted tuples for x_i and x_j. If any modifications were applied to a domain, the function returns *TRUE* (line 5) to signal a change in a domain to the propagation algorithm.

Function REVISE(i,j)

```
1 change := FALSE
2 for a_i ∈ dom(x_i) do
3 |   if ∄a_j ∈ dom(x_j) such that ⟨a_i, a_j⟩ ∈ R_{x_i,x_j} then
4 |   |   Delete a_i from dom(x_i)
5 |   |   change := TRUE
6 Return change
```

Algorithm *AC*-1 was published by Mackworth (1977) and is ascribed to Rosenfeld et al. (1976). It repeatedly iterates over all binary constraints' variables until the last iteration has not changed any domain. The pseudocode is presented in Algorithm 1 on the next page. The list Q (line 1) contains all pairs of variables (x_i, x_j) that are linked by the problem's constraint set: C_{ij} denotes the binary constraints with x_i and x_j in their scope and $\mathcal{C}$ is the set of all constraints. The algorithm calls REVISE(i,j) for all pairs in Q and removes values from the domain $dom(x_i)$

that violate a constraint that includes x_i and x_j. *AC*-1 is a *brute force algorithm* that tries all combinations in Q. In line 5 the old value of *change* and the boolean result of REVISE(i,j) must be *false* to set *change* to *true* as the term is a disjunction. If there were no changes—that is *change* is *false* at the end of the loop—the procedure terminates and returns the arc consistent problem. Otherwise, all pairs in Q are revised again. The algorithm has a time complexity of $\mathcal{O}(n^3d^3)$ in the worst case, where n is the number of variables and d the maximum size of the domains (Ghédira, 2013, p. 36).

Algorithm 1: AC-1

```
1 Q := {(i,j)|C_ij ∈ C, i ≠ j}
2 repeat
3 |   change :=FALSE
4 |   for (i,j) ∈ Q do
5 |   |   change := (REVISE (i,j) ∨ change)
6 until change == FALSE
```

In our TSP example we have to check all variable pairs to receive an arc consistent problem. However, in the first propagation step no pruning will result from Constraints (3.7) using *AC*-1—as in all algorithms presented later on—since no variable is fixed initially. However, in the search process (more) pruning will take place as we will illustrate in Section 3.6.

Mackworth (1977) introduced *AC*-2 to improve the propagation process. Instead of iterating over all variables after every propagation operation, only variables connected to the modified variable by another constraint are examined. To draw the picture in a constraint graph: If a domain $dom(x_i)$ is pruned because of an arc (constraint) induced with x_j, only the arcs (k,i) must be checked for arc consistency again, where $k \leq i$ and $k \neq j$. For, only the nodes that have already been revised must be considered again because all other arcs will be checked in the future anyway. Furthermore, checking $k = j$ is not necessary because it has been checked by the function REVISE(i,j) already.

Algorithm 2: AC-2

```
1  for i = 0 to n − 1 do
2  |   Q := {(i,j)|C_ij ∈ C, j < i}
3  |   Q' := {(j,i)|C_ji ∈ C, j < i}
4  |   while Q ≠ ∅ do
5  |   |   while Q ≠ ∅ do
6  |   |   |   Pop (k,m) from Q
7  |   |   |   if REVISE (k,m) == TRUE then
8  |   |   |   |   Q' := Q' ∪ {(p,k)|C_pk, p ≤ i, p ≠ m}
9  |   |   |   Q := Q'
10 |   |   |   Q' := ∅
```

The algorithm's pseudocode is given in Algorithm 2. The variables x_i are pruned successively in a loop (line 1): In every iteration the algorithm operates on two sets Q and Q' (lines 2 and 3). The former one contains arcs that direct away from a node i, while the latter

contains nodes directing to a node i, where in both sets only the node that have been checked before are considered (as the others will be checked in later iterations). The algorithm first iterates over all elements of the set Q (inner while loop). It selects node combinations (k, m) from this set and calls REVISE(k,m) (line 7). If this function modifies $dom(x_k)$, all arcs that direct towards node x_k (are present in a binary constraint C_{pk}) are added to Q' (line 8). Again, nodes that will be checked in later iterations can be ignored at the moment. These arcs must be revised again because pruning a value from $dom(x_k)$ may make values in other variables' domains infeasible. The set Q' includes all arcs that direct towards x_i and arcs that have been checked in former iterations but must be revised again because a domain of another variable was changed. Arcs that have been revised are deleted from the set Q; the size of the set decreases by one in every iteration of the inner while loop. After all arcs of the set Q have been checked set Q' is copied to Q and Q' is cleared (Q' is now a cache for the inner loop). The algorithm now considers all nodes leading to a node and nodes that must be revised again (outer while loop). In further iterations only nodes that must be inspected again are included in Q, respectively Q'. If there are no further nodes that must be inspected again (Q' is empty), the next variable will be pruned.

The time complexity of AC-2 is around $O(n^2d^3)$. However, note that some tuples of nodes that are already included in the data structure could be added again and will be examined repeatedly.

To avoid these unnecessary calls of REVISE(i,j), the algorithm AC-3 relies on the same idea but has only a single data structure *queue* Q instead of two sets. Because the queue contains each element only once, adding an element that is already included in the queue will be rejected. Initially, this queue includes pairs of all variables that are connected by a binary constraint (line 1). The algorithm (pseudocode is presented Algorithm 3) iterates over the tuples (x_i, x_j) and checks consistency using the function REVISE(i,j) (line 4). If a value is deleted from $dom(x_i)$, the arcs directing to that variable are added to the queue *if they are not yet included in it* (line 5). Otherwise, REVISE(i,j) is called anyway for this combination and a second call would not result in better pruning. The time complexity of algorithm AC-3 is $\mathcal{O}(md^3)$, where m is the number of binary constraints. Zhang and Yap (2001) provided an improved version of AC-3 that has a better time complexity of $\mathcal{O}(md^2)$ in worst cases.

Algorithm 3: AC-3

```
1 Q := {(i, j) | C_ij ∈ C, j ≠ i}
2 while Q ≠ ∅ do
3 |   Pop arc (i, j) from Q
4 |   if REVISE (i, j) == TRUE then
5 |   |   Q := Q ∪ {(k, i) | C_ki ∈ C, k ≠ i, k ≠ j}
```

The pruning algorithm AC-4 (Mohr and Henderson, 1986) is not based on a revise function. Instead, it computes a value that gives the *support from variable* x_j for each value $a_i \in dom(x_i)$, i.e. the number of the values in the domain of x_j that are consistent with a_i. If a value a_i has no support from any neighbor in the constraint graph, the value will be pruned from $dom(x_i)$. We use arc-label pairs $[(i, j), a]$ to denote the arc from node i to node j with value a in node i. For

every value $b \in dom(x_j)$ there is a set $S_{jb} = \{(i,a)|x_i = a \text{ supports } x_j = b\}$. This set contains all variable-value pairs that gain support from the $x_j = b$. The set S_{jb} can be used to look up relations in the further propagation process. For, if b is deleted from $dom(x_j)$ there is one value less that supports $x_i = a$. Therefore, each counter $Counter([i,j],a)$ that was supported by $x_j = b$ must be decremented by one. Additionally, a list L is present to manage nodes that must be propagated and boolean variables $M[i,a]$ are used to mark values a that have been deleted from $dom(x_i)$.

Algorithm 4: AC-4

```
1  L := ∅                                                    /* (1) Initialization */
2  for a ∈ dom(x_i) do
3  |  M[i,a]:= FALSE
4  |  S_ia := ∅
5  for (i,j)|C_ij ∈ C do
6  |  for a ∈ dom(x_i) do
7  |  |  total := 0
8  |  |  for b ∈ dom(x_j) do
9  |  |  |  if ⟨a,b⟩ ∈ R_{x_i,x_j} then
10 |  |  |  |  total := total + 1
11 |  |  |  |  S_jb := S_jb ∪ {(i,a)}
12 |  |  if total == 0 then
13 |  |  |  M[i,a] := TRUE
14 |  |  |  dom(x_i) := dom(x_i) − {a}
15 |  |  else
16 |  |  |  Counter[(i,j),a] := total
17 Initialize L with {(i,a)|M[i,a] == TRUE}
18 while L ≠ ∅                                               /* (2) Propagation */
19 do
20 |  Pop (j,b) from L
21 |  for (i,a) ∈ S_jb do
22 |  |  Counter[(i,j),a] := Counter[(i,j),a] − 1
23 |  |  if Counter[(i,j),a] == 0 and M[i,a] == FALSE then
24 |  |  |  L := L ∪ {(i,a)}
25 |  |  |  M[i,a] := TRUE
26 |  |  |  dom(x_i) := dom(x_i) − {a}
```

One can see from the pseudocode of Algorithm 4 that propagation is done in two steps: (1) The algorithm computes the support values for every element in the domains of all variables that take part in binary constraints. The number *total* is computed and the sets S are constructed (line 11). If a value has no support, it is marked (line 13) and deleted from the domain (line 14). All node-value combinations that have been deleted are added to L in line17. (2) The list L is processed: The algorithm looks up all elements (i,a) of the set $S_{j,b}$. It decreases all counters $Counter[(i,j),a]$ (line 22) where the value $a \in dom(x_i)$ was supported by the value $x_j = b$. If a counter is set to 0 (line 23), the value is deleted, the combination is marked and added to the end of list L (lines 24–26). The time complexity of AC-4 is $\mathcal{O}(md^2)$ in worst case (Mohr and Henderson, 1986).

In our TSP example all values in the domains of all variables are supported. Therefore, the list L is empty in line 26 of Algorithm 4 and the propagation procedure does not start.

AC-6 (Bessière, 1994) improves *AC*-4 in a way that it is not required to know the exact support value (i.e. the number of values that support a specific member of the domain) but it is sufficient to know if there is any support for a specific value. Therefore, the algorithm only searches for the first support of any value and seeks for an additional value that supports the current member only if the support value has been deleted. The general pseudocode is given in Algorithm 5. It is very similar to the code given in Algorithm 4. However, there is a new function NEXTSUPPORT(i,j,a,b) that returns the next support value (if one exists) (line 9). The function is called every time a value is deleted and a new support value is required. If there are not any supporters anymore, the unsupported value will be deleted. Note that the meaning of $M[i,a]$ has changed: It now indicates $a \in dom(x_i)$.

Algorithm 5: AC-6

```
1  L := ∅                                                    /* (1) Initialization */
2  for i = 0 to n − 1 do
3  |   for a ∈ dom(x_i) do
4  |   |   if S_ia == ∅ then
5  |   |   |   M[i,a]:= TRUE
6  for (i,j)|C_ij ∈ C do
7  |   for a ∈ dom(x_i) do
8  |   |   b := 0
9  |   |   if NEXTSUPPORT (i,j,a,b) == TRUE then
10 |   |   |   S_jb := S_jb ∪ (i,a)
11 |   |   else
12 |   |   |   dom(x_i) := dom(x_i) − {a}
13 |   |   |   M[i,a] := FALSE
14 |   |   |   L := L ∪ (i,a)
                                                             /* (2) Propagation */
15 while L ≠ ∅ do
16 |   Pop (j,a) from L
17 |   for (i,a) ∈ S_jb do
18 |   |   if M[i,a]== TRUE then
19 |   |   |   c := b
20 |   |   |   if NEXTSUPPORT (i,j,a,c) == TRUE then
21 |   |   |   |   S_jc := S_jc ∪ (i,a)
22 |   |   |   else
23 |   |   |   |   dom(x_i) := dom(x_i) − {a}
24 |   |   |   |   M[i,a] := FALSE
25 |   |   |   |   L := L ∪ (i,a)
```

Function NEXTSUPPORT(i,j,a,b) works as follows: The input parameters are variables x_i and x_j as well as the values from the domains $a \in dom(x_i)$ and $b \in dom(x_j)$. The procedure returns a boolean variable *emptysupport* and identifies a new integer b that supports a, if any exists. In the pseudocode $last(dom(x_j))$ returns the greatest value in $dom(x_j)$. The function first searches for the smallest value of $dom(x_j)$ that is greater or equal the input parameter b (line 4).

Afterwards, it checks if $\langle a, b\rangle$ is a consistent solution to $C_{i,j}$ (line 5). If this is not the case, the function $next(b, dom(x_j))$ returns the next value from $dom(x_j)$ that is greater than b (line 7). If no support was found and the entire domain was checked the algorithm returns that there was no support (line 9). The time complexity of *AC*-6 is in $\mathcal{O}(n^2d^2)$.

Function NEXTSUPPORT(i,j,a,b)

```
1  if b ≤ last(dom(x_j)) then
2  |  emptysupport := FALSE
3  |  while M(j,b) == TRUE do
4  |  |  b := b + 1
5  |  while ⟨a,b⟩ ∉ R_{x_i,x_j} and emptysupport == FALSE do
6  |  |  if b < last(dom(x_i)) then
7  |  |  |  b := next(b, dom(x_j))
8  |  |  else
9  |  |  |  emptysupport := TRUE
10 else
11 |  emptysupport := TRUE
12 Return emptysupport
```

AC-2000 (Bessière and Régin, 2001) improves the propagation algorithm *AC*-3. For, if in *AC*-3 a value b is removed from $dom(x_j)$ REVISE(i,j) checks all constraints that are defined over x_i. Every value in $dom(x_i)$ will be checked even if b was not a support for some values. To avoid these useless checks Bessière and Régin (2001) introduce a second data structure $\Delta(x_j)$ that includes all values that have been removed since the last propagation of the variable x_j. Therefore it is sufficient to check if one of the values $a \in dom(x_i)$ has lost its support by iterating over $\Delta(x_j)$ and testing if a support value has been removed. If this applies, one must check if the value must be pruned. This may be efficient on small sets of $\Delta(x_j)$, but iterating over a large set $\Delta(x_j)$ may consume much time and the probability of a match is higher. Therefore, it may be beneficial to skip this step and proceed immediately to the search of a support (and the removal of value a from $dom(x_i)$, if applicable) as this step must be done certainly. The authors introduce a *lazymode* to deal with these situations: A threshold is introduced that decides if this mode is active or passive. If $\Delta(x_j)$ contains more than $r \cdot |dom(x_j)|$ elements, where r is a given parameter, the same propagation procedure as in *AC*-3 will be carried out. This can be seen in line 3 of procedure REVISE2000(i,j,lazymode) on page 43: Only if *lazymode* is *true*, the algorithm checks the second part of the condition and only if the support was deleted, the further steps in lines 4–7 are executed. Otherwise, the algorithm checks the set $\Delta(x_j)$ for the presence of b and only if the value was removed, steps in lines 4–7 are used to find a new support value.

The function PROPAGATE2000(Q) takes the list generated by REVISE2000(i,j,lazymode) as input and performs propagation on the elements of the list Q. It picks an element out of Q and deletes it. The decision if *lazymode* is set to true depends on the size of $\Delta(x_j)$ (line 3). The procedures calls REVISE2000(i,j,lazymode) for all combinations of x_i and x_j that are connected by binary constraints. If changes in the domain of x_i are made, the variable is added to the set Q. If a domain becomes empty, the procedure returns *FALSE* to signal that the problem is infeasible.

Function REVISE2000(i,j,lazymode)

```
1 change := FALSE
2 for a ∈ dom(x_i) do
3     if ¬lazymode or ∃b ∈ Δ(x_j)|(a,b) ∈ R_{x_i,x_j} then
4         if ∄b ∈ dom(x_j)|(a,b) ∈ R_{x_i,x_j} then
5             dom(x_i) := dom(x_i) − {a}
6             Δ(x_i) := Δ(x_i) ∪ {a}
7             change := TRUE
8 Return change
```

Function PROPAGATE2000(Q)

```
1  while Q ≠ ∅ do
2      Pop x_j from Q
3      lazymode := (|Δ(x_j)| < r · |dom(x_j)|)
4      for x_i such that C_ij ∈ C do
5          if REVISE2000(x_i, x_j, lazymode) then
6              if dom(x_i) == ∅ then
7                  Return FALSE
8              Q := Q ∪ {x_i}
9      Reset Δ(x_j)
10 Return TRUE
```

Algorithm 6 presents the general framework for *AC*-2000. It iterates over all binary constraints and adds all nodes whose domains were modified to the set Q. Because all sets Δ are empty *lazymode* is set to *false*. If a variable has an empty domain in the first iteration, the CSP has no solution (line 7). If REVISE2000(i,j,lazymode) has pruned values from the node's domain it is added to Q. Finally, the function PROPAGATE2000(Q) is called to do the further propagation steps as it is known from *AC*-3. Note that in line 3 the boolean variable *lazymode* is set. The time complexity of *AC*-2000 is $\mathcal{O}(nd^3)$.

Algorithm 6: AC2000

```
1 Q := ∅
2 for x_i ∈ X do
3     for x_j such that C_ij ∈ C do
4         if REVISE 2000(x_i, x_j, false) then
5             if dom(x_i) == ∅ then
6                 Return FALSE
7             Q := Q ∪ {x_i}
8 Return PROPAGATE2000(Q)
```

AC-2001 (Bessière and Régin, 2001) improves *AC*-2000 by adding an additional data structure $Last(x_i, a, x_j)$ to the algorithm that stores the value that the REVISE2001(i,j,lazymode) has found as support for a in the last iteration. One has simply to check if the value $Last(x_i, a, x_j)$ is still in $dom(x_j)$. This is done in line 3 of the pseudocode of REVISE2001(i,j,lazymode). If the value was deleted, a new support value must be found. We speed up this process using the as-

sumption that all domains are ordered increasingly: We can start after the element $Last(x_i, a, x_j)$ because all of its predecessors have been checked in former iterations (line 4).

Function REVISE2001(i,j,lazymode)

```
1  change := FALSE
2  for a ∈ dom(x_i) do
3      if Last(x_i, a, x_j) ∉ dom(x_j) then
4          if ∃b ∈ dom(x_j) >_d Last(x_i, a, x_j) ∧ ⟨a, b⟩ ∈ R_{i,j} then
5              Last(x_i, a, x_j) := b
6              dom(x_i) := dom(x_i) − {a}
7          else
8              Δ(x_i) := Δ(x_i) ∪ {a}
9              change := TRUE
10 Return change
```

Function PROPAGATE2001(Q)

```
1  while Q ≠ ∅ do
2      Pop x_j from Q
3      for x_i such that C_ij ∈ C do
4          if REVISE 2001(x_i, x_j, false) == TRUE then
5              if dom(x_i) == ∅ then
6                  Return FALSE
7              Q := Q ∪ {x_i}
8      Reset Δ(x_j)
9  Return TRUE
```

To get the complete pseudocode of *AC*-2001 one can replace REVISE2000 by REVISE2001 in line 4 of Algorithm 6 and PROPAGATE2000(Q) in line 8 by PROPAGATE2001(Q). These modifications reduce the time complexity to $\mathcal{O}(nd^2)$.

A generalization of arc consistency to arbitrary constraints is hyper-arc consistency. A constraint on k variables $x_0, \ldots, x_{k-1}$ is called hyper-arc consistent iff for every $i \in [0, \ldots, k-1]$ and $a \in dom(x_i)$ there is a solution $sol(x_0, \ldots, x_{k-1})$ where $x_i = a$. Thus, for binary constraints hyper-arc consistency coincides with arc consistency. A CSP is hyper-arc consistent iff all of its constraints are hyper-arc consistent. Hyper-arc consistency guarantees that every $c_i \in C$ has a solution but not that the CSP has a solution. Some authors denoted hyper-arc consistency as *domain consistency* or *generalized arc consistency* (Bessière, 2006, p. 38).

AC-5 (Hentenryck, 1992) and *AC*-7 (Bessière et al., 1995, 1999) are propagation algorithms that deal with hyper-arc consistency. Because both algorithms are generalizations of *AC*-4 and *AC*-6 we will not consider them in detail here. Further illustrations and explanations for these algorithms can be found in Ghédira (2013, Sec. 2.2.5) for *AC*-5 and in Ghédira (2013, Sec. 2.2.7) for *AC*-7.

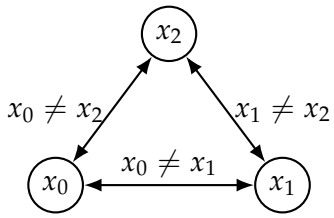
From the explanations above it can be concluded that there was a lot of research on propagation techniques that can tighten the domains. The algorithms evolved over time and have

improved worst case time complexity to achieve AC. It may be questionable at the moment to spend effort on constraint propagation but Section 3.6 will stress the importance of good propagation algorithms within the search process.

3.4.3 Path Consistency

Path Consistency can be applied to normalized CSPs (Montanari, 1974). In a *normalized CSP* for each subsequence $x_i, x_j \in \mathcal{X}$ of its variables there exists at most one constraint in P that has x_i and x_j in its scope. Every CSP of atomic constraints can be normalized by merging the constraints that share the same scope. An example constraint graph is illustrated in Figure 3.5.

Figure 3.5: Constraint Network with Three Binary Constraints



A set of two variables $\{x_i, x_j \in \mathcal{X}\}$ is ***path consistent*** relative to variable $x_k \in \mathcal{X}$ iff for every consistent assignment $\langle a_i, a_j \rangle \in R_{x_i,x_j}$ there is a value $a_k \in dom(x_k)$ such that $\langle a_i, a_k \rangle \in R_{x_i,x_k}$ and $\langle a_j, a_k \rangle \in R_{x_j,x_k}$ are consistent.

CP algorithms to enforce PC are closely related to AC algorithms. The algorithms to achieve arc consistency were extended to deal with path consistency, too. Therefore every time a new AC algorithm emerged, a new consistency technique for PC appeared later. A brief overview on the progress in the field of PC algorithms and the corresponding AC algorithms is given in Bessière (2006, p. 51). Simple examples for PC can be found in Bessière (2006, p. 50), Dechter (2009, p. 61–62), and Apt (2009, pp. 151–152).

3.4.4 *k*-Consistency

A very general definition of consistency is ***k-consistency***. A CSP is *k-consistent* iff for any tuple of k variables $x_0, \ldots, x_{k-1} \in \mathcal{X}$ any $(k-1)$-consistent assignment may be extended to a consistent assignment with the k-th variable. It can be seen that *node consistency* can be denoted as 1-consistency, *arc consistency* as 2-consistency and *path consistency* is equivalent to 3-consistency in normalized networks.

However, k-consistency does not guarantee that a CSP is globally consistent, even if k is large compared to the number of variables. Furthermore, a k-consistent problem must not be necessarily $(k-1)$-consistent. Because our running example is not suitable to demonstrate this property, we refer to the simpler examples in Bessière and Régin (2001) and (Apt, 2009, pp. 159-160). To overcome these difficulties, we define *strong k-consistency*: A CSP is called ***strongly* k-*consistent*** iff P is i-consistent for every $i \in [1, \ldots, k]$, where $k \geq 1$. A CSP with n variables that is strongly n-consistent is *globally consistent*. A propagation algorithm that achieves strong k-consistency can be found in Cooper (1989).

3.4.5 Bounds Consistency

A constraint $c \in C$ of k variables is ***bounds consistent*** if the smallest and the largest value in the domain of every variable x_i, $a_i = \min dom(x_i)$ ($b_i = \max dom(x_i)$, respectively), can be extended to a tuple $\langle a_0, \ldots, a_i, \ldots, a_{k-1} \rangle \in R_c$, $\langle a_0, \ldots, b_i, \ldots, a_{k-1} \rangle \in R_c$, respectively. A CSP is bounds consistent if each of its constraint is bounds consistent. A sketch of a general propagation algorithm is presented in Marriott and Stuckey (1999, Sec. 3.4). Furthermore, there is a number of specialized algorithms on specific global constraints, see e.g. Lopéz-Ortiz et al. (2003) for the constraint *ALLDIFFERENT($\mathcal{X}$)*.

Achieving bounds consistency is much cheaper than applying k-consistency techniques because only the extremes of a domain must be checked. Especially large domains of integers can imply heavy computational effort. An example would be an extension of the TSP by including a load constraint: Each customer of the salesman requires an amount of capacity and the total capacity is limited. To keep track of the load we can add a new variable to every node that gives the remaining capacity of the salesman's vehicle at the moment it arrives therein. In a pick-up scenario the variables' values at every customer node are equal to the vehicle's capacity at the beginning. Every time the vehicle picks up load at a customer the remaining capacity decreases. To check if an additional customer could be serviced by the salesman, it is sufficient to evaluate the largest value in the domain of the customer. If this value is consistent, all other (smaller) values could be part of the solution, too. However, it would be useless to check every value in the domain one by one.

Note that the total demand cannot exceed the vehicle's capacity in this example because the presence of only one vehicle would turn the problem to be infeasible. In this situation, more vehicles would be needed to service all customers and the resulting problem would become a Multi Traveling Salesman Problem or a VRP. Therefore, for the sake of simplicity we assume sufficient capacity. Furthermore, we waive the formalization of these constraints because they will be introduced in Chapters 4 and 5.

In this section, we have presented several algorithms to achieve different levels of consistency. It may be obvious to enforce a possibly high consistency level, preferably n-consistency to get information on the feasibility of a problem. However, enforcing consistency requires computational effort that in many cases will exceed the time saved in the search process. It is usual to determine the consistency level for every constraint individually by estimating the importance of a variable's domain size for the search process. We will refer to this fact in the following chapters.

3.5 Global Constraints

As explained in Section 3.3 a *global constraint* describes relations between a non-fixed number of variables. This kind of constraint can be seen as *shortcuts* because the same restriction on the variables' values can be modeled by using several atomic constraints. Therefore, global constraints simplify the modeling respectively programming tasks. Furthermore, the usage

of global constraints guides the solution process because global constraints can provide information on the structure of a problem to the solver that can use specific algorithms in the propagation process (Hoeve and Katriel, 2006, p. 169).

Global constraints are one of the main aspects in CP. Some authors (Hooker, 2002; Freuder and O'Sullivan, 2014) claim that modeling with global constraints will increase the meaning of optimization in the real world. Their visions is that users can work with toolkits that allow the application of constraints by *drag&drop*, i.e. programs offer lists of global constraints that can be picked and combined. Hereby, global constraints become vocabularies to describe real-world problems and no knowledge about modeling or the solution process is required at the end user's level. On the other hand, experts on a certain field may be able to identify structures in their area that cannot captured by CP specialists. Thus, the exchange of knowledge from these two areas enriches CP and propagation techniques.

An extensive and very detailed overview on all global constraints can be found in the Global Constraint Catalog (Beldiceanu et al., 2015). In this document every global constraint is described by an example, its first occurrence in literature is mentioned, a sketch of the propagation algorithm and an illustration through a constraint graph as well as an overview of the solvers that support this global constraint is given. The last point is important because not every constraint solver supports every global constraint and—if they are included—the names and the underlying propagation algorithms may differ between solvers. Additionally, there is no standard definition for the constraints, e.g. the input parameters of a global constraint can be different for two solvers. A survey on global constraints can be found in Régin (2011).

In this section we will only sketch some global constraints to illustrate the idea and show the advantages of this concept.

ALLDIFFERENT($\mathcal{X}$)

A very well-known example for a global constraint is the constraint *ALLDIFFERENT($\mathcal{X}$)*, where $\mathcal{X}$ is a set of variables. The constraint was introduced by Lauriere (1978) and enforces all variables $x_i \in \mathcal{X}$ to take pairwise distinct values; no two variables can take the same value. The formal description is as follows (Hoeve and Katriel, 2006):

$$ALLDIFFERENT(x_0, \ldots, x_{n-1}) = \{(a_0, \ldots, a_{n-1}) | a_i \in dom(x_i), a_i \neq a_j, i \neq j\}$$

The global constraint can also be described as a conjunction of $n(n-1)$ atomic constraints ($x_i \neq x_j, i \neq j$), where n is the number of variables in the CSP.

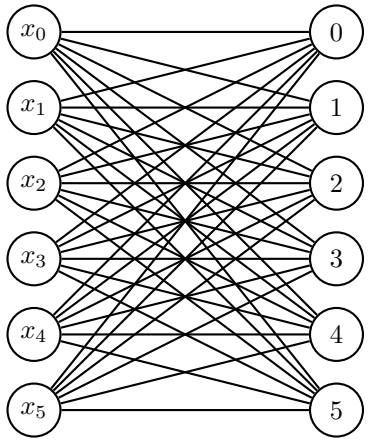
A propagation algorithm for hyper-arc consistency was found independently by Régin (1994) and Costa (1994), where the set of variables $\mathcal{X}$ and the conjunction of all domains $\mathcal{D}$ is represented in a value graph G. This is a bipartite graph, where one set contains only the variables and the other set the domain values. Therefore the intersection of the two sets is empty. In the filtering algorithm propagation is executed by principles of graph theory. Thus, a tuple $(a_0, \ldots, a_{n-1}) \in ALLDIFFERNT(a_0, \ldots, a_{n-1})$ iff $M = \{\{x_0, a_0\}, \ldots, \{x_{n-1}, a_{n-1}\}\}$ is a matching in the graph G. A matching in an undirected graph $G = (V, E)$ is a set of $M \subset E$

of disjoint edges, i.e. no two edges in M share a vertex. The *cardinality* of a matching M is the number of edges $|M|$ in it. A matching of maximum cardinality is called a *maximum matching*.

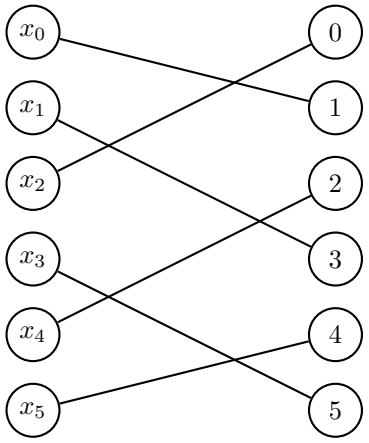
Figure 3.6 shows a bipartite graph for the *ALLDIFFERENT($\mathcal{X}$)* constraint in our TSP example. The nodes on the left-hand side of each graph are the variables and the domains' values are on the opposite side—all values are present in all domains before the first propagation. Therefore all nodes are connected with each other in Figure 3.6a. A maximum matching—which is a solution of the problem—is given in Figure 3.6b.

Figure 3.6: Illustration of a Bipartite Graph and Its Maximum Matching for the TSP Example

(a) Bipartite Graph



(b) Maximum Matching



Using matching algorithms from graph theory is advantageous because Hopcroft and Karp (1973) described an algorithm to identify a matching covering $\mathcal{X}$ in a bipartite graph with a time complexity of $\mathcal{O}(\sqrt{|\mathcal{X}|m})$. Alt et al. (1991) improved the algorithm and yield an algorithm with time complexity of $\mathcal{O}(|\mathcal{X}|^{\frac{3}{2}}\sqrt{m\log|\mathcal{X}|})$. Therefore efficient propagation can be executed for this global constraint using the representation as a graph. There are further algorithms that achieve bounds consistency (Guernalec and Colmerauer, 1997; Lopéz-Ortiz et al., 2003; Mehlhorn and Thiel, 2000) using the graph representation, too. More details on the constraint can be found in Downing et al. (2012).

ELEMENT(y,z,$\mathcal{X}$)

A further global constraint that is useful in our context is the constraint *ELEMENT(y,z,$\mathcal{X}$)*. It enforces the variable z to take the value of the y^{th} item in a set of variables $\mathcal{X} = \{x_0, \ldots, x_{n-1}\}$, that is $z = x_y$. Thus, it is possible to index a variable with another variable. Formally (Hoeve and Katriel, 2006):

$$ELEMENT(y, z, x_0, \ldots, x_n) = \{(e, f, a_0, \ldots, a_{n-1}) | e \in dom(y), f \in dom(z), \\ a_i \in dom(x_i), f = a_e\}$$

The constraint was presented first by Hentenryck and Carillon (1988). It has no specialized propagation algorithm but it useful for modeling: As we have stated above, this constraint allows to index variables with other variables.

In our running example there can be symmetric solutions. It is not of importance if the vehicle passes a tour forward or backward, both tours would be feasible and the objective function value would be the same. To avoid symmetric solutions we add a variable p and the following constraints to our problem:

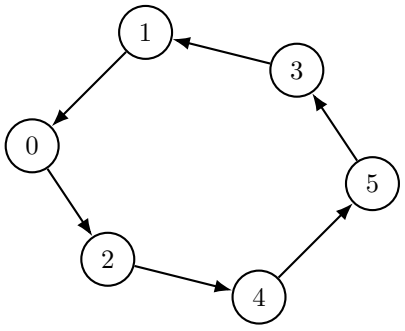
$$ELEMENT(p, 0, \mathcal{X}) \tag{3.10}$$

$$p < x_0 \tag{3.11}$$

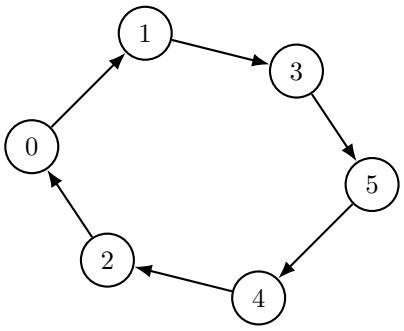
Constraint (3.10) can be rewritten as $x_p = 0$. Thus, it looks for the node that has node 0 as its successor. Because of Constraint (3.11) this value must be smaller than the value of x_0 and the tour can only have one direction. Figure 3.7 illustrates the situation: The tour in Figure 3.7a would be feasible, because $x_1 = 0$, that is $p = 1$ and $p = 1 < x_0 = 2$. However, the tour in Figure 3.7b—that has the reversed order of nodes—is infeasible because $p = 2$ and $x_0 = 1$.

Figure 3.7: Avoiding Symmetric Solutions in the TSP Example

(a) Feasible Tour



(b) Infeasible Tour



TOUR($\mathcal{X}$)

Another global constraint that is known from our running example is *TOUR($\mathcal{X}$)*, where $\mathcal{X}$ is a set of variables (nodes) again. This constraint enforces to cover a graph G described by the set of nodes V (associated with variables $\mathcal{X}$) with a Hamiltonian cycle. A Hamiltonian cycle is a cycle that visits each vertex in a graph once (Jungnickel, 2013, Sec. 1.4). According to the *Global Constraint Catalog* this constraint was introduced by Althaus et al. (2002) but Caseau and Laburthe (1997) introduced the idea and a propagation algorithm earlier.

Formally, this global constraint is stated as follows:

$$TOUR(x_0, \ldots, x_{n-1}) = \{(a_0, \ldots, a_{n-1}) | a_i \in dom(x_i), a_0, \ldots, a_{n-1} \text{ is cyclic}\}$$

We will describe the propagation algorithm by Caseau and Laburthe (1997) henceforth. During the propagation process we store the *start node*, *end node* and *length* of each chain that includes the depot node 0. We use the notation $start_0$, end_0 and $length_0$, where the length is the number of arcs in the chain. Upon each assignment—we assign $x = y$—the algorithm checks if there are already subchains adjacent to x or y. The end of the chain starting at y is denoted by b, and the start of the chain ending in x is called a, i.e. the new chain is $a \rightarrow \ldots \rightarrow x \rightarrow y \rightarrow$

$\ldots \rightarrow b$. The number of arcs in the chain connected with a is $length_a$ and $length_b$ for the chain including b.

Three rules are applied to avoid a solution that includes subcycles:

- If $x = end_0$ and $length_0 + length_b < n - 2$, we infer $x_b \neq start_0$.
- If $y = start_0$ and $length_0 + length_a < n - 2$, we infer $x_{end_0} \neq a$.
- Otherwise, a is removed from $dom(b)$.

The first rule deals with a second chain (with start node y) that is added to the end of the chain including node 0 (*depot chain*). If the length of both tours is less than $n - 2$, the successor of the node at the end of the second chain cannot be the beginning node a of the depot chain. Therefore, the value a ($= start_0$) is deleted from $dom(x_b)$.

The second rule applies in situations where a chain of nodes (*first chain* starting at node a) is added before the beginning of the *depot chain*. If the length of the tours that are joined is less than $n - 2$ the end node of the *depot chain* b cannot be connected with start node in the *first chain*; value a is pruned from $dom(x_{end_0})(= dom(x_b))$.

The third rule deals with the standard situation: The arc that would establish a (sub)cycle is forbidden.

Figure 3.8 on the next page illustrates the situation: Figure 3.8a gives an example for the first rule, Figure 3.8b depicts the second situation and the standard case is shown in Figure 3.8c. In Figure 3.8a there are two chains—(0,1) with $length_0 = 1$ and (4,5) with $length_b = 1$, both marked by solid arcs—and $x_1 = 4$ shall be assigned (dashed arcs). Because $x_1 = end_0$ and the length of both chains is $2 < (6 - 2)$, the instantiation $x_5 = 0$ is forbidden (dotted arc): $dom(x_5) := dom(x_5) - \{0\}$. Note that the domains under the nodes state the domains before the current assignment. Figures 3.8b and Figure 3.8c can be interpreted analogously.

More details on the global constraint *TOUR($\mathcal{X}$)*—that is called *CIRCUIT($\mathcal{X}$)*, too—can be found in Francis and Stuckey (2014). Another (incomplete) propagation algorithm can be found in Kaya and Hooker (2006).

From the three examples given above one can guess the variety of global constraints. It is obvious that modeling is simpler with these constraints. Furthermore, it is illustrated that propagation algorithms have been developed for efficient reasoning. These algorithms can be applied on the graph formulations (*ALLDIFERENT($\mathcal{X}$)*) or be highly problem-specific (*TOUR($\mathcal{X}$)*).

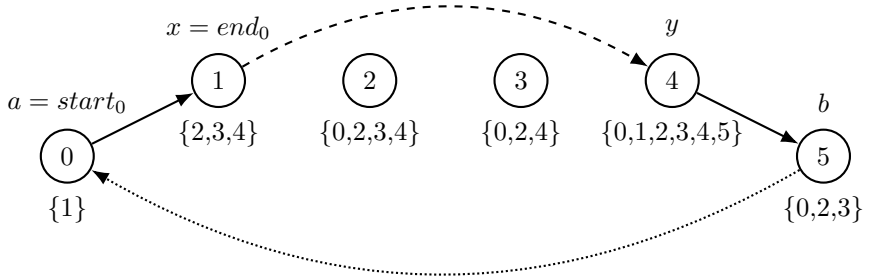
Beside the usage of the predefined global constraints it is also possible to define user-specific constraints. They may be needed because no global constraint fits the requirements in a specific model or a solver does not support a global constraint and it must be introduced manually to the algorithm. We will present problem-specific global constraints in Chapters 4 and 5.

3.6 Search Algorithms

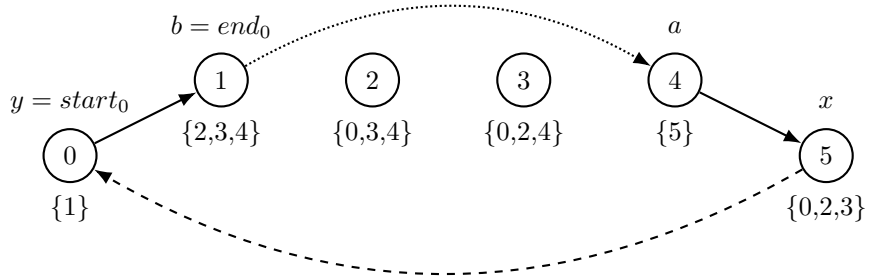
Propagation techniques may be sufficient to find an arbitrary solution for some problems or state the (in)feasibility of an instance. However, for most of the problems it will be required to

Figure 3.8: Illustration of Propagation Rules for the TOUR($\mathcal{X}$) Constraint

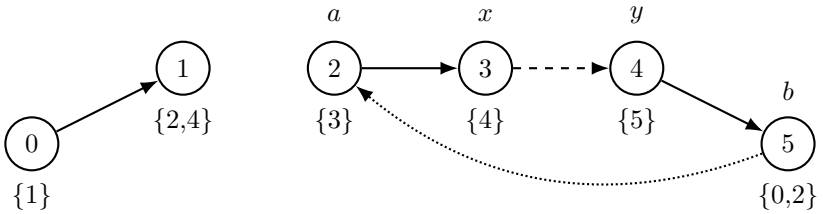
(a) First Rule: $x = end_0$



(b) Second Rule: $y = start_0$



(c) Third Rule: Depot not in Current Chains



apply further search techniques to the problem to explore all solutions. Search algorithms in CP are similar to those that are known from OR but include propagation algorithms. Within the area of CP search algorithms can be distinguished by the fact if they are complete or incomplete. Complete search algorithms ensure that *all* solutions will be found (if at lest one exists). Furthermore they are able to prove that no solution exists or to find optimal solutions. Incomplete algorithms are limited in the sense that only some of the solutions will be detected and/or no final decision on the global consistency can be made.

In this chapter we will introduce some of the common search algorithms and use pseudocode (that is mainly based on Apt (2009)) again to illustrate the behavior of the algorithms exactly. The term a_i states the value instantiated to the i-th variable. Due to reasons of brevity we moreover use the following procedures:

- `CHECK(inst,j,a)` returns if the assignment of a to variable x_j is consistent with the instantiation of $j-1$ variables $inst = \langle a_0, \ldots, a_{j-1} \rangle$. No propagation algorithm is executed within this procedure.
- `PROPAGATE(j,`$\mathcal{D}$`)` executes the pre-defined pruning steps (cf. Section 3.4) on the domains $dom(x_{j+1}), \ldots, dom(x_{n-1}) \in \mathcal{D}$. The function returns *true* if a failure occurs during propagation, i.e. the problem turns out to be infeasible. Otherwise it returns *true*.

This sections starts with a focus on algorithms for the CSP. Afterwards, we will give an overview on optimization algorithms where we will consider exact and heuristic approaches.

3.6.1 Algorithms on Satisfaction Problems

Backtracking Algorithms

Backtracking search generally takes place in a search tree. A tree consists of a root, nodes, and leafs. The *root node* is the top node and has no predecessor, while *leafs* have no successor. A node is a *child* y of a (*parent*) node x if the edge $x \rightarrow y$ is the last one in a simple path from the root to node y. The *length* of a simple path from the root node to the node x is the depth of x in the tree, the length of the longest simple path form a root node to a leaf is the height of a tree (Cormen and Sussman, 2009, pp. 1176–1177). A complete assignment in a leaf node is a *solution* of a problem.

The remainder of this chapter will discuss the following backtracking algorithms whose characteristics are summarized in Table 3.2:

1. Naive Backtracking (BT): This simple method assigns variables step by step. If an inconsistency is detected, it tries another instantiation.
2. Backjumping (BJ): Instead of backtracking chronologically this algorithm *jumps* back to the source of the conflict.
3. Forward Checking (FC): The algorithms are designed to move efficiently forward within a search tree. On every level a given instantiation of variables is extended.

Table 3.2: Search Algorithms as Combinations of Tree Search and Constraint Propagation (Kumar, 1992)

Algorithm	Chronological Backtracking	Constraint Propagation
BT	yes	no
BJ	no	no
MAC	yes	arc consistency
FC	yes	arc consistency[1]
MC-k	yes	strong k-consistency

4. Maintaining Arc Consistency (MAC): This search algorithm is defined only for binary constraints and imposes arc consistency on future variables.

5. Maintaining Strong k-consistency (MC-k): The search algorithm executes constraint propagation to achieve strong k-consistency in the entire search process.

An extensive survey on tree search and AC can be found in Nadel (1988).

Generally, backtracking assigns incrementally variables to values. In every a node an additional constraint is posted compared to its parent node. The deeper the level of a node in a search tree the more assignments have been posted in this node.

The shape of the search tree heavily depends on the order of branching variables and chosen values. Therefore specific *variable selection heuristics* and *value selection heuristics* must be defined. *Variable selection heuristics* determine the choice of the variable that shall be instantiated next. One natural way is to use the variable with the smallest domain. The idea behind this rule is that the resulting search tree will be the smallest. Another rule would be to select the variable with the highest degree $deg(x)$, i.e. the variable that is present in most of the constraints. Fixing this variable will result in a propagation process that affects various variables. Beside this intuitive ideas there are some more selection rules in the literature, e.g. the variable that has the smallest or largest value in its domain, the variable with the smallest difference between the minimum and maximum element in the domain etc. *Value selection* for a variable determines the value that is used for branching by the heuristic. One can use simple rules (smallest/largest value, value closest to the median etc.) or more specific heuristics that try to evaluate selections by a value. For detailed descriptions we refer to Ghédira (2013, pp. 107-116).

For naive branching there exist various branching strategies: In *enumeration* every value of a domain is posted. The number of child nodes equals the number of values in the domain of the parent nodes. *Two-way branching* selects a value out of the domain $dom(x)$ and creates two nodes: The first one sets the variable x to a specific value $x = a$, the second constraint forbids this value in the subtree, i.e. $x \neq a$. Finally, *domain splitting* reduces the domains of the variables in every subproblem: Constraints $x \leq b$ and $x > b$ are posted, where $b \in dom(x)$. This branching rule does not necessarily assign the variable x but triggers propagation. Furthermore, there exist branching strategies that are specific to a class of problems and branching

[1] On constraints with exactly one uninstantiated variable.

strategies with non-unary constraints. However, the latter class of branching strategies seems to require knowledge on the problems structure, too. Therefore, we skip details on that branching rules here but refer to Caseau and Laburthe (1994) for an example on job shop scheduling.

Naive Backtracking

Naive Backtracking is a depth-first search. Its search process works in a recursive manner and calls the procedure BACKTRACK(j,$\mathcal{D}$) that takes the set of domains $\mathcal{D}$ and the level j as input parameters. Therein, the algorithm copies the domain of x_j and chooses a value $d \in dom(x_j)$, deletes it from the copy of the domain and checks if it is a feasible instantiation (lines 2–3). Next, the current instantiation is checked for inconsistencies (line 7). If none were found, the algorithm will call BACKTRACK(j,$\mathcal{D}$) for the next variable (line 8), i.e. the next level of the tree is explored. However, if the current node was a leaf node ($j == n-1$) the algorithm signals success (line 5) and the further exploration of the search space is skipped. If the instantiation was infeasible, the next value of the domain of x_j is checked. If a domain becomes empty, no value could be found that is consistent with the current assignment in this node. Therefore, the algorithm jumps back to the parent node and selects the next value. The algorithm itself is given in Algorithm 7: It sets the boolean variable *success* to *FALSE* (line 1) and carries out propagation in the root node (line 2). If no infeasibility is detected, the algorithm calls BACKTRACK(j,$\mathcal{D}$) for the next level 1 (line 3). This function is called recursively until a solution was found and returns *success* == *TRUE* (line 4). If the entire tree was explored and no solution was found, the algorithm will return the initial value of *success* to signal failure.

Function BACKTRACK(j,$\mathcal{D}$)

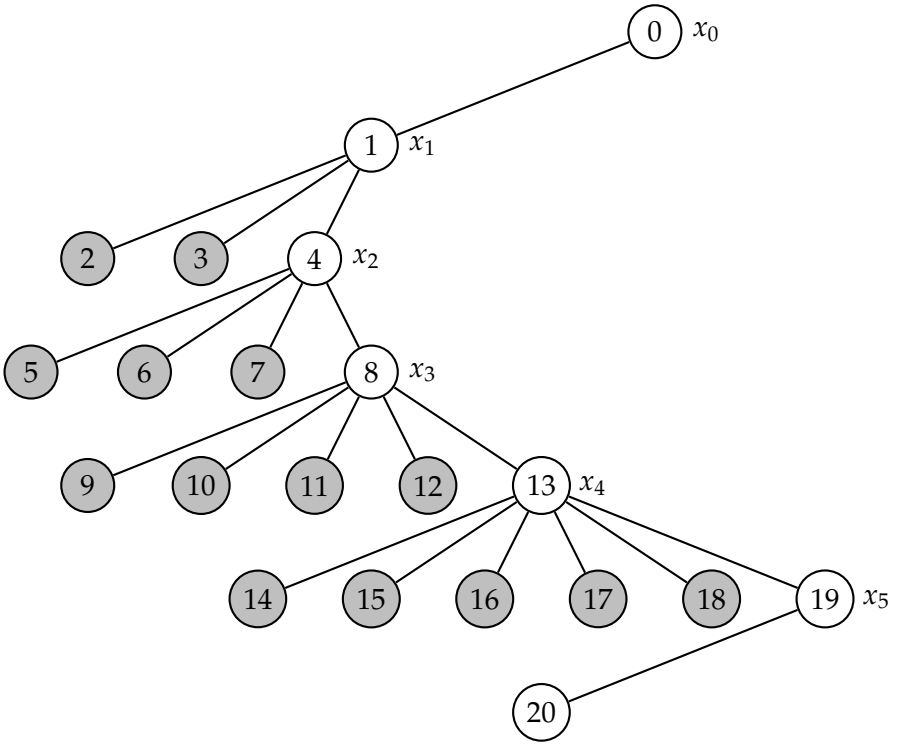
```
1 while dom(x_j) ≠ ∅ ∧ ¬success do
2 |   Pop a from dom(x_j)
3 |   if Instantiation {(x_0,a_0),...,(x_{j-1},a_{j-1}),(x_j,a)} is consistent then
4 |   |   x_j := d
5 |   |   success := (j == (n-1))
6 |   |   if success == FALSE then
7 |   |   |   if CHECK(j,D) == FALSE then
8 |   |   |   |   success := BACKTRACK (j+1,D)
9 Return success
```

Algorithm 7: Naive Backtracking

```
1 success := FALSE
2 if CHECK(0,D) == TRUE then
3 |   BACKTRACK (1,D)
4 Return success
```

Figure 3.9 on the next page illustrates the search tree for the TSP example without optimization. To keep the example simple, the variables are assigned in the order of their (increasing) indexes. The branching heuristic selects the smallest (so far not tested) value in the domain of

Figure 3.9: Naive Backtracking Search Tree for the TSP Example



the variable for the assignment. Remember that the domains are ordered. In the illustration white nodes represent feasible assignments, grey ones are infeasible.

On level 0 all (successor) nodes are left in the domain and the value 1 is chosen first (the assignment $x_0 = 0$ is in conflict with the unary Constraint (3.6) and we assume all variables to be node consistent). Next, on level 1 the algorithm tries $x_1 = 0$ which would build a subtour and $x_1 = 1$ which violates Constraints (3.7) and (3.8) again. The algorithm proceeds until a solution is found: The resulting tour is $0 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 0$. Note that a large number of inconsistent nodes has been explored.

Backjumping

The Backjumping algorithm improves the backtracking behavior of the BT algorithm. The algorithm introduces the concept of a *culprit variable* that causes the inconsistency of the current value. The culprit variable x_b relative to an inconsistent leaf $\langle a_0, \ldots, a_i \rangle$ is defined by $b = \min_j \{j \leq i | a_j \text{ conflicts with } x_{i+1}\}$. We will present the BJ algorithm of Gaschnig (1977) that jumps back to the culprit variable only if the tuple is in conflict with x_{i+1}. Therefore, the procedure BACKJUMP(j,$\mathcal{D}$) searches for the most recent variable whose instantiation is in conflict with the current assignment b. It saves in the variable $latest_j$ for every x_j the node of the most recent assignment of a value to a variable that does not conflict with the current assignment, i.e. the predecessor of the culprit node: The value $latest_j$ is updated to the next node as long as the assignment is consistent. If an inconsistent assignment was detected, it will be checked if the most current assignment is on a deeper level than the value $latest_j$. For the case an instantiation with j variables was found, the algorithm returns this solution. If all values

have been removed from $dom(x_j)$, the function returns a failure. If this is the case, the general Algorithm 8 jumps back to the node before the culprit node $latest_j$ (line 6). Otherwise, it proceeds with the assignment of the next variable (line 8).

Function BACKJUMP(j,$\mathcal{D}$)

```
1  while dom(x_j) ≠ ∅ do
2  |  Pop b from dom(x_j)
3  |  consistent := TRUE
4  |  k := 0
5  |  while k < j ∧ consistent == TRUE do
6  |  |  if k > latest_j then
7  |  |  |  latest_j := k
8  |  |  if Instantiation ⟨a_0, ..., a_k, b⟩ is not consistent then
9  |  |  |  consistent := FALSE
10 |  |  else
11 |  |  |  k := k + 1
12 |  |  if consistent == TRUE then
13 |  |  |  Return TRUE
14 Return FALSE
```

Algorithm 8: Backjumping

```
1  success := FALSE
2  i := 0
3  latest_0 := 0
4  while 0 < i < (n − 1) do
5  |  if BACKJUMP (j,D) == FALSE then
6  |  |  i := latest_i
7  |  else
8  |  |  i := i + 1
9  |  |  latest_i = 0
```

This algorithm is useful in situations where the inconsistent assignment has been made at the beginning of the search tree, while the inconsistency is detected only later. The naive BT algorithm detects the same inconsistencies more often as it backtracks chronologically. Jumping back to the culprit variable is therefore more efficient. However, this is not true for the TSP example—the BJ search tree is identical to Figure 3.9. Note that the algorithms in this and the following sections would explore the entire tree and report all solutions. However, the resulting search trees are spacious and we therefore prefer trees that only show the way to the first solution to exhaustive ones.

Besides the algorithm of Gaschnig (1977) there are the conflict-directed (Prosser, 1993) and the graph-based BJ algorithm (Dechter, 1990).

Forward Checking

Forward Checking (Haralick and Elliot, 1980; Bacchus and Grove, 1995) is a *look ahead* algorithm

that extends the simple BT algorithm. It applies specific propagation algorithms not only to the current variable but to all not yet instantiated variables. The pseudocode of the recursive algorithm is given in Algorithm 9. It starts with the entire set of variables $V = \mathcal{X} = \{x_0, \ldots, x_{n-1}\}$ and empty set of instantiations $I = \emptyset$ and calls FC(V,I). This function selects a variable according to the variable ordering heuristic (line 4) and tests all its values (line 5) in the function FORWARDCHECK(i,a,$\mathcal{X}$) for consistency. Therein, all values (line 4) in the domains of all remaining variables (line 2) are tested for consistency and inconsistent values are deleted (line 6). An empty domain signals that the assignment is inconsistent (line 8). If the assignment is consistent, function FC(V,I) recursively calls FC(V,I) with a reduced set of variables and the current instantiation (line 7). A solution is found if the set V is empty and therefore no variables are left for assignment (line 2).

Function FORWARDCHECK(i,a,$\mathcal{X}$)

```
1 consistent :=TRUE
2 for x_j ∈ X − {x_i} do
3 |  while consistent == TRUE do
4 |  |  for b ∈ dom(x_j) do
5 |  |  |  if ∄⟨a, b⟩ ∈ R_{x_i,x_j} then
6 |  |  |  |  dom(x_j) := dom(x_j) − {b}
7 |  |  if dom(x_j) == ∅ then
8 |  |  |  consistent := FALSE
9 |  Return consistent
```

Function FC(V,I)

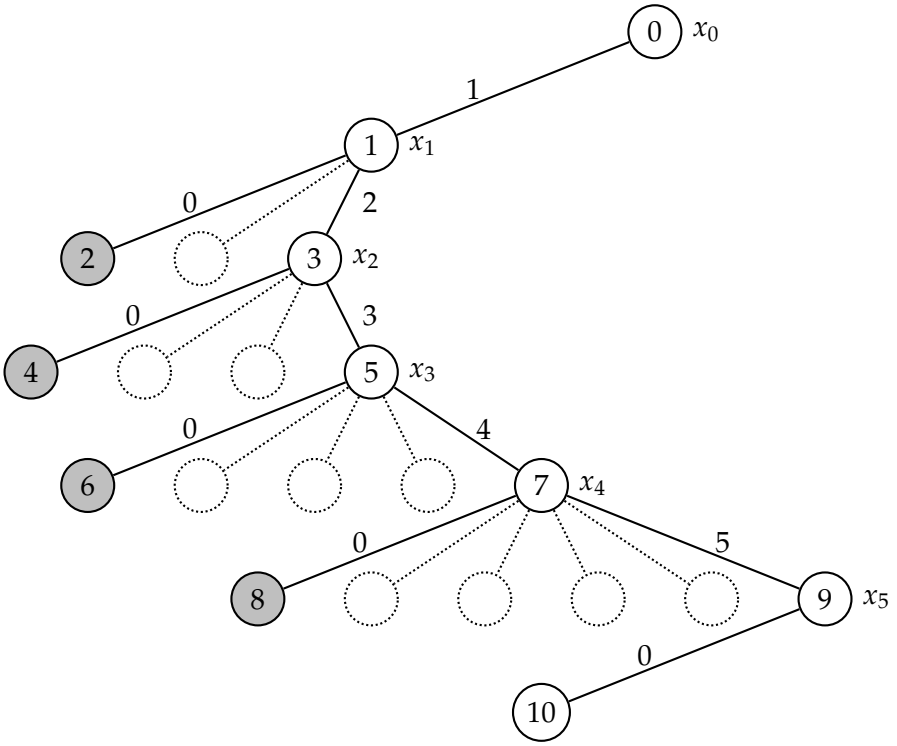
```
1 if V = ∅ then
2 |  I is a solution
3 else
4 |  Pop x_i ∈ X
5 |  for a ∈ dom(x_i) do
6 |  |  if FORWARDCHECK x_i, a, X) then
7 |  |  |  FC (X − {x_i}, I ∪ x_i = a)
```

Algorithm 9: ForwardChecking

```
1 V := X
2 I := ∅
3 FC (V,I)
```

Figure 3.10 on the next page demonstrates the search process up to the first solution with the same ordering heuristics as for BT (because FC only considers binary constraints, global constraints *ALLDIFFERENT($\mathcal{X}$)* and *TOUR($\mathcal{X}$)* are ignored in FORWARDCHECK(i,a,$\mathcal{X}$)). Nodes that are not explored compared to Figure 3.9 are printed using dashed lines. Therefore, the algorithm instantiates $x_0 = 1$ and the value 1 is deleted from all upcoming variables' domains due to Constraint (3.7). This works analogously in all other nodes, however the algorithm

Figure 3.10: Forward Checking Search Tree for the TSP Example



repeatedly tries to assign the value 0 to variables as the subtours are not forbidden by a binary but a global constraint. It is obvious that propagation reduces the size of the search tree enormously.

Maintaining Arc Consistency and Maintaining Strong k-consistency

MAC was discussed by Gaschnig (1974) for the first time and describes a backtracking algorithm where arc consistency is enforced in every node of the search tree with at least one uninstantiated variable. According to Sabin and Freuder (1994) MAC is different from FC in the following points:

- The constraint propagation algorithm is executed at the beginning, therefore search starts on an arc consistent constraint network.
- If any variable x_i is assigned to a value a_i, all other values are deleted from $dom(x_i)$ and the knowledge is used to propagate all the other constraints in the network. Thus, not only the nodes in the neighborhood are propagated.

MAC dedicates more time to propagation but this effort leads to a more compact search tree with less nodes (cf. Ghédira, 2013, p. 90).

At each node of the search tree, a pruning algorithm that achieves AC is executed. Arc consistency algorithms must only be executed on the constraint that was posted in the branching process. If this modifies the domain of a variable, arc consistency is enforced for these

constraints until no further changes occur. If propagation deletes all values of a domain, the branching is rejected. Otherwise it is accepted and search continues on the next level.

Maintaining Strong k-consistency follows the same idea as MAC but achieves higher consistency levels. Again, k denotes the level of consistency (Section 3.4). The idea is that every value in a domain can be part of an assignment of k variables. However, this approach was only tested on single problems, statements on general CSPs cannot be made (Beek, 2006).

Beside BT there is a backtrack-free search that is a left-most depth-first method and assigns values to variables successively. It terminates if an inconsistency was detected or a solution was found (cf. Apt, 2009, Sec. 8.51). Freuder (1982) states some properties of CSPs where backtrack-free search can find all solutions (if one exists).

Limited Discrepancy Search

Limited Discrepancy Search (LDS) is a tree-based search engine to find feasible solutions, too. It acts in a deep-first manner and was presented by Harvey and Ginsberg (1995).

LDS is based on the idea that the variable ordering heuristic is good enough to guide the search process to a feasible solution. That is, following the branching and the variable ordering heuristic should lead to a feasible solution in most cases. In all other cases, deviating from the branching heuristic in a limited number of cases is allowed and should result in a feasible solution. Such a deviation is a *discrepancy*. Harvey and Ginsberg (1995) call a discrepancy a "wrong turn", i.e. a situation where the search cannot choose the *first* or *best* decision according to the branching rule but the second, third,...best alternative. Choosing the second best alternative would mean a discrepancy of 1, choosing the third alternative a discrepancy of 2 and so forth. The search engine takes the maximum number of discrepancies as a parameter that must not be applied on one level of the search tree but may be distributed all over it. The search process traverses the tree iteratively with an increasing number of discrepancies per path (starting with 0 up to the maximum number of discrepancies).

Figure 3.11: Limited Discrepancy Search Tree with Discrepancy Levels 0 (a) and 1 (b–g)

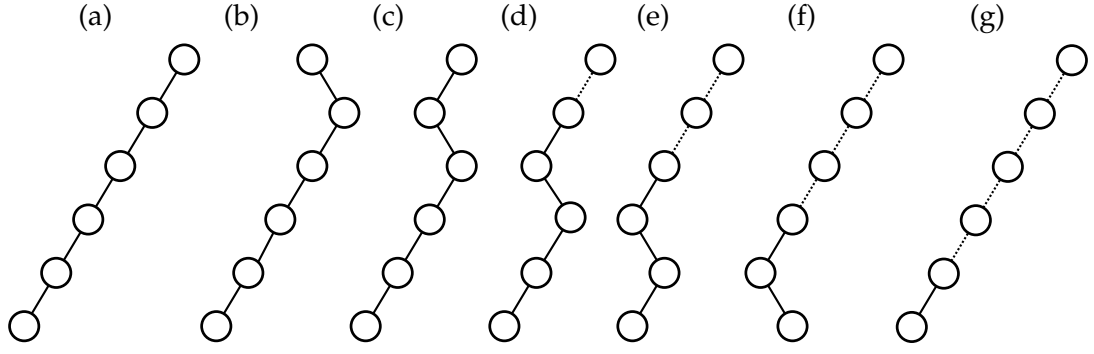


Figure 3.11 illustrates the search process on a binary search tree (only one discrepancy can be caused by a single variable). The branching heuristic prefers left-hand side over right-hand side. In the first iteration (Figure 3.11a) the discrepancy level is zero, the search choose the left-most path. In the second iteration the discrepancy level is increased to one (Figures 3.11b–g).

One can see one "turn" to the right. Finally, the solution from Figure 3.11a is still present in Figure 3.11g because there is only a maximum discrepancy level—not a minimum—the algorithm finds this solution again. The dashed edges in the figure denote the backtracking behavior: No backtracking to those nodes took part since the last picture. Therefore, the solid edges illustrates the search process within the current iteration. It is clear that if the maximum discrepancy level equals the depth of the search tree, the engine performs a full tree search.

Consider the TSP with its associated cost matrix (Table 3.1). Furthermore, assume that the variable selection heuristic remains unchanged—variables are handled in the order of their index—but the value selection heuristic determines that the successor with the smallest cost value is chosen. This should result in a tour with low costs. Indeed, the solution is the shortest tour $0 \rightarrow 1 \rightarrow 3 \rightarrow 5 \rightarrow 4 \rightarrow 2 \rightarrow 0$ with a length of 27 which is an optimal solution. Note that a discrepancy level of 1 would lead to the same unique feasible solution if the value selection heuristic's decision is distinct (there are no further successors with the same cost value).

Furthermore, recall that the result of the search process depends on the quality of the variable ordering heuristic. If no sufficiently good heuristic is available, search will claim the problem to be infeasible (if the number of discrepancies is too small) or consume a lot of time. Besides, the application of LDS is limited to problems with very simple branching decisions. Even the presence of a second variable (with an own branching rule) may cause problems because the discrepancy limit could be exhausted before a solution was found. Consider a more complex variable for time windows: The variable takes the point of time in arrival. If the value selection rule tells the algorithm to select the smallest value in every variables' domain, the number of discrepancies grows very fast even for small deviations in travel time. In summary, this algorithm is not applicable to our problem MDVRPTW-RH as load variables and arrival time variables are present.

3.6.2 Optimization Algorithms

The search engines presented in this chapter seek a best solution for a problem. It is mandatory that the underlying problem has an objective function $obj : Sol \rightarrow \mathbb{R}$ (cf. Section 3.3) that evaluates a solution and to guide the search to the maximum resp. minimum objective function value. We will limit the explanations to minimization problems, but maximization problems can be solved analogously. For our running example, the total travel distance shall be minimized ($\min \sum_{i=0}^{|N|-1} cost_{i,x_i}$) (Section 3.3).

In the following two sections we present an exact method—Branch&Bound Search—and sketch the idea of LS methods that are applied to COP.

Branch&Bound Search

The B&B search engine in CP acts in the same way as the general B&B framework which is based on bounds: Given a minimization problem a *lower bound* is the objective function value of a heuristic but possibly infeasible solution, e.g. a solution for the TSP that contains subtours. Given a lower bound it is clear that another solution with a lower objective than the lower

bound is infeasible, too. The lower bound approximates the optimal solution from below. On the other hand, an upper bound is a feasible solution, i.e. all constraints are fulfilled. Note, that there can be other solutions with better objectives. Therefore, one seeks for the smallest upper bound and rejects solutions with worse upper bounds. In the optimal solution the upper and lower bound are identical. We refer to (Winston, 2005, Sec. 9.3) for a detailed description of the B&B method in OR.

B&B (Algorithm 10) is a tree-based search that uses these bounds to prune the search tree and calls the function BRANCH-AND-BOUND(j,$\mathcal{D}$,$\mathcal{C}$,solution,bound).

To handle the objective function we introduce two additional abbreviations for the pseudo-code:

- `OBJ(I)` takes the current instantiation I and returns a value *obj* based on this instantiation. This function takes a complete instantiation as input parameter and cannot be used to evaluate partial assignments of values to variables. The result is an upper bound.
- `HEUR(I,j,`$\mathcal{D}$`)` returns a lower bound. It evaluates the solution on basis of the sequence $a_0, a_1, \ldots, a_j, dom(x_{j+1}), \ldots, dom(x_{n-1})$. It can be seen that only a partial assignment has been made so far. While we have exact values for variables $x_0, \ldots, x_j$, for variables $x_{j+1}, \ldots, x_{n-1}$ an approximation must be found, e.g. the minimum value in the domain. This solution is a lower bound.

Algorithm 10: B&B Search

```
1 bound := ∞
2 solution := NULL
3 if PROPAGATE(0,D) == FALSE then
4 |   BRANCH-AND-BOUND(1,D,C,solution,bound)
```

Function BRANCH-AND-BOUND(j,$\mathcal{D}$,$\mathcal{C}$,solution,bound)

```
1  while dom(x_j) ≠ ∅ do
2  |   Pop a from dom(x_j)
3  |   if Instantiation of x_j = a is consistent with all former instantiations then
4  |   |   x_j := a
5  |   |   if j == n then
6  |   |   |   if obj <bound then
7  |   |   |   |   bound := obj
8  |   |   |   |   solution := I
9  |   |   |   |   C := C ∪ {OBJ(I) < bound}
10 |   |   else
11 |   |   |   if PROPAGATE(0,D) == FALSE then
12 |   |   |   |   if HEUR(I,j,d) <bound then
13 |   |   |   |   |   BRANCH-AND-BOUND(j+1,D,C,solution,bound)
```

In the root node the initial upper bound is set to ∞ as no feasible solution has been found yet (line 1). The algorithm executes the propagation procedure (with an empty instantiation)

to delete inconsistent variables from the domains and to check if the problem is feasible at all (line 3). If so, it calls the function BRANCH-AND-BOUND(j,$\mathcal{D}$,$\mathcal{C}$,solution,bound) for the next assignment.

The procedure picks a value from the domain of x_j and checks if it is a consistent instantiation. If the instantiation is incosistent, the procedure precedes with the next value. Otherwise, it evaluates the assignment: If the number of assigned variables equals the number of total variables, the subprocedure will be called to get an upper bound. If this is better than the so far best-known bound, the best solution is updated and a constraint is added to the global constraint set, that all solutions must be better (less) than the current value of *bound* (line 9). For incomplete assignments the procedure executes the propagation algorithm and evaluates the solution heuristically. Only if this evaluation is better than the value of *bound* BRANCH-AND-BOUND(j,$\mathcal{D}$,$\mathcal{C}$,solution,bound) is called recursively to explore this part of the subtree. Otherwise, the next value of $dom(x_j)$ will be explored.

If there exists a solution, the B&B algorithm will return the solution with the best value for *obj* (depending on the aim of the objective function). Otherwise, the algorithm reports that no solution exists.

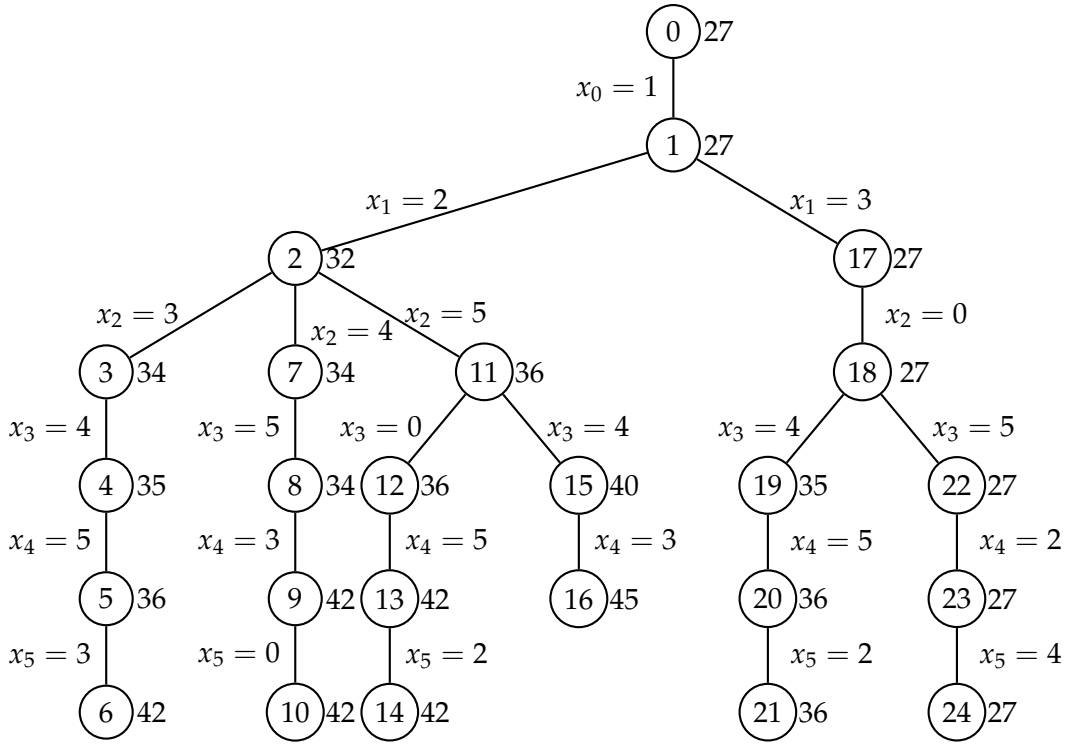
For our TSP example we can set the bounds in the root node as follows: The maximum objective would be the longest tour, i.e. the vehicle has to go to the farthest successor in every node. Therefore the upper bound can be computed by summing up the maximum element of every line in the cots matrix: $UB_0 = \sum_i \max_j(c_{ij})$ (Note that this solution may be infeasible, but is a better approximation than ∞). The lower bound can be computed analogously: It is the sum of the minimum element in every line of the cost matrix: $LB_0 = \sum_i \min_j(c_{ij})$. A shorter tour cannot be found if every node must have a successor. Figure 3.12 on the next page illustrates the B&B search tree. Variables are assigned in the order of their index and the smallest element in every domain is assigned first (after constraint propagation was applied). The number to the right of the node is the current value of the lower bound, the node number describes the traversal of the tree. The first solution was detected in node 6, in node the heuristic evaluation returns a value that is larger than the bound, this part of the tree will not be further explored. In node 25 the solution of 27 equals the upper bound. Thus, the optimal solution was found and B&B search aborts. Note that we have chosen a naive value selection heuristic to demonstrate all aspects of a B&B tree, the selection of the node with least cost as successor would result in a tree identical to the LDS tree in Figure 3.11, i.e. a tree without backtracking.

Details of B&B for CP can be found in (Apt, 2009, Ch. 8.7) and (Ghédira, 2013, Ch. 7.2.1).

Local Search Methods

Generally, Local Search methods start from a solution and try to find a solution in their neighborhood with a better objective function value. A solution in the neighborhood is a solution that is close to the origin solution in some sense. A neighborhood again is defined by a neighborhood function that returns a number of solutions. A very basic example is the *Iterated Local Search*: Every time a better solution is found it replaces the original solution and search continues in the neighborhood. If no improving neighbor can be identified, the current solution is *locally optimal*.

Figure 3.12: BAB Tree for the TSP Example



The quality of the solution highly depends on the efficiency of the neighborhood function that has to fit the problem's characteristics. For the TSP an example is the function by Lin and Kernighan (1973): Neighboring solutions are all solutions that can be yielded from the current solution by deleting k edges of the graph G and replacing them by k other edges that build a Hamiltonian cycle.

However, this simple version has the drawback that it may get stuck in (poor) local optima. Therefore, the search must be broadened. A straightforward approach is to use Multi-Start Local Search that explores the search space starting from different initial solutions or Variable Neighborhood Search (Hansen and Mladenović, 2001; Hansen et al., 2010) that changes neighborhoods during search. Very Large-Scale Neighborhood Search (Ahuja et al., 2002) applies LS principles to neighborhoods of exponential size. Finally, Guided Local Search (Voudouris et al., 2010) works with penalty terms in the objective function whose evaluation depends on the fact if a certain feature is present in the solution. To escape from a local optimum features that matter most in the current solution are penalized to guide the search away from this solution, other features may have a lower weight. Therefore, search will move into an area where only low penalties are added to the objective.

One thing is characteristic for neighborhood search algorithms in CP: Adding more constraints to a problem improves the performance of the search process concerning the objective function. This is paradox as an *underconstrained* problem has more solutions and feasible solutions can be found easily (Clark et al., 1996). However, a problem with more constraints—involving fewer solutions—has less local optima, too. Therefore, the search space can be exploited more efficiently (Yokoo, 1997).

Beside the standard LS there is a wide range of search engines, which are based on analogies with processes in natural sciences. These search procedures can solve COPs but the application is highly problem-dependent. Examples are SA, TS, GA, and ACO. Hentenryck and Michel (2005) explain the ability of these methods to solve COPs. A textbook on the solution of CSP using ACO is Solnon (2010). Backer et al. (1997) and Backer and Furnon (1999) present a framework for TS to solve CSPs and demonstrate it on the VRP. A review on LS in CP can be found in Hoos and Tsang (2006), more general explanations in Pisinger and Ropke (2010).

For our running example we will illustrate the idea of constraint-based Large Neighborhood Search (LNS) (Shaw, 1998): Consider a feasible solution of the TSP. In this solution every node has its successor. The search process now allows a number of these nodes to search for a new successor. Formally, some assignments are frozen and cannot be changed, but some nodes can be re-assigned. In Table 3.3 frozen nodes are marked by an *X*. Because the degree of freedom is quite small (only a limited number of nodes has been relaxed) B&B search can solve this problem to optimality. In many cases the structure of constraints assigns some or even all nodes to their old value. An exchange is accepted if it improves the objective. Frozen nodes are selected at random. Note that this behavior is the same as suggested by Lin and Kernighan (1973): Freeing a successor is analog to deleting an arc, both methods relax the connection between to nodes. More promising than this naive approach to select all nodes randomly is to regard relations between nodes, e.g. to incorporate the distance between two nodes. For, it is more likely that customers that are close to each other exchange their position than customer that are widely scattered. We will provide more details in Section 5.2.3.

Table 3.3: Example of Constrained-Based LNS

	Node	0	1	2	3	4	5	Objective
Initial Solution	Successor	2	0	5	4	1	3	42
1^{st} Iteration	Frozen?		X			X		—
	New Successor	3	0	4	5	1	2	44
2^{nd} Iteration	Frozen?				X	X		—
	New Successor	2	0	5	4	1	3	40

The example given in Table 3.3 shows that in the first iteration the objective function value was not improved (it is even worse than the initial soultion), while it gets better in the second iteration. As a consequence, the neighborhood solutions for all iterations in this example are derived from the first solution. The solution from the first iteration will be rejected while the second one will be accepted.

3.7 Constraint Programming and Vehicle Routing Problems

In this section we focus on applications of CP on VRPs. It turns out that CP is often used only to solve parts of the problem because it is not as efficient as state-of-the-art OR techniques, but allows to add constraints easily and model complex constraints from real-world situations (Kilby et al., 2000; Kilby and Shaw, 2006).

A first solution approach for the TSP using CP is introduced by Caseau and Laburthe (1997). They provide a global constraint to avoid subtours (Section 3.5) and a specific branching rule. Pesant et al. (1998) present a framework to solve the Traveling Salesman Problem with Time Windows (TSPTW) based on cost-based propagation techniques. The CP model is linked with the relaxation of an assignment problem to achieve reduced costs and bounds. These information are used for propagation. In a second step, cutting planes are added to tighten the relaxation of the assignment problem. Pesant et al. (1999) show that CP provides flexibility to adapt the model to a TSP with multiple time windows. Focacci et al. (1999b) use the TSP to demonstrate a global constraint that takes into account information from the objective function. While earlier generations of global constraints performed pruning only on information on the variables this constraint prunes values regarding dual values and bound from the B&B search process.

In Shaw (1998) a LNS is used to solve the VRP with and without Time Windows. Nodes are chosen according to the fact, how *related* they are. These nodes are removed from their current position and re-inserted during the optimization process. The best insertion position is identified by a B&B algorithm with CP elements. Caseau and Laburthe (1999) present a heuristic approach which is able to solve large-scale VRP with side constraints. It is based on an insertion heuristic and local optimization. In the case of the VRPTW a CP solver is used to improve the individual tours represented by a TSPTW. Backer et al. (2000) present a LS heuristic, where only the feasibility check for solutions is made through CP. It is implemented as an active and passive representation. In the former one, new solutions are created while the latter one checks the feasibility of the constrained variables. A CG approach to solve the VRPTW is provided by Rousseau et al. (2002a). The subproblem is solved by a CP model, where additional implicit constraints are added to reduce the size of the search tree. The solution time is worse than in pure OR methods. Rousseau et al. (2002b) combine operators for LNS and CP using backtracking and optimization techniques of CP.

Focacci et al. (2002) extend the global constraint *path* to take the costs of the tour into account to strengthen the relationship between the constraints and the objective function, which is weak in general. They incorporate OR techniques into the global constraint to improve the propagation from a cost-based point-of-view. Rousseau et al. (2004) present a CG approach to solve the VRPTW. In this B&P approach the subproblem of finding a negative reduced cost path is solved by a CP model. Two formulations for the Chinese Postman Problem with Time Windows are presented by Aminu and Eglese (2006): A basic formulation and a second one, which is transformed into a VRP. A CP version of the TSPTW is also applied in the post-optimization phase. Cruz-Chavez et al. (2007) adapt a search procedure that can solve the job shop deadline scheduling problem to handle the VRPTW. A GA that optimizes long-distance transportation in a vertical cooperation was proposed by Onoyama et al. (2008). The algorithm uses a pre-checking procedure that uses propagation techniques.

Finally, Berbeglia et al. (2012) develop an algorithm based on CP and TS for the Dynamic Dial-a-Ride Problem. To decide whether a request is accepted, TS and the CP algorithm are executed in parallel. It is expected that in a tight problem CP proves if a tour is feasible or not, while TS can return a feasible solution in problems with few constraints. A request is only accepted if neither the problem is infeasible nor no solution could be identified after a given

time.

Chapter 4

A Constraint Programming-Based Genetic Algorithm

We have shown in Section 2.4 that *cluster first, route second* approaches were often applied to the MDVRP and seen that GA were successfully used to solve this problem. Therefore, we will combine these approaches in this chapter and develop a heuristic approach that makes use of both principles. The first level of the algorithm assigns every customer to a specific depot and a single vehicle. This is done by a GA whose population is built of different assignments of customers to vehicles respectively depots. The second level makes the routing decisions for the single vehicles through their customer set whereby the individual travel distance should be minimized. A disadvantage of this double-stage design is that an assignment made in the first step, can be infeasible in the second step because of the time windows or the capacity required. On the other hand it is advantageous that there are smaller routing problems at the second stage. An earlier version of this algorithm has been published by Kimms and Reiners (2013).

GAs are common knowledge from the field of OR and the main principles are not modified in this thesis. However, one important operation—the computation of the fitness value (to evaluate the quality of a solution)—is done by CP methods. In this sense, this algorithm belongs to the class of *matheuristics* that combines exact mathematical methods with (meta)heuristic principles (Caserta and Voß, 2009, p. 19). Note that this definition is not exact because of the fact that CP is not a mathematical programming method in a narrow sense but provides optimal solutions which seems the main criteria for the characterization in our opinion. During the solution process the algorithm alternates clustering and optimization. Furthermore, the initial population that is necessary for this kind of algorithm is computed by CP. To the best of our knowledge this is the first time GA and CP are combined in that way to solve a problem.

The outline of this chapter is as follows: In Section 4.1 we will summarize the foundations of GAs in general. Afterwards, in Section 4.2 we will give a broader review on literature that is specific for this chapter, i.e. present a survey on GAs for solving VRPTW in general and not only the multi-depot variant (as done in Section 2.4.2). In Section 4.3 we will present the single elements of our GA and sketch the superior process . In particular, we will model a CSP and a COP that will be used within the overall algorithm. This version of the algorithm can only deal with static planning scenarios yet, the rolling horizon framework will be introduced in

Section 4.4. Finally, in Section 4.5 the result of the computational experiments for the static and dynamic setting are presented.

4.1 Fundamentals

The fundamental idea of GAs relies on the principle of evolution in nature. Generally, the genesis of GAs is ascribed to Holland (1975), but there are other sources that deal with the same idea and have been published earlier or nearly at the same time (Rechenberg, 1973; Schwefel, 1977). The description of the GA is made in a biological terminology: The fundamental element is an *individual* that represents a single solution and has a *fitness value*. This value measures the quality of an individual. Solutions of a higher quality (measured by some objective function) have better fitness values. The entirety of all individuals is the *population*, while an encoded individual is denoted as *chromosome*. Each chromosome consists of several *genes* as its basic parts. The value of a gene is named *allele* and its position in a string is called *locus*.

In contrast to other evolutionary algorithms, like Evolutionary Programming, GAs emphasize the aspect of reproduction, which describes the process where two individuals exchange parts of their genes to produce an offspring with a better fitness value (*crossover*). Additionally, the concept of mutation is applied. Herein, a fixed number of genes is modified randomly.

A basic GA consists of the following steps: First of all, an initial population is generated. Out of this population some individuals are selected for reproduction. Afterwards, mutation is performed. Finally, some of the individuals of the whole population (aggregation of offspring and the "old" population) are selected to build the basis for the next iteration. The solution of the algorithm is the best individual in the population at the moment when a termination criterion is met.

Genetic Algorithms are stochastic algorithms because the chances of selection, reproduction and mutation depend on random variables.

4.2 Literature Review

There exists a vast amount of literature that deals with VRPTW. An overview can be found in Bräysy and Gendreau (2005a,b). We therefore focus on the VRPTW solved by Genetic Algorithms.

A GA for the VRPTW was first presented by Thangiah et al. (1991, 1993). In *GIDEON* they use the idea of evolution in the first step of a *cluster first, route second* approach. The customers are divided into clusters by genetic search first and routed to serve the nodes in their cluster afterwards. Berger et al. (1998) create crossover and mutation operators for the VRPTW taking knowledge of the problem structure and expected solutions into account. In the recombination step customers are rescheduled using route construction heuristics and the mutation operator reduces the number of short tours.

Most of the GAs developed to solve the VRPTW are hybridized with other heuristic approaches: In Kopfer et al. (1994) the search process of the GA is improved by combining it with

the idea of the *sweep* resp. *savings* approach (cf. Section 2.4.2). Potvin and Bengio (1996) also use a GA in a *cluster first, route second* approach in the former step, while routing is made through a greedy heuristic. In Jung and Moon (2002) the offspring during the evolutionary process are improved by a local optimization heuristics. In a similar way Ho et al. (2001) apply TS to the new individuals created in a GA. The intensity of this improvement step depends on the progress of the GA: In later generations the improvement step through TS is more often executed than in early generations. The hybrid GA by Vidal et al. (2013b) contains an *education* and *repair* phase to improve the offspring solutions and fix infeasible solutions based on neighborhood search. Chiang and Hsu (2014) developed a GA that can deal with two objectives—minimizing the number of required vehicles and minimizing the total travel distance—simultaneously.

A pure GA for the VRPTW is presented by Blanton and Wainwright (1993) and develops specific crossover operators. Another variant is introduced by Zhu (2000), where chromosomes describe the sequence of customer visits for all nodes. Baker and Ayechew (2003) describe another GA for VRP. This straightforward approach swaps customers between vehicles. Through the usage of TS promising results are achieved. A GA working with two populations is developed by Berger et al. (2003): The first population aims to minimize the total travel distance. The second population tries to minimize the violated time window constraints and construct feasible solutions. If a new best feasible solution appears in the second population, the first population is replaced by the second one. Tan et al. (2001) describe a Messy Genetic Algorithm for the VRPTW. In this algorithm the genes in a chromosome are presented by a tuple of *locus* and *value*. This allows redundant and missing genes and enables the algorithm to escape from local optima. In a GA to solve a static VRPTW by Cheng and Wang (2009) the customers are first clustered and afterwards the vehicles are routed in a TSP by a problem-specific heuristic.

Hanshar and Ombuki-Berman (2007) present the solution approach most similar to our framework. The orders arrive over time as well and the dynamic VRP is solved as a sequence of static VRPs. New orders from the previous period are handled by an *Event Scheduler*, which forwards static VRP instances to a Genetic Algorithm at the end of each period. During the optimization process the customers are committed to a current vehicle and fixed for the following periods. The *Event Scheduler* also keeps track of the vehicles' capacity restrictions and calls them back to the depot, if required. Alvarenga et al. (2007) provide a set partitioning model for the VRPTW where the set of potential routes is computed by a GA procedure. The authors use some relations between the global and local optima to guide the search process.

4.3 Description of the Genetic Algorithm

In this section we describe the chromosome encoding and the operators that are used within the GA. Finally, the complete algorithm is presented.

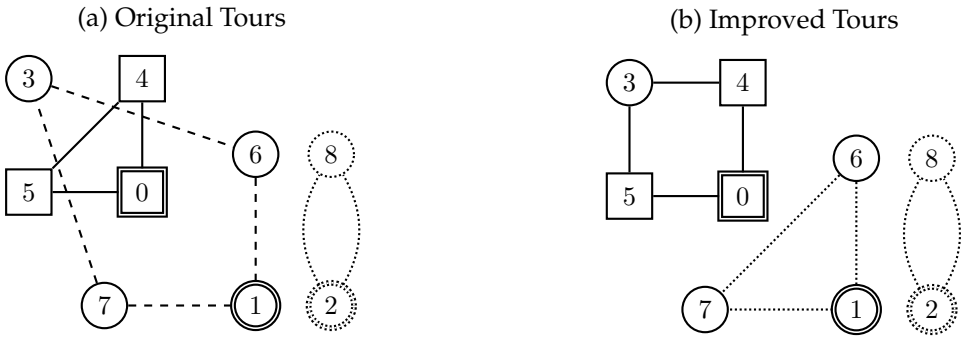
4.3.1 Chromosome Encoding

In our problem setting the depots are able to communicate with each other to ask vehicles of other depots for fulfilling an order or demand. This mechanism—and the possibility to swap

orders between vehicles—enables carriers to improve the service level without traveling long distances because another vehicle from a partner's depot could be closer to the actual demand node than all own vehicles. We neglect aspects of competition on customers at this stage in our model and assume all carriers to share any information on demand. As a result, we are in a situation where a central decision maker that has full information on all vehicles' routes and customers' demand. We will reflect and evaluate these simplifying assumptions in Chapter 6.

Figure 4.1 illustrates the advantage of the exchange process: Rectangles represent orders from customers of depot 0, the circles stand for those of depot 1. The solid edges represent the route of vehicle 0, the dashed lines the tour of vehicle 1. Serving node 3 means a detour for vehicles from depot 1 (Figure 4.1a), while by transferring node 3 to depot 0 the total travel distance decreases (Figure 4.1b). The dashed circles stand for a third tour (starting at depot 2).

Figure 4.1: Vehicle Routes Before (a) and After (b) Swapping Customer 3 between Depots



The proposed GA does the clustering in the *cluster first, route second* approach: It swaps demand nodes between different vehicles. Each gene of a chromosome represents a single node, the value of the gene gives the number of the vehicle serving it. Figure 4.2 demonstrates the design: In the first line the index numbers of the customers are given. The boxes below show two different chromosomes C_0 and C_1 that encode Figure 4.1a and 4.1b. In C_0 node 3 is served by vehicle 1, node 4 is visited by vehicle 0 and so forth. Notice that nodes 0–2 are the depot nodes. The given representation has the advantage that every node will be visited exactly once and this condition is valid for every solution created by the GA (Assumption 1 in Section 2.1).

Figure 4.2: Example of Two Chromosomes

	0	1	2	3	4	5	6	7	8
C_0	0	1	2	1	0	0	1	1	2
C_1	0	1	2	0	0	0	1	1	2

4.3.2 Operators

Initialization

To generate an initial population for the GA we use the CSP (4.1)–(4.12), which does not have any objective function. It only searches for feasible solutions on the given constraints, but does

not evaluate the different solutions by a cost term. This is advantageous because we need a large and diverse population to start the algorithm. Evaluating the solution would guide the search process in a current direction and the solutions would be similar. Simply searching for feasible solutions will allow more diverse routes. This aspect is emphasized by the fact that the value selection heuristic selects values randomly.

The model to create the initial population was inspired by the Open Vehicle Routing Problem (Sariklis and Powell, 2000) and builds a giant tour that contains all customer nodes and depots. To deal with different vehicles there is a unique depot for every vehicle which is a duplicate of the origin depot, i.e. the number of depots equals the number of vehicles. Duplicated depots are included in the set D, too. The depot are inserted between the customers and every time the path crosses a depot the vehicle is changed, i.e. the current vehicle ends its tour and a further one starts a new trip. Every time a change occurs, the arrival time (start time) at the depot and its load are set to 0 because a new vehicle starts at its depot. In this way the CSP generates solutions that are feasible according to time windows and loads. However, the Open VRP does not enforce that vehicles return to the depot; a route can finish at any node. Therefore, some tours could be infeasible according to the arrival time at the own depot. This would not be detected by the model for initial solutions, but turn out as an infeasible solution in the second step. However, the algorithm is designed for the rolling horizon setting and only a fraction of the entire customer set will be considered at the beginning. Thus, sufficient load capacity and time should be available as more customers must be added to the tours. Furthermore, we assume that restrictions on load and tour duration are more important than the opening hours of the depots. If there is a deadline for arriving at the depot for some reason (e.g. the depot is part of a hub&spoke in express mail) the problem can be modeled with start and end depots where the travel time between these nodes is 0.

The CSP to generate an initial population that can be used as input for the GA was modeled as follows:

$$PATH(1, \mathcal{X}) \tag{4.1}$$

$$ALLDIFFERENT(\mathcal{X}) \tag{4.2}$$

$$INT\_VALUE\_PRECEDE\_CHAIN(depots, \mathcal{X}) \tag{4.3}$$

$$x_i \neq i \qquad i \in N \tag{4.4}$$

$$t_i = 0 \qquad i \in D \tag{4.5}$$

$$x_i = j \implies t_j = t_i + c_{ij} + s_i \qquad i \in N, j \in C \tag{4.6}$$

$$e_i \leq t_i \leq l_i \qquad i \in N \tag{4.7}$$

$$u_i = 0 \qquad i \in D \tag{4.8}$$

$$x_i = j \implies u_j = u_i + d_j \qquad i \in N, j \in C \tag{4.9}$$

$$u_i \leq Q \qquad i \in N \tag{4.10}$$

$$\tau_i = i \qquad i \in D \tag{4.11}$$

$$x_i = j \implies \tau_j = \tau_i \qquad i \in N, j \in C \tag{4.12}$$

The decision variable x_i gives the successor of every customer and depot node $i \in N$; the variables $x_i, i \in N$ are summarized in the set $\mathcal{X}$. The initial domain is $x_i \in \{0, \ldots, n-1\}$,

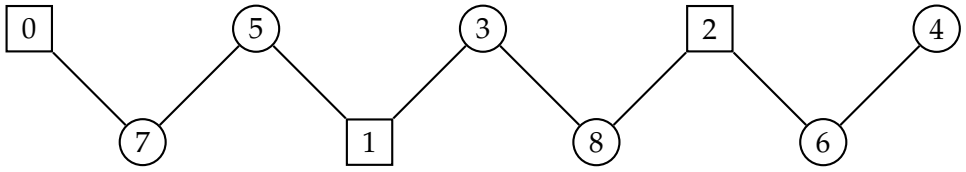
where n is the number of nodes. For example, if a vehicle goes directly from node 4 to node 2, the result would be presented as $x_4 = 2$. Therefore, the domain of x_i in every node i consists of all other depot and customer nodes. The variable t_i saves the arrival time (begin of service, more precisely) of a vehicle at every customer node i and has a (global) initial domain of $[0, \ldots, \max_i l_i]$, that is from the start of the planning horizon to the latest allowed arrival time of all nodes. The set $\mathcal{T}$ contains all arrival time variables $t_i, i \in N$. The load is controlled by the variable u_i that is the used capacity at arrival in node $i \in N$ and ranges from an empty vehicle to the maximum capacity, that is $u_i \in \{0, \ldots, Q\}, i \in N$. Finally, τ_i keeps the number of the vehicle that services customer node i. Its initial domain is $\tau_i \in \{0, \ldots, (d-1)\}$ where d is the number of depots $|D|$.

Constraint (4.1) forces the variables to form a Hamiltonian path (the first argument states the number of paths that shall cover all nodes) (Assumption 1 in Section 2.1). We assume that the path starts in depot 0. Because no objective function is present in the model the total length of the tour is not important. Therefore the run back to the "home depot" can be replaced by a trip to another depot. The time windows at the depot nodes are sufficient to arrive for every vehicle in time. The path finishes at a customer node, the way back to the depot is disregarded.

Constraint (4.2) enforces all variables $x_i \in \mathcal{X}$ to take distinct values (Section 3.5). This constraint is not mandatory because Constraint (4.1) ensures the same, but the redundant constraint is added to allow additional pruning steps: We have shown in Section 3.5 that both global constraints use different propagation algorithms and therefore the combination of both constraints can detect more inconsistent values in the domains. Constraint (4.3) fixes the sequence of depots. The ordered set *depot* contains all depot nodes. In general, if d denotes the number of items of the set *depots*, the following condition will be valid for every $i \in \{0, \ldots, d-2\}$: The first occurrence of the $(i+1)^{th}$ value of *depots* should be preceded by the first occurrence of the i^{th} value. Propagation results in hyper-arc consistency. Here, the constraint states that depot 0 has to precede depot 2 in a feasible solution, e.g. the variable's index i of the variable that takes value 1 has to be smaller than the index of the variable which takes the value 2. Given an example where the set of depots is $D = \{1, 2, 3\}$ and the set of customers is $C = \{4, 5, 6, 7, 8\}$, the tour *1*-4-5-*2*-6-7-*3*-8 would be a feasible tour, whereas *2*-4-5-*1*-6-7-*3*-8 is infeasible because depot 2 would be visited before depot 1. We use this constraint for reasons of symmetry breaking. Otherwise it would be possible to build a new solution by just swapping the depots with the same coordinates between the routes. If one would evaluate the solutions, it would yield to an identical tour length. However, this would result in very similar members of the population and the algorithm would perform poorly.

Constraint (4.4) ensures that a node cannot precede itself. Constraint (4.6) keeps track of the arrival time t_j of the vehicle at each node $j \in C$. It is a logical implication, e.g. if the left-hand side of the constraint is true the right hand-side has to be true as well. The left-hand side checks if a vehicle visits node j immediately after it leaves node i. The right hand side adds the travel time from node i to node j c_{ij} and the service time s_i at node i to the arrival time at node i. The start time at the depot nodes $i = 0, \ldots, (d-1)$ is set to the beginning of the planning horizon $(t_i = 0, i = 0, \ldots, (n-1))$ by Constraint (4.5). Constraint (4.7) ensures that time windows with earliest ready time e_i and latest due date l_i in all nodes $i \in N$ are kept (Assumption 5). Using an *ELEMENT(x,z,$\mathcal{X}$)* constraint $t_{x_i} = t_i + c_{i,x_i} + s_i, i \in N$ is not possible here because there would

Figure 4.3: Example of a Feasible Initial Solution



x_i	7	5	1	3	8	2	6	4	—
τ_i	0	0	0	1	1	1	2	2	2
t_i	0	3	5	0	2	4	0	4	6

be no way to distinguish between customer and depot nodes.

Constraint (4.9) updates the current load of all vehicles. If there is a connection between two customers, the load of the vehicle leaving node j equals the sum of the capacity after leaving the previous node i and the demand of node j. Constraint (4.8) resets the load in the depot nodes and Constraint (4.10) ensures that the capacity of each vehicle is not exceeded (Assumption 3). The latter constraint is redundant: The domain of u_i already limits the maximum capacity.

Finally, Constraint (4.12) ensures that a node and its successor are served by the same vehicle. It is a logical implication, e.g. if the left-hand side of the constraint is true the right hand-side has to be true as well. The idea behind this constraint is similar again. If one node is an immediate successor of the other, it will be served by the same vehicle if none of the nodes is a depot. Nodes following a depot get the same τ_i value as the previous depot, connections between customer nodes and depots are not considered. If node i is a depot, Constraint (4.11) sets the vehicle number τ_i to the depot number i.

Figure 4.3 illustrates an example of a feasible solution: Customer nodes are represented by circles and depots are drawn as rectangles. The variable x_i is the number of the node which is the successor of i, while τ_i presents the number of the vehicle which serves node i. Every time the vehicle passes a depot node τ_i is updated to its number. t_i is also reset to 0, because in fact a new tour is started by a new vehicle from the point-of-view of a VRPTW.

The search process is a depth-first search that branches randomly on unassigned τ_i variables and assigns random values from their domains to them (and forbids them in the other child node).

Evaluation

At the start of the algorithm the initial population is evaluated, in later iterations the descendants' fitness value must be computed.

An individual is generally evaluated by solving a Constraint Optimization Problem. Before this step starts, some preparation of the chromosome is necessary: The chromosome structure in our algorithm is taken from Baker and Ayechew (2003) and consists of the vehicle numbers which serve a node. In a first step, the algorithm filters the nodes which are served by one vehicle. If the chromosome contains the genes $\langle 0, 1, 2, 1, 2, 0, 1, 1, 2 \rangle$, it concludes that vehicle 0 serves nodes 0 and 5, vehicle 1 serves customers 1, 3, 6, and 7, and vehicle 2 serves customers 2, 4, and 8. For each vehicle the shortest cycle through the tour's nodes starting and ending at

the own depot is computed (in the standard case) by the CP solver with the TSP considering all time windows and the total tour length. The results of all vehicles summed up give the global objective function value (total travel time). In the GA these values represent the fitness of the individual. To speed up computation and avoid calling a solver for trivial problems, three cases are distinguished according to the number of nodes q in a tour. In the special cases $q = 1$ and $q = 2$ only simple calculations are required while in the standard case of $q > 2$ optimization is needed:

- $q = 1$: These tours contain only the depot node and have an objective function value of 0. The vehicle will remain in the depot. Time windows must not be considered.
- $q = 2$: If a tour contains two nodes, one depot and one customer, the total duration is twice the time between these nodes. The time window restriction is kept because of the design of the test instances.
- $q > 2$: For tours consisting of three or more nodes, we need optimization to determine the shortest tour. However, before the CP solver is called, the algorithm computes the total demand and compares it to the vehicle's capacity. Only if this test is passed, the COP described below will be solved considering the time window restrictions. Otherwise the repair procedure is called immediately.

To evaluate a tour, an objective function has to be included in the CP model to compute its length. To find a single tour with one vehicle we extended the TSP presented in Pesant et al. (1998) with the same notation and initial domains as above. Additionally, the predecessor of node i is given by the decision variable y_i, all $y_i \in N$ are summarized in $\mathcal{Y}$. Their initial domains are the same as for the variable x_i.

$$\min \sum_{i \in N} c_{i,x_i} \tag{4.13}$$

s. t.

$$TOUR(\mathcal{X}) \tag{4.14}$$

$$ALLDIFFERENT(\mathcal{X}) \tag{4.15}$$

$$x_i \neq i \qquad i \in N \tag{4.16}$$

$$t_0 = 0 \tag{4.17}$$

$$t_{x_i} = t_i + c_{i,x_i} + s_i \qquad i \in N \tag{4.18}$$

$$e_i \leq t_i \leq l_i \qquad i \in N \tag{4.19}$$

$$x_i = j \iff y_j = i \qquad i, j \in N \tag{4.20}$$

$$REDUCE\_TIME\_WINDOWS(\mathcal{X}, \mathcal{Y}, \mathcal{T}) \tag{4.21}$$

The objective function (4.13) minimizes the duration of the tour (Assumption 10 from Section 2.1). This is the sum of travel times between every node i and its successor. The variables x_i are enforced to build a Hamiltonian cycle by Constraint (4.14) (Assumption 9). Constraint (4.15) ensures that every node is visited exactly once (Assumption 1 and 4) and Constraint (4.16)

forbids a variable to be the successor of itself. Constraints (4.17)–(4.19) are analog to (4.5)–(4.7) (Assumptions 5 and 6). However, this time we can replace the implication by *ELEMENT(y,z,$\mathcal{X}$)*; Constraint (4.18) has a variable as an index which allows a more intuitive readability of the model. There are no constraints (Assumption 3) on the load in this problem because the tour was checked for feasibility according to capacity limitations before. This preprocessing step saves computation time.

This model also contains two redundant constraints: The *TOUR($\mathcal{X}$)* constraint (4.14) already ensures that all values are pairwise distinct as enforced by Constraint (4.15). Again, the combination improves the interaction between global constraints and using different pruning algorithms strengthens the propagation process which shrinks the search space, but they are inconclusively needed.

Constraint (4.20) establishes a connection between the successor and the predecessor variable: If there is the connection $i \rightarrow j$, j is the successor of i ($x_i = j$) and i is the predecessor of j ($y_j = i$). We use this for the global constraint *REDUCE_TIME_WINDOWS($\mathcal{X}$,$\mathcal{Y}$,$\mathcal{T}$)* to shrink the domains of the arrival time variables. It was introduced by (Pesant et al., 1998) and implements rules that were introduced by Langevin et al. (1993). Propagation works as follows:

For every node i there is a set of nodes that has to be serviced before ($\mathcal{B}_i$) or after ($\mathcal{A}_i$) the current node i because of the time window restriction:

$$\mathcal{A}_i = \{k \in dom(x_i) | t_k^{\min} + c_{ki} + s_k > t_i^{\max}\} \tag{4.22}$$

The set $\mathcal{A}_i$ contains all nodes that can only be serviced after node i (Equation 4.22). It takes the earliest time a service can start in node k and adds the service duration in node k and the travel distance between node k and i to get the earliest arrival time in node k. If that point in time is later than the latest allowed arrival time in node i (taken from $dom(t_i)$), node k can only be serviced after node i. Figure 4.4a on the following page demonstrates the situation (service times at and travel distances between all customers are assumed to be equal). Consider node $i = 0$, customer 1 is obviously not included in the set $\mathcal{A}_0$. Customer 2 is clearly included in the set $\mathcal{A}_0$. The same is true for customer 3—even if service at customer 3 starts as early as possible there is not sufficient time to travel from customer 3 to 0 and start the service within the time window of customer 0 (dashed arcs).

$$\mathcal{B}_i = \{k \in dom(y_i) | t_i^{\min} + c_{ik} + s_i > t_k^{\max}\} \tag{4.23}$$

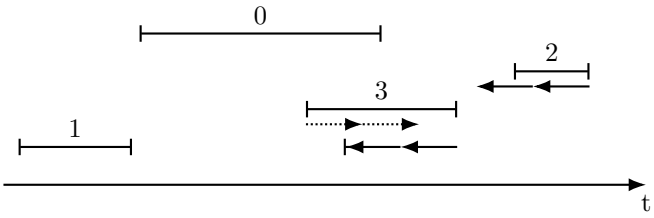
Equation (4.23) works analogously for set $\mathcal{B}_i$: If one starts at node i at the earliest point in time (the minimum element in $dom(t_i)$) with the service and leaves immediately to node k, but cannot arrive within the time window, then the node k must be visited before node i. This case is presented in Figure 4.4b: Customers 1 and 2 are inserted in set $\mathcal{B}_0$, because there is no chance to service customer 0 first, travel to customer 1 and start the service within the time window (dashed arcs).

By using these sets the size of the time variables' domains can be reduced as follows:

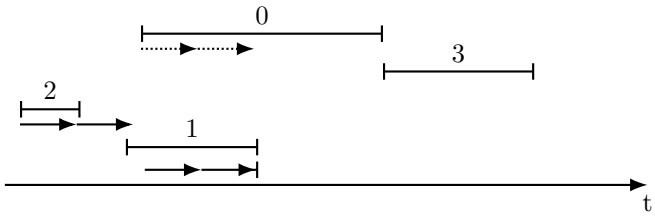
$$t_i \leq \min_{k \in \mathcal{A}_i} (t_k^{max} - c_{ik} - s_k) \qquad i \in V \tag{4.24}$$

Figure 4.4: Example of the *REDUCE_TIME_WINDOWS($\mathcal{X}$,$\mathcal{Y}$,$\mathcal{T}$)* Constraint

(a) Pruning the End of Time Windows



(b) Pruning the Begin of Time Windows



$$t_i \geq \max_{k \in \mathcal{B}_i} (t_k^{min} + c_{ki} + s_k) \qquad i \in V \tag{4.25}$$

In Equation (4.24) we consider all nodes in the set $\mathcal{A}_i$ that can only be visited later in the tour than i: The solid arcs in Figure 4.4a demonstrate the pruning. The service at customer 3 must start not later than t_3^{max}, but the service at customer 0 must be done previously and the vehicle must travel between both customers. Thus, the upper bound of $dom(t_0)$ will be strengthened. In an analog way Equation (4.25) updates the lower bound of $dom(t_0)$ as depicted in Figure 4.4b.

This propagator is executed before branching if the upper or lower bound of any node is changed and achieves bounds consistency.

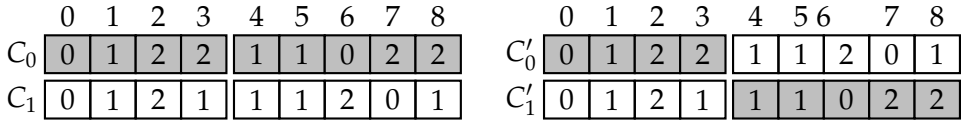
To solve the COP (4.13)–(4.21) we use B&B search. Because the aim is to minimize costs, the search process branches on the individual cost variables first: It selects the variable by its regret, i.e. the variable where the difference between the largest ("most expensive") and the second largest in the domain is largest. In one child node the values in the domain must be smaller than the mean of the currently smallest and the largest value in the domain (most expensive values are excluded) respectively larger in the second child node. Afterwards we branch on the successor variable. The variable with the smallest element in its domain is selected and assigned to this value in one child node and this value is forbidden in the other child node.

Selection

For selection of candidates for the reproduction operator the roulette wheel method is applied (Goldberg, 1989, Ch. 3). Analogous to a real roulette game a "wheel" is turned and a single individual of the population is selected, where the probability of selection is proportional to the individual's fitness.

Fitness scaling can essentially improve the performance of a GA (Kreinovich et al., 1993): It avoids that some very good individuals dominate the selection process at the beginning of

Figure 4.5: One-Point Crossover



the search process and increases selection pressure when the population converges at the end of the search process. Instead of an individual's raw fitness value f—that is the result of the evaluation operator—a scaled fitness value f' is used on the process of selection. In our GA fitness values are balanced by linear scaling (Goldberg, 1989) during the selection process. By applying this method the average value of the raw fitness $\bar{f}$ equals the average value of the scaled fitness $\bar{f}'$: $\bar{f} = \bar{f}'$ and the maximum value of the scaled fitness ${f'}_{max}$ is twice of its mean: ${f'}_{max} = 2\bar{f}$. The scaled fitness f' for every individual is computed by

$$f' = a \cdot f + b.$$

The factor a is computed by

$$a = \bar{f}/(\bar{f} - f_{max}),$$

where $\bar{f}$ is the average raw fitness, and the second factor by b

$$b = \frac{\bar{f}(f_{max} - 2\bar{f})}{(\bar{f} - f_{max})}.$$

To implement the roulette wheel method, the individual's fraction on the scaled total fitness of the population is computed. The sum of these values gives the cumulative probability for every individual. Selection takes place by a randomly generated number from the interval $[0, 1]$. The individual whose cumulative probability is met or is the next largest is the first candidate for the crossover, a second one is chosen analogously.

Reproduction

The algorithm uses the following two ways to build offspring from the individuals selected in the former step. The choice of a specific operator is at random.

The *one-point crossover* is performed as follows: Two candidates selected in the former step are recombined around a selection point. All alleles before and including the crossing point are kept in the current chromosome, while alleles behind the crossing point change from the current to the second chromosome and vice versa. Figure 4.5 shows an example of the one-point crossover. Chromosome C_0 is shaded. On the left-hand side the original chromosomes are shown, the right-hand side is the situation after the crossover was applied. To find the crossing point an integer number out of the range $[1, (n-1)]$ is randomly generated, where n is the number of genes in the chromosome.

The *two-point crossover* works analogously but two crossing points are generated: The first point a is taken from the interval $[1, (n-2)]$ and the second point b from $[a, (n-1)]$. The offspring consists of the first a alleles from its own, $b - a$ alleles from the second individual and $n - b$ alleles from itself again.

Mutation

The decision whether the mutation operator is applied or not depends on randomness. For every chromosome of the population a random number r is generated out of the interval $[0, 1]$. This value is compared to the given mutation rate p_{mut} describing the percentage of the population which should be mutated. If $r \leq p_{mut}$ the mutation operator is applied: The locus is found by generating another random number from the interval $[0, (n-1)]$, where n is again the number of genes in the chromosome. The new allele is randomly generated from the interval $[0, (m-1)]$, where m is the number of vehicles respectively tours (Properties 2 and 7).

Practically speaking, mutation changes the vehicle which will serve a customer. In every individual at most one gene is mutated.

Repair Process

If it turns out that one part of the subproblem has no solution, e.g. a time window in one tour cannot hold, a repair mechanism is called. Its basic idea is that the narrowest time window in a tour is most difficult to fulfill and it tends to be easier satisfying this restriction in a smaller tour with fewer dependencies.

If a tour is infeasible and there are some tours left for evaluation, the customer node i with the narrowest time window $\arg\min_i(l_i - e_i)$, where e_i is the earliest starting time and l_i is the latest possible due date, is deleted from the current and added to the shortest tour (the one containing least nodes) of the not yet evaluated tours. If there is more than one tour with an identical number of nodes, out of these a random tour is chosen. Afterwards, the computation for the current vehicle is repeated.

If the tour of the last vehicle is infeasible, the node with the narrowest time window is transferred to a randomly chosen shortest tour. The varied chromosomes must be re-evaluated. If there are still violated time windows, one more node is exchanged and the process continues until all time windows can hold or a maximum number of repair trials was executed. In this case the individual is deleted from the population. Otherwise the fitness value is calculated from the sum of all tours in the chromosome.

The goal of this procedure is to save computational time: If more than one tour is left, only one tour must be recomputed, while by selecting the "global" shortest tour the total duration of travel of this tour changes and must be calculated again. This is only necessary in the second case because no more tours are available to transfer the node to.

Complete Algorithm

The complete algorithm is described in Algorithm 11 on page 80. P_0 is a set of initial solutions. The GA terminates if there exists no sufficient number of individuals in the initial population. N_{it} and N_o are parameters for the number of iterations and offspring respectively. P_i is the population of iteration i and P_o are the offspring in a specific generation. P_f is used for the union of P_i and P_o; P_i and P_o are deleted at the end of each iteration (after the union). α gives

the mutation rate. Π_0 and Π_1 are the parents in the recombination step, π_0 and π_1 are the offspring. In the list I_e we keep individuals that have been evaluated in former iterations. Note that the line 24 is only required in the rolling horizon framework and will be explained more detailed in Section 4.4.

In a first step, the initial population is generated and evaluated; the CP models introduced above are applied. The initial population is generated by the Constraint Satisfaction Problem (4.1)–(4.12) as described above. Once a sufficient number of individuals was found the evaluation function is called, that optimizes tours with more than one customer nodes using the Constraint Optimization Problem (4.13)–(4.21). Otherwise, the algorithm terminates.

The function Evaluate(I)—which is described in pseudocode on page 81—works as follows: In a first step, the information about the tour for each of the $N_{vehicles}$ vehicles are extracted. The algorithm uses a sorted set L to save the tours that must be explored (again)—tour indexes are sorted increasingly in this set and appear only once. Thus, the algorithm evaluates the tour with the smallest index next. The way of computing the shortest tour depends on the number of nodes in this tour: Depots without any customer nodes have a travel distance of 0, tours with one customer node have an objective function value twice the distance between customer node and depot, and for all other tours the TSP (4.13)–(4.21) must be solved to find the tour's objective function value. d_{total} is the sum of duration of all tours and gives the fitness value of the individual. l_i and e_i denote the latest and earliest allowed service time for each node. The variable *success* saves if a feasible solution for the current vehicle can be generated. The boolean variable *passed* signals if all nodes have been evaluated. This is important in the repair process because nodes must be recomputed in this case during the repair process (line 34), otherwise nodes are simply moved to the remaining tours. *stall* counts the number of attempts to repair the individual by exchanging nodes. If this counter exceeds a specified limit, the variable *infeasible* is set to *true*. The function will return a *NULL* value in this case to signal the GA that the individual is infeasible. Otherwise the fitness value is returned.

After the initial evaluation was done, the GA repeats *Repruction*, *Mutation*, and *Evaluation* as described above. At the end of each iteration, out of the current population and the offspring the fittest individuals are selected to build the population for the following iteration. If one of the termination criteria (maximum time, maximum number of iterations, and maximum number of iterations without improvement) is met, the fittest individual is output as the result of the algorithm.

4.4 Rolling Horizon Framework

The planning process considers a rolling horizon setting: The tours for the upcoming periods are designed on the basis of the current knowledge. While the vehicles are on tour, new customer demand appears. At the end of every period the pool of customer demand is updated: new orders are added and already served customers must be removed (Assumption 11 in Section 2.1). Using this information the tours are updated to serve all known customers. In the planning process the dispatcher can swap demand between different vehicles, even demand from the former period can be reassigned as long as the customer has not been visited. If a

Algorithm 11: GA

Generate initial population P_0 consisting of N_{init} individuals using CSP (4.1)–(4.12)
if *No population with a sufficient number of individuals can be built* **then**
    Return NULL
Initialize $d_{total} = 0$ and $P_o = \emptyset$
`Evaluate` $(\mathtt{I_k})$ all individuals $I_k, k \in P_0$
for $i = 0$ **to** N_{it} **do**
    $j := 0.$
    while $j \leq N_o$ **do**
        Select two parents Π_0 and Π_1 out of P_i at random `/* Recombination */`
        Generate random number ψ in range $[0, 1]$ `/* Crossover */`
        if $\psi < 0.5$ **then**
            Apply one-point crossover to Π_0 and Π_1 to yield π_0 and π_1. Select crossover point at random
        else
            Apply two-point crossover to Π_0 and Π_1 to yield π_0 and π_1. Select crossover points at random
        for *k=0* **to** *1* **do**
            `/* New individual? */`
            if $\pi_k \in I_e$ **then**
                Discard individual
            Add new individual π_k to P_o
            $j := j + 1$
    for $k \in P_o$ **do**
        Generate random number γ in range $[0, 1]$ `/* Mutation */`
        if $\gamma \geq \alpha$ **then**
            Apply mutation to k-th individual of P_o
        Add fixed nodes to chromosome
        `Evaluate` $(\mathtt{I_k})$
        if $d_{total} \geq 0$ **then**
            Set fitness value to d_{total}
        else
            Delete k-th individual of P_o
    Merge individuals of P_o and P_i to yield P_f `/* Prepare next iteration */`
    Sort P_f in increasing order of fitness values
    Select N_{init} fittest individuals of P_f to yield P_{i+1}
    Clear P_f and P_o
Return fittest individual

Function Evaluate(I)

$d_{total} := 0$, $d_i := 0, i \in N_{vehicles}$, *infeasible* := FALSE, and *passed* := FALSE
Initialize sorted set $L := 1, \ldots, N_{vehicles}$.
Extract tours from individual I's chromosome and save them in the set *Tour*
while $L \neq \emptyset$ **do**
    $i :=$ first element in L
    if $i == N_{vehicles}$ **then**
        passed := TRUE
    switch *number of nodes in* $Tour_i$ **do**
        case *1*
            Delete i from L `/* Depot is the only node */`
        case *2*
            $d_i := 2 \times c_{0i}$ `/* Depot and one additional node */`
            $d_{total} := d_{total} + d_i$
            Delete i from L
        otherwise
            stall := 0 and *success* := *FALSE* `/* Depot and multiple nodes */`
            while *(stall* $\leq stall_{max}$*)* $\wedge$ *(success == FALSE)* **do**
                if *(Sufficient Capacity)* $\wedge$ *(COP* (4.13)–(4.21) *has a solution)* `/* Use COP to compute shortest path with length` d_i `*/`
                then
                    $d_{total} := d_{total} + d_i$
                    success := TRUE
                    Delete i from L
                else
                    if *passed == FALSE* **then**
                        $g := \arg\min_j (l_j - e_j)$ `/* Repair mechanism */`
                        Find tour z with least nodes that has not been evaluated yet ($z \in L$)
                        Erase g from individual i and add it to z
                        stall := *stall* + 1
                    else
                        $g := \arg\min_j l_j - e_j$
                        Find tour z with least nodes
                        Erase g from individual i and add it to z
                        $d_{total} := d_{total} - d_z$
                        Add z to L
                        stall := *stall* + 1
            if *success == FALSE* **then**
                infeasible := TRUE
if *infeasible == FALSE* **then**
    Return total travel time d_{total}
else
    Return NULL

customer was serviced by a vehicle it becomes "fixed" and neither its position in the tour (and the arrival time) nor the number of the vehicle servicing it can be changed anymore. From an algorithmic point of view, reoptimization takes place in every period.

To model the rolling horizon problem we embedded our GA into a heuristic framework inspired by the Dynamic VRP (Kilby et al., 1998; Gendreau et al., 1999; Montemanni et al., 2005).

The time horizon is divided into several periods, where the start and end time of every period is known in advance but the demand occurring is unknown a priori. The algorithm starts optimizing the current set of orders at the beginning of the planning horizon ($t = 0$) and presents a solution for every vehicle to serve all orders known so far. During the first period all vehicles start their tours at the depots and visit the earliest nodes. Note that some vehicles will stay in the depot because of Assumption 12. While the vehicles are on tour, new orders occur (for future periods only). These are collected until the end of the current period. Now, the route for every vehicle is reoptimized. The order of nodes that already have been visited must be fixed, because it cannot be changed anymore and the new orders have to be inserted into the vehicles' tours.

For that reason the GA is modified as follows: After the first period all nodes whose departure time in the best individual is less or equal to the current time (so called *fixed nodes*) as well as all successors of fixed nodes are deleted from the current individual and moved to a cache. This ensures that no modifications in reproduction or mutation can be made. These operators can modify the non-fixed nodes only. Furthermore, new nodes are added randomly to one of the tours with the smallest number of nodes. We assume that it is easier to find a feasible solution with respect to the time windows when fewer restrictions exist. This version is referred to as *GA-RH*.

Figure 4.6: Illustration of Gene Modification

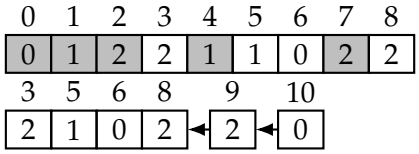


Figure 4.6 illustrates the adjustment of a chromosome. The shaded alleles in the upper chromosome are fixed nodes and cut off in the next period (lower chromosome). Some new nodes are added at the end of the revised chromosome. The resulting chromosome establishes the basis for the next iteration. During the optimization process arrival times and the arrangement of the fixed nodes are fixed by subjoining additional constraints. The TSP Model (4.13)–(4.21) therefore has to be extended by the following constraints:

$$t_i = \hat{t}_i \qquad i \in Nodes_{fixed} \tag{4.26}$$

$$x_i = \hat{x}_i \qquad i \in Nodes_{fixed} \tag{4.27}$$

Constraint (4.26) forces the arrival time of the fixed nodes to equal the saved date $\hat{t}_i$. Constraint (4.27) ensures that the relationship between the predecessor and the successor of the fixed nodes $\hat{x}_i$ is maintained. In a *CP* model the fixation of some nodes simplifies the search

for a solution. Because of the additional constraints domain propagation is more efficient and accelerates the search process.

The overall GA-RH algorithm is described in Algorithm 12, where $\underline{H_i}$ denotes the beginning of the i-th time horizon and $\overline{H_i}$ its end. H_N is the last horizon.

An additional difference to the proposed static Algorithm 11 is that individuals' fitness must be updated after adding nodes. Since the problem has more nodes in every step the current value is obsolete. If it would not be changed, the individuals of the last iteration always would be chosen because of the shorter duration of travel. Note that line 24 in Algorithm 11 must be executed to compute the correct fitness value. Otherwise, only a tour consisting of non-fixed nodes would be found.

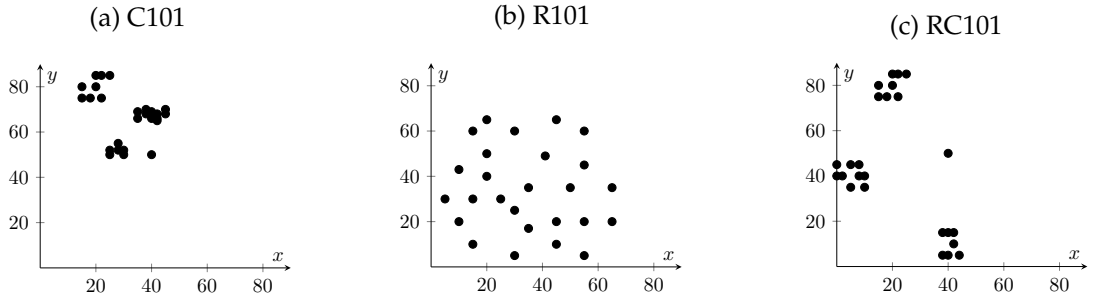
Algorithm 12: Rolling Horizon Framework

```
1  H_1: Run algorithm 11 for initial customer demand
2  overline(H_1): Fix order and time of already visited nodes in the best chromosome and delete fixed
   nodes from all chromosomes
3  while i ≤ H_N do
4  |  H_i: Add new nodes to all chromosomes and Evaluate (I) all individuals
5  |  if d_total ≥ 0 then
6  |  |  Set fitness value to d_total
7  |  else
8  |  |  Delete individual
9  |  Run Algorithm 11 for new (truncated) population
10 |  overline(H_i): Fix order and time of already visited nodes
11 |  Delete fixed nodes from all chromosomes
12 |  i := i + 1
13 H_N: Add new nodes
14 Run Algorithm 11 for complete customer demand
```

4.5 Computational Study

We tested the GA and the rolling horizon framework on the Solomon instances (Solomon, 1987) and MDVRPTW instances generated by Cordeau et al. (2001). The Solomon instances consist of 100 customers, but there are subsets that contain only the first 25 respectively 50 customer nodes. Instances do not only differ in the number of customers but also in the spatial distribution of the customers in a coordinate graph: Customers' positions may be distributed in clusters (nodes in the same cluster are close to each other), randomly generated by a uniform distribution, or may be a combination of clusters and random values. We will mark these instances by *C*, *R*, and *RC*, respectively. Figure 4.7 illustrates the spatial distribution of randomly chosen instances *C101* (Figure 4.7a), *R101* (Figure 4.7b), and *RC101* (Figure 4.7c). Each node has a specific time window, demand, and service time. Instance numbers that start with 1 have narrower time windows at the depot than those that start with 2, therefore a vehicle can service fewer customers on its tour. The number of vehicles is 25 in all instances.

Figure 4.7: Illustration of Instance Types C, R, and RC



We use the Euclidean distance to compute the travel costs. Because the CP subproblem requires integer values for the variables—and (total) travel distances are variables—we use rounded down distance values. The reduced precision results in better solutions compared to other heuristic and exact methods from literature that use a higher precision on distance values.

We modified the Solomon instances by the use of five depots instead of one. For this purpose the problem set was divided into four areas. Each border was placed exactly in the middle between the "maximum" and "minimum" node. For the horizontal case (x-axis) the border value was set to $l_x = ((\max x - \min x)/2 + \min x)$, in the vertical case $l_y = ((\max y - \min y)/2 + \min y)$. A depot was placed in the center of each area. The fifth depot was inserted on the intersection of the boundaries, while the initial depot of the problem sets were removed. Values for demand, service times, and the time windows are cloned from the origin depot.

The characteristics for the instances of Cordeau et al. (2001) are given in Table 4.1, the additional column *Steps* gives the number of reoptimization steps in the rolling horizon setting (12 customers are added in each iteration). Customer nodes are placed randomly around predefined seeds that act as depots. Thus, these instances contain clusters. Instances *pr01–pr10* have narrow time windows, while instances *pr11-pr20* allow a wider span to visit a node (all other values are equal). The time windows at the depot are equal in both cases.

For the rolling horizon framework both instance sets must be prepared as follows to meet the requirements of a rolling planning horizon. According to Assumption 11 from Section 2.1 new demand can only occur for future periods. To guarantee that, first of all, the customer nodes are sorted in a non-decreasing order of due dates. This set is divided into a number of smaller subsets according to the number of steps to be taken into account. Each set contains $N_i/step$ nodes, where N_i is the total number of customer nodes in instance i and *step* gives the number of periods in which new requests can be received. Fractional values are adjusted downward, remaining nodes are added to the last step. We set the parameter *step* to 3 in problems with 25, to 5 in instances 50 customer nodes, and to 8 in 100 customer instances in the Solomon instances, and to the values given in Table 4.1 for the Cordeau instances. The point in time for fixing nodes is the ready time of the $(N_i/step)$-th node in the current step. In this design, some instances are not applicable because in further steps nodes are added with a ready time that has already been fixed. Therefore, it is not possible to add these nodes and the instances are discarded (and are not presented in the following tables).

For the entire computational study a computer with 2.67 GHz Duo Core CPU and 8 GB RAM running *Windows 7 (64 bit)* was used. The CP solver used was *Gecode 4.3.2* (Schulte et al.,

Table 4.1: Problem Instance Characteristics of Cordeau Instances

Instance	Depots	Vehicles	Customers	Max. Length	Max. Load	Steps
pr01	4	2	48	500	200	4
pr02	4	3	96	480	195	8
pr03	4	4	144	460	190	12
pr04	4	5	192	440	185	16
pr05	4	6	240	420	180	20
pr06	4	7	288	400	175	24
pr07	6	2	72	500	200	6
pr08	6	3	144	475	190	12
pr09	6	4	216	450	180	18
pr10	6	5	288	425	170	24
pr11	4	1	48	500	200	4
pr12	4	2	96	480	195	8
pr13	4	3	144	460	190	12
pr14	4	4	192	440	185	16
pr15	4	5	240	420	180	20
pr16	4	6	288	400	175	24
pr17	6	1	72	500	200	6
pr18	6	2	144	475	190	12
pr19	6	3	216	450	180	18
pr20	6	4	288	425	170	24

2015) and the GA as well as the framework were implemented using *Visual C*$^{++}$ 2012.

We will describe the experiments to set the parameters for the GA in the next section. In Section 4.5.2 the results of the rolling horizon approach will be explained. Afterwards, we present the results of the GA on the original Solomon instances (without rolling horizon planning and a single depot) to have a benchmark on the quality of the GA with larger instances.

4.5.1 Adjusting the Search Parameters

We have performed experiments to fine tune the parameters of the GA. The population size was fixed to 80 individuals and the maximum computation time to 300 seconds, all other parameters were varied:

- Maximum number of iterations (per step).
- Maximum number of iterations without improvement (per step) (so called *stall iterations*).
- Mutation rate: Percentage of individuals the mutation operator will be applied to.
- Elitism rate: Percentage of individuals that will be selected from the parent population to be part of the next generation.

The two parameters mentioned first were stop criteria, i.e. if the value was exceeded the algorithm terminates. Each combination of parameters was run ten times. We have tested one instance of each type of the Solomon instances (*C101*, *R101*, *RC101*) with three steps, in total

720 variants have been evaluated. Table 4.2 gives the results on a different number of iterations, the first value in each cell is the mean objective value of ten runs, the first value in brackets is the minimum and the second value is the maximum objective in ten runs. Best values for each are printed bold.

Table 4.2: Average, Best, and Worst Objective for Different Numbers of Iterations

C101			
Iterations	Step 0	Step 1	Step 2
100	126.5 (100; 146)	217.1 (174; 249)	338.7 (281; 385)
200	122.4 (97; 138)	211.1 (175; 241)	328.8 (270; 366)
300	119.1 (**93**; 136)	204.0 (16; 230)	319.2 (274; 359)
400	117.1 (100; **134**)	201.6 (170; 232)	314.6 (256; 360)
500	**116.6** (95; **134**)	**199.0** (**165**; **225**)	**310.6** (**248**; **349**)
R101			
Iterations	Step 0	Step 1	Step 2
100	263.6 (228; 284)	472.1 (371; 537)	732.8 (**548**; 832)
200	255.9 (**204**; 283)	457.7 (395; 528)	713.2 (623; 803)
300	254.7 (222; 279)	454.6 (388; 513)	703.1 (613; 791)
400	251.5 (218; 272)	445.9 (**345**; 523)	696.3 (578; 788)
500	**247.9** (218; **271**)	**439.0** (384; **495**)	**686.2** (571; **770**)
RC101			
Iterations	Step 0	Step 1	Step 2
100	231.5 (179; 282)	487.5 (357; 551)	740.8 (605; 865)
200	218.6 (187; 255)	461.0 (**332**; 540)	698.9 (**531**; 812)
300	212.6 (162; 249)	453.8 (376; 503)	681.0 (598; 783)
400	212.3 (162; 249)	446.3 (377; **501**)	673.6 (571; 770)
500	**209.9** (**160**; **247**)	**441.9** (348; 506)	**663.1** (564; **761**)

From the test of 100, 200, 300, 400, and 500 iterations one can see that the highest number of iterations provides the best results. Even if some minimum values are better for lower iteration numbers, the maximum values are higher compared to the one from the 500 iteration test.

As mentioned above, the time limit was set to 300 seconds per step, but no instance was aborted because of the time limit in the current setting. In all configurations about 82% of the instances were aborted because the maximum number of iterations has been reached, the remaining 18% because of the maximum number of iterations without improvement. These values remain almost identical for a higher number of stall iterations. Objective function values deviate by 0.03% between the settings with 50 and 100 stall iterations. However, the better individual values were found more often at 100 stall iterations. Therefore, we have used the higher value of 100 iterations in the computational study and accepted slightly longer computation times but ensuring that better solutions are found. The detailed results can be found in Table 4.3.

We have tested the values of 17% and 33% for the elitism rate, i.e. 17% respectively 33% of the individuals were chosen from the (unmodified) parent population when the population for the next generation was build; the remaining individuals were taken from the population that

Table 4.3: Average, Best, and Worst Objective Function Values for Different Numbers of Stall Iterations

C101			
Stall Iterations	Step 0	Step 1	Step 2
50	**120.3** (95; **144**)	207.1 (**165**; 249)	322.8 (256; 385)
100	120.4 (**93**; 146)	**206.0** (169; **240**)	**322.0** (**248**; **377**)
R101			
Stall Iterations	Step 0	Step 1	Step 2
50	**254.6** (218; **284**)	455.2 (**345**; 537)	708.8 (574; **804**)
100	254.8 (**204**; 284)	**452.5** (371; **528**)	**703.8** (**548**; 832)
RC101			
Stall Iterations	Step 0	Step 1	Step 2
50	217.2 (162; **282**)	**457.5** (**332**; 551)	691.5 (**531**; 865)
100	**216.8** (**160**; **282**)	458.7 (357; **543**)	**691.4** (571; **837**)

was created in the current iteration. In the results given in Table 4.4 no trend can be found and the differences are small. We have chosen the lower value of 17% in the hope that the children are more diverse and provide better solutions.

Table 4.4: Average, Best, and Worst Objective Function Values for Different Elitism Rates

C101			
Elitism Rate	Step 0	Step 1	Step 2
0.17	116.9 (**95**; **130**)	199.9 (**165**; **225**)	311.6 (**248**; 349)
0.33	**116.3** (99; 134)	**198.0** (177; **225)**	**309.7** (266; **348**)
R101			
Elitism Rate	Step 0	Step 1	Step 2
0.17	**247.1** (**218**; **270**)	440.7 (**384**; 495)	687.5 (574; **768**)
0.33	248.6 (**218**; 271)	**437.4** (395; **484**)	**684.8** (**571**; 770)
RC101			
Elitism Rate	Step 0	Step 1	Step 2
0.17	210.5 (164; 247)	**441.3** (360; 506)	**658.2** (**564**; **753**)
0.33	**209.2** (**160**; **229**)	442.5 (**348**; **489**)	668.0 (581; 761)

The differences between the mean values are small when we evaluate the mutation rates of 20% and 40%, too. As can be seen from Table 4.5 (on page 88) the values are very similar. However, the best individual solution was found more often in the setting with the higher mutation rate of 40%. Thus, we have chosen this value. In summary, the algorithm was run at most 500 iterations, 100 iterations without improvement or 300 seconds. The population contains 80 individuals, the mutation rate is 40% and the elitism rate is 17%. Additionally, we allow 50 repair trials. This setting will be used for all experiments in this section.

Table 4.5: Average, Best, and Worst Objective Function Values for Different Mutation Rates

C101			
Mutation Rate	Step 0	Step 1	Step 2
0.2	**120.3** (95; 146)	*206.5* (**165**; **241**)	**322.0** (266; 385)
0.4	120.4 (**93**; **144**)	206.7 (173; 249)	322.8 (**248**; **374**)
R101			
Mutation Rate	Step 0	Step 1	Step 2
0.2	**254.7** (**204**; **284**)	454.7 (379; 537)	706.4 (571; 832)
0.4	**254.7** (218; **284**)	**453.0** (**345**; **528**)	**706.3** (**548**; **811**)
RC101			
Mutation Rate	Step 0	Step 1	Step 2
0.2	**216.7** (162; **282**)	458.5 (359; **549**)	692.8 (**531**; 865)
0.4	217.2 (**160**; **282**)	**457.7** (**332**; 551)	**690.0**(571; **854**)

4.5.2 Rolling Horizon Setting

We started our evaluation with the Solomon instances. Table 4.6 on the facing page shows the percentage of instances that could be solved in each step. Instances are grouped by the number of customers (column *Cust.*), the instance type (column *Type*), and the length of the time window (column *TW*). All instances were run ten times, the values are the average result of all runs. For the instances with 25 customers all data sets could be solved. This is not surprising because the number of customers equals the number of vehicles. The same is true for the first two steps in the larger instance sets. All data sets with 50 customer nodes except *R101* and *RC106* could be solved at least once. Generally, instances with a wider planning horizon were solved more often than those with a shorter time window at the depot (type 1). For the largest data sets only two instances could be solved in all steps (*C206* and *C207*).

The computation time for the GA varied in every step, but tended to increase in general. Table 4.7 on the next page shows the mean computation time over ten runs. Solution times that were higher than the limit of 300 seconds result from the fact that the termination criterion was checked at the end of any iteration and terminated afterwards. This was the case if the repair procedure was called often and/or solving the B&B subproblem consumed a lot of time because a tour consists of many nodes. Computation time did not strongly depend on the problem type in the steps 1–3. The range increased from step 4 on, however this variation could be explained with the different number of solved instances in all types: The less instances could be solved the higher is the computation time.

The increasing duration per iteration is illustrated in Figure 4.8 on page 90, too. It shows the reasons for the termination of the GA. In the step 0 all iterations could be executed, but starting from step 1 the problem size increased and about half of the steps were terminated because of the time limit and the other half because of the maximum number of iterations. Again, the values for steps 6 and 7 were based on at most four instances and the fact that one of these steps was terminated because of the maximum iterations without improvements demonstrated the difficulties with problems of this size.

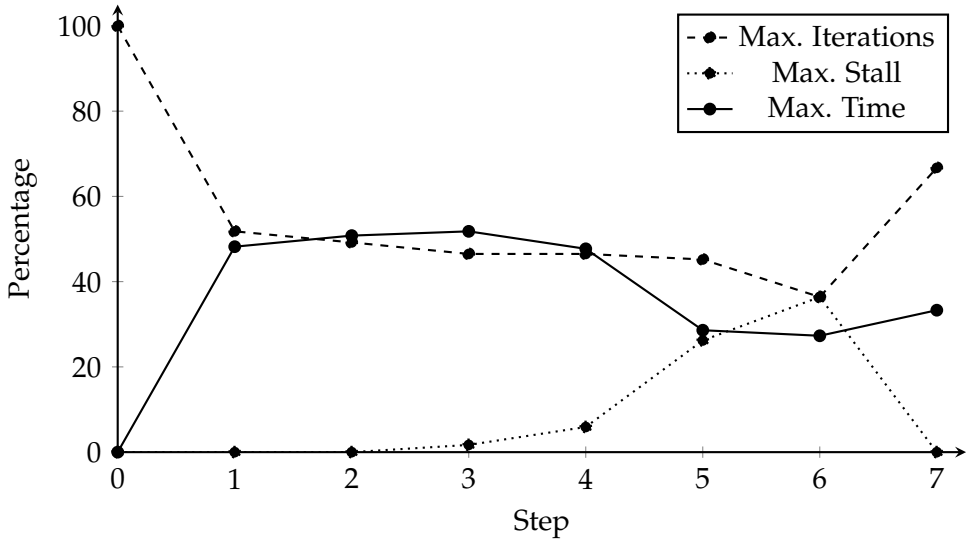
Table 4.6: Percentage of Solved Solomon Instances in Each Step

Cust.	Type	TW	Steps 0	1	2	3	4	5	6	7
25	C	1	100%	100%	100%					
		2	100%	100%	100%					
	R	1	100%	100%	100%					
		2	100%	100%	100%					
	RC	1	100%	100%	100%					
		2	100%	100%	100%					
50	C	1	100%	100%	100%	86.7%	65%			
		2	100%	100%	98%	94%	86%			
	R	1	100%	100%	85%	76.7%	55%			
		2	100%	100%	98%	98%	90%			
	RC	1	100%	100%	92%	58%	20%			
		2	100%	100%	100%	98%	84%			
100	C	1	100%	100%	98.3%	49.2%	13.6%	0%	0%	0%
		2	100%	100%	98%	80%	60%	32%	8%	6%
	R	1	100%	100%	94.9%	71.2%	11.9%	0%	0%	0%
		2	100%	100%	93.8%	81.3%	41.7%	27.1%	12.5%	0%
	RC	1	100%	100%	96%	68%	8%	0%	0%	0%
		2	100%	100%	98.2%	84.2%	45.6%	24.6%	3.5%	0%

Table 4.7: Computation Time For Each Step in the Solomon Instances (in Seconds)

Cust.	Type	TW	Step 0	1	2	3	4	5	6	7
25	C	1	2.7	7.5	17.9					
		2	2.7	8.7	19.1					
	R	1	2.6	7.6	16.6					
		2	2.7	8.2	19.2					
	RC	1	2.6	7.4	16.2					
		2	2.7	8.5	19.4					
50	C	1	3.6	10.2	34.0	69.8	123.1			
		2	3.9	12.5	35.0	71.6	128.5			
	R	1	3.8	10.6	31.4	70.2	135.6			
		2	3.9	11.3	34.6	70.7	124.7			
	RC	1	3.7	9.6	27.4	62.0	120.3			
		2	3.9	10.8	35.2	74.0	135.4			
100	C	1	4.6	15.6	49.2	110.9	150.3			
		2	4.8	16.5	54.4	117.1	210.7	354.2	576.5	594.0
	R	1	4.6	17.2	63.0	148.8	343.5			
		2	5.0	18.1	65.0	160.3	365.1	528.7	781.4	
	RC	1	4.7	16.4	49.4	99.7	144.3			
		2	5.0	17.5	58.4	210.4	448.0	498.5	558.0	

Figure 4.8: Reasons for Termination of the GA in the Solomon Instances (in Percent)



We have listed the mean objective function values per instance type and step in Table 4.8 on the next page (detailed values can be found in Tables A.1–A.3 in Appendix A). It is obvious that the objective increased in every step because tours were built of more customers and the fixed nodes were included in the objective. We will compare these values to another heuristic approach in Section 5.5. If we have a look at the variety of objective function values within a single instance, it can be seen that randomness strongly influences the results: The mean difference (over ten runs) between the largest and smallest value was about 20% in all steps of the 25 customer instances, this value grew to 66% in steps 3 and 4 in 50 customer instances (especially values in *RC105* were twice the best value) but decreased again in the last step. The difference for 100 customer instances varied between 14% and 31%.

To evaluate the possibility to exchange nodes between vehicles, we analyzed how often an order was reassigned. Therefore we counted if a request was fulfilled by another vehicle in the best solution of a specific step compared to the best solution of the previous one (that is why step 0 is missing). The ratios are summarized in Table 4.9 on the facing page, detailed values can be found in Tables A.4–A.6 in Appendix A. As fixed nodes are included in the computation values from step 2 on will never result in a value of 100% even if all not fixed nodes are exchanged.

In general, nodes were more often exchanged between vehicles in instances of type 1. We have stated above that it is more difficult to solve these problems, because the time windows at the depot were tighter. To find a feasible solution it was crucial to have short tours and the algorithm made more effort to get those solutions. Additionally, the repair procedure was called more often to fix infeasible solutions, which meant that customers were exchanged between vehicles. In summary, we can conclude that the algorithm works properly for the rolling horizon setting. Nodes were exchanged in most of the steps to obtain better solutions on the basis of the new information.

We have also tested Cordeau instances but it turned out that the algorithm was not able to solve the problem in acceptable time because the subproblem rapidly grows too large. Out of the 20 instances 12 could be used within our framework. The first two steps could be solved by

Table 4.8: Objective Function Values for Each Step in Solomon Instances

Cust.	Type	TW	Step 0	1	2	3	4	5	6	7
25	C	1	114.3	221.7	355.8					
		2	121.9	312.9	404.5					
	R	1	223.8	476.2	742.2					
		2	220.7	444.4	683.3					
	RC	1	222.5	481.7	787.8					
		2	216.6	429.4	660.0					
50	C	1	194.4	423.9	640.1	876.6	1,164.3			
		2	125.7	428.5	702.5	935.1	1,104.9			
	R	1	248.7	612.4	941.0	1,353.7	1,871.8			
		2	277.8	608.4	893.2	1,187.2	1,508.9			
	RC	1	432.5	932.4	1,436.8	2,006.9	2,659.1			
		2	448.7	914.7	1,336.4	1,756.2	2,181.7			
100	C	1	334.0	829.4	1,368.0	2,171.0	3,313.6			
		2	242.8	667.5	1,127.4	1,639.2	2,199.2	3,341.3	3,612.3	4,305.0
	R	1	249.4	627.9	1,123.0	1,870.5	2,786.9			
		2	303.1	709.6	1,155.2	1,566.8	1,960.1	2,512.9	3,286.0	
	RC	1	333.3	888.4	1,566.5	2,425.7	3,649.0			
		2	381.7	965.0	1,507.0	2,083.0	2,739.5	3,469.6	4,342.5	

Table 4.9: Percentage of Exchanged Nodes in Solomon Instances

Cust.	Type	TW	Step 1	2	3	4	5	6	7
25	C	1	6.3%	6.5%					
		2	1.8%	1.9%					
	R	1	2.9%	5.4%					
		2	0.1%	0.1%					
	RC	1	3.2%	4.4%					
		2	1.4%	0.9%					
50	C	1	3.6%	1.8%	1.3%	2.4%			
		2	0.0%	0.3%	0.4%	1.0%			
	R	1	2.7%	4.8%	7.7%	8.7%			
		2	3.3%	3.8%	2.4%	3.6%			
	RC	1	7.3%	12.4%	9.0%	7.6%			
		2	5.3%	5.0%	2.0%	1.9%			
100	C	1	6.8%	9.4%	8.5%	28.5%			
		2	5.4%	4.1%	6.9%	24.3%	18.3%	13.5%	8.9%
	R	1	6.6%	10.8%	12.3%	35.7%			
		2	8.0%	8.9%	8.4%	25.3%	20.9%	16.4%	0.0%
	RC	1	7.8%	12.1%	13.6%	34.0%			
		2	5.0%	6.8%	6.8%	26.9%	20.8%	16.9%	

the GA within the time limit for all but two instances; the problem contains 24 customer nodes which were almost the same as in the smallest Solomon instances but fewer vehicles were available. However, the algorithm failed in the following steps to find a solution: Three steps were completed for instance *pr08* and *pr14* and a fourth step for instance *pr19*. The computation of the first step took 110 seconds on average, while the second step consumed 1,763 seconds which is more than the triple of the time limit and shows the problems to find a solution in a single iteration.

4.5.3 Static Setting

We have also tested the GA for the complete problem set of Solomon Instances. The framework can be easily adapted to this situation: We set all depots to be in the same location and the number of steps to 1. This allowed us to evaluate the results of the algorithm as we can compare the results to the best known solutions for Solomon instances (*www.sintef.no/projectweb/top/vrptw/solomon-benchmark/*). Table 4.10 on the next page gives the result for this experiment: Columns *Cust.* and *Type* specify the instance characteristics *Mean Obj. GA* are the mean of the GA's objective function values computed over five runs. *Best Known Sol.* refers to the optimal or best heuristic solution found in the literature. *Gap* quantifies the difference between our and the best solution and is computed by $(\textit{Mean Obj. GA}/\textit{Best Known Sol.}) - 1$. Remember that we use distance values with low precision, thus the gap may be underestimated. The last column gives the mean computation time the algorithms to complete 500 iterations.

It can be seen that the algorithm was not very competitive, especially on larger instances it performed poorly. The mean objective function values were at least 97.4% and at most 311.1% worse than the mean optimal solution. As we mentioned above, this is not necessarily a drawback in dynamic problems. Solutions that are locally bad may give potential for the remaining steps because they can provide more flexible solutions (more customers may be rearranged) or the usage of more vehicles will be advantageous in the future. In some cases, higher travel costs can therefore incur higher revenues. This is a very general statement and must be investigated in further studies, but it stresses that the algorithm may not be discarded because of these results.

The algorithm worked best on randomized instances; for both problem sizes the gap was least for these problems. This can be explained by the fact that the operators of the GA are not designed to detect clusters in the problem set and do not guide the solution process towards tours that make use of the spatial distribution. Thus, customers in one cluster may be serviced by different vehicles. The impact of this effect was less in randomly generated instances which gives better results for these problems.

Results for the instances with 100 customers are missing because the algorithm had problems to find feasible solutions. It either aborted because no initial solution was found or got stuck in early iterations. Because we guess that the gap will increase with a higher number of customers, we skipped further studies after some initial experiments. Especially the solution time was not competitive, the time limit of 300 seconds was often exceeded in the first iteration.

Table 4.10: Results of the Static Setting for Solomon Instances

Cust.	Type	Mean Obj. GA	Best Known Sol.	Gap	Sol. Time GA (Sec.)
	C	486.9	201.8	141.3%	15.9
25	R	822.1	424.5	93.7%	30.2
	RC	881.1	334.8	163.2%	16.2
	C	1,341.9	359.5	273.2%	115.8
50	R	1,661.7	660.1	151.7%	153.7
	RC	2,452.7	596.6	311.1%	31.3

Chapter 5

A Branch&Price Approach with Constraint Programming-Based Local Search

This chapter describes another approach to solve the MDVRPTW-RH. At first, we give a literature review in Section 5.1 followed by a description of the CG model for the static problem (Section 5.2). Because the variables in the solution of the CG procedure will not necessarily be integers, the algorithm is extended to a B&P approach. This is necessary to have a solution that clearly indicates whether a tour is selected or not for the rolling horizon framework as well as to give clear instructions to the drivers in real-world problems. In Section 5.3 we describe the general branching steps and the problem-dependent branching rules. Section 5.4 sketches how the rolling planning horizon is handled, i.e. how to deal with fixed nodes and postponements. Penalties for delayed arrival at a node have been neglected in Chapter 4 but will introduced here. Section 5.5 shows the results of the computational study that have be done on two different test sets. Additionally, we compare the results of this approach to the ones achieved by the GA in Chapter 4.

5.1 Literature Review

There exists a wide field of publications that are related to CG, B&P and some VRP variants, therefore we will limit this review to VRPs with time windows. Branch&Price is a combination of CG and B&B and was introduced by Barnhart et al. (1998). Further introductory information on CG can be found in Desrosiers and Lübbecke (2005) and Lübbecke and Desrosiers (2005). A tutorial on B&P for the VRP (without time windows) can be found in Feillet et al. (2010). Baldacci et al. (2012) published a review on B&P and other exact approaches.

The first work on B&P and the VRPTW is by Desrochers et al. (1992) who relax the subproblem to the shortest path problem. More efficient ways to solve the subproblem are presented by Boland et al. (2006), Chabrier (2006), Feillet and Gendreau (2007), Irnich and Villeneuve (2006), and Righini and Salani (2008). Additional improvements to the performance can be achieved

by adding *cuts*, but we will not provide the details here and refer to the above-mentioned review instead. Azi et al. (2010) apply this methods to a VRPTW with multiple use of vehicles, Gutiérrez-Jarpa et al. (2010) to a VRPTW with deliveries and selective pick-ups. Bettinelli et al. (2011) introduce a Branch&Cut&Price approach to solve the VRPTW with a heterogeneous fleet and multiple depots, Dohn et al. (2011) and Dabia et al. (2013) use a similar approach for a VRPTW with temporal dependencies. Baldacci et al. (2011) present a column and cut generation algorithm for the set partioning formulation of the VRPTW. Salani and Vacca (2011) give an adaption of the solution method to solve the VRPTW with split deliveries. Tabu Search was added to the subproblem solver of Column Generation to speed up the solution process for the same problem by Archetti et al. (2011) . Athanasopoulos and Minis (2013) address the Multi-Period VRPTW with B&P.

The combination of B&P respectively CG and CP was discussed by several authors in the last decade. A general explanation of these solution approaches can be found in Milano and Wallace (2010) and Easton et al. (2004), extended surveys in Gualandi and Malucelli (2009, 2013). Combinations of CG and CP are applied to multiple problem, e. g. crew assignement (Junker et al., 1999; Yunes et al., 2000, 2005; Fahle et al., 2002; Sellmann et al., 2002), tail assignment (Grönkvist, 2005, 2006; Gabteni and Grönkvist, 2009), knapsack problems (Fahle and Sellmann, 2002), and routing (Rousseau et al., 2002a). Furthermore, there are B&P approaches combined with CP to solve scheduling (Easton et al., 2003; Puchinger et al., 2011) and routing (Rousseau et al., 2004; Cortés et al., 2014) problems.

There are many authors that investigate the combination of CP and LS techniques. General papers introduce frameworks to set up a search engine: Jussien and Lhomme (2002) present a generic framework that combines LS and *nogood*-based search. Based on a partial assignment propagation is executed to decide on the feasibility and to extend or repair the current solution. Mairy et al. (2010) suggest generic heuristics that decide which part of the solution should be relaxed. Mouthuy et al. (2012) introduce a framework for a very large-scale neighborhood search. Kiziltan et al. (2012) develop a local branching procedure that examines the neighborhood of a solution for better ones. Finally, Prud'homme et al. (2014) suggest a problem-independent LNS that makes use of subsets of constraints that cause conflicts within the search.

LS is applied to the TSP by Pesant and Gendreau (1996) and to its generalization, theVRP, by Shaw (1998), Backer et al. (1997, 2000) and Backer and Furnon (1999). Cambazard et al. (2012) deal with the course timetabling problem. A general review on Local Search for VRPs can be found in Funke et al. (2005), an overview on LS techniques in CP is provided by Hoos and Tsang (2006).

5.2 Column Generation Model

This section deals with a set partitioning model (Garfinkel and Nemhauser, 1969) for the MDVRP. We explicitly add a variable for every feasible tour. These binary variables indicate whether a tour is selected in the solution. Every variable is connected with a $0-1$ vector that represents if the i-th customer is part of the tour and the starting depot. Furthermore, costs that result from servicing this tour are known. Thus, a tour can be seen as a column in an integer

program. One now seeks for the set of tours/columns that visits every customer exactly once at minimal costs.

This formulation results in an extraordinary number of variables because there may be a huge number of feasible tours that can be combined with several depots. On the other hand, only a very small subset of tours is selected (at most the number of vehicles available in total). Therefore it may be sufficient to consider only a subset of tours instead of the complete enumeration. Additional tours that can further improve the solution can be generated on the fly. It is common in literature to call the general set partitioning problem *master problem* and the subprocedure to find new columns *subproblem* (sometimes also *oracle* or *column generator*).

Column generation starts with an initial set of columns that produces a feasible solution for the set partitioning problem. From the LP theory it is known that the objective can be improved by including a variable (column) with negative reduced costs in the basis. To obtain a column with negative reduced costs one uses the structure of the underlying problem and computes a solution with a procedure that shows good results on the problem. This is a TSP in our case.

One (or more) columns identified by this subprocedure are added to the master problem which is solved again. As a result the values of the dual variables change and are put in the subproblem again. This procedure continues until the subproblem cannot find any new columns with negative reduced costs. Because no more columns can improve the solution of the set partitioning problem the optimal solution is found. Note that the solution is not optimal if one uses a heuristic in the subprocedure. Because there is no guarantee for finding the optimal solution with a heuristic, there may be ”better” columns that cannot be found when the heuristic gets stuck in a local optimum. If these columns are not sent to the set covering problem, the solution will not be optimal.

In the next sections we will give a formulation of the master problem and state the subproblem as a CP model. Additionally, we will describe the global constraints that have been used in the model and the LNS method to solve the subproblem.

5.2.1 Master Problem

The Master Problem (5.1)–(5.4) is a set partitioning model that chooses tours from a set of feasible tours R. The binary decision variable $x_r, r \in R$ indicates whether a tour was selected ($x_r = 1$), $x_r = 0$ otherwise. The costs of tour r are given by the parameter c_r. All customers are included in the set C, set D the contains all depots. Set N contains all nodes, that is $N = C \cup D$. The binary parameter $\delta_{ir}, i \in N, r \in R$ equals 1 if node i is part of tour r, $\delta_{ir} = 0$ otherwise. Finally, $v_i, i \in D$ is the maximum number of vehicles in depot i. Using this notation the problem can be stated as follows:

$$\textbf{PMP: } \min \sum_{r \in R} c_r x_r \tag{5.1}$$

s. t.

$$\sum_{r \in R} \delta_{ir} x_r = 1 \qquad i \in C \tag{5.2}$$

$$\sum_{r \in R} \delta_{ir} x_r \leq v_i \qquad i \in D \tag{5.3}$$

$$x_r \in \{0,1\} \qquad r \in R \tag{5.4}$$

The objective function (5.1) minimizes the total travel distance that is the sum of travel distances of all selected tours r with costs c_r (Assumption 10). Constraint (5.2) states that every customer i must be serviced exactly once (Assumption 1). The maximum number of vehicles that start in depot $i \in D$ is restricted by Constraint (5.3) to be less or equal v_i for all $i \in D$ (Assumption 8).

The relaxed master problem *RPMP* is given by (5.1)–(5.3) and

$$x_r \geq 0 \qquad r \in R \tag{5.5}$$

This means that the integrality of the decision variable will be relaxed and x_r replaced by a continuous variable. There is no need to introduce an explicit upper bound on the decision variable in our problem because selecting values $x_r > 1$ is forbidden by Constraint (5.2).

To get the dual problem *DRMP* of *RPMP* we assign dual variables $\psi_i, i \in N$ to the i-th constraint.

$$\textbf{DRMP: } \max \sum_{i \in C} \psi_i - \sum_{i \in D} v_i \psi_i \tag{5.6}$$

s. t.

$$\sum_{i \in C} \delta_{ir} \psi_i - \sum_{i \in D} \delta_{ir} \psi_i \geq c_r \qquad r \in R \tag{5.7}$$

$$\psi_i \in \mathbb{R} \qquad i \in C \tag{5.8}$$

$$\psi_i \geq 0 \qquad i \in D \tag{5.9}$$

Due to the Theorem of Strong Duality the optimal objective function value of the *RPMP* is equal to the optimal objective function value of the *DRMP*. To decrease the objective function value of *RPMP* the objective function value of the *DRMP* must be decreased, too. This can be enforced by an additional constraint that is built like Constraint (5.7) and excludes the optimum from the current solution. It can be identified by searching for a constraint of the form of Constraint (5.7) but must be violated, i.e. by Constraint (5.10):

$$\sum_{i \in C} \delta_{ir} \psi_i - \sum_{i \in D} \delta_{ir} \psi_i < c_r \qquad r \in R \tag{5.10}$$

Thus, we search for a column with negative reduced costs for the master problem to improve its solution. The reduced cost $\bar{c_r}$ of a column r can be computed by

$$\bar{c_r} = c_r - \sum_{i \in C} \delta_{ir} \psi_i + \sum_{i \in D} \delta_{ir} \psi_i \qquad r \in R. \tag{5.11}$$

It is obvious that there are two components in Equation (5.11) to compute the reduced cost of the complete tour: Costs depending on inter-customer routes and costs connected with the selected depot. Because only one of the depots can be selected in a specific tour r, the second expression $\sum_{i \in D} \delta_{ir} \psi_i$ can be reduced to one term ψ_0^r, where the subscript 0 denotes the depot in the tour r, in the actual tour $r \in R$. Equation (5.11) can therefore be rewritten to

$$\bar{c_r} = c_r - \sum_{i \in C} \delta_{ir} \psi_i + \psi_0^r \qquad r \in R. \tag{5.12}$$

According to Desrochers et al. (1992) the cost of a route $(i_0, \ldots, i_K, i_{K+1})$, where i_0 and i_{K+1} are depot nodes, are defined by $c_r = \sum_{k=0}^{K} c_{i_k, i_{k+1}}$ and Equation (5.12) can be written as

$$\bar{c_r} = \sum_{k=0}^{K} c_{i_k, i_{k+1}} - \sum_{k=1}^{K} \psi_{i_k} + \psi_0^r \qquad r \in R. \tag{5.13}$$

In Equation (5.13) the total travel distance is computed and reduced by the dual value of all constraints connected to the customer nodes in the tour plus the dual variables of the constraints connected to the depot node. By splitting up this equation into one term for arcs between customers and another one for arcs including the depot we get

$$\bar{c_r} = \sum_{k=1}^{K-1} (c_{i_k,i_{k+1}} - \psi_{i_k}) + (c_{i_0 i_1} + c_{i_K i_{K+1}} + \psi_0^r) \qquad r \in R. \tag{5.14}$$

Finally, the marginal costs of a single arc (i, j), where $i, j \in C$, for inter-customer routes as well computed by

$$\bar{c_{ij}} = c_{ij} - \psi_i \qquad i, j \in C. \tag{5.15}$$

The resulting Equation (5.15) for the marginal cost of an arc will be advantageous in the formulation of the subproblem as we will show in Section 5.2.3.

5.2.2 Initial Solution

To solve the *RPMP* for the first time and to obtain the dual variables' values we need an initial set of columns. Therefore, we add three types of columns to the problem:

- For every combination of a depot $i \in D$ and customer $j \in C$ we add a tour depot $\rightarrow$ customer $\rightarrow$ depot: In total we get $|C| \times |D|$ tours. Costs are computed by $c_r = 2c_{ij}$. The arrival time at node j is given by $\max\{e_i + s_i + c_{ij}, e_j\}$. At the moment, it may be confusing that the earliest start time e_i and the service time s_i at every depot is not 0; but we will show the necessity of this general formulation later in Section 5.4.
- An infeasible tour starting and ending at the first depot that contains all nodes and is weighted by ∞. This guarantees a feasible solution of the *PRMP*. The arrival times for all nodes will be set to ∞, too.
- At most $\sum_{i \in D} v_i$ feasible tours obtained by the heuristic described below.

To generate a non-trivial initial set of tours we adapted the heuristic by Gambardella et al. (1999) and Cruz et al. (2013) and extended it to fit for multi-depot problems. It constructs multiple tours considering information about distances and time windows.

The heuristic is a greedy approach that assigns customers based on a score $H(d, p, s)$, where d is the start depot, p is the current position of a vehicle, and s is the successor node, that incorporates information about the spatial distribution and the time window:

$$H(d, p, s) = \frac{1}{\phi_s(\max\{t_p + s_p + c_{ps}, e_s\} - v_p)(l_s - t_p)} \tag{5.16}$$

Both factors that influence the score are in the denominator of Equation (5.16). That means that more similar nodes must have smaller values for both information to increase their chance of being picked next.

The information about the spatial distribution is regarded by angular coordinates:

$$\phi_s = (\min\{(|\theta_s - \bar{\theta}^r|, 2\pi - |\theta_s - \bar{\theta}^r|\})^\nu \tag{5.17}$$

The angular coordinate of node s is denoted by θ_s and the mean coordinate of tour r by $\bar{\theta}^r$. The mean is calculated from all nodes that are actually part of tour r. The parameter ν improves the convergence in $H(d, p, s)$ and is set to $\nu = \frac{1}{3}$ as suggested by Cruz et al. (2013).

Within the multi-depot problem we adjust the coordinates of the customers with respect to the depot. To get the proper angular coordinates for depot i $(\tilde{x}^i, \tilde{y}^i), i \in D$ we compute for every customer j the original coordinates (x_j, y_j) the new coordinates $(\tilde{x}^i_j, \tilde{y}^i_j)$ by

$$\tilde{x}^i_j = x_j - x^i_0$$

and

$$\tilde{y}^i_j = y_j - y^i_0,$$

where $j \in N$ and (x^i_0, y^i_0) represent the coordinates of depot $i \in D$.

The second and third factor to compute the score $H(d, p, s)$ in Equation (5.16) depend on the time window: At first, the ready time is considered. The more time is left to the begin of the time window of node s the smaller is the value of $H(d, p, s)$ and vice versa. Note that if the start time has been exceeded, this term will be a constant factor because the maximum of the actual arrival time and the start time is considered. Finally, the last term in the denominator of Equation (5.16) states that a small difference between the latest allowed arrival time in node s and the current time in node p will increase the value $H(d, p, s)$ c. p. Service times are neglected in the computation of $H(d, p, s)$ because the service must start in the interval but not be completed before the due date, but are included in the *max* operator to get the accurate end time at a node.

The heuristic starts with a set of $\sum_{i \in D} v_i$ empty tours, that means the position p of every vehicle is its depot. Now the $H(d, p, s)$ values for the current situation are computed and the node with the highest value for $H(d, p, s)$ is selected. The algorithm tries to assign the node to the vehicle's tour. In the assignment procedure the node's compatibility with the time window is checked as well as the loading constraint. If the arrival time is within the time windows and there is sufficient capacity left, the node is added to the tour. Otherwise, the next largest value of $H(d, p, s)$ is selected. After one node s was added to a tour the position of the current vehicle is updated ($p = s$) and the algorithm computes $H(d, p, s)$ values for the remaining nodes. This procedure is repeated iteratively until no more nodes are left for assignment or at least one customer node cannot be assigned to any tour due to capacity or time window restrictions. In the former case a feasible solution was found, in the latter case the the algorithm would signal that it has failed and terminate the overall column generation process.

5.2.3 Subproblem

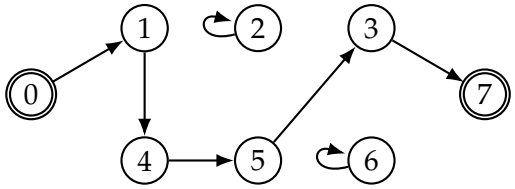
In this section we will introduce the subproblem that generates new columns for the master problem. The subproblem takes existing tours as input and tries to improve them using the

dual variable values. For vehicles that are idle the subproblem tries to find a completely new tour. To save computation time, this step is performed only for one of the vehicles that are in the same start depot. In the next section we sketch the general idea and give a COP formulation of the problem. Afterwards, we introduce additional problem-specific global constraints. The search algorithm to solve the subproblem will be described in Section 5.2.3. In our experiments we experienced that CP is only able to solve VRP instances of limited size optimally. Therefore we use a CP-based LNS and describe the problem-specific implementation.

Constraint Optimization Problem Formulation

To generate new columns for the master problem we solve the following COP that searches for additional solutions with minimal reduced costs. It assigns a successor x_i to every node i. Because the underlying problem is a VRP it is possible that other vehicles service some nodes. We therefore allow nodes to succeed themselves (with costs of 0). This means that these nodes are not part of the current tour. We duplicate the depot node (there is only a single depot because we improve existing tours) to get different depots for the start and the end of the tour. The successor of the start depot can only be a customer node or an end depot, but not the depot itself. The problem can again be stated as an directed graph $G = (N, A)$, where N denotes the set of customer and both depot nodes. Its subset $C \subset N$ includes the m customer nodes $C = \{1, \ldots, m\}$. The set A includes all weighted arcs that connect two nodes. An example for the problem graph is illustrated in Figure 5.1. The route starts in depot 0, visits customers $1, 4, 5$, and 3 and ends in depot 7. Nodes 2 and 6 are not part of the tour.

Figure 5.1: Example of COP Problem



Set $\mathcal{X}$ contains all $x_i, i \in N$. Every node i has an arrival (respectively start) time t_i and and load variable u_i. These variables indicate when the vehicle starts the service at this node and how much capacity of this vehicle is used at arrival at node i. Sets $\mathcal{T}$ and $\mathcal{U}$ contain all arrival time respectively load variables. The variable $c_i, i \in C \cup \{0\}$ gives the costs of leaving a node i, i.e. $x_i = j \implies c_i = c_{ij}$. All variables $c_i, i \in C \cup \{0\}$ are summarized in set $\mathcal{C}$.

Note that we have to distinguish between two matrices to create a CP model that can be solved efficiently. The first one represents travel times and is denoted as *time*. The values within this table are computed as euclidean distances from the customers' coordinates. The second matrix *costs* includes the distance values and the dual variable values; the entries are computed by Equation (5.15) for the customer nodes and the dual variable values of the depot Constraint (5.3) are added to the depot node. The *time* matrix is necessary to keep track of time windows and the total travel time, while the *cost* matrix is used to compute the objective function value. The entries in the *cost* matrix are denoted as $costs_{ij}$, the parameters of the *time*

matrix as $time_{ij}$. Time windows are determined by the earliest e_i and latest l_i start time for the service at node i. The parameter a_i denotes the demand in node i.

The variables' initial domains are constructed as follows: It is obvious that only tours with negative reduced costs improve the solution of the master problem. Therefore all solution with positive total costs can be excluded and all positive values are forbidden for *total*. The lower bound of the domain of *total* is obtained by summing up the minimum values of the *costs* matrix for every customer, that is $total \in \{\sum_{i \in N}(\min_{j \in N} costs_{ij}), \ldots, -1\}$. For the successor variable x_i the domain is $x_i \in \{1, \ldots, N\}$: The start depot 0 cannot succeed itself, the way back to it is forbidden for all customer nodes but it is possible to go to all customer nodes and the end depot. The domain of c_i is bounded by the minimum and the maximum element in the *cost* matrix, i.e. the costs to the customer that is closest and farthest to node i: $c_i \in \{\min_{j \in N} cost_{ij}, \ldots, \max_{j \in N} costs_{ij}\}, i \in N$. The arrival time t_i at node i must be in the time window, therefore e_i and l_i limit the domain of t_i, that is $t_i \in \{e_i, \ldots, l_i\}, i \in N$ (Assumption 5). Because a vehicle arrives without load in a node i or has just sufficient capacity to service that node in extreme cases, the domain of u_i is set up with $u_i \in \{0, \ldots, Q - a_i\}, i \in N$ (Assumption 3).

$$\textbf{Sub:} \min total \tag{5.18}$$

s.t.

$$total = \sum_{i \in \mathcal{C}} c_i \tag{5.19}$$

$$NOSUBTOUR(\mathcal{X}, \mathcal{C}, costs) \tag{5.20}$$

$$ALLDIFFERENT(\mathcal{X}) \tag{5.21}$$

$$t_i + time_{i,x_i} + s_i \leq t_{x_i} \qquad i \in N \tag{5.22}$$

$$u_i + a_i = u_{x_i} \qquad i \in N \tag{5.23}$$

$$ARC\_ELIM\_BY\_COST(\mathcal{X}, \langle \psi_0, \ldots, \psi_{(n-1)} \rangle, costs) \tag{5.24}$$

$$ARC\_ELIM\_BY\_LOAD(\mathcal{X}, \mathcal{U}, \langle a_0, \ldots, a_{(n-1)} \rangle, Q) \tag{5.25}$$

$$LOAD\_ELIM(\mathcal{X}, \mathcal{U}, \langle a_0, \ldots, a_{(n-1)} \rangle) \tag{5.26}$$

$$REDUCE\_TIME\_WINDOWS(\mathcal{X}, \mathcal{T}) \tag{5.27}$$

The objective function (5.18) is to minimize the reduced cost term *total*. It is computed in Constraint (5.19) as the sum of leaving all nodes but the end depot. The global constraint *NOSUBTOUR*($\mathcal{X}$,*cost*) (Constraint (5.20)) is madantory to build a complete tour (without subtours, Assumption 9) and assigns the values to $c_i, i \in C \cup \{0\}$ from *cost* depending on the successor of $x_i, i \in C \cup \{0\}$. *ALLDIFFERENT*($\mathcal{X}$) (Constraint (5.21)) enforces distinct values for the variables in $\mathcal{X}$. Constraints (5.22) and (5.23) update the arrival time and the load of the vehicle (Assumptions 3 and 5). We have not modeled the maximum load and the start of the service explicitly this time because the initial domains of the variables u_i and t_i were set up to guarantee feasibility. Constraints (5.24)–(5.27) are global constraints to improve propagation, where ψ_i are the values of the dual variables. Constraint (5.27) is already known from Section 4.3.2 but we have to slightly modify it to deal with the current model: Because no predecessor variable is present (in a path the start depot has no predecessor) the set $\mathcal{Y}$ that is included in

the explanation in Section 4.3.2 must be constructed from the successor variables $\mathcal{X}$ by iterating over all but the current variable and checking if the current variable can be a successor. All global constraints will be explained in detail below.

Problem-Specific Propagators

To shrink the search space and speed up search we have implemented problem-specific global constraints. These constraints are customized global constraints and not included in the Global Constraint Catalog (Beldiceanu et al., 2015). We explain the underlying idea and present the technical details of the propagation steps. Throughout this section we use the following notation: The superscripts *min* and *max* stand for the smallest and largest value in the domain of a variable. Furthermore, we will state under which conditions a propagator is *scheduled*—that means it is added to the list of operations that will be carried out before the next branching—and the stop criterion, that means the propagator is not considered anymore because it will not prune any domain.

Eliminate Arcs By Load Information

The global constraint *ARC_ELIM_BY_LOAD($\mathcal{X}$,$\mathcal{U}$,$\langle a_0, \ldots, a_{(n-1)} \rangle$,Q)* filters domains of the successor variable according to the current load of a vehicle and the demand of potential successors in a pick-up problem (Assumption 3 from Section 2.1) and was inspired by Rousseau et al. (2004). Consider a vehicle that is currently in a customer node i and can choose the next customer out of the domain of variable x_i. If the current load in node i u_i plus the demand in the next node a_j exceeds the capacity the solution will be infeasible. Therefore this successor can be pruned by the algorithm. Formally:

$$u_i + a_j > Q \Rightarrow x_i \neq j \qquad\qquad j \in dom(x_i), i \in C : x_i \text{ is unfixed.} \tag{5.28}$$

This propagation algorithm is only active if capacity is scarce ($\sum_{i \in C} a_i > Q$) because there would be no pruning otherwise. If it is active, the constraint is scheduled every time the bound of one load variable's domain changes. During search more of the vehicles' capacity will be used and therefore additional nodes will be excluded as successors. Propagation would not benefit from monitoring the successor variable additionally, because fixing a successor results in fixing the corresponding load variable (and changing its bounds). The global constraint *ARC-ELIM-BY-LOAD($\mathcal{X}$,$\mathcal{U}$,$\langle a_0, \ldots, a_{(n-1)} \rangle$,Q)* will be called until all successor variables are assigned.

Shrink Domains of Load Variables

As we have shown that arcs can be eliminated based on the bounds of the load variables' domains, it is crucial to get tight bounds on the domain of the load variable $u_i, i \in N$. We introduce the global constraint *LOAD_ELIM($\mathcal{X}$,$\mathcal{U}$,$\langle a_0, \ldots, a_{(n-1)} \rangle$)* to shrink the domains (Rousseau et al., 2004). In every node we use three different ideas based on the current node j, the successor

nodes $k \in dom(x_j)$ and the preceding nodes $i \in Y_j$. All explanations are based on a pick-up problem again (Assumption 3).

It is obvious that in every node j the load when leaving the node is at least equal to the demand in the node, that is $u_j \geq a_j, j \in C$. By considering the successors $dom(x_j)$ we can reduce the upper bound of the domain in Equation (5.29):

$$u_j \leq \max_{k \in dom(x_j)} \{u_k^{max} - a_k\} \qquad j \in C \tag{5.29}$$

All values that are larger than $\max_{k \in dom(x_j)}\{l_k^{max} - a_k\}$ are pruned from $dom(u_j)$. For an example, consider a vehicle with a capacity of 100 units that is in node j and has three nodes as potential successors and exactly one shall be selected (no pruning has been done on the current load variable's domain). Because of prior assignments each of the customers' successor is fixed. Therefore, it is known that the maximum capacity at this node can be 50, 60, and 70 units. Assuming that the current demand rate equals 10 in all nodes, the vehicle's remaining capacity can be at most 60 units when leaving node j.

Next, we check all predecessors for the lower bound. We will show below that the problem cannot be modeled with a predecessor variable as it was applicable in the TSP in Section 4.3.2. Therefore, we iterate over all other variables and check if node j can be a successor to construct a set y_j that contains all predecessors of customer j. The idea works analogously to the upper bound: For all possible predecessors $i \in Y_j$ of node j we check for the minimum in the domain (including the demand in i) in Equation (5.30):

$$u_j \geq \min_{i \in Y_j}\{u_i^{min} + a_i\} \qquad j \in C \tag{5.30}$$

Again, the values smaller than $\min_{i \in Y_j}\{u_i^{min} + dem_i\}$ are deleted from $dom(u_j)$. For, the load of a vehicle that leave node j can not be less than its load at the customer it visited before.

Compared to the previous global constraint *LOAD_ELIM*($\mathcal{X}$,$\mathcal{U}$,$\langle a_0, \ldots, a_{(n-1)}\rangle$) prunes the load variables' domains. It is scheduled every time a successor variable's or the bounds of a load variable's domain changes as long as not all load variables are fixed.

Forbidding Subtours

In the previous chapter we dealt with problems where all nodes had to be part of the tour. Therefore we could use the standard global constraints *TOUR*($\mathcal{X}$) implemented in *Gecode* to avoid subtours. However, this constraint necessitates visiting all nodes which does not match the current problem. To address the issue that only a subset of nodes is part of the tour we set up the *NOSUBTOUR*($\mathcal{X}$,$\mathcal{C}$,*costs*) constraint: It ensures that there is one path from the start depot (source) to the end depot (sink). Every node is either included in this path or a successor of itself, i.e. not part of the tour.

The fact that there exists a path is derived from the characteristics of the problem. A start depot cannot succeed itself, e.g. it must have a customer node or an end depot as its successor. The selected customer node cannot succeed itself because all successor variables must be pairwise distinct and it already has been selected as a successor for the depot. Therefore, another

customer node or an end depot must be its successor. This requirement applies to all nodes that are part of the tour and guarantees a path between two depots.

The underlying propagation algorithm to forbid subtours of two or more nodes is based on a global constraint introduced by Caseau and Laburthe (1997) and its general idea was described in Section 3.5. The implementation of the propagation steps is as follows:

For every variable it is checked if it was assigned. If the assignment is $x_i = i$, the algorithm skips further checks as the variable fulfills the requirements. Otherwise, for every variable assigned to $x_i = j : i \neq j$ a path is build using the direct successors of the variable and the upcoming successors until the algorithm detects an unassigned variable, which is the current end of the path ending in node end_i. To avoid subtours the start node of the path is excluded from the end node's domain: $x_{end_i} \neq i$. All nodes on the path are marked by saving the end node end_i to avoid the inspection of the same path more often. For every node on the path no pruning is necessary because all successor variables had been assigned to a single value.

To accelerate the propagation process the algorithm jumps directly to the end of the path if it detects that a node has been marked as checked instead of inspecting the path again. If a path is extended by the current node, the information about its end must be updated. If the arc is added at the end, the end_i information on each node of the path will be changed to the new end_i. If it is the new start node of the path, it copies the end information from the first node in the previous iteration.

This constraint is posted if the problem consists of more than three customer nodes: Returning to the start depot is forbidden by definition and one node is connected to the end depot. A third node could not build a subtour because it cannot head to any depot node and the node after the start depot is assigned as a successor; a second assignment is forbidden by *ALLDIFFERENT($\mathcal{X}$)* (posting this constraint is mandatory for *NOSUBTOUR($\mathcal{X}$,$\mathcal{C}$,costs)*). Therefore, only problems with at least four customer nodes can include subtours.

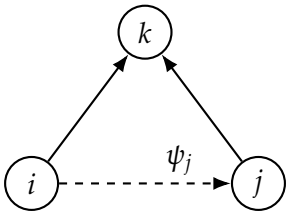
The pruning is only executed every time a successor variable has been fixed and ignores changes in the domains of the remaining variables as these information do not influence pruning. If all but one variables are fixed, the constraint will be ignored by the solver: The remaining variable would have to be at the end of a path to build a subtour (with its own path) but exactly this connection has been forbidden in the previous iterations.

Cost-Based Arc Elimination

The global constraints *ARC_ELIM_BY_COST($\mathcal{X}$, $\langle \psi_0, \ldots, \psi_{(n-1)} \rangle$,costs)* is a constraint that uses cost-based propagation as suggested in Focacci et al. (1999a). Information on the dual variables from the master problem are used for propagation. The constraint has the effect that the triangular inequality is not valid any more (Assumption 1 from Section 2.1) because the cost of a tour do not rely on the travel distance between two nodes but also on the dual variables' values Rousseau et al. (2002a, 2004):

Consider the arc $i \rightarrow j$ in Figure 5.2: If it is cheaper for all other customers $k \in dom(x_j)$ to go immediately from i to k at cost $costs_{ik}$ than to travel $i \rightarrow j \rightarrow k$ (at cost of $time_{ij} + time_{jk} - \psi_j$), arc $i \rightarrow j$ will never be part of the optimal tour and can be eliminated, i.e. $x_i \neq j$.

Figure 5.2: Graphical Illustration of the *ARC_ELIM_BY_COSTS*($\mathcal{X}$, $\langle \psi_0, \ldots, \psi_{(n-1)} \rangle$,*costs)* Constraint (Rousseau et al., 2002a)



Search Method

The search for a solution within the subproblem is done by a local search heuristic called Large Neighborhood Search (Shaw, 1998) and was introduced in Section 3.6. This class of heuristics starts from a feasible solution and tries to improve the objective function value by exploring neighboring solutions. If such a better solution was found, search will move to that neighbor and start again from its new position. A neighborhood $N(x) \subset X$ of a solution $x \in X$ is defined by all solutions that can be built from a feasible solution by applying an elementary operation. Search terminates if no elementary solution can improve the solution.

A neighbor is found by *freezing* most of the variables to the values in the current solution but relaxing some of them. The algorithm searches for a new combination of nodes that incurs less costs. Because of dependencies between the nodes in our problem a solution can be found in many cases just by applying the propagation step, otherwise the resulting problem can be solved by the standard B&B search engine. Because there are only a few variables whose values can be changed, the algorithm can find a solution very efficiently. The number of non-fixed variables is a parameter that increases over time. If no better solution can be found in a given number of iterations, the number is increased. By this modification it is possible to escape from local minima and further improve the solution. After the LNS found a better solution the number of non-fixed variables is reset to the minimum value. Search terminates after a pre-defined number of iterations.

In our problem freezing a node means that a connection between node i and its successor x_i is enforced in the neighboring solutions. However, a new successor can be assigned to relaxed nodes. The start solution for LNS is a single tour taken from the *PRMP*. The search procedure alternates between two procedures to find a new neighbor. The first one is based on a random selection of nodes that should be relaxed in the neighboring solution; nodes are picked with the same probability. To increase the probability of a swap we adapt the concept of *relatedness* by Shaw (1998). For every randomly chosen node that should be relaxed a second node is added that is *related* to the first one: Nodes are more related, if they are served by the same vehicle and are close to each other. Thus, the relatedness $Rel(i, j)$ of two node i and j can be expressed by

$$Rel_{ij} = \frac{1}{time_{ij} + O_{ij}}. \tag{5.31}$$

The term O_{ij} equals 1 if both nodes i and j are served by the same (the current) vehicle, 0 otherwise. The most related node j to the randomly selected node i ($arg\,min_j\ Rel_{ij}$) is supplementary added to the set of relaxed nodes.

The second procedure to generate neighboring solutions is called *Cost Impact Guided LNS* (Lombardi and Schaus, 2014). It captures information about the problem and ranks variables on their impact on the objective function. Variables with a higher impact on the objective should be relaxed more probably (Carchrae and Beck, 2009). The impact is estimated by the difference between the lower bounds of a problem with n fixed variables compared to the lower bound with $n+1$ fixed variables.

To evaluate the impact Lombardi and Schaus (2014) introduced a *dive*, that is the progressive re-application of the current solution in a rearranged order: Let π be a permutation of the variables in $\mathcal{X}$ and let k be the position of x_i in π. The term $lb_{\tau_{\pi,k}}$ denotes the lower bound that can be found by applying propagation algorithms, where π is the current order and the first k variables are fixed to their value in the current solution. The the cost impact of x_i with respect to a solution ϕ is the quantity:

$$\mathcal{I}(x_i, \phi, \pi) = lb_{\tau_{\pi,k}} - lb_{\tau_{\pi,k-1}} \tag{5.32}$$

where

$$S(\tau_{\pi,k}) = \{x_{\pi(i)} \mid j = 0 \dots k\} \tag{5.33}$$

$$\tau_{\pi,k}(x_i) = \sigma(x_i) \qquad x_i \in S(\tau_{\pi,k}) \tag{5.34}$$

Clearly, the cost impact $\mathcal{I}(x_i, \phi, \pi)$ is the difference in the lower bounds of the problem with $k-1$ and k fixed variables. $\tau_{\pi,k}$ forces the first k variables to take their value from the solution ϕ.

Next, the algorithm computes the mean value for the cost impact of variable x_i over a set of dives Π:

$$\mathcal{I}(x_i, \phi, \Pi) = \frac{1}{|\Pi|} \sum_{\pi \in \Pi} \mathcal{I}(x_j, \phi, \pi) \tag{5.35}$$

The number of dives $|\Pi|$ is restricted and the permutation π is based on a uniform distribution.

The selection of variables x_i that should be relaxed is based on a score s_i:

$$s_i = 0.5 \cdot \mathcal{I}(x_i, \phi, \Pi) + 0.5 \cdot \frac{1}{|X|} \sum_{x_j \in X} \mathcal{I}(x_j, \phi, \pi) \tag{5.36}$$

The convex expression in Figure (5.36) assigns a higher score to variables with big impact on the objective function. These variables are preferably selected as they should be in the best position and relaxing them during search supports this necessity. The uniform quantity term guarantees that even variables with zero impact can be selected as free variables.

The *Cost Impact Guided LNS* is summarized in pseudocode in Algorithm 13.

Because *Cost Impact Guided LNS* causes a lot of computational overhead it is only repeated after a given number of iterations. In all other iterations the random-based search operator described above is used.

Algorithm 13: Cost Impact Guided LNS (Lombardi and Schaus, 2014)

```
1 Assign a score s_i to each variable x_i
2 r := Σ_{x_i} s_i
3 while variables for relaxation left do
4 |   Pick a random value v in [0, r]
5 |   forall the not selected x_i do
6 |   |   v := v − s_i
7 |   |   if v ≤ 0 then
8 |   |   |   r := r − s_i
9 |   |   |   Select x_i for relaxation and go back to line 2
```

5.3 Branch&Price Algorithm

The results from the column generation model must not necessarily be integer values. In these cases no clear assignment can be made because more than one vehicle is assigned to serve some customers which may be impossible (or at least undesirable) in practical situations. Furthermore, for the rolling horizon approach it is substantial to have a precise arrival time at a customer node. Therefore we extend the solution method to a B&P approach.

In every node of the B&B tree the column generation algorithm produces a solution that is a lower bound for the *PMP* (because it may be infeasible for the reasons mentioned above). This lower bound *LB* is compared to the best known feasible solution—the upper bound *UB**, initially $UB^* = \infty$. If $LB > UB^*$, this part of the tree can be pruned because applying more restrictions will generally worsen the solution. Otherwise the algorithm uses the branching rules described below. Every time a feasible and integer solution with an objective smaller than *UB** was found the upper bound is updated.

It is known that branching on the decision variables of the *RPMP* model x_r would change the structure of the subproblem as single tours must be enforced or forbidden which increases the complexity of the pricing problem (Desaulniers et al., 2004). To address this issue we use two common branching schemes from literature (Bettinelli et al., 2011; Luo et al., 2014):

Branching on the number of vehicles: This branching rule creates two nodes in the B&B tree that modify the number of vehicles that start at a depot. The (possibly fractional) number of vehicles that depart from a depot $d \in D$ can be computed by $m'_d = \sum_{r \in R} \delta_{dr} x_r$. The brancher chooses the depot $i \in D$ whose fractional part of m'_i is closest to 0.5 and constrains the problem in one child node to use more than $\lceil m'_i \rceil$ vehicles from this depot in one node and less than $\lfloor m'_i \rfloor$ vehicles in the other node of the search tree. To include this condition in the master problem, we replace Equation (5.3) by by Equations (5.37a) and (5.37b)

$$\sum_{r \in R} \delta_{ir} x_r \geq \underline{v_i} \qquad i \in D \tag{5.37a}$$

$$\sum_{r \in R} \delta_{ir} x_r \leq \overline{v_i} \qquad i \in D \tag{5.37b}$$

For the first child node we set $\underline{v_i} = \lceil m'_r \rceil$ and leave $\overline{v_i}$ unchanged, while in the second node we constrain $\overline{v_i} = \lfloor m'_r \rfloor$ and keep the original value for $\underline{v_i}$. The values of the the other depots $i \in D \setminus \{d\}$ remain unchanged.

Incorporating these constraints in the model does not change the general problem structure but tightens the bound for the number of vehicles in every depot (Formerly it was $[0..v_i], i \in D$). Note that adding a constraint to the problem implies an extra dual variable. Thus, the computation of the reduced costs must be adapted to

$$\bar{c_r} = c_r - \sum_{i \in C} \delta_{ir} \psi_i + \sum_{i \in D} \delta_{ir} \omega_i - \sum_{i \in D} \delta_{ir} \rho_i \tag{5.38}$$

where ψ, ω_i, and ρ_i are the dual variables of Constraints (5.2), (5.37a), and (5.37b) respectively. The values in the cost matrix for the subproblem must be modified accordingly.

Branching on arcs: Another branching is applied on the arcs in the problem graph. A part of these arcs is forbidden. The rule identifies the customer $i \in C$ that is split among the largest number of tours. The set of tours is ordered according to the index of the tour and split into two parts. It is forbidden to visit the first half of the outgoing arcs in one child node and the second half in the other child node. This can be achieved by setting the distance to a sufficient large number. These nodes are pruned in the preprocessing step of the constraint solving algorithm because that instantiation would be infeasible. Thus, there is no need to modify the CP model, it is sufficient to update the *cost* matrix. If the number of tours is odd, the second child node contains one more node than the first one.

The search tree is traversed in a depth-first manner and explores the subtree that provides the better LB first. The branching schemes are applied in the order as they are presented above.

5.4 Rolling Horizon Framework

To extend the algorithm to deal with a rolling planning horizon we have introduced two ideas that consider fixed nodes and announced visits.

Fixed nodes have already been visited and their position in a tour will not change, i.e. all arcs between these nodes are fixed. The set F_k contains all fixed nodes of a vehicle k, i.e. the start depot of the tour, all nodes that have been visited by the vehicle in the past, and—if the vehicle is not in a customer node at the moment of reoptimization—the node that will be reached next. Now, we want to neglect these fixed nodes in the search process. Therefore, the last fixed node in a tour is replaced by a new "artificial" depot node. This node has the coordinates of the last node in the set F_k, that is the depot respectively customer that was visited by the vehicle at the moment of optimization or the node that will be reached next. It acts as a new start depot for the current tour in the reoptimization, all other fixed nodes will be ignored. However, one must consider that the vehicle's capacity and travel time may be influenced by the fixed nodes. To avoid infeasible solutions according to these parameters the characteristics of the new depot node must be adapted. The demand of this new depot node k equals the summed up demand of the fixed node in the vehicle's tour:

$$a_k = \sum_{i \in F_k} a_i$$

Thus, the load is the same as in a situation where fixed nodes had been included in the tour; the capacity will not be exceeded.

The begin of the time window for the new depot node is more complex: In general, it can not be earlier than the original start date. Furthermore, service cannot start before the vehicle arrives from the previous node. Finally, the vehicle cannot start its trip to the next customer until the new tour is known in the moment of reoptimization t^{opt}. In the former two cases the service time must be considered; we add the origin service time s'_k in node k to the earliest start time and set the new value $s_k = 0$. The following equation captures all characteristics and can be used to compute the correct time:

$$e_k = \max\{e_k + s'_k, t_{k-1} + c_{k-1,k} + s'_k, t^{opt}\}$$

We include the service in the start time and replace it with 0 afterwards because service starts immediately. If we would add the original service time to t^{opt} but the vehicle had arrived earlier, the vehicle would be idle and wait for the service. Consider the following example: A vehicle arrives at the last fixed node at $t^a = 90$, the service lasts 20 time units, reoptimization is done at $t^{opt} = 100$. The vehicle can start its trip to the next customer already at $t^d = 110$, adding the service time to t^{opt} would postpone the vehicle incorrectly ($t^d = 120$).

As the adapted node acts as a starting point in the next step, a new depot is added to the model and therefore the total number of depots increases. Because the number of vehicles is constant over the entire planning horizon, we must update the number of vehicles available at a depot. Due to the facts that at most one vehicle can serve a customer node (Equation (5.7)) and that one vehicle is currently in this node, in every "artificial depot" must be exactly one vehicle. If a vehicle is placed at an "artificial depot", the number available at its original depot must be reduced by one. This assures that the total number of vehicles stays constant.

Because of Constraint (5.2) in the *RPMP* it is possible that a vehicle is in an artificial depot but no tour is selected by the set partitioning problem relaxation. Vehicles stay in the last fixed nodes until they are scheduled again. To ensure a complete tour for each vehicle, e.g. that the vehicle arrives at its end depot, a postprocessing step collects all vehicles and sends them to their original depot at the end of the planning horizon. All these changes can be done by postprocessing the current solution without any modification to the model.

The announced visits are identified by the postprocessing step, too. The algorithm creates a set P which contains all nodes that could cause penalties of c^p if their arrival time deviates from the announced time by an amount larger that the tolerance value w_i. To incorporate that the Constraint (5.39) is added to *Sub*:

$$|t_i - t'_i| > w_i \rightarrow p_i = c^p \qquad i \in P, \tag{5.39}$$

where p_i is the penalty value that results from visiting node $i \in N$ too late or too early (without loss of generality, we assume that the same penalties apply for both cases). Additionally, Equation (5.19) is replaced by the new objective function

$$total = \sum_{i \in C \cup \{0\}} c_{i,x_i} + \sum_{i \in P} p_i. \tag{5.40}$$

5.5 Computational Study

In this section we describe the results of the test that we have carried out to evaluate the performance of the algorithm. The problem sets are the same as in Section 4.5. All results were obtained on a PC with a 2.67 GHz Duo Core CPU and 8 GB RAM running *Windows 7*. The framework was written in *Visual C*$^{++}$2012. To solve the *CP* problems *Gecode 4.3.2* (Schulte et al., 2015) was used and *Gurobi 6.0* (*www.gurobi.com*) to solve the master problem in CG.

The search parameters for the LNS were set to at least 4 and at most 8 customer nodes to keep the instances for the B&P solver tractable. For a single problem LNS was allowed to use at most 100 seconds of computation time. The cost-based operator was executed in every fifteenth iteration and five dives were performed. Duration of CG was limited to 500 seconds, the entire B&P search to 2,000 seconds. At most three (or less, if no sufficient number of solutions was found) new columns were added to the master problem from every solution of the subproblem.

As we did in Section 4.5 we will describe the results for the rolling horizon setting first and present findings on the static setting afterwards. Both cases will be compared to results found by the GA in Chapter 4.

5.5.1 Rolling Horizon Setting

The experimental setting was the same as the setting to evaluate the GA in Section 4.5. We used the Solomon and Cordeau instances again. The customer sets of the Solomon instances were divided into 3, 5, and 8 steps; the number of steps for the Cordeau instances was given in Table 4.1 on page 85. The algorithm was run twice and the results presented below are averaged. For the first results penalty terms are absent.

Table 5.1 on the next page shows the percentage of instances that could be solved in each step (aggregated from two runs (exceptionally not averaged)). Column *Cust.* gives the number of customers in the instance, *Type* signals if the instance is clustered, randomly generated or mixed , and *TW* specifies the width of the time window at the depot. We say that an instance was solved if a feasible tour could be constructed. Thus, a feasible solution obtained from the initial heuristic was sufficient, no improvement from further parts of the algorithm was required to mark a step as solved. The reason for failures in a step was as follows: The initial heuristic took the solution from the former step as a basis for time windows and capacity. This solution could be poor according to travel distances or a large number of vehicles was used. This made it difficult to construct an initial solution. Therefore, a solution could be found for all instances in the first step but the percentage decreased with every additional step. Remember that the overall algorithm was aborted if the initial heuristic could not find a solution; the depot-customer-depot tours and the infeasible (too expensive) tour were not sufficient to build a solution if the heuristic fails. To stress the influence of the previous solution we give the percentage of instances that could be solved in at least one of the two runs: It can be seen from Table 5.2 on the facing page that most of the instances could be solved. More type 2 instances (with larger scheduling horizon) can be solved can be solved than type 1 instances. This is the same as in the GA. In general, the B&P approach is able to solve more instances with many vehicles but fails more often in the smaller instances.

Table 5.1: Percentage of Solved Solomon Instances in Each Step

Cust.	Type	TW	Step 0	1	2	3	4	5	6	7
25	C	1	100%	85.7%	78.6%					
		2	100%	75%	75%					
	R	1	100%	68.8%	56.3%					
		2	100%	91.7%	83.3%					
	RC	1	100%	83.3%	58.3%					
		2	100%	91.7%	83.3%					
50	C	1	100%	85.7%	64.3%	57.1%	42.9%			
		2	100%	62.5%	50%	50%	37.5%			
	R	1	100%	66.7%	50%	41.7%	33.3%			
		2	100%	90%	90%	70%	60%			
	RC	1	100%	40%	40%	40%	40%			
		2	100%	100%	80%	70%	70%			
100	C	1	100%	91.7%	66.7%	58.3%	16.7%	0%	0%	0%
		2	100%	90%	60%	50%	40%	0%	0%	0%
	R	1	100%	63.6%	63.6%	36.4%	36.4%	27.3%	27.3%	27.3%
		2	100%	90%	90%	90%	90%	80%	80%	80%
	RC	1	100%	100%	87.5%	50%	50%	50%	25%	25%
		2	100%	100%	80%	80%	80%	80%	70%	60%

Table 5.2: Percentage of Solved Solomon Instances in at Least One Run

Cust.	Type	TW	Step 0	1	2	3	4	5	6	7
25	C	1	100%	100%	100%					
		2	100%	100%	100%					
	R	1	100%	75%	75%					
		2	100%	100%	85.7%					
	RC	1	100%	100%	100%					
		2	100%	100%	100%					
50	C	1	100%	87.5%	62.5%	62.5%	62.5%			
		2	100%	80%	75%	75%	75%			
	R	1	100%	66.7%	66.7%	50%	50%			
		2	100%	100%	100%	100%	100%			
	RC	1	100%	60%	60%	60%	60%			
		2	100%	100%	100%	100%	100%			
100	C	1	100%	100%	83.3%	83.3%	16.7%	0%	0%	0%
		2	100%	100%	80%	60%	60%	60%	60%	60%
	R	1	100%	66.7%	66.7%	50%	50%	50%	50%	50%
		2	100%	100%	100%	100%	100%	80%	80%	80%
	RC	1	100%	100%	100%	75%	75%	75%	25%	25%
		2	100%	100%	100%	100%	100%	100%	80%	80%

We give the computation time in Table 5.3. It is obvious that computation takes longer but there is no unique trend that depends on the number of steps. Two reasons can be named for this behavior: The computation time depended on the number of columns that has been generated. The more columns were added the more time was consumed to solve the problem. Secondly, branching influenced the solution time. If branching was applied, the CG procedure was called in every node of the search tree. Therefore, computing integer solutions for these instances took much longer than for instances where the CG procedure returns integer solutions and no branching is required.

The algorithm was terminated in about 10% of all instances and steps because the time limit has been exceeded. For the smallest instances the algorithm was never aborted because of the time limit. The highest value (50%) was on *RC* instances of type with 50 customers. LNS consumed the major part of time in the CG, only 0.1% of the total computation time was used to solve the master problem. Therefore, we had not implemented techniques to manage the pool of columns, e.g. to remove columns that have not been used for a given number of iterations.

Table 5.3: Computation Time for Each Step in Solomon Instances (in Seconds)

			Step							
Cust.	Type	TW	0	1	2	3	4	5	6	7
25	C	1	178.8	1,961.9	801.8					
		2	165.1	1,800.9	614.9					
	R	1	119.1	661.5	1,315.9					
		2	162.8	1,129.3	950.1					
	RC	1	215.4	1,565.6	4,285.0					
		2	215.7	688.1	560.2					
50	C	1	214.1	1,038.0	917.3	1,237.8	1,066.0			
		2	214.9	1,305.6	2,359.0	1,470.8	1,865.7			
	R	1	140.4	1,074.1	1,402.3	607.0	1,118.0			
		2	178.1	1,228.4	768.9	1,846.4	1,372.2			
	RC	1	457.4	1,753.8	2,242.0	1,795.3	1,906.0			
		2	347.6	988.1	1,303.5	1089.1	515.6			
100	C	1	174.6	783.0	740.0	786.7	574.0	—	—	—
		2	743.4	1,574.4	1,602.5	1,085.0	1,095.3	628.3	647.5	1,045.3
	R	1	574.3	1,537.7	820.4	770.8	866.3	891.3	820.3	1,638.0
		2	427.6	759.6	578.8	806.8	738.6	690.4	701.1	665.8
	RC	1	383.9	1,138.0	713.3	745.8	900.3	775.5	891.0	904.0
		2	525.0	639.7	622.6	628.9	885.1	810.5	851.0	763.3

The branching procedure was called in 76.3% of all steps because no integer solution has been found. The first branching rule (branch on vehicle number in depots) was called on average 2.4 times in 25 customers instances, 0.85 times in 50 customers instances, and 1.0 times in 100 customers instances. The second branching rule (branching on arcs) was only applied if the number of vehicles that start at all depots was integer. It was used averaged 1.2 times in the search trees of 25 and 50 customer instances and 1.1 time for 100 customer data sets. The average tree height was 2.9, the highest values could be found for the instances *RC106* (height of 14), *RC107* (18), and *RC108* (12).

The effect of rising computation times can be explained by the problem size: The number of nodes that were added to the problem is predetermined by the number of steps. In contrast, the number of fixed nodes resulted from the computation. The averaged numbers are summarized in Table 5.4 on the following page. In column *New* the number of customer nodes that were added in this step is given, column *Fixed* includes the number of nodes that is fixed in the current step (from all previous steps). Step 0 is not presented in the table because there cannot be any fixed node by definition. New (artificial) depot nodes are not included in the numbers as they are customer nodes, too.

Table 5.4: Average Number of New and Fixed Nodes in Solomon Instances

		Step													
		1		2		3		4		5		6		7	
Cust.	Type	New	Fixed	New	Fixed	New	Fixed	New	Fixed	New	Fixed	New	Fixed	New	Fixed
	C	8	6.1	7	15.7	—	—	—	—	—	—	—	—	—	—
25	R	7.9	5.5	7	13.7	—	—	—	—	—	—	—	—	—	—
	RC	7.8	6.7	7	13.7	—	—	—	—	—	—	—	—	—	—
	C	10	9.1	10.2	18.3	10	27.9	7.7	37.6	—	—	—	—	—	—
50	R	10.1	5.5	9.7	14.4	10.1	23.7	7.9	33.6	—	—	—	—	—	—
	RC	10	5.6	10.2	13.9	9.5	23.4	8	34.4	—	—	—	—	—	—
	C	12	9.3	12	19.1	12	29.4	12	39.9	12	50.1	12	62.6	14	74.9
100	R	11.9	6.3	11.8	14.2	12	23.8	12.2	34.9	11.5	46.4	12	60.1	14	70.9
	RC	11.6	7.4	12.2	14.5	12.2	23.5	12	34.4	11.4	45.4	12	58.4	14	70.6

Table 5.5: Comparison of Objective Function Values of GA and BAP for Solomon Instances (in Percent)

Cust.	Type	TW	Step 0	1	2	3	4	5	6	7
25	C	1	50.1	75.9	131.3					
		2	40.6	120.0	124.8					
	R	1	22.6	60.8	86.0					
		2	19.0	47.3	72.2					
	RC	1	40.3	135.8	269.7					
		2	38.3	121.5	123.1					
50	C	1	62.9	215.7	182.4	205.6	295.2			
		2	42.4	78.6	59.1	126.3	122.8			
	R	1	19.4	81.0	84.2	90.4	161.6			
		2	40.7	54.4	70.4	68.4	73.2			
	RC	1	103.4	237.7	150.8	214.6	329.0			
		2	79.9	141.2	150.9	176.9	205.8			
100	C	1	10.1	52.5	175.2	223.9	—	—	—	—
		2	78.8	109.2	78.5	92.7	113.7	108.5	134.6	194.1
	R	1	27.8	52.6	138.7	138.8	184.8	—	—	—
		2	67.7	59.3	80.3	85.3	107.2	112.3	129.0	—
	RC	1	36.0	103.1	140.1	178.3	230.2	—	—	—
		2	44.5	99.9	115.5	109.0	134.9	149.9	85.0	—

In many cases the number of new nodes is larger than the growth in fixed nodes and therefore the number of nodes that require a decision increases. Consider the 100 customers instances as an example: The problem starts with 12 customer nodes (not shown in the table) and in the last step there are more than 24 nodes that are not fixed. Thus, solving the subproblem consumes more time. This effect is intensified by the fact that there are more tours that must be checked in later steps because the algorithm checks tours for every depot only once and there may be tours with more than one vehicle. With a higher number of vehicles that are on tour the subproblem solver will be called more often.

The mean objective function values of the B&P algorithm for every instance and every step are given in Tables B.1–B.3 in Appendix B. Additionally, we have added the mean objective of the GA to evaluate the performance. The column *Gap* is computed by $(GA/BAP) - 1$ and expresses the amount the GA solution is worse than the solution from the B&P approach for this instance. The higher the gap the higher the difference between both solutions. We have summarized the results for every step and every type in Table 5.5.

The B&P procedure performed much better than the GA which warranted the longer computation times. In general, the gap values increased with the number of steps. The increase between the first steps was higher than the rise between later steps. At least, in three instance with 25 customers the GA found cheaper results by mean than the B&P method (Table B.1 in Appendix B). All this confirmed our findings that the GA performed better on small instances.

To evaluate the performance of the approach most precisely we computed an optimal solution for one instance of each type with 25 and 50 customers. We solved only the first step for these instances. For, the results for the further steps depend on the solution of the former

Table 5.6: Comparison of Optimal Solution and Both Heuristic Approaches for Solomon Instances

25 customers					
Instance	Opt	Best GA	Gap GA	Best BAP	Gap BAP
C101	64	107	67.2%	78	21.9%
C201	81	104	28.4%	88	8.6%
R101	199	218	9.5%	252	26.6%
R201	178	225	26.4%	212	19.1%
RC101	164	192	17.1%	168	2.4%
RC201	140	196	40.0%	168	20.0%
50 customers					
Instance	Opt	Best GA	Gap GA	Best BAP	Gap BAP
C101	102	—	—	146	43.1%
C201	79	99	25.3%	90	13.9%
R101	231	273	18.2%	262	13.4%
R201	188	210	11.7%	201	6.9%
RC101	198	463	133.8%	241	21.7%
RC201	200	383	91.5%	221	10.5%

step and as these results will be different for all approaches a comparison would not lead to a properly interpretable result. The input data for the MILP used herein were the same as for the GA and B&P approach. We used the model (2.1)–(2.20), the modeling language *AMPL* (Fourer et al., 2003), and *Gurobi 5.6.3* as solver. Because only the solution of the first step was computed, penalty-related parameters are set to 0.

Table 5.6 compares the best solution found by both heuristic approaches with the optimal solution. Column *Opt* is the optimal solution of the Mixed Integer Program, *Best GA* and *Best BAP* give the gaps that are computed by $(\textit{Best GA}/\textit{Opt}) - 1$ and $(\textit{Best BAP}/\textit{Opt}) - 1$, respectively.

It can be seen that the gap between the optimal solution and solutions of the B&P approach is still large, even if it clearly outperforms the GA. 25 customers instances are by mean worse 16.4% than the optimal solution, the larger instances with 50 customers 18.3%. The span of the B&P is smaller and never exceeds 50%, while for the GA the objective is more than twice the optimal solution.

We have tested this algorithm on the data set of Cordeau as well. Like the GA the B&P approach was not able to solve all steps of at least one instance. However, it could solve more steps than the other approach: 37.1% of the steps for the instances with narrow time windows (*pr01–pr19*) and 26.6% of the steps for instances with wider time windows (*pr11–pr20*) were completed; in total 31.8% of all steps. The highest number of solved steps was 12 (compared to 4 in the GA). Thus, this supports our findings that the B&P performs better on instances that contain a larger number of customers.

Additionally, we conducted further experiments to evaluate the influence of the penalty term on the solution. The values for the penalty term and the tolerance depended on the specific problem instance. The penalty terms were derived from the maximum travel costs

between two nodes (customer and depots). These values were multiplied with two different factors: In one scenario with 0.33 and in the other one with 0.66. To compute the tolerance value we have determined the average length of all time windows and multiplied this value with one of three scenario-specific factors (0.05, 0.1, and 0.25). Table 5.7 summarizes the six scenarios that result from the combination of these values.

Table 5.7: Tolerance and Penalty Values for Different Scenarios

Scenario	Tolerance	Penalty
A	0.05	0.33
B	0.05	0.66
C	0.1	0.33
D	0.1	0.66
E	0.25	0.33
F	0.25	0.66

The detailed results of objectives after the last step and penalties because of postponed nodes are given in Tables B.4 and B.5 in Appendix B. Table 5.8 presents the summarized values aggregated by the number of vehicles. Column *Objective* is the mean objective value of all nodes, *Penalty* is that is included in the objective function value. *Postponed* is the number of nodes that could potentially cause a penalty, *Nodes w. Penalty* states the number of nodes that caused a penalty. The last column, *Penalty per Node*, is the penalty that applies if a node is postponed. It can be clearly seen for the 25 customer instances that high penalty costs reduced the average number of penalized nodes. In scenario *E* the algorithm avoided penalties completely but accepted a higher objective function value. Figure 5.3 on the following page illustrated the 25 customers instances. The left bar is the averaged total objective function value after step 2, the right bar gives the averaged penalty of all steps for every scenario. Scenarios with lower penalty terms (*A*, *C*, *E*) tended to have lower objective function values than those with higher penalties. The low tolerance in scenario *A* resulted in a higher objective function value. In general, the penalties were small compared to the routing cost and only a small fraction of possible deviations was used. This and the fact that the objective function values ranged over 18% show that the algorithm preferred detours over accepting penalties because of postponement. The averaged values of all instances were better than those in the setting without penalties (275.1). This may sound paradoxically but was caused by the randomness that influenced the search process.

Figure 5.4 on page 119 shows the results for the instances with 100 customers. Because the penalty terms are much higher than in the other scenario (Table 5.8 on the following page), the algorithm avoids penalties in most cases (even with a higher number of nodes that can be postponed). Only if the tolerance span is widest, penalties are accepted. The mean objective function values must be interpreted carefully because (like in the scenario without penalties) only 4 out of 56 instances could be solved completely. Nevertheless, the difference between the smallest and the largest objective function value is 15%, the mean objective in the scenario without penalties was 1,706.5 but is subject to the same restrictions as the smaller instances.

Table 5.8: Results for Scenarios with Different Penalty Parameters

25 Customers					
Scenario	Objective	Penalty	Postponed	Nodes w. Penalty	Penalty per Node
A	267.4	3.1	10.3	3.7	11.0
B	278.3	5.1	9.9	5.0	23.0
C	249.7	1.3	8.8	1.3	11.0
D	303.7	11.1	9.5	11.0	23.0
E	280.3	0.4	9.1	0.0	11.0
F	264.9	5.6	10.7	5.6	23.0
100 Customers					
Scenario	Objective	Penalty	Postponed	Nodes w. Penalty	Penalty per Node
A	1,841.5	0.0	23.0	0.0	30.0
B	1,739.0	0.0	21.8	0.0	60.0
C	1,731.0	0.2	24.8	0.0	30.0
D	1,593.8	0.0	24.5	0.0	60.0
E	1,724.5	19.4	24.3	0.7	31.0
F	1,777.8	35.0	26.5	0.5	66.0

Figure 5.3: Objective Function Values and Penalties for Solomon Instances with 25 Customers

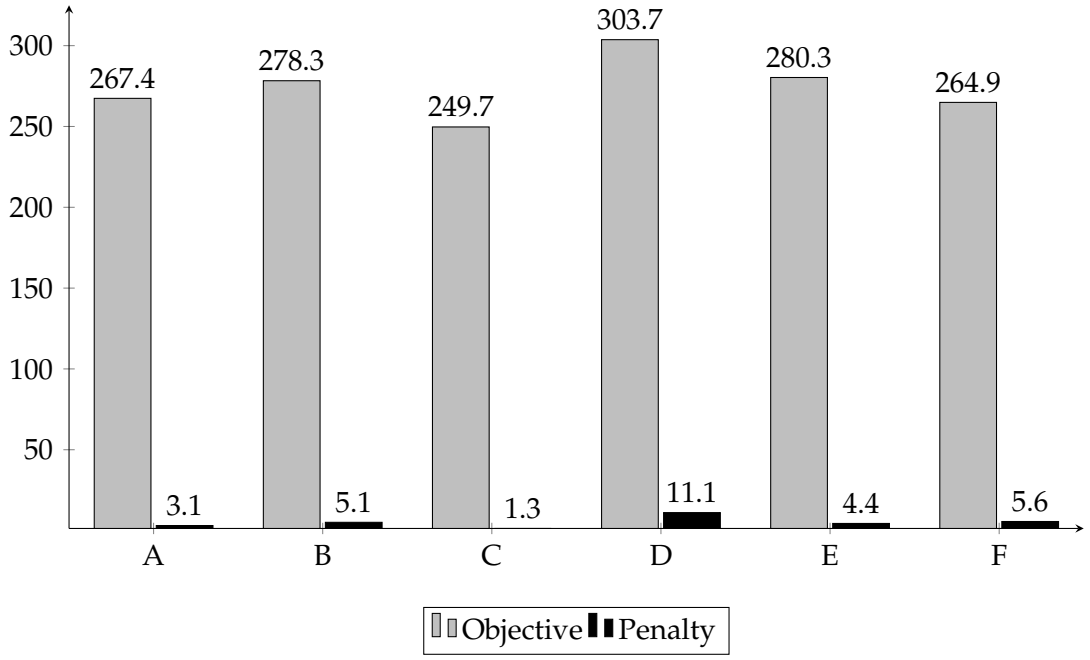
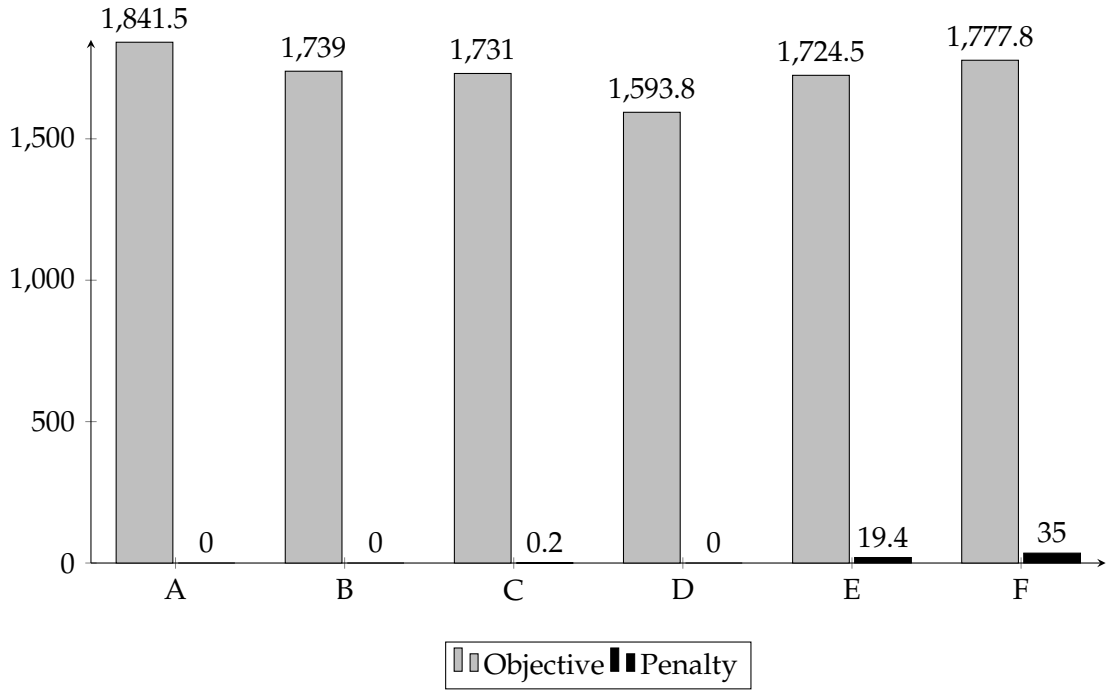


Figure 5.4: Objective Function Values and Penalties for Solomon Instances with 100 Customers



5.5.2 Static Setting

We have evaluated the performance of the GA on the complete instances in Section 4.5.3 and will do the same for the CG approach in the following. We restrict the experiments to the CG instead of the B&P approach because we want only to assess the quality of the solution. Like in Section 4.5.3, we defined all depots to be in the same place and set the number of steps to 1. As a result we get the original Solomon instances.

The results are presented in Table 5.9 on the next page. The first two columns describe the characteristics of the instances, *Mean Obj. B&P* is the average objective found by the B&P procedure. Again, we use low precision on the distances. *Best Known Sol.* are the best solutions for the instances from literature and *Gap* is computed by $(\textit{Mean Obj. B\&P}/\textit{Best Known Sol.}) - 1$. The last column includes the average computation time of the B&P algorithm.

The approach could solve all but one instances with 25 customers, 48 of 56 instances with 50 customers, and 27 of 56 instances with 100 customers. The GA could not solve the largest instances at all. The average gap of all instances was 77.7% compared to 177.2% in the GA which was a significant improvement. However, the difference to the best solution was still very large, but might be advantageous as we discussed in Section 4.5.3. Like for the GA best results were obtained on instances with randomly distributed customers. A drawback was the long computation time of the CG approach that is up to 41 times higher compared to the GA.

Table 5.9: Results of the Static Setting for Solomon Instances

Cust.	Type	Mean Obj. B&P	Best Known Sol.	Gap	Sol. Time B&P (Sec.)
25	R	395.4	201.8	96.8%	662.2
	C	619.5	424.5	45.9%	685.7
	RC	585.2	337.2	73.5%	2,382.5
50	R	782.1	360.2	171.2%	578.6
	C	1,059.8	684.5	54.8%	566.8
	RC	1,088.1	662.6	64.2%	552.7
100	C	1,327.8	587.4	126.0%	358.1
	R	1,703.7	986.5	72.7%	305.5

Chapter 6

Conclusions and Future Research

This thesis focused on rolling horizon planning for Dynamic VRPs. In this class of problems not all future customer demand has been known when optimization was done; new orders arrived while the vehicles are on tour. Therefore, reoptimization was carried out at specific points in time and new routes are sent to the vehicles that replaced the future part of the current plans. This planning method will gain more importance due to the raise in the transportation sector in general and *E-commerce* in particular. Additionally, multiple depots were considered as starting points for the vehicles. Because of the spatial distribution of the depots it could be advantageous that a vehicle from another depot than the one that has received the order might visit a customer to avoid detours (and save costs) or service the customer faster (to increase profits). Thus, there was an exchange mechanism included in the problem.

In Chapter 2 the general problem considered in this thesis was described in detail and relevant literature was reviewed. First of all, we explained the specifics of the problem considered in this thesis. Besides well-known restrictions from general routing problems we introduced assumptions related to the rolling horizon planning, i.e. on the nature of demand and reoptimization. On the basis of these reflections, a mathematical model was built to guarantee a clear understanding of the problem. Because this model is very general it can be applied to multiple planning scenarios that can be distinguished by the type of the vehicle and the length of the planning horizon. Some of these areas and involved chances were derived from the literature. By reviewing previous works on the aspects of the MDVRPTW-RH it turned out that there are scarce results on the Dynamic VRP. Opposed to that the field of MDVRP is very wide as many variants and solution approaches have been developed in the last decades. Best results have been achieved by Local Search and metaheuristics.

Chapter 3 introduced the general aspects of CP that were relevant to the solution methods we suggested in the remainder. We have shown that this optimization method originates from Artificial Intelligence and has emerged from a *language stream* and an *algorithmic stream*. Characteristics of both of them were present in CP as we have shown in the sections about the basic components: Modeling was made easier by the introduction of global constraints, which are shortcuts for complex systems of relations in a model and consist only of a single term. We have explained some predefined global constraints in detail, but it is possible to create problem specific global constraints as well. Additionally, logical modeling operators, like the implication, simplify formulating a model for a given problem. The second aspect, reasoning, played a

major role in the solution of these problems, which mainly relies on tree search algorithms. The size of the search trees was pruned by domain propagation which removed values from the solution space that could not be part of a feasible solution and thus avoided discovering (some) failures. We have introduced different levels of consistency and described the development of one propagation techniques in detail to give an insight into the research field. Considering search, different search methods, their interaction with constraint propagation, and their effect on the shape of the search tree were presented. Most of these aspects were illustrated by the running example of the TSP which is strongly connected to the general topic of this work.

Chapter 4 seized the results of the literature review and presented a GAs to solve the Multi-Depot Vehicle Routing Problem with Time Windows and a Rolling Planning Horizon. It was a two-phase approach that divided the customers into clusters by a GA first and computed afterwards the shortest route for every vehicle by a CP model. This design had two advantages: On the one hand, decomposing the problem into subproblems that seek for the shortest route only for a single vehicle reduced the problem to a capable size and subproblems could be solved optimally. On the other hand, by choosing CP as a solver for the evaluation we allowed flexibility in the model because additional constraints could be added easily. Especially in road transportation there are multiple restriction on travel times etc.

In the chromosome of the GA the alleles held the number of the vehicle that serviced the customers. To enable this framework to deal with a rolling planning horizon we developed a way to modify the genes accordingly. During mutation and reproduction only genes were present in the chromosome that could be modified. Fixed nodes (nodes that have been already visited) were protected from reallocation. This approach allowed flexibility during the optimization process and is intuitively understandable. Additionally, it was not limited to the specific approach and its operators but could be used together with a variety of genetic operators that could be found in the literature. Our algorithm was able to find a solution in a reasonable time for small instances of well-known problems from literature, but was time-consuming or failed for larger customer sets.

To address these shortcomings we have developed a B&P that used LNS to solve the subproblem within the CG procedure. LNS was a search technique that performed well on COPs and was promising according to the literature review, but was a heuristic method. Therefore, it was not guaranteed that all columns that could further improve the solution would be found and B&P would not return the optimal solution but definitely a feasible one. The parts of the algorithm were presented in Chapter 5: We have adopted a heuristic to build initial solutions for the CG procedure. Using the information of the basic solution the LNS procedure was called to generate additional columns. To strengthen propagation and speed up search in the subproblem we had implemented some problem-specific global constraints. Two different branching rules that do not require changing the subproblem formulation were applied to non-integer solutions.

To allow rolling horizon planning without modifying the algorithm we replaced the customer nodes the vehicles have visited in the moment of optimization by an ”artificial” depot node. Thus, the vehicles could start their routes at the depot and return to a depot (with other coordinates than the ”artificial” start depot, specifically). To guarantee feasible a solution, demand of and travel times between so far visited customers was aggregated in the new depot

node of very vehicle. This planning framework was very general, especially it did not rely on CP. For example, it was possible to replace the concrete search algorithm in CG by other techniques like dynamic programming or other search procedures to get optimal solutions.

The B&P approach was able to solve larger instances and the quality of the solution was much better compared to the GA. However, for some comparisons with optimal solution there was still a large gap between the algorithm's best and the optimal solution. Nevertheless, we could accept this because the entire planning process is heuristic and future demand was unknown. In this situation worse solutions could be advantageously. For instance, consider a situation where a bad heuristic solution guides the vehicle on a detour and new customer demand—that must be serviced immediately—appears in this region. In some circumstances the heuristic solution will be globally better than the one that is optimal in every step. This brings up the question if a pure cost calculation is the right objective for this problem. Alternative formulations could be oriented on profits, i.e. servicing a customer earlier results in higher profits, or measure the quality of the service. Future research should focus on this topic.

So far, we have neglected important aspects of collaboration as we assumed that the depots are profit centers but owned by the same firm. This will relax the problem of sharing full information to some extend but it may be arguable that this assumption is realistically. For, every profit center in a company will try to be better off compared to the others. This leads to the question how revenues or costs should be shared in the coalition of profit centers to keep it stable. Even if the organizational units are forced to work together they may act fraudulently if they have no incentive to cooperate. Therefore, the frameworks must be extended by some of the methods presented in Chapter 2 to address these issues.

Appendix A

Detailed Results for the Genetic Algorithm

Table A.1: Average, Minimum, and Maximum Objective Function Values for Solomon Instances with 25 Customers

	Step 0			Step 1			Step 2		
Instance	Average	Min	Max	Average	Min	Max	Average	Min	Max
C101	115.1	107	128	207.5	193	227	325.4	301	360
C102	121.9	109	130	265.5	220	290	370.3	305	432
C105	112.2	90	125	220.7	197	256	393.2	355	435
C106	115.0	101	121	221.6	205	245	379.6	352	416
C107	112.3	98	124	201.7	178	220	326.9	311	351
C108	113.3	89	128	221.6	204	259	365.9	332	431
C109	110.6	100	121	213.2	195	235	328.5	267	359
C201	114.3	104	132	313.4	293	332	392.7	361	415
C202	139.2	119	149	342.9	301	370	460.2	426	504
C205	127.1	113	133	312.9	289	330	400.6	385	422
C206	115.9	87	138	298.3	250	328	383.0	304	419
C207	118.6	88	132	306.8	279	345	390.8	349	433
C208	116.5	104	132	303.3	276	321	405.0	364	431
R101	240.8	218	256	472.3	442	493	722.8	669	775
R102	202.2	189	214	476.2	419	533	734.5	668	807
R105	246.0	230	256	462.2	427	480	717.1	671	748
R106	197.6	177	211	476.4	455	534	734.8	667	873
R109	243.6	233	256	468.8	404	504	740.2	642	827
R110	245.3	233	256	498.2	432	544	743.4	624	840
R111	192.5	178	203	484.4	446	530	798.9	736	873
R112	222.0	200	233	471.1	416	506	743.4	671	825
R201	237.7	225	254	435.3	397	481	663.7	612	704
R202	198.5	186	207	480.3	450	517	710.4	606	793
R205	240.9	223	263	423.5	387	456	668.0	622	720
R206	197.6	186	211	487.4	453	546	734.4	661	815
R209	239.8	234	252	439.3	402	482	662.9	588	753
R210	190.5	168	197	389.4	363	413	657.3	617	702
R211	243.5	220	256	454.0	422	494	692.3	607	752
RC101	210.7	192	220	463.4	417	495	757.6	674	862
RC102	239.3	230	249	476.0	444	532	766.7	671	862
RC105	222.9	198	254	499.1	447	542	741.7	682	775
RC106	210.0	187	227	456.4	414	504	745.6	627	856
RC107	228.2	215	234	497.9	463	542	856.4	791	953
RC108	224.1	201	243	497.1	466	523	858.6	753	959
RC201	214.3	196	222	430.5	396	448	650.2	604	701
RC202	237.2	220	250	439.3	381	505	653.7	547	771
RC205	217.6	191	228	435.7	412	463	661.9	586	722
RC206	214.4	195	222	426.6	383	473	654.5	626	694
RC207	202.5	189	220	398.9	356	430	611.9	539	674
RC208	209.1	178	222	443.7	418	465	717.7	642	808

Table A.2: Average, Minimum, and Maximum Objective Function Values for Solomon Instances with 50 Customer

	Step 0			Step 1			Step 2			Step 3			Step 4		
Instance	Average	Min	Max	Average	Min	Max	Average	Min	Max	Average	Min	Max	Average	Min	Max
C105	209.2	189	227	434.4	397	467	638.8	592	883	809.1	592	883	1,067.6	891	1,232
C106	207.6	186	227	428.2	404	458	653.0	609	942	877.4	609	942	1,192.8	963	1,412
C107	203.2	184	227	432.8	402	465	647.9	578	924	880.7	578	924	1,261.6	1,030	1,808
C108	209.9	184	231	437.9	390	474	646.4	554	1,016	878.4	554	1,016	1,187.2	1,079	1,482
C109	180.8	143	196	438.3	405	474	668.4	642	1,009	923.2	642	1,009	1,167.4	1,064	1,281
C201	126.9	99	141	431.1	386	462	699.7	657	968	935.8	657	968	1,127.3	1,002	1,554
C205	125.3	100	142	432.8	360	485	701.1	650	1,005	938.6	650	1,005	1,075.0	982	1,173
C206	129.4	124	140	435.4	399	464	692.6	611	1,010	922.2	611	1,010	1,089.8	986	1,306
C207	117.8	89	129	407.0	378	427	715.3	658	979	920.2	658	979	1,077.0	995	1,188
C208	128.9	124	133	436.4	391	477	702.8	670	1,104	956.2	670	1,104	1,143.2	1,011	1,518
R101	302.3	273	311	645.2	560	696	968.0	968	968						
R105	270.2	251	279	647.8	595	719	919.1	848	1,470	1,295.3	848	1,470	1,990.5	1,487	2,849
R109	264.0	240	282	628.5	575	674	952.9	881	1,382	1,344.3	881	1,382	1,899.0	1,651	2,262
R110	234.4	220	251	615.7	535	662	997.3	958	1,560	1,379.4	958	1,560	1,832.0	1,652	2,109
R111	198.5	174	216	565.1	507	596	912.3	834	1,439	1,346.9	834	1,439	1,864.0	1,604	2,142
R112	222.8	210	237	571.8	497	618	920.9	853	1,493	1,392.6	853	1,493	1,810.3	1,675	1,923
R201	289.1	270	306	598.9	552	698	869.4	766	1,265	1,129.0	766	1,265	1,407.6	1,272	1,617
R205	286.0	263	301	602.1	572	636	890.9	831	1,375	1,177.5	831	1,375	1,573.7	1,416	2,256
R209	317.7	296	336	686.4	648	718	971.7	865	1,451	1,308.1	865	1,451	1,557.7	1,428	1,721
R210	255.1	234	275	563.1	541	606	872.6	820	1,239	1,140.4	820	1,239	1,484.6	1,378	1,581
R211	241.1	220	262	591.4	542	631	859.2	820	1,237	1,176.1	820	1,237	1,521.4	1,365	1,678
RC101	491.1	463	518	975.3	915	1,053	1,432.2	1,289	2,328	2,009.3	1,289	2,328	2,476.0	2,476	2,476
RC105	423.7	394	455	881.2	706	1,071	1,377.2	1,224	2,455	2,039.0	1,224	2,455	2,636.5	2,536	2,737
RC106	425.1	351	476	950.5	830	1,058	1,469.4	1,332	2,015	1,957.3	1,332	2,015			
RC107	424.3	380	448	938.6	850	1,046	1,460.8	1,282	2,192	1,998.9	1,282	2,192	2,646.7	2,468	2,765
RC108	398.1	323	435	916.4	840	963	1,441.0	1,263	2,180	2,015.6	1,263	2,180	2,725.5	2,478	3,017
RC201	470.5	383	511	873.2	784	1,011	1,229.3	952	1,779	1,582.4	952	1,779	1,971.2	1,791	2,270
RC205	430.4	340	466	919.7	827	1,078	1,316.3	1,209	1,899	1,736.6	1,209	1,899	2,175.7	1,876	2,434
RC206	465.6	438	499	924.4	795	1,022	1,336.6	1,141	2,181	1,758.0	1,141	2,181	2,221.5	1,853	2,695
RC207	456.6	412	487	934.7	849	1,005	1,353.6	1,191	2,011	1,785.5	1,191	2,011	2,252.3	1,953	2,414
RC208	420.3	366	457	921.7	883	958	1,446.2	1,338	2,227	1,916.6	1,338	2,227	2,294.0	1,971	2,687

Table A.3: Average, Minimum, and Maximum Objective Function Values for Solomon Instances with 100 Customer

	Step 0			Step 1			Step 2			Step 3			Step 4		
Instance	Average	Min	Max	Average	Min	Max	Average	Min	Max	Average	Min	Max	Average	Min	Max
C101	430.6	413	448	852.5	781	919	1,398.6	1,279	1,627	2,434.7	2,070	2,833			
C105	365.4	330	385	860.8	806	913	1,375.5	1,287	1,476	2,277.6	2,087	2,436			
C106	355.5	341	377	851.4	786	929	1,445.4	1,362	1,516	2,104.7	1,987	2,226			
C107	306.6	274	317	808.8	767	860	1,349.9	1,273	1,416	2,185.0	1,986	2,344	2,972.5	2,912	3,033
C108	295.5	279	314	840.9	748	921	1,313.7	1,261	1,419	2,111.2	1,913	2,247	3,642.7	2,985	4,667
C109	241.1	225	261	754.7	684	849	1,328.9	1,119	1,441	2,060.9	1,851	2,299	3,161.0	2,586	3,736
C201	254.1	239	261	659.7	625	704	1,091.2	982	1,184	1,565.4	1,422	1,727	2,190.8	1,816	2,564
C205	241.5	233	255	648.6	613	691	1,075.5	971	1,192	1,542.3	1,430	1,664	2,234.8	2,066	2,472
C206	242.1	221	255	664.5	624	764	1,150.7	1,043	1,316	1,679.3	1,423	1,861	2,142.5	1,869	2,409
C207	238.7	222	254	679.8	647	723	1,190.3	1,079	1,278	1,786.5	1,441	1,926	2,229.3	1,844	2,436
C208	237.8	212	255	685.0	622	717	1,125.8	1,022	1,226	1,597.9	1,359	1,816	2,203.6	1,772	2,510
R101	322.9	309	343	704.6	647	761	1,164.3	1,098	1,239	2,910.0	1,636	5,320			
R105	297.4	262	326	675.2	634	713	1,141.7	1,064	1,224	1,828.2	1,611	2,191			
R109	237.0	206	255	650.5	604	700	1,178.6	1,107	1,254	1,788.6	1,612	2,123			
R110	224.9	212	236	585.9	502	625	1,086.4	1,026	1,209	1,787.3	1,661	2,070			
R111	208.6	191	226	581.0	522	618	1,089.6	912	1,167	1,702.7	1,532	1,903	3,089.5	2,425	3,754
R112	203.4	182	215	566.1	482	633	1,087.8	1,013	1,156	1,854.7	1,737	2,060	2,665.8	2,402	3,060
R201	446.8	432	457	688.9	645	725	1,110.8	1,020	1,183	1,482.0	1,410	1,594	1,878.0	1,818	2,059
R205	389.8	375	414	696.3	636	748	1,124.6	1,056	1,181	1,531.3	1,344	1,715	1,962.6	1,812	2,262
R209	275.4	255	290	813.8	731	883	1,205.5	1,114	1,353	1,600.4	1,484	1,850	1,999.0	1,834	2,164
R210	232.4	204	241	707.3	678	757	1,112.0	1,010	1,305	1,537.8	1,396	1,845	1,945.3	1,857	2,004
R211	187.5	170	200	650.6	593	684	1,203.0	1,084	1,323	1,646.8	1,441	1,894	2,071.5	1,739	2,308
RC101	399.1	373	420	893.6	852	1,014	1,556.0	1,429	1,771	2,538.8	2,190	2,817			
RC105	350.3	327	370	975.8	837	1,075	1,628.1	1,442	1,756	2,331.7	2,063	2,529			
RC106	351.0	330	368	921.4	842	990	1,590.2	1,445	1,817	2,509.7	2,234	3,123			
RC107	279.1	265	300	852.5	779	963	1,545.4	1,438	1,659	2,477.8	2,221	2,717	3,849.0	3,074	5,214
RC108	287.2	277	298	798.6	748	851	1,517.8	1,417	1,602	2,296.5	2,176	2,639	3,049.0	3,049	3,049
RC201	490.0	455	517	881.0	813	986	1,403.9	1,277	1,544	1,930.0	1,679	2,164	2,569.4	2,221	3,078
RC205	349.2	297	373	1,026.3	935	1,106	1,493.2	1,340	1,741	2,089.1	1,898	2,308	2,723.9	2,428	3,197
RC206	440.8	400	480	947.1	867	1,024	1,502.4	1,339	1,610	2,024.3	1,805	2,234	2,522.0	2,159	3,024
RC207	337.8	324	350	1,001.2	912	1,085	1,513.4	1,316	1,658	2,098.6	1,924	2,253	2,990.8	2,497	3,528
RC208	308.0	294	325	940.3	886	978	1,624.5	1,531	1,761	2,278.6	2,142	2,372	3,117.5	2,890	3,345

Table A.3: Average, Minimum, and Maximum Objective Function Values for Solomon Instances with 100 Customers (ctd.)

	Step 5			Step 6			Step 7		
Instance	Average	Min	Max	Average	Min	Max	Average	Min	Max
C101									
C105									
C106									
C107									
C108									
C109									
C201	3,985.8	2,266	5,633						
C205	3,251.5	2,896	3,616	4,102.0	4,102	4,102			
C206	2,945.0	2,852	3,038	3,660.0	3,660	3,660	4,696.0	4,696	4,696
C207	2,792.0	2,609	2,975	3,075.0	3,075	3,075	3,914.0	3,914	3,914
C208	2,532.5	2,409	2,656	3,612.0	3,612	3,612			
R101									
R105									
R109									
R110									
R111									
R112									
R201	2,426.5	2,169	2,704	3,780.5	3,080	4,481			
R205	2,590.8	2,500	2,760	2,950.5	2,808	3,093			
R209	2,659.0	2,659	2,659						
R210	2,524.0	2,456	2,592						
R211	2,379.0	2,379	2,379	2,968.0	2,968	2,968			
RC101									
RC105									
RC106									
RC107									
RC108									
RC201	3,438.5	3,029	3,709	5,317.0	5,317	5,317			
RC205	3,789.6	3,127	4,632						
RC206	2,848.0	2,688	3,008	3,368.0	3,368	3,368			
RC207	3,392.0	3,182	3,528						
RC208									

Table A.4: Detailed Results of Exchanged Customers for Solomon Instances with 25 Customers

	Step 1			Step 2		
Instance	Average	Min	Max	Average	Min	Max
C101	0.0%	0.0%	0.0%	9.5%	0.0%	95.2%
C102	12.3%	0.0%	23.1%	4.3%	0.0%	14.3%
C105	10.8%	7.7%	15.4%	10.0%	9.5%	14.3%
C106	10.0%	0.0%	15.4%	18.1%	9.5%	95.2%
C107	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
C108	3.8%	0.0%	15.4%	2.4%	0.0%	9.5%
C109	6.9%	0.0%	15.4%	1.0%	0.0%	4.8%
C201	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
C202	10.8%	0.0%	23.1%	11.4%	0.0%	85.7%
C205	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
C206	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
C207	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
C208	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
R101	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
R102	0.0%	0.0%	0.0%	4.8%	4.8%	4.8%
R105	0.8%	0.0%	7.7%	9.0%	0.0%	90.5%
R106	1.5%	0.0%	7.7%	5.7%	4.8%	9.5%
R109	3.8%	0.0%	7.7%	2.4%	0.0%	4.8%
R110	2.3%	0.0%	7.7%	4.3%	0.0%	14.3%
R111	6.9%	0.0%	23.1%	12.4%	4.8%	19.0%
R112	7.7%	0.0%	23.1%	4.8%	0.0%	9.5%
R201	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
R202	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
R205	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
R206	0.8%	0.0%	7.7%	0.5%	0.0%	4.8%
R209	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
R210	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
R211	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
RC101	0.0%	0.0%	0.0%	1.9%	0.0%	4.8%
RC102	0.8%	0.0%	7.7%	4.8%	0.0%	9.5%
RC105	2.3%	0.0%	15.4%	1.4%	0.0%	9.5%
RC106	0.8%	0.0%	7.7%	1.9%	0.0%	4.8%
RC107	7.7%	0.0%	15.4%	7.6%	0.0%	14.3%
RC108	9.2%	0.0%	15.4%	10.5%	4.8%	14.3%
RC201	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
RC202	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
RC205	7.7%	7.7%	7.7%	4.8%	4.8%	4.8%
RC206	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
RC207	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
RC208	0.8%	0.0%	7.7%	0.5%	0.0%	4.8%

Table A.5: Detailed Results of Exchanged Customers for Solomon Instances with 50 Customers

	Step 1			Step 2			Step 3			Step 4		
Instance	Average	Min	Max	Average	Min	Max	Average	Min	Max	Average	Min	Max
C105	0.0%	0.0%	0.0%	0.4%	0.0%	4.2%	1.1%	0.0%	2.9%	2.5%	0.0%	4.4%
C106	2.2%	0.0%	11.1%	0.8%	0.0%	4.2%	0.6%	0.0%	2.9%	2.7%	0.0%	4.4%
C107	4.4%	0.0%	22.2%	0.8%	0.0%	4.2%	1.6%	0.0%	5.7%	2.2%	0.0%	4.4%
C108	7.8%	0.0%	22.2%	2.9%	0.0%	8.3%	1.4%	0.0%	8.6%	1.8%	0.0%	4.4%
C109	3.3%	0.0%	11.1%	3.8%	0.0%	8.3%	2.0%	0.0%	5.7%	3.0%	0.0%	6.7%
C201	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	1.0%	0.0%	4.4%
C205	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.6%	0.0%	2.2%
C206	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.3%	0.0%	2.9%	0.8%	0.0%	4.4%
C207	0.0%	0.0%	0.0%	0.8%	0.0%	4.2%	0.6%	0.0%	2.9%	1.7%	0.0%	4.4%
C208	0.0%	0.0%	0.0%	0.4%	0.0%	4.2%	0.9%	0.0%	2.9%	1.1%	0.0%	4.4%
R101	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%						
R105	3.3%	0.0%	11.1%	2.9%	0.0%	8.3%	3.6%	0.0%	8.6%	7.4%	2.2%	11.1%
R109	2.2%	0.0%	11.1%	7.1%	0.0%	12.5%	7.6%	5.7%	11.4%	7.1%	4.4%	11.1%
R110	5.6%	0.0%	22.2%	9.2%	0.0%	16.7%	9.5%	5.7%	14.3%	9.3%	6.7%	13.3%
R111	2.2%	0.0%	11.1%	5.0%	0.0%	8.3%	10.0%	5.7%	17.1%	11.1%	6.7%	15.6%
R112	2.2%	0.0%	11.1%	7.9%	4.2%	20.8%	10.6%	5.7%	14.3%	14.1%	11.1%	17.8%
R201	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.7%	0.0%	2.2%
R205	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.9%	0.0%	5.7%	1.7%	0.0%	6.7%
R209	7.8%	0.0%	22.2%	7.1%	0.0%	12.5%	0.3%	0.0%	2.9%	0.9%	0.0%	4.4%
R210	6.7%	0.0%	11.1%	4.2%	0.0%	8.3%	0.3%	0.0%	2.9%	1.3%	0.0%	4.4%
R211	3.3%	0.0%	11.1%	3.3%	0.0%	12.5%	2.3%	0.0%	5.7%	3.0%	0.0%	6.7%
RC101	1.1%	0.0%	11.1%	7.4%	0.0%	12.5%	6.4%	2.9%	8.6%	6.7%	6.7%	6.7%
RC105	4.4%	0.0%	11.1%	9.7%	4.2%	25.0%	6.9%	2.9%	8.6%	4.4%	4.4%	4.4%
RC106	4.4%	0.0%	11.1%	15.0%	8.3%	20.8%	11.4%	5.7%	14.3%			
RC107	7.8%	0.0%	22.2%	16.2%	8.3%	25.0%	10.7%	5.7%	14.3%	11.1%	6.7%	13.3%
RC108	18.9%	0.0%	33.3%	13.9%	8.3%	20.8%	9.4%	2.9%	14.3%	8.3%	4.4%	17.8%
RC201	0.0%	0.0%	0.0%	0.4%	0.0%	4.2%	0.6%	0.0%	2.9%	1.2%	0.0%	2.2%
RC205	7.8%	0.0%	22.2%	3.3%	0.0%	8.3%	0.6%	0.0%	2.9%	1.1%	0.0%	2.2%
RC206	2.2%	0.0%	11.1%	2.1%	0.0%	4.2%	0.6%	0.0%	5.7%	0.9%	0.0%	4.4%
RC207	4.4%	0.0%	11.1%	7.5%	0.0%	12.5%	1.7%	0.0%	5.7%	2.2%	0.0%	4.4%
RC208	12.2%	0.0%	33.3%	11.7%	0.0%	25.0%	6.6%	0.0%	14.3%	3.9%	2.2%	6.7%

Table A.6: Detailed Results of Exchanged Customers for Solomon Instances with 100 Customers

	Step 1			Step 2			Step 3			Step 4		
Instance	Average	Min	Max	Average	Min	Max	Average	Min	Max	Average	Min	Max
C105	4.7%	0.0%	11.8%	11.7%	3.4%	20.7%	11.2%	9.8%	12.2%			
C106	11.2%	5.9%	17.6%	11.9%	3.4%	17.2%	6.1%	4.9%	7.3%			
C107	5.9%	0.0%	11.8%	7.6%	3.4%	13.8%	7.3%	4.9%	12.2%	27.4%	26.4%	28.3%
C108	5.9%	0.0%	11.8%	10.7%	3.4%	17.2%	9.3%	4.9%	12.2%	26.4%	26.4%	26.4%
C109	12.9%	0.0%	23.5%	12.4%	10.3%	17.2%	11.3%	7.3%	14.6%	30.2%	30.2%	30.2%
C201	0.0%	0.0%	0.0%	0.4%	0.0%	3.4%	1.7%	0.0%	4.9%	23.6%	20.8%	26.4%
C205	0.6%	0.0%	5.9%	0.3%	0.0%	3.4%	1.2%	0.0%	2.4%	23.4%	22.6%	24.5%
C206	5.9%	0.0%	17.6%	3.8%	0.0%	6.9%	2.0%	0.0%	7.3%	22.3%	18.9%	24.5%
C207	8.8%	5.9%	17.6%	9.0%	3.4%	17.2%	6.7%	0.0%	14.6%	25.5%	22.6%	30.2%
C208	11.8%	5.9%	17.6%	7.2%	3.4%	10.3%	23.0%	0.0%	95.1%	26.8%	18.9%	41.5%
R101	0.0%	0.0%	0.0%	2.2%	0.0%	6.9%	4.1%	0.0%	7.3%			
R105	3.5%	0.0%	11.8%	6.9%	3.4%	17.2%	6.8%	2.4%	9.8%			
R109	6.5%	0.0%	17.6%	10.7%	3.4%	20.7%	14.0%	9.8%	19.5%			
R110	12.4%	5.9%	17.6%	14.9%	3.4%	20.7%	15.7%	12.2%	22.0%			
R111	7.1%	0.0%	11.8%	15.5%	3.4%	24.1%	15.2%	7.3%	22.0%	35.8%	35.8%	35.8%
R112	10.0%	0.0%	23.5%	14.5%	10.3%	20.7%	17.8%	12.2%	22.0%	35.5%	30.2%	39.6%
R201	0.0%	0.0%	0.0%	1.1%	0.0%	3.4%	13.0%	0.0%	92.7%	26.4%	20.8%	43.4%
R205	5.3%	0.0%	11.8%	4.6%	0.0%	6.9%	4.6%	2.4%	7.3%	24.9%	22.6%	26.4%
R209	11.8%	5.9%	17.6%	7.8%	0.0%	17.2%	3.5%	0.0%	9.8%	22.6%	22.6%	22.6%
R210	10.9%	0.0%	23.5%	13.3%	3.4%	27.6%	9.1%	2.4%	22.0%	24.5%	22.6%	28.3%
R211	12.3%	0.0%	35.3%	17.8%	6.9%	31.0%	11.5%	4.9%	19.5%	27.8%	20.8%	34.0%
RC101	2.9%	0.0%	11.8%	7.3%	3.4%	10.3%	10.6%	7.3%	14.6%			
RC105	12.9%	0.0%	23.5%	12.3%	6.9%	20.7%	8.9%	2.4%	14.6%			
RC106	7.6%	0.0%	17.6%	10.3%	3.4%	17.2%	12.6%	7.3%	17.1%			
RC107	9.4%	0.0%	17.6%	15.9%	6.9%	20.7%	17.7%	14.6%	26.8%	34.0%	32.1%	37.7%
RC108	5.9%	0.0%	11.8%	14.8%	6.9%	20.7%	18.3%	12.2%	24.4%	34.0%	34.0%	34.0%
RC201	0.0%	0.0%	0.0%	2.4%	0.0%	6.9%	3.7%	0.0%	9.8%	25.3%	22.6%	28.3%
RC205	6.3%	0.0%	17.6%	5.0%	0.0%	10.3%	4.1%	0.0%	9.8%	24.5%	20.8%	30.2%
RC206	6.5%	0.0%	17.6%	5.2%	0.0%	10.3%	3.7%	0.0%	7.3%	25.2%	22.6%	28.3%
RC207	4.5%	0.0%	17.6%	6.4%	0.0%	13.8%	5.9%	0.0%	9.8%	25.2%	22.6%	28.3%
RC208	7.6%	0.0%	23.5%	15.2%	6.9%	27.6%	16.8%	12.2%	26.8%	34.6%	34.0%	35.8%

Table A.6: Detailed Results of Exchanged Customers for Solomon Instances with 100 Customers (ctd.)

	Step 5			Step 6			Step 7		
Instance	Average	Min	Max	Average	Min	Max	Average	Min	Max
C105									
C106									
C107									
C108									
C109									
C201	20.8%	18.5%	23.1%						
C205	20.4%	18.5%	23.1%	16.9%	16.9%	16.9%			
C206	19.2%	16.9%	21.5%	14.3%	14.3%	14.3%	13.3%	13.3%	13.3%
C207	17.7%	16.9%	18.5%	14.3%	14.3%	14.3%	13.3%	13.3%	13.3%
C208	13.3%	0.0%	20.0%	8.4%	0.0%	16.9%			
R101									
R105									
R109									
R110									
R111									
R112									
R201	16.0%	0.0%	23.1%	10.8%	0.0%	16.9%			
R205	20.0%	18.5%	21.5%	17.5%	16.9%	18.2%			
R209	21.5%	21.5%	21.5%						
R210	22.3%	21.5%	23.1%						
R211	24.6%	24.6%	24.6%	20.8%	20.8%	20.8%			
RC101									
RC105									
RC106									
RC107									
RC108									
RC201	20.8%	20.0%	21.5%	16.9%	16.9%	16.9%			
RC205	22.5%	20.0%	24.6%						
RC206	18.5%	16.9%	20.0%	16.9%	16.9%	16.9%			
RC207	21.5%	20.0%	23.1%						
RC208									

Appendix B

Detailed Results for the Branch&Price Approach

Table B.1: Comparison of Both Approaches on Solomon Instances with 25 Customers

	Step 0			Step 1			Step 2		
Instance	GA	BAP	Gap	GA	BAP	Gap	GA	BAP	Gap
C101	115.1	83.0	38.7%	207.5	145.0	43.1%	325.4	186.5	74.5%
C102	121.9	82.0	48.7%	265.5	100.0	165.5%	370.3	122.0	203.5%
C105	112.2	78.5	42.9%	220.7	132.0	67.2%	393.2	272.5	44.3%
C106	115.0	88.0	30.7%	221.6	138.5	60.0%	379.7	156.0	143.4%
C107	112.3	76.0	47.8%	201.7	119.0	69.5%	326.9	163.0	100.6%
C108	113.3	65.0	74.3%	221.6	124.0	78.7%	365.9	115.0	218.2%
C109	110.6	66.0	67.6%	213.2	145.0	47.0%	328.5	140.0	134.6%
C201	114.3	88.0	29.9%	313.4	172.0	82.2%	392.7	219.0	79.3%
C202	139.2	90.0	54.7%	342.9	162.5	111.0%	460.2	181.0	154.3%
C205	127.1	89.0	42.8%	312.9	135.0	131.8%	400.6	167.0	139.9%
C206	115.9	87.0	33.2%	298.3	132.0	126.0%	383.0	183.0	109.3%
C207	118.6	83.0	42.9%	306.8	131.5	133.3%	390.8	167.5	133.3%
C208	116.5	83.0	40.4%	303.3	130.0	133.3%	405.0	174.0	132.8%
R101	240.8	252.0	-4.4%	472.3	—	—	722.8	—	—
R102	202.2	174.0	16.2%	476.2	—	—	734.5	—	—
R105	246.0	249.0	-1.2%	462.2	318.0	45.3%	717.1	461.0	55.6%
R106	197.6	164.0	20.5%	476.4	300.5	58.5%	734.8	438.0	67.8%
R109	243.6	189.0	28.9%	468.8	295.5	58.6%	740.2	360.0	105.6%
R110	245.3	198.0	23.9%	498.2	279.0	78.6%	743.4	361.0	105.9%
R111	192.5	165.0	16.7%	484.4	298.0	62.6%	798.9	406.0	96.8%
R112	222.0	151.0	47.0%	471.1	292.0	61.3%	743.4	403.5	84.2%
R201	237.7	212.0	12.1%	435.3	409.0	6.4%	663.7	—	—
R202	198.5	205.0	-3.2%	480.3	311.0	54.4%	710.4	425.0	67.2%
R205	240.9	197.5	22.0%	423.5	293.5	44.3%	668.0	389.0	71.7%
R206	197.6	161.0	22.7%	487.4	260.0	87.5%	734.4	373.0	96.9%
R209	239.8	190.0	26.2%	439.3	319.0	37.7%	662.9	395.5	67.6%
R210	190.5	150.0	27.0%	389.4	282.0	38.1%	657.3	430.0	52.9%
R211	243.5	193.0	26.2%	454.0	279.0	62.7%	692.3	391.5	76.8%
RC101	210.7	172.0	22.5%	463.4	236.0	96.4%	757.6	—	—
RC102	239.3	134.0	78.6%	476.0	225.0	111.6%	766.7	233.0	229.1%
RC105	222.9	176.0	26.6%	499.1	199.0	150.8%	741.7	—	—
RC106	210.0	156.0	34.6%	456.4	189.0	141.5%	745.6	220.5	238.1%
RC107	228.2	169.0	35.0%	497.9	209.0	138.2%	856.4	224.0	282.3%
RC108	224.1	155.0	44.6%	497.1	180.0	176.2%	858.6	200.0	329.3%
RC201	214.3	168.5	27.2%	430.5	175.5	145.3%	650.2	440.5	47.6%
RC202	237.2	138.5	71.3%	439.3	164.0	167.9%	653.7	352.5	85.4%
RC205	217.6	179.0	21.6%	435.7	190.0	129.3%	661.9	281.5	135.1%
RC206	214.4	152.0	41.1%	426.6	206.0	107.1%	654.5	256.0	155.7%
RC207	202.5	156.0	29.8%	398.9	223.0	78.9%	611.9	259.0	136.3%
RC208	209.1	150.5	38.9%	443.7	221.5	100.3%	717.7	257.5	178.7%

Table B.2: Comparison of Both Approaches on Solomon Instances with 50 Customers

	Step 0			Step 1			Step 2			Step 3			Step 4		
Instance	GA	BAP	Gap	GA	BAP	Gap	GA	BAP	Gap	GA	BAP	Gap	GA	BAP	Gap
C105	209.2	129.0	62.2%	434.4	135.0	221.8%	638.8	171.5	272.5%	809.1	531.0	52.4%	1,067.6	328.0	225.5%
C106	207.6	129.0	60.9%	428.2	134.0	219.6%	653.0	212.5	207.3%	877.4	243.0	261.1%	1,192.8	317.0	276.3%
C107	203.2	137.0	48.3%	432.8	139.0	211.4%	647.9	236.0	174.5%	880.7	286.5	207.4%	1,261.6	285.5	341.9%
C108	209.9	113.0	85.8%	437.9	162.5	169.5%	646.4	—	—	878.4	—	—	1,187.2	—	—
C109	180.8	115.0	57.2%	438.3	123.0	256.3%	668.4	252.0	165.2%	923.2	230.0	301.4%	1,167.4	267.0	337.2%
C201	126.9	90.0	41.0%	431.1	237.0	81.9%	699.7	—	—	935.8	—	—	1,127.3	—	—
C205	125.3	91.5	36.9%	432.8	201.0	115.3%	701.1	473.0	48.2%	938.6	412.0	127.8%	1,075.0	464.0	131.7%
C206	129.4	84.5	53.1%	435.4	377.0	15.5%	692.6	—	—	922.2	—	—	1,089.8	—	—
C207	117.8	85.5	37.8%	407.0	221.5	83.7%	715.3	560.0	27.7%	920.2	432.5	112.8%	1,077.0	566.0	90.3%
C208	128.9	90.0	43.2%	436.4	222.0	96.6%	702.8	349.0	101.4%	956.2	401.0	138.5%	1,143.2	464.0	146.4%
R101	302.3	262.0	15.4%	645.2	—	—	968.0	—	—	—	—	—	—	—	—
R105	270.2	261.0	3.5%	647.8	—	—	919.1	—	—	1,295.3	—	—	1,990.5	—	—
R109	264.0	221.5	19.2%	628.5	347.5	80.9%	952.9	400.0	138.2%	1,344.3	—	—	1,899.0	—	—
R110	234.4	173.5	35.1%	615.7	355.5	73.2%	997.3	580.0	71.9%	1,379.4	781.0	76.6%	1,832.0	687.0	166.7%
R111	198.5	152.5	30.2%	565.1	287.5	96.6%	912.3	585.0	55.9%	1,346.9	599.0	124.9%	1,864.0	608.0	206.6%
R112	222.8	197.0	13.1%	571.8	330.0	73.3%	920.9	539.0	70.9%	1,392.6	820.0	69.8%	1,810.3	856.0	111.5%
R201	289.1	201.0	43.8%	598.9	393.0	52.4%	869.4	504.0	72.5%	1,129.0	659.0	71.3%	1,407.6	786.0	79.1%
R205	286.0	177.0	61.6%	602.1	352.5	70.8%	890.9	481.0	85.2%	1,177.5	686.0	71.6%	1,573.7	819.0	92.1%
R209	317.7	202.0	57.3%	686.4	547.0	25.5%	971.7	650.5	49.4%	1,308.1	849.0	54.1%	1,557.7	1,039.0	49.9%
R210	255.1	201.5	26.6%	563.1	407.0	38.4%	872.6	473.5	84.3%	1,140.4	631.0	80.7%	1,484.6	785.0	89.1%
R211	241.1	211.0	14.3%	591.4	320.0	84.8%	859.2	534.5	60.7%	1,176.1	716.5	64.1%	1,521.4	976.0	55.9%
RC101	491.1	241.0	103.8%	975.3	—	—	1,432.2	—	—	2,009.3	—	—	2,476.0	—	—
RC105	423.7	255.0	66.2%	881.2	—	—	1,377.2	—	—	2,039.0	—	—	2,636.5	—	—
RC106	425.1	315.5	34.7%	950.5	292.0	225.5%	1,469.4	676.0	117.4%	1,957.3	680.0	187.8%	—	643.0	—
RC107	424.3	164.0	158.7%	938.6	261.0	259.6%	1,460.8	676.0	116.1%	1,998.9	633.0	215.8%	2,646.7	517.0	411.9%
RC108	398.1	157.0	153.6%	916.4	279.5	227.9%	1,441.0	452.0	218.8%	2,015.6	583.5	245.4%	2,725.5	787.5	246.1%
RC201	470.5	235.5	99.8%	873.2	467.0	87.0%	1,229.3	475.0	158.8%	1,582.4	608.0	160.3%	1,971.2	824.0	139.2%
RC205	430.4	266.0	61.8%	919.7	466.0	97.4%	1,316.3	423.0	211.2%	1,736.6	575.0	202.0%	2,175.7	688.0	216.2%
RC206	465.6	228.0	104.2%	924.4	376.0	145.9%	1,336.6	696.5	91.9%	1,758.0	862.5	103.8%	2,221.5	793.5	180.0%
RC207	456.6	282.5	61.6%	934.7	292.0	220.1%	1,353.6	599.0	126.0%	1,785.5	548.0	225.8%	2,252.3	588.0	283.0%
RC208	420.3	244.0	72.3%	921.7	360.5	155.7%	1,446.2	542.0	166.8%	1,916.6	655.5	192.4%	2,294.0	738.5	210.6%

Table B.3: Comparison of Both Approaches on Solomon Instances with 100 Customers

	Step 0			Step 1			Step 2			Step 3			Step 4		
Instance	GA	BAP	Gap	GA	BAP	Gap	GA	BAP	Gap	GA	BAP	Gap	GA	BAP	Gap
C101	430.6	392.0	9.8%	852.5	798.0	6.8%	1,398.6	—	—	2,434.7	—	—	—	—	—
C105	365.4	392.0	-6.8%	860.8	538.5	59.9%	1,375.5	497.50	176.5%	2,277.6	624.5	264.7%	—	1,194.0	—
C106	355.5	315.0	12.9%	851.4	550.0	54.8%	1,445.4	798.50	81.0%	2,104.7	882.0	138.6%	—	—	—
C107	306.6	386.0	-20.6%	808.8	432.5	87.0%	1,349.9	434.00	211.0%	2,185.0	607.0	260.0%	2,972.5	—	—
C108	295.5	223.0	32.5%	840.9	669.0	25.7%	1,313.7	451.00	191.3%	2,111.2	613.0	244.4%	3,642.7	—	—
C109	241.1	181.5	32.8%	754.7	417.0	81.0%	1,328.9	420.00	216.4%	2,060.9	661.0	211.8%	3,161.0	—	—
C201	254.1	137.0	85.5%	659.7	304.5	116.7%	1,091.2	—	—	1,565.4	—	—	2,190.8	—	—
C205	241.5	136.5	76.9%	648.6	240.0	170.3%	1,075.5	459.00	134.3%	1,542.3	—	—	2,234.8	—	—
C206	242.1	143.5	68.7%	664.5	317.0	109.6%	1,150.7	761.00	51.2%	1,679.3	884.5	89.9%	2,142.5	843.0	154.2%
C207	238.7	124.0	92.5%	679.8	378.0	79.8%	1,190.3	736.00	61.7%	1,786.5	851.5	109.8%	2,229.3	1,159.5	92.3%
C208	237.8	139.5	70.5%	685.0	404.0	69.6%	1,125.8	675.00	66.8%	1,597.9	895.0	78.5%	2,203.6	1,132.0	94.7%
R101	322.9	258.0	25.2%	704.6	—	—	1,164.3	—	—	2,910.0	—	—	—	—	—
R105	297.4	271.0	9.7%	675.2	—	—	1,141.7	—	—	1,828.2	—	—	—	—	—
R109	237.0	169.0	40.2%	650.5	474.0	37.2%	1,178.6	430.00	174.1%	1,788.6	—	—	—	—	—
R110	224.9	166.0	35.5%	585.9	326.0	79.7%	1,086.4	412.50	163.4%	1,787.3	718.0	148.9%	—	975.0	—
R111	208.6	171.5	21.6%	581.0	372.0	56.2%	1,089.6	512.50	112.6%	1,702.7	708.0	140.5%	3,089.5	981.0	214.9%
R112	203.4	151.0	34.7%	566.1	413.0	37.1%	1,087.8	531.00	104.9%	1,854.7	817.0	127.0%	2,665.8	1,047.0	154.6%
R201	446.8	234.5	90.5%	688.9	445.0	54.8%	1,110.8	539.00	106.1%	1,482.0	921.0	60.9%	1,878.0	793.0	136.8%
R205	389.8	236.0	65.2%	696.3	457.0	52.4%	1,124.6	670.00	67.8%	1,531.3	895.0	71.1%	1,962.6	999.5	96.4%
R209	275.4	159.5	72.7%	813.8	484.0	68.1%	1,205.5	676.50	78.2%	1,600.4	834.0	91.9%	1,999.0	970.5	106.0%
R210	232.4	143.5	62.0%	707.3	524.0	35.0%	1,112.0	710.00	56.6%	1,537.8	812.0	89.4%	1,945.3	1,244.0	56.4%
R211	187.5	126.5	48.2%	650.6	349.0	86.4%	1,203.0	623.50	92.9%	1,646.8	772.0	113.3%	2,071.5	862.0	140.3%
RC105	350.3	215.5	62.6%	975.8	520.0	87.7%	1,628.1	649.50	150.7%	2,331.7	—	—	—	—	—
RC106	351.0	288.0	21.9%	921.4	567.5	62.4%	1,590.2	627.00	153.6%	2,509.7	652.0	284.9%	—	685.0	—
RC107	279.1	226.5	23.2%	852.5	337.5	152.6%	1,545.4	580.00	166.4%	2,477.8	1,059.0	134.0%	3,849.0	853.0	351.2%
RC108	287.2	210.5	36.4%	798.6	381.0	109.6%	1,517.8	801.00	89.5%	2,296.5	1,063.0	116.0%	3,049.0	1,457.0	109.3%
RC201	490.0	290.0	69.0%	881.0	458.5	92.1%	1,403.9	646.00	117.3%	1,930.0	823.0	134.5%	2,569.4	957.0	168.5%
RC205	349.2	246.0	41.9%	1,026.3	514.5	99.5%	1,493.2	746.00	100.2%	2,089.1	1,200.0	74.1%	2,723.9	1,451.0	87.7%
RC206	440.8	305.5	44.3%	947.1	486.5	94.7%	1,502.4	718.00	109.2%	2,024.3	1,108.0	82.7%	2,522.0	1,171.5	115.3%
RC207	337.8	239.5	41.0%	1,001.2	534.5	87.3%	1,513.4	723.50	109.2%	2,098.6	1,071.0	95.9%	2,990.8	1,278.0	134.0%
RC208	308.0	244.0	26.2%	940.3	416.5	125.8%	1,624.5	672.00	141.7%	2,278.6	884.5	157.6%	3,117.5	1,159.0	169.0%

Table B.3: Comparison of Both Approaches on Solomon Instances with 100 Customers (ctd.)

Instance	Step 5			Step 6			Step 7		
	GA	BAP	Gap	GA	BAP	Gap	GA	BAP	Gap
C101	—	—	—	—	—	—	—	—	—
C105	—	—	—	—	—	—	—	—	—
C106	—	—	—	—	—	—	—	—	—
C107	—	—	—	—	—	—	—	—	—
C108	—	—	—	—	—	—	—	—	—
C109	—	—	—	—	—	—	—	—	—
C201	3,985.8	—	—	—	—	—	—	—	—
C205	3,251.5	—	—	4,102.0	—	—	—	—	—
C206	2,945.0	1,432.0	105.7%	3,660.0	1,329.0	175.4%	4,696.0	1,329.0	253.3%
C207	2,792.0	1,152.5	142.3%	3,075.0	1,474.0	108.6%	3,914.0	1,667.0	134.8%
C208	2,532.5	1,426.0	77.6%	3,612.0	1,643.0	119.8%	—	1,577.0	—
R101	—	—	—	—	—	—	—	—	—
R105	—	—	—	—	—	—	—	—	—
R109	—	—	—	—	—	—	—	—	—
R110	—	815.0	—	—	906.0	—	—	676.0	—
R111	—	1,122.0	—	—	1,446.0	—	—	1,854.0	—
R112	—	1,111.0	—	—	1,543.0	—	—	1,939.0	—
R201	2,426.5	—	—	3,780.5	—	—	—	—	
R205	2,590.8	1,232.0	110.3%	2,950.5	1,394.5	111.6%	—	1,573.0	—
R209	2,659.0	1,242.5	114.0%	—	1,592.0	—	—	1,671.0	—
R210	2,524.0	1,227.0	105.7%	—	1,523.5	—	—	1,701.0	—
R211	2,379.0	1,085.0	119.3%	2,968.0	1,204.5	146.4%	—	1,578.0	—
RC105	—	—	—	—	—	—	—	—	—
RC106	—	813.0	—	—	—	—	—	—	—
RC107	—	831.0	—	—	—	—	—	—	—
RC108	—	1,428.0	—	—	1,575.0	—	—	1,948.0	—
RC201	3,438.5	1,073.0	220.5%	5,317.0	—	—	—	—	
RC205	3,789.6	1,583.0	139.4%	—	1,681.0	—	—	1,837.0	—
RC206	2,848.0	1,492.0	90.9%	3,368.0	1,820.5	85.0%	—	2,000.0	—
RC207	3,392.0	1,362.0	149.0%	—	1,587.0	—	—	1,912.0	—
RC208	—	1,425.5	—	—	1,523.5	—	—	1,675.5	—

Table B.4: Objectives and Penalties for Solomon Instances with 25 Customers

	Scenario A		Scenario B		Scenario C		Scenario D		Scenario E		Scenario F	
Instance	Objective	Penalty	Objective	Penalty	Objective	Penalty	Objective	Penalty	Objective	Penalty	Objective	Penalty
C101	195	0.3	127	0	183	0.3	127	0	196	3.7	235	0
C102	164	1.0	153	23	143	2.0	127	0	—	—	129	3.7
C105	78	0	79	0	186	0	166	0	196	3.7	231	0
C106	165	0.3	88	0	78	0	88	0	143	5.5	174	7.7
C107	165	0.3	76	0	76	0	78	0	—	—	77	0
C108	65	0	121	1.0	64	0	90	0	124	7.3	120	0.7
C109	141	0.3	142	0	116	1.0	179	15.3	124	7.3	118	11.3
C201	204	0	197	0	179	0	202	0	201	0	187	0
C202	214	0.7	181	0	177	0	211	0	176	0	184	0
C205	198	0	168	0	137	0	184	0	134	0	187	0
C206	231	0	188	0	199	0	196	0	164	0	186	0
C207	175	29	182	0.7	203	1.7	240	11.3	163	0	166	34.0
C208	126	0.5	205	0	136	17.0	83	0	221	0	185	0
R101	252	0	252	0	252	0	252	0	252	0	252	0
R102	178	0	165	0	170	0	170	0	165	0	178	0
R105	469	0	455	0	469	0	459	0	249	0	471	0
R106	432	14.7	294	22.5	389	7.3	167	0	164	0	420	30.0
R109	189	0	189	0	189	0	407	15.3	189	0	199	0
R110	389	8.0	357	0	198	0	387	0	198	0	198	0
R111	287	0	369	15.3	160	0	423	30.7	416	3.7	203	0
R112	143	0	390	15.0	160	0	374	15.3	377	7.3	366	15.0
R201	323	0	323	0	297	0	414	0	408	0	323	0
R202	—	—	—	—	—	—	—	—	161	0	—	—
R205	449	0	484	15.0	202	0	198	0	424	0	396	0
R206	—	—	161	0	169	0	161	0	161	0	674	75.3
R209	383	7.7	451	0	190	0	384	0	452	0	429	0
R210	274	0	164	0	360	0	366	0	158	0	155	0
R211	384	0	295	0	395	7.3	195	0	368	0	392	0

Table B.4: Objectives and Penalties for Solomon Instances with 25 Customers (ctd.)

	Scenario A		Scenario B		Scenario C		Scenario D		Scenario E		Scenario F	
Instance	Objective	Penalty	Objective	Penalty	Objective	Penalty	Objective	Penalty	Objective	Penalty	Objective	Penalty
RC101	232	10.0	176	0	209	0	168	0	201	5.5	168	0
RC102	267	0.7	265	36.3	263	0.3	134	0	186	0	237	0
RC105	176	0	260	0.3	176	0	192	0	192	0	240	0
RC106	—	—	—	—			257	54.0	—	—	—	—
RC107	—	—	—	—	175	0	399	92.0	—	—	—	—
RC108	350	18.3	364	18.0	251	0.3	372	90.0	155	0	256	0
RC201	224	0	305	0	226	0	220	0	223	0	176	0
RC202	199	0	172	0	172	0	—	—	134	0	222	0
RC205	269	1.7	282	0.7	—	—	257	0	—	—	262	0
RC206	265	0.3	—	—	153	0	150	0	253	0	150	0
RC207	259	0.7	210	0	248	0	152	0	257	0	248	0
RC208	262	0	237	0	255	0	272	18.0	245	0	244	0

Table B.5: Objectives and Penalties for Solomon Instances with 100 Customers

	Scenario A		Scenario B		Scenario C		Scenario D		Scenario E		Scenario F	
Instance	Objective	Penalty	Objective	Penalty	Objective	Penalty	Objective	Penalty	Objective	Penalty	Objective	Penalty
C101	798	0	793	0	757	0	817	0	761	0	817	0
C106	507	0	921	0	1,287	0.3	704	0	457	0	446	0
C108	230	0	223	0	1,320	0	789	0	—	—	553	3.7
C109	905	0	1,211	0	660	0	336	0	1,517	10.6	1,454	32.25
C201	516	0	246	0	671	0	158	0	177	0	145	0
C205	134	0	182	0	152	0	1,395	0	—	—	383	3.7
C206	537	0	629	0	566	0	210	0	448	33	1,580	142.5
C207	1,253	0	1,395	0	1,684	0	127	0	—	—	—	—
C208	138	0	389	0	141	0	1,577	0	1,168	52.8	1,047	157.5
R101	258	0	258	0	258	0	258	0	258	0	258	0
R105	394	0	639	0	229	0	427	0	229	0	229	0
R106	—	—	—	—	—	—	—	—	767	6.6	—	—
R109	355	0	508	0	324	0	364	0	—	—	378	0
R110	165	0	223	0	490	0	168	0	373	0	1,378	11
R111	1,619	0	594	0	1,033	0	163	0	—	—	1,010	48.4
R112	142	0	415	0	1,640	0	498	0	—	—	—	—
R201	658	0	622	0	1,000	0	233	0	393	0	1,156	44.6
R205	817	0	733	0	238	0	1,185	0	1,643	28.9	733	44
R209	1,823	0	151	0	155	0	1,367	0	—	—	1,707	68.3
R210	1,173	0	448	0	816	0	771	0	1,255	36.7	761	60
R211	384	0	1,403	0	645	0	1,230	0	348	0	—	—
RC105	223	0	686	0	540	0	1,087	0	766	3.7	824	22
RC106	301	0	677	0	676	0	—	—	900	5.5	1,098	0
RC107	367	0	293	0	879	0	—	—	222	0	225	0
RC108	1,257	0	718	0	508	0	1,861	0	—	—	2,100	24.8
RC201	1,511	0	526	0	273	0	750	0	285	0	671	22
RC205	1,580	0	902	0	1,820	0	1,362	0	—	—	—	—
RC206	1,860	0	2,075	0	2,247	0	1,858	0	2,007	49.5	289	0
RC207	1,138	0	1,894	0	1,758	0	974	0	1,831	24.8	2,024	57.8
RC208	433	0	987	0	1,680	0	1,672	0	—	—	—	—

Bibliography

Abrache, J., Crainic, T. G., Gendreau, M., and Rekik, M., (2007), Combinatorial Auctions, *Annals of Operations Research*, **153**, pp. 131–164. (Cited on page 15.)

Agra, A., Christiansen, M., Figueiredo, R., Hvattum, L. M., Poss, M., and Requejo, C., (2013), The Robust Vehicle Routing Problem with Time Windows, *Computers & Operations Research*, **40**, pp. 856–866. (Cited on page 13.)

Ahuja, R. K., Ergun, Ö., Orlin, J. B., and Punnen, A. P., (2002), A Survey of Very Large-Scale Neighborhood Search Techniques, *Discrete Applied Mathematics*, **123**, pp. 75–102. (Cited on page 63.)

Alt, H., Blum, N., Mehlhorn, K., and Paul, M., (1991), Computing a Maximum Cardinality Matching in a Bipartite Graph in Time $\mathcal{O}(n^{1.5})$, *Information Processing Letters*, **37**, pp. 237–240. (Cited on page 48.)

Althaus, E., Bockmayr, A., Elf, M., Jünger, M., Kasper, T., and Mehlhorn, K., (2002), SCIL—Symbolic Constraints in Integer Linear Programming, in: Möhring, R. and Raman, R., eds., *Algorithms*, *Lecture Notes in Computer Science*, vol. 2461, Springer, Berlin, pp. 75–87. (Cited on page 49.)

Alvarenga, G., Mateus, G. R., and Tomi, G. d., (2007), A Genetic and Set Partitioning Two-Phase Approach for the Vehicle Routing Problem with Time Windows, *Computers & Operations Research*, **34**, pp. 1561–1584. (Cited on page 69.)

Aminu, U. and Eglese, R. W., (2006), A Constraint Programming Approach to the Chinese Postman Problem with Time Windows, *Computers & Operations Research*, **33**, pp. 3423–3431. (Cited on page 65.)

Angelelli, E., Bianchessi, N., Mansini, R., and Speranza, M. G., (2009), Short Term Strategies for a Dynamic Multi-Period Routing Problem, *Transportation Research Part C: Emerging Technologies*, **17**, pp. 106–119. (Cited on page 12.)

Angelelli, E., Bianchessi, N., Mansini, R., and Speranza, M. G., (2010), Comparison of Policies in Dynamic Routing Problems, *Journal of the Operational Research Society*, **61**, pp. 686–695. (Cited on page 12.)

Angelelli, E., Grazia Speranza, M., and Savelsbergh, M. W. P., (2007), Competitive Analysis for Dynamic Multiperiod Uncapacitated Routing Problems, *Networks*, **49**, pp. 308–317. (Cited on page 12.)

Ansótegui, C., Bofill, M., Palahí, M., Suy, J., and Villaret, M., (2013), Solving Weighted CSPs with Meta-Constraints by Reformulation into Satisfiability Modulo Theories, *Constraints*, **18**, pp. 236–268. (Cited on page 35.)

Applegate, D. L., (2006), *The Traveling Salesman Problem: A Computational Study*, Princeton University Press, Princeton. (Cited on page 31.)

Apt, K. R., (2009), *Principles of Constraint Programming*, Cambridge University Press, Cambridge. (Cited on pages 29, 32, 36, 45, 52, 59, and 62.)

Aras, N., Aksen, D., and Tuğrul Tekin, M., (2011), Selective Multi-Depot Vehicle Routing Problem with Pricing, *Transportation Research Part C: Emerging Technologies*, **19**, pp. 866–884. (Cited on pages 20, 25, and 27.)

Archetti, C., Bouchard, M., and Desaulniers, G., (2011), Enhanced Branch and Price and Cut for Vehicle Routing with Split Deliveries and Time Windows, *Transportation Science*, **45**, pp. 285–298. (Cited on page 95.)

Archetti, C. and Speranza, M. G., (2012), Vehicle Routing Problems with Split Deliveries, *International Transactions in Operational Research*, **19**, pp. 3–22. (Cited on page 15.)

Athanasopoulos, T. and Minis, I., (2013), Efficient Techniques for the Multi-Period Vehicle Routing Problem with Time Windows within a Branch and Price Framework, *Annals of Operations Research*, **206**, pp. 1–22. (Cited on page 95.)

Azi, N., Gendreau, M., and Potvin, J.-Y., (2010), An Exact Algorithm for a Vehicle Routing Problem with Time Windows and Multiple Use of Vehicles, *European Journal of Operational Research*, **202**, pp. 756–763. (Cited on page 95.)

Bacchus, F. and Grove, A., (1995), On the Forward Checking Algorithm, in: Montanari, U. and Rossi, F., eds., *Principles and Practice of Constraint Programming*, *Lecture Notes in Computer Science*, vol. 976, Springer, Berlin, pp. 292–309. (Cited on page 56.)

Backer, B. d. and Furnon, V., (1999), Local Search in Constraint Programming: Experiments with Tabu Search on the Vehicle Routing Problem, in: Voß, S., Martello, S., Roucairol, C., and Osman, I. H., eds., *Meta-Heuristics*, Kluwer, Boston, pp. 63–76. (Cited on pages 64 and 95.)

Backer, B. d., Furnon, V., Kilby, P., Prosser, P., and Shaw, P., (1997), Local Search in Constraint Programming: Application to the Vehicle Routing Problem, in: *CP97 Workshop on Industrial Constraint-Directed Scheduling*. (Cited on pages 64 and 95.)

Backer, B. d., Furnon, V., Shaw, P., Kilby, P., and Prosser, P., (2000), Solving Vehicle Routing Problems Using Constraint Programming and Metaheuristics, *Journal of Heuristics*, **6**, pp. 501–523. (Cited on pages 65 and 95.)

Baker, B. M. and Ayechew, M. A., (2003), A Genetic Algorithm for the Vehicle Routing Problem, *Computers & Operations Research*, **30**, pp. 787–800. (Cited on pages 69 and 73.)

Baldacci, R., Mingozzi, A., and Roberti, R., (2011), New Route Relaxation and Pricing Strategies for the Vehicle Routing Problem, *Operations Research*, **59**, pp. 1269–1283. (Cited on page 95.)

Baldacci, R., Mingozzi, A., and Roberti, R., (2012), Recent Exact Algorithms for Solving the Vehicle Routing Problem under Capacity and Time Window Constraints, *European Journal of Operational Research*, **218**, pp. 1–6. (Cited on page 94.)

Ball, M. O., Golden, B. L., Assad, A. A., and Bodin, L. D., (1983), Planning for Truck Fleet Size in the Presence of a Common-Carrier Option, *Decision Sciences*, **14**, pp. 103–120. (Cited on pages 18, 24, and 26.)

Barkaoui, M. and Gendreau, M., (2013), An Adaptive Evolutionary Approach for Real-Time Vehicle Routing and Dispatching, *Computers & Operations Research*, **40**, pp. 1766–1776. (Cited on page 14.)

Barnhart, C., Johnson, E. L., Nemhauser, G. L., Savelsbergh, M. W. P., and Vance, P., (1998), Branch-and-Price: Column Generation for Solving Huge Integer Programs, *Operations Research*, **46**, pp. 316–329. (Cited on page 94.)

Barták, R., (1999), Constraint Programming - What is Behind?, in: *Proceedings of CPDC99 Workshop*, pp. 7–15. (Cited on page 29.)

Barták, R., (2011), History of Constraint Programming, in: Cochran, J. J., Cox, L. A., Keskinocak, P., Kharoufeh, J. P., and Smith, J. C., eds., *Wiley Encyclopedia of Operations Research and Management Science*, vol. 3, Wiley, Hoboken, pp. 2205–2215. (Cited on page 29.)

Beasley, J. E., (1983), Route First-Cluster Second Methods for Vehicle Routing, *Omega*, **11**, pp. 403–408. (Cited on page 18.)

Beek, P. v., (2006), Backtracking Search Algorithms, in: Rossi, F., Beek, P. v., and Walsh, T., eds., *Handbook of Constraint Programming*, Elsevier, Amsterdam, pp. 85–134. (Cited on page 59.)

Beldiceanu, N., Carlsson, M., and Rampon, J.-X., (2015), Global Constraint Catalog. (Cited on pages 47 and 102.)

Benavent, E. and Martínez, A., (2013), Multi-Depot Multiple TSP: A Polyhedral Study and Computational Results, *Annals of Operations Research*, **207**, pp. 7–25. (Cited on page 16.)

Benhamou, F. and Granvilliers, L., (2006), Continous and Interval Constraints, in: Rossi, F., Beek, P. v., and Walsh, T., eds., *Handbook of Constraint Programming*, Elsevier, Amsterdam, pp. 571–603. (Cited on page 33.)

Benslimane, M. T. and Benadada, Y., (2013), Ant Colony Algorithm for the Multi-Depot Vehicle Routing Problem in Large Quantities by a Heterogeneous Fleet of Vehicles, *INFOR*, **51**, pp. 31–40. (Cited on pages 22, 25, and 28.)

Bent, R. W. and van Hentenryck, P., (2004), Scenario-Based Planning for Partially Dynamic Vehicle Routing with Stochastic Customers, *Operations Research*, **52**, pp. 977–987. (Cited on page 14.)

Benton, W. C., (1986), Evaluating a Modified Heuristic for the Multiple-Vehicle Scheduling Problem, *IIE Transactions*, **18**, pp. 70–78. (Cited on pages 16, 24, and 26.)

Berbeglia, G., Cordeau, J.-F., and Laporte, G., (2012), A Hybrid Tabu Search and Constraint Programming Algorithm for the Dynamic Dial-a-Ride Problem, *INFORMS Journal on Computing*, **24**, pp. 343–355. (Cited on page 65.)

Berger, J., Barkaoui, M., and Bräysy, O., (2003), A Route-Directed Hybrid Genetic Approach for the Vehicle Routing Problem with Time Windows, *INFOR*, **41**, pp. 179–194. (Cited on page 69.)

Berger, J., Salois, M., and Begin, R., (1998), A Hybrid Genetic Algorithm for the Vehicle Routing Problem with Time Windows, in: Mercer, R. E., ed., *Advances in Artificial Intelligence*, *Lecture Notes in Computer Science*, vol. 1418, Springer, Berlin, pp. 114–127. (Cited on page 68.)

Bertossi, A. A., Carraresi, P., and Gallo, G., (1987), On Some Matching Problems Arising in Vehicle Scheduling Models, *Networks*, **17**, pp. 271–281. (Cited on page 11.)

Bessière, C., (1994), Arc-Consistency and Arc-Consistency Again, *Artificial Intelligence*, **65**, pp. 179–190. (Cited on page 41.)

Bessière, C., (2006), Constraint Propagation, in: Rossi, F., Beek, P. v., and Walsh, T., eds., *Handbook of Constraint Programming*, Elsevier, Amsterdam, pp. 29–83. (Cited on pages 36, 44, and 45.)

Bessière, C., Freuder, E. C., and Régin, J.-C., (1995), Using Inference to Reduce Arc Consistency Computation, in: Mellish, C. S., ed., *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, Kaufmann, pp. 592–598. (Cited on page 44.)

Bessière, C., Freuder, E. C., and Régin, J.-C., (1999), Using Constraint Metaknowledge to Reduce Arc Consistency Computation, *Artificial Intelligence*, **107**, pp. 125–148. (Cited on page 44.)

Bessière, C. and Régin, J.-C., (2001), Refining the Basic Constraint Propagation Algorithm, in: Nebel, B., ed., *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence*, Kaufmann, San Francisco, pp. 309–315. (Cited on pages 42, 43, and 45.)

Bettinelli, A., Ceselli, A., and Righini, G., (2011), A Branch-and-Cut-and-Price Algorithm for the Multi-Depot Heterogeneous Vehicle Routing Problem with Time Windows, *Transportation Research Part C: Emerging Technologies*, **19**, pp. 723–740. (Cited on pages 22, 25, 27, 95, and 107.)

Blanton, J. L. J. and Wainwright, R. L., (1993), Multiple Vehicle Routing with Time Capacity Constraints using Genetic Algorithms, in: Forrest, S., ed., *Proceedings of the Fifth International Conference on Genetic Algorithms*, Kaufmann, San Mateo and Calif, pp. 452–459. (Cited on page 69.)

BMVBS, (2007), Prognose der deutschlandweiten Verkehrsverflechtungen 2025 (Kurzfassung). (Cited on page 1.)

Bodin, L. and Golden, B., (1981), Classification in Vehicle Routing and Scheduling, *Networks*, **11**, pp. 97–108. (Cited on pages 15, 16, 17, and 26.)

Boland, N., Dethridge, J., and Dumitrescu, I., (2006), Accelerated Label Setting Algorithms for the Elementary Resource Constrained Shortest Path Problem, *Operations Research Letters*, **34**, pp. 58–68. (Cited on page 94.)

Bratko, I., (2001), *Prolog: Programming for Artificial Intelligence*, 3rd edn., Addison Wesley, Harlow and New York. (Cited on page 30.)

Bräysy, O. and Gendreau, M., (2005a), Vehicle Routing Problem with Time Windows, Part I: Route Construction and Local Search Algorithms, *Transportation Science*, **39**, pp. 104–118. (Cited on pages 15 and 68.)

Bräysy, O. and Gendreau, M., (2005b), Vehicle Routing Problem with Time Windows, Part II: Metaheuristics, *Transportation Science*, **39**, pp. 119–139. (Cited on pages 15 and 68.)

Cambazard, H., Hebrard, E., O'Sullivan, B., and Papadopoulos, A., (2012), Local Search and Constraint Programming for the Post Enrolment-Based Course Timetabling Problem, *Annals of Operations Research*, **194**, pp. 111–135. (Cited on page 95.)

Carchrae, T. and Beck, J. C., (2009), Principles for the Design of Large Neighborhood Search, *Journal of Mathematical Modelling and Algorithms*, **8**, pp. 245–270. (Cited on page 106.)

Carpaneto, G. and Toth, P., (1980), Some New Branching and Bounding Criteria for the Asymmetric Travelling Salesman Problem, *Management Science*, **26**, pp. 736–743. (Cited on page 22.)

Caseau, Y. and Laburthe, F., (1994), Improved CLP Scheduling with Task Intervals, in: *Eleventh International Conference on Logic Programming (ICLP'94)*. (Cited on page 54.)

Caseau, Y. and Laburthe, F., (1997), Solving Small TSPs with Constraints, in: Naish, L., ed., *Logic Programming*, MIT Press, Cambridge, pp. 316–330. (Cited on pages 49, 65, and 104.)

Caseau, Y. and Laburthe, F., (1999), Heuristics for Large Constrained Vehicle Routing Problems, *Journal of Heuristics*, **5**, pp. 281–303. (Cited on page 65.)

Caserta, M. and Voß, S., (2009), Metaheuristics: Intelligent Problem Solving, in: Maniezzo, V., Stützle, T., and Voß, S., eds., *Matheuristics*, Springer, New York, pp. 1–38. (Cited on page 67.)

Cassidy, P. J. and Bennett, H. S., (1972), TRAMP—A Multi-Depot Vehicle Scheduling System, *Operational Research Quarterly*, **23**, pp. 151–163. (Cited on pages 19, 24, and 26.)

Chabrier, A., (2006), Vehicle Routing Problem with Elementary Shortest Path Based Column Generation, *Computers & Operations Research*, **33**, pp. 2972–2990. (Cited on page 94.)

Chan, Y., Carter, W. B., and Burnes, M. D., (2001), A Multiple-Depot, Multiple-Vehicle, Location-Routing Problem with Stochastically Processed Demands, *Computers & Operations Research*, **28**, pp. 803–826. (Cited on pages 23, 24, and 27.)

Chandru, V. and Hooker, J. N., (1999), *Optimization Methods for Logical Inference*, Wiley, New York. (Cited on page 2.)

Chao, I.-M., Golden, B. L., and Wasil, E. A., (1993), A New Heuristic for the Multi-Depot Vehicle Routing Problem that Improves upon Best-Known Solutions, *American Journal of Mathematical and Management Sciences*, **13**, pp. 371–406. (Cited on pages 18, 24, and 26.)

Chen, P. and Xu, X., (2008), A Hybrid Algorithm for Multi-Depot Vehicle Routing Problem, in: *IEEE International Conference on Service Operations and Logistics, and Informatics*, vol. 2, IEEE, Piscataway, pp. 2031–2034. (Cited on pages 21, 25, and 27.)

Chen, Z.-L. and Xu, H., (2006), Dynamic Column Generation for Dynamic Vehicle Routing with Time Windows, *Transportation Science*, **40**, pp. 74–88. (Cited on pages 12 and 14.)

Cheng, C.-B. and Wang, K.-P., (2009), Solving a Vehicle Routing Problem with Time Windows by a Decomposition Technique and a Genetic Algorithm, *Expert Systems with Applications*, **36**, pp. 7758–7763. (Cited on page 69.)

Chiang, T.-C. and Hsu, W.-H., (2014), A Knowledge-Based Evolutionary Algorithm for the Multiobjective Vehicle Routing Problem with Time Windows, *Computers & Operations Research*, **45**, pp. 25–37. (Cited on page 69.)

Christie, A. and James, S., (2006), Saving Our Energy Sources and Meeting Kyoto Emission Reduction Targets While Minimizing Costs with Application of Vehicle Logistics Optimization, in: *Annual Conference of the Transportation Association of Canada*. (Cited on page 2.)

Clark, D. A., Frank, J., Gent, I. P., MacIntyre, E., Tomov, N., and Walsh, T., (1996), Local Search and the Number of Solutions, in: Freuder, E. C., ed., *Principles and Practice of Constraint Programming (CP96)*, *Lecture Notes in Computer Science*, vol. 1118, Springer, Berlin, pp. 119–133. (Cited on page 63.)

Clarke, G. and Wright, J. W., (1964), Scheduling of Vehicles from a Central Depot to a Number of Delivery Points, *Operations Research*, **12**, pp. 568–581. (Cited on page 16.)

Contardo, C. and Martinelli, R., (2014), A New Exact Algorithm for the Multi-Depot Vehicle Routing Problem under Capacity and Route Length Constraints, *Discrete Optimization*, **12**, pp. 129–146. (Cited on pages 23, 25, and 28.)

Cooper, M. C., (1989), An Optimal k-Consistency Algorithm, *Artificial Intelligence*, **41**, pp. 89–95. (Cited on page 45.)

Cordeau, J.-F., Gendreau, M., and Laporte, G., (1997), A Tabu Search Heuristic for Periodic and Multi-Depot Vehicle Routing Problems, *Networks*, **30**, pp. 105–119. (Cited on pages 19, 24, and 27.)

Cordeau, J.-F., Gendreau, M., Laporte, G., Potvin, J.-Y., and Semet, F., (2002), A Guide to Vehicle Routing Heuristics, *The Journal of the Operational Research Society*, **53**, pp. 512–522. (Cited on page 26.)

Cordeau, J.-F., Laporte, G., and Mercier, A., (2001), A Unified Tabu Search Heuristic for Vehicle Routing Problems with Time Windows, *The Journal of the Operational Research Society*, **52**, pp. 928–936. (Cited on pages 19, 24, 27, 83, and 84.)

Cormen, T. H. and Sussman, J., (2009), *Introduction to Algorithms*, 3rd edn., MIT Press, Cambridge. (Cited on page 52.)

Cornillier, F., Boctor, F., and Renaud, J., (2012), Heuristics for the Multi-Depot Petrol Station Replenishment Problem with Time Windows, *European Journal of Operational Research*, **220**, pp. 361–369. (Cited on pages 23, 25, and 27.)

Cortés, C. E., Gendreau, M., Rousseau, L.-M., Souyris, S., and Weintraub, A., (2014), Branch-and-Price and Constraint Programming for Solving a Real-Life Technician Dispatching Problem, *European Journal of Operational Research*, **238**, pp. 300–312. (Cited on page 95.)

Costa, M.-C., (1994), Persistency in Maximum Cardinality Bipartite Matchings, *Operations Research Letters*, **15**, pp. 143–149. (Cited on page 47.)

Crevier, B., Cordeau, J.-F., and Laporte, G., (2007), The Multi-Depot Vehicle Routing Problem with Inter-Depot Routes, *European Journal of Operational Research*, **176**, pp. 756–773. (Cited on pages 20, 25, and 27.)

Cruz, J. J., Paternina-Arboleda, C. D., Cantillo, V., and Montoya-Torres, J. R., (2013), A Two-Pheromone Trail Ant Colony System—Tabu Search Approach for the Heterogeneous Vehicle Routing Problem with Time Windows and Multiple Products, *Journal of Heuristics*, **19**, pp. 233–252. (Cited on pages 98 and 99.)

Cruz-Chavez, M. A., Diaz-Parral, O., Hernández, J. A., Zavala-Diaz, J. C., and Martinez-Rangel, M. G., (2007), Search Algorithm for the Constraint Satisfaction Problem of VRPTW, in: *Electronics, Robotics and Automotive Mechanics Conference, 2007*, IEEE, Los Alamitos, pp. 746–751. (Cited on page 65.)

Dabia, S., Ropke, S., Woensel, T. v., and Kok, T. d., (2013), Branch and Price for the Time-Dependent Vehicle Routing Problem with Time Windows, *Transportation Science*, **47**, pp. 380–396. (Cited on page 95.)

Dechter, R., (1990), Enhancement Schemes for Constraint Processing: Backjumping, Learning, and Cutset Decomposition, *Artificial Intelligence*, **41**, pp. 273–312. (Cited on page 56.)

Dechter, R., (2009), *Constraint Processing*, Kaufmann, San Francisco. (Cited on pages 29, 32, and 45.)

Dechter, R. and Dechter, A., (1988), Belief Maintenance in Dynamic Constraint Networks, in: *Proceedings of the National Conference on Artificial Intelligence*, AAAI Press, pp. 37–42. (Cited on page 35.)

Demir, E., Bektas, T., and Laporte, G., (2003), A Review of Recent Research on Green Road Freight Transportation, Working Paper, Eindhoven University of Technology. (Cited on page 1.)

Desaulniers, G., Desrosiers, J., loachim, I., Solomon, M. M., Soumis, F., and Villeneuve, D., (2004), A Unified Framework for Deterministic Time Constrained Vehicle Routing and Crew Scheduling Problems, in: Crainic, T. G., ed., *Fleet Management and Logistics*, Kluwer, Boston, pp. 57–93. (Cited on page 107.)

Desaulniers, G., Lavigne, J., and Soumis, F., (1998), Multi-Depot Vehicle Scheduling Problems with Time Windows and Waiting Costs, *European Journal of Operational Research*, **111**, pp. 479–494. (Cited on page 15.)

Desrochers, M., Desrosiers, J., and Solomon, M. M., (1992), A New Optimization Algorithm for the Vehicle Routing Problem with Time Windows, *Operations Research*, **40**, pp. 342–354. (Cited on pages 94 and 97.)

Desrosiers, J. and Lübbecke, M. E., (2005), A Primer in Column Generation, in: Desaulniers, G., Desrosiers, J., and Solomon, M. M., eds., *Column Generation*, Springer, New York, pp. 1–32. (Cited on page 94.)

Dewilde, T., Cattrysse, D., Coene, S., Spieksma, F. C. R., and Vansteenwegen, P., (2013), Heuristics for the Traveling Repairman Problem with Profits, *Computers & Operations Research*, **40**, pp. 1700–1707. (Cited on page 12.)

Dohn, A., Rasmussen, M. S., and Larsen, J., (2011), The Vehicle Routing Problem with Time Windows and Temporal Dependencies, *Networks*, **58**, pp. 273–289. (Cited on page 95.)

Dondo, R. and Cerdá, J., (2007), A Cluster-Based Optimization Approach for the Multi-Depot Heterogeneous Fleet Vehicle Routing Problem with Time Windows, *European Journal of Operational Research*, **176**, pp. 1478–1507. (Cited on pages 22, 25, and 27.)

Downing, N., Feydy, T., and Stuckey, P. J., (2012), Explaining Alldifferent, in: Reynolds, M. and Thomas, B., eds., *Australasian Computer Science Conference*, *CRPIT*, vol. 122, ACS, Melbourne, Australia, pp. 115–124. (Cited on page 48.)

Easton, K., Nemhauser, G. L., and Trick, M., (2003), Solving the Traveling Tournament Problem: A Combined Integer Programming and Constraint Programming Approach, in: Burke, E. K., ed., *Practice and Theory of Automated Timetabling IV*, *Lecture Notes in Computer Science*, vol. 2740, Springer, Berlin, pp. 100–109. (Cited on page 95.)

Easton, K., Nemhauser, G. L., and Trick, M., (2004), CP Based Branch-and-Price, in: Milano, M., ed., *Constraint and Integer Programming*, Kluwer, Boston. (Cited on page 95.)

Eksioglu, B., Vural, A. V., and Reisman, A., (2009), The Vehicle Routing Problem: A Taxonomic Review, *Computers & Industrial Engineering*, **57**, pp. 1472–1483. (Cited on pages 4 and 16.)

Escobar, J. W., Linfati, R., Toth, P., and Baldoquin, M. G., (2014), A Hybrid Granular Tabu Search Algorithm for the Multi-Depot Vehicle Routing Problem, *Journal of Heuristics*, **20**, pp. 483–509. (Cited on pages 21, 25, and 28.)

Fahle, T., Junker, U., Karisch, S., Kohl, N., Sellmann, M., and Vaaben, B., (2002), Constraint Programming Based Column Generation for Crew Assignment, *Journal of Heuristics*, **8**, pp. 59–81. (Cited on page 95.)

Fahle, T. and Sellmann, M., (2002), Cost Based Filtering for the Constrained Knapsack Problem, *Annals of Operations Research*, **115**, pp. 73–93. (Cited on page 95.)

Feillet, D. and Gendreau, M., (2007), New Refinements for the Solution of Vehicle Routing Problems with Branch and Price, *INFOR*, **45**, pp. 239–256. (Cited on page 94.)

Feillet, D., Laporte, G., Semet, F., Cordeau, J.-F., Iori, M., and Salazar-González, J.-J., (2010), A Branch-and-Cut Algorithm for the Pickup and Delivery Traveling Salesman Problem with LIFO Loading, *Networks*, **55**, pp. 46–59. (Cited on page 94.)

Ferrucci, F., Bock, S., and Gendreau, M., (2013), A Pro-Active Real-Time Control Approach for Dynamic Vehicle Routing Problems Dealing with the Delivery of Urgent Goods, *European Journal of Operational Research*, **225**, pp. 130–141. (Cited on page 12.)

Fikes, R. E., (1970), REF-ARF: A System for Solving Problems Stated as Procedures, *Artificial Intelligence*, **1**, pp. 27–120. (Cited on pages 30 and 37.)

Focacci, F., Lodi, A., and Milano, M., (1999a), Cost-Based Domain Filtering, in: Jaffar, J., ed., *Principles and Practice of Constraint Programming*, *Lecture Notes in Computer Science*, vol. 1713, Springer, Berlin, pp. 189–203. (Cited on page 104.)

Focacci, F., Lodi, A., and Milano, M., (2002), Embedding Relaxations in Global Constraints for Solving TSP and TSPTW, *Annals of Mathematics and Artificial Intelligence*, **34**, pp. 291–311. (Cited on page 65.)

Focacci, F., Lodi, A., Milano, M., and Vigo, D., (1999b), Solving TSP Through the Integration of OR and CP Techniques, *Electronic Notes in Discrete Matehmatics*, **1**, pp. 13–25. (Cited on page 65.)

Foster, A. and Elcock, E. W., (1969), Absys 1: An Incremental Compiler for Assertions: An Introduction, in: Meltzer, B., ed., *Machine Intelligence*, vol. 4, University Press, Edinburgh, pp. 423–429. (Cited on page 30.)

Fourer, R., Gay, D. M., and Kernighan, B. W., (2003), *AMPL: A Modeling Language for Mathematical Programming*, 2nd edn., Thomson/Brooks/Cole, Pacific Grove. (Cited on page 116.)

Francis, K. G. and Stuckey, P. J., (2014), Explaining Circuit Propagation, *Constraints*, **19**, pp. 1–29. (Cited on page 50.)

Freuder, E. C., (1982), A Sufficient Condition for Backtrack-Free Search, *Journal of the ACM*, **29**, pp. 24–32. (Cited on page 59.)

Freuder, E. C., (1997), In Pursuit of the Holy Grail, *Constraints*, **2**, pp. 57–61. (Cited on page 30.)

Freuder, E. C. and Mackworth, A. K., (2006), Constraint Satisfaction: An Emerging Paradigm, in: Rossi, F., Beek, P. v., and Walsh, T., eds., *Handbook of Constraint Programming*, Elsevier, Amsterdam, pp. 13–27. (Cited on page 29.)

Freuder, E. C. and O'Sullivan, B., (2014), Grand Challenges for Constraint Programming, *Constraints*, **19**, pp. 150–162. (Cited on page 47.)

Freuder, E. C. and Wallace, M. G., (1992), Partial Constraint Satisfaction, *Artificial Intelligence*, **58**, pp. 21–70. (Cited on page 35.)

Frühwirth, T. and Abdennadher, S., (1997), *Constraint-Programmierung: Grundlagen und Anwendungen*, Springer, Berlin. (Cited on page 29.)

Funke, B., Grünert, T., and Irnich, S., (2005), Local Search for Vehicle Routing and Scheduling Problems: Review and Conceptual Integration, *Journal of Heuristics*, **11**, pp. 267–306. (Cited on page 95.)

Gabteni, S. and Grönkvist, M., (2009), Combining Column Generation and Constraint Programming to Solve the Tail Assignment Problem, *Annals of Operations Research*, **171**, pp. 61–76. (Cited on page 95.)

Gambardella, L. M., Taillard, E., and Agazzi, G., (1999), MACS-VRPTW: A Multiple Ant Colony System for Vehicle Routing Problems with Time Windows, Technical Report, IDISA. (Cited on page 98.)

García, A., Jodrá, P., and Tejel, J., (2002), A Note on the Traveling Repairman Problem, *Networks*, **40**, pp. 27–31. (Cited on page 12.)

Garey, M. R. and Johnson, D. S., (1979), *Computers and Intractability: A Guide to the Theory of NP-completeness*, W.H. Freeman, San Francisco. (Cited on page 11.)

Garfinkel, R. S. and Nemhauser, G. L., (1969), The Set-Partitioning Problem: Set Covering with Equality Constraints, *Operations Research*, **17**, pp. 848–856. (Cited on page 95.)

Gaschnig, J., (1974), A Constraint Satisfaction Method for Inference Making, in: *Proceedings of the Twelfth Annual Allerton Conference on Circuit and System Theory*, University of Illinois, pp. 866–874. (Cited on page 58.)

Gaschnig, J., (1977), A General Backtrack Algorithm that Eliminates Most Redundant Tests, in: *Fifth International Joint Conference on Artificial Intelligence*, vol. 1, Kaufmann, Los Altos, CA, p. 457. (Cited on pages 55 and 56.)

Gendreau, M., Guertin, F., Potvin, J.-Y., and Séguin, R., (2006), Neighborhood Search Heuristics for a Dynamic Vehicle Dispatching Problem with Pick-Ups and Deliveries, *Transportation Research Part C: Emerging Technologies*, **14**, pp. 157–174. (Cited on page 12.)

Gendreau, M., Guertin, F., Potvin, J.-Y., and Taillard, E., (1999), Parallel Tabu Search for Real-Time Vehicle Routing and Dispatching, *Transportation Science*, **33**, pp. 381–390. (Cited on pages 14 and 82.)

Gendreau, M., Hertz, A., and Laporte, G., (1992), New Insertion and Postoptimization Procedures for the Traveling Salesman Problem, *Operations Research*, **40**, pp. 1086–1094. (Cited on pages 19 and 20.)

Ghédira, K., (2013), *Constraint Satisfaction Problems: CSP Formalisms and Techniques*, Wiley, London. (Cited on pages 29, 32, 36, 38, 44, 53, 58, and 62.)

Ghiani, G., Guerriero, F., Laporte, G., and Musmanno, R., (2003), Real-Time Vehicle Routing: Solution Concepts, Algorithms and Parallel Computing Strategies, *European Journal of Operational Research*, **151**, pp. 1–11. (Cited on pages 2 and 14.)

Ghiani, G., Laporte, G., and Musmanno, R., (2004), *Introduction to Logistics Systems Planning and Control*, Wiley, Hoboken. (Cited on page 13.)

Gillett, B. E. and Johnson, J. G., (1976), Multi-Terminal Vehicle-Dispatch Algorithm, *Omega*, **4**, pp. 711–718. (Cited on pages 18, 24, and 26.)

Gillett, B. E. and Miller, L. R., (1974), A Heuristic Algorithm for the Vehicle-Dispatch Problem, *Operations Research*, **22**, pp. 340–349. (Cited on page 18.)

Giosa, I. D., Tansini, I. L., and Viera, I. O., (2002), New Assignment Algorithms for the Multi-depot Vehicle Routing Problem, *Journal of the Operational Research Society*, **53**, pp. 977–984. (Cited on pages 19, 24, and 27.)

Goldberg, D. E., (1989), *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, Reading. (Cited on pages 76 and 77.)

Golden, B. L., Magnanti, T. L., and Nguyen, H. Q., (1977), Implementing Vehicle Routing Algorithms, *Networks*, **7**, pp. 113–148. (Cited on pages 16, 18, and 22.)

Golden, B. L., Raghavan, S., and Wasil, E. A., eds., (2008), *The Vehicle Routing Problem: Latest Advances and New Challenges*, Springer, New York. (Cited on page 4.)

Groiez, M., Desaulniers, G., Hadjar, A., and Marcotte, O., (2013), Separating Valid Odd-Cycle and Odd-Set Inequalities for the Multiple Depot Vehicle Scheduling Problem, *EURO Journal on Computational Optimization*, **1**, pp. 283–312. (Cited on page 16.)

Grönkvist, M., (2005), The Tail Assignment Problem, Ph.D. Thesis, Chalmers University of Technology and Göteborg University, Göteborg. (Cited on page 95.)

Grönkvist, M., (2006), Accelerating Column Generation for Aircraft Scheduling Using Constraint Propagation, *Computers & Operations Research*, **33**, pp. 2918–2934. (Cited on page 95.)

Gualandi, S. and Malucelli, F., (2009), Constraint Programming-Based Column Generation, *4OR*, **7**, pp. 113–137. (Cited on page 95.)

Gualandi, S. and Malucelli, F., (2013), Constraint Programming-Based Column Generation, *Annals of Operations Research*, **204**, pp. 11–32. (Cited on page 95.)

Guernalec, N. B. and Colmerauer, A., (1997), Narrowing a $2n$-Block of Sortings in $\mathcal{O}(n \log n)$, in: Smolka, G., ed., *Principles and Practice of Constraint Programming*, *Lecture Notes in Computer Science*, vol. 1330, Springer, Berlin, pp. 2–16. (Cited on page 48.)

Gulczynski, D., Golden, B. L., and Wasil, E. A., (2011), The Multi-Depot Split Delivery Vehicle Routing Problem: An Integer Programming-Based Heuristic, New Test Problems, and Computational Results, *Computers & Industrial Engineering*, **61**, pp. 794–804. (Cited on pages 22, 25, and 27.)

Gutiérrez-Jarpa, G., Desaulniers, G., Laporte, G., and Marianov, V., (2010), A Branch-and-Price Algorithm for the Vehicle Routing Problem with Deliveries, Selective Pickups and Time Windows, *European Journal of Operational Research*, **206**, pp. 341–349. (Cited on page 95.)

Hadjar, A., Marcotte, O., and Soumis, F., (2006), A Branch-and-Cut Algorithm for the Multiple Depot Vehicle Scheduling Problem, *Operations Research*, **54**, pp. 130–149. (Cited on page 15.)

Hadjiconstantinou, E. and Baldacci, R., (1998), A Multi-Depot Period Vehicle Routing Problem Arising in the Utilities Sector, *The Journal of the Operational Research Society*, **49**, pp. 1239–1248. (Cited on pages 19, 24, and 27.)

Hansen, P. and Mladenović, N., (2001), Variable Neighborhood Search: Principles and Applications, *European Journal of Operational Research*, **130**, pp. 449–467. (Cited on page 63.)

Hansen, P., Mladenović, N., and Moreno Pérez, J. A., (2010), Variable Neighbourhood Search: Methods and Applications, *Annals of Operations Research*, **175**, pp. 367–407. (Cited on page 63.)

Hanshar, F. T. and Ombuki-Berman, B. M., (2007), Dynamic Vehicle Routing Using Genetic Algorithms, *Applied Intelligence*, **27**, pp. 89–99. (Cited on page 69.)

Haralick, R. M. and Elliot, G. L., (1980), Increasing Tree Search Efficiency for Constraint Satisfaction Problems, *Artificial Intelligence*, **14**, pp. 263–313. (Cited on page 56.)

Harvey, W. D. and Ginsberg, M. L., (1995), Limited Discrepancy Search, in: Mellish, C. S., ed., *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, vol. 1, Kaufmann, pp. 607–615. (Cited on page 59.)

Hentenryck, P. v., (1989), *Constraint Satisfaction in Logic Programming*, MIT Press, Cambridge. (Cited on page 30.)

Hentenryck, P. v., (1992), A Generic Arc-Consistency Algorithm and its Specializations, *Artificial Intelligence*, **57**, pp. 291–321. (Cited on page 44.)

Hentenryck, P. v. and Carillon, J.-P., (1988), Generality Versus Specificity: An Experience with AI and OR Techniques, in: Mitchell, T. M. and Smith, R. G., eds., *Proceedings of the Seventh National Conference on Artificial Intelligence*, The AAAI Press, Menlo Park, pp. 660–664. (Cited on page 48.)

Hentenryck, P. v. and Michel, L., (2005), *Constraint Based Local Search*, MIT Press, Cambridge. (Cited on pages 30 and 64.)

Hewitt, C., (1968), PLANNER: A Language for Proving Theorems in Robots, Ph.D. Thesis, Massachusetts Institute of Technology. (Cited on page 30.)

Ho, W., Ho, G. T., Ji, P., and Lau, H. C. W., (2008), A Hybrid Genetic Algorithm for the Multi-Depot Vehicle Routing Problem, *Engineering Applications of Artificial Intelligence*, **21**, pp. 548–557. (Cited on pages 22, 25, and 27.)

Ho, W.-K., Ang, J. C., and Lim, A., (2001), A Hybrid Search Alogrithm for the Vehicle Routing Problem with Time Windows, *International Journal on Artificial Intelligence Tools*, **10**, pp. 431–449. (Cited on page 69.)

Hoeve, W.-J. v. and Katriel, I., (2006), Global Constraints, in: Rossi, F., Beek, P. v., and Walsh, T., eds., *Handbook of Constraint Programming*, Elsevier, Amsterdam, pp. 169–208. (Cited on pages 47 and 48.)

Holland, J. H., (1975), *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor. (Cited on page 68.)

Hooker, J., (2002), Logic, Optimization, and Constraint Programming, *INFORMS Journal on Computing*, **14**, pp. 295–321. (Cited on pages 2 and 47.)

Hooker, J., (2012), Integrated Methods for Optimization. (Cited on page 2.)

Hooker, J. N., (1994), Logic-Based Methods for Optimization: A Tutorial. (Cited on page 29.)

Hoos, H. H. and Tsang, E. P. K., (2006), Local Search Methods, in: Rossi, F., Beek, P. v., and Walsh, T., eds., *Handbook of Constraint Programming*, Elsevier, Amsterdam, pp. 135–167. (Cited on pages 64 and 95.)

Hopcroft, J. E. and Karp, R. M., (1973), An $n^{5/2}$ Algorithm for Maximum Matchings in Bipartite Graphs, *SIAM Journal on Computing*, **2**, pp. 225–231. (Cited on page 48.)

Ichoua, S., Gendreau, M., and Potvin, J.-Y., (2000), Diversion Issues in Real-Time Vehicle Dispatching, *Transportation Science*, **34**, p. 426. (Cited on page 12.)

Irnich, S., (2000), A Multi-Depot Pickup and Delivery Problem with a Single Hub and Heterogeneous Vehicles, *European Journal of Operational Research*, **122**, pp. 310–328. (Cited on pages 23, 24, and 27.)

Irnich, S. and Villeneuve, D., (2006), The Shortest-Path Problem with Resource Constraints and k-Cycle Elimination for $k \geq 3$, *INFORMS Journal on Computing*, **18**, pp. 391–406. (Cited on page 94.)

Jaffar, J. and Lassez, J.-L., (1987), Constraint Logic Programming, in: *Conference Record of the Fourteenth Annual ACM Symposium on Principles of Programming Languages (POPL '87)*, ACM, New York, pp. 111–119. (Cited on page 30.)

Jaffar, J. and Maher, M., (1994), Constraint Logic Programming: A Survey, *The Journal of Logic Programming*, **19-20**, pp. 503–581. (Cited on page 30.)

Jaffar, J., Michaylov, S., Stuckey, P. J., and Yap, R. H. C., (1992), The CLP(R) Language and System, *ACM Transactions on Programming Languages and Systems*, **14**, pp. 339–395. (Cited on page 30.)

Jeon, G., Leep, H. R., and Shim, J. Y., (2007), A Vehicle Routing Problem Solved by Using a Hybrid Genetic Algorithm, *Computers & Industrial Engineering*, **53**, pp. 680–692. (Cited on pages 21, 25, and 27.)

Jung, S. and Moon, B. R., (2002), A Hybrid Genetic Algorithm for the Vehicle Routing Problem with Time Windows, in: Langdon, W. B., ed., *Proceedings of the Genetic and Evolutionary Computation Conference*, Kaufmann, San Francisco, pp. 1309–1316. (Cited on page 69.)

Jungnickel, D., (2013), *Graphs, Networks, and Algorithms*, 4$^{\text{th}}$ edn., Springer, Berlin. (Cited on pages 34 and 49.)

Junker, U., Karisch, S., Kohl, N., Vaaben, B., Fahle, T., and Sellmann, M., (1999), A Framework for Constraint Programming Based Column Generation, in: Jaffar, J., ed., *Principles and Practice of Constraint Programming*, *Lecture Notes in Computer Science*, vol. 1713, Springer, Berlin, pp. 261–275. (Cited on page 95.)

Jussien, N. and Lhomme, O., (2002), Local Search with Constraint Propagation and Conflict-based Heuristics, *Artificial Intelligence*, **139**, pp. 21–45. (Cited on page 95.)

Kallehauge, B., (2008), Formulations and Exact Algorithms for the Vehicle Routing Problem with Time Windows, *Computers & Operations Research*, **35**, pp. 2307–2330. (Cited on page 15.)

Kallehauge, B., Larsen, J., Madsen, O. B. G., and Solomon, M. M., (2005), Vehicle Routing Problem with Time Windows, in: Desaulniers, G., Desrosiers, J., and Solomon, M. M., eds., *Column Generation*, Springer, New York, pp. 67–98. (Cited on page 15.)

Kara, I. and Bektas, T., (2006), Integer Linear Programming Formulations of Multiple Salesman Problems and its Variations, *European Journal of Operational Research*, **174**, pp. 1449–1458. (Cited on page 16.)

Karp, R. M., (1972), Reducibility Among Combinatorial Problems, in: Miller, R. E., Thatcher, J. W., and Bohlinger, J. D., eds., *Complexity of Computer Computations*, The IBM Research Symposia Series, Springer, Boston, MA, pp. 85–103. (Cited on page 11.)

Kaya, L. G. and Hooker, J. N., (2006), A Filter for the Circuit Constraint, in: Benhamou, F., ed., *Principle and Practice of Constraint Programming - CP 2006*, vol. 4204, Springer, Berlin, pp. 706–710. (Cited on page 50.)

Kilby, P., Prosser, P., and Shaw, P., (1998), Dynamic VRPs: A Study of Scenarios, Technical Report, University of Strathclyde. (Cited on pages 14 and 82.)

Kilby, P., Prosser, P., and Shaw, P., (2000), A Comparison of Traditional and Constraint-Based Heuristic Methods on Vehicle Routing Problems with Side Constraints, *Constraints*, **5**, pp. 389–414. (Cited on page 64.)

Kilby, P. and Shaw, P., (2006), Vehicle Routing, in: Rossi, F., Beek, P. v., and Walsh, T., eds., *Handbook of Constraint Programming*, Elsevier, Amsterdam, pp. 801–836. (Cited on page 64.)

Kimms, A., (1998), Stability Measures for Rolling Schedules with Applications to Capacity Expansion Planning, Master Production Scheduling, and Lot Sizing, *Omega*, **26**, pp. 355–366. (Cited on page 6.)

Kimms, A. and Kozeletskyi, I., (2013), Core-Based Cost Allocation in the Cooperative Traveling Salesman Problem under Rolling Planning Horizon, Working Paper, University of Duisburg-Essen. (Cited on page 15.)

Kimms, A. and Kozeletskyi, I., (2015), Shapley Value-Based Cost Allocation in the Cooperative Traveling Salesman Problem under Rolling Horizon Planning, *EURO Journal on Transportation and Logistics*, to appear. (Cited on page 15.)

Kimms, A. and Reiners, C., (2013), The Vehicle Routing Problem with Time Windows, Collaboration Between Freight Carriers, and a Rolling Planning Horizon, Working Paper, University of Duisburg-Essen. (Cited on page 67.)

Kiziltan, Z., Lodi, A., Milano, M., and Parisini, F., (2012), Bounding, Filtering and Diversification in CP-based Local Branching, *Journal of Heuristics*, **18**, pp. 353–374. (Cited on page 95.)

Kliewer, N., Mellouli, T., and Suhl, L., (2006), A Time-Space Network Based Exact Optimization Model for Multi-Depot Bus Scheduling, *European Journal of Operational Research*, **175**, pp. 1616–1627. (Cited on page 15.)

Klundert, J. v. d. and Otten, B., (2011), Improving LTL Truck Load Utilization on Line, *European Journal of Operational Research*, **210**, pp. 336–343. (Cited on page 13.)

König, D., (1916), Über Graphen und ihre Anwendung auf Determinantentheorie und Mengenlehre, *Mathematische Annalen*, **77**, pp. 453–465. (Cited on page 16.)

Kopfer, H., Pankratz, G., and Erkens, E., (1994), Entwicklung eines hybriden Genetischen Algorithmus zur Tourenplanung, *OR Spectrum*, **16**, pp. 21–31. (Cited on page 68.)

Kreinovich, V., Quintana, C., and Fuentes, O., (1993), Genetic Algorithms: What Fitness Scaling is Optimal?, *Cybernetics and Systems*, **24**, pp. 9–26. (Cited on page 76.)

Krumke, S. O., de Paepe, Willem E., Poensgen, D., and Stougie, L., (2003), News from the Online Traveling Repairman, *Theoretical Computer Science*, **295**, pp. 279–294. (Cited on page 12.)

Kuhn, H. W., (1955), The Hungarian Method for the Assignment Problem, *Naval Research Logistics*, **2**, pp. 83–97. (Cited on page 16.)

Kumar, V., (1992), Algorithms for Constraint-Satisfaction Problems: A Survey, *AI Magazine*, **13**, pp. 32–44. (Cited on pages 29 and 53.)

Kuo, Y. and Wang, C.-C., (2012), A Variable Neighborhood Search for the Multi-Depot Vehicle Routing Problem with Loading Cost, *Expert Systems with Applications*, **39**, pp. 6949–6954. (Cited on pages 21, 25, and 27.)

Langevin, A., Desrochers, M., Desrosiers, J., Gélinas, S., and Soumis, F., (1993), A Two-Commodity Flow Formulation for the Traveling Salesman and the Makespan Problems with Time Windows, *Networks*, **23**, pp. 631–640. (Cited on page 75.)

Laporte, G., (1992), The Vehicle Routing Problem: An Overview of Exact and Approximate Algorithms, *European Journal of Operational Research*, **59**, pp. 345–358. (Cited on page 4.)

Laporte, G., (2009), Fifty Years of Vehicle Routing, *Transportation Science*, **43**, pp. 408–416. (Cited on page 4.)

Laporte, G., Nobert, Y., and Taillefer, S., (1988), Solving a Family of Multi-Depot Vehicle Routing and Location-Routing Problems, *Transportation Science*, **22**, pp. 161–172. (Cited on pages 22, 24, and 26.)

Lau, H. C. W., Chan, T. M., Tsui, W. T., and Pang, W. K., (2010), Application of Genetic Algorithms to Solve the Multidepot Vehicle Routing Problem, *IEEE Transactions on Automation Science and Engineering*, **7**, pp. 383–392. (Cited on pages 22 and 25.)

Laudon, K., (2015), *E-Commerce*, 11th edn., Pearson, Boston. (Cited on page 1.)

Lauriere, J.-L., (1978), A Language and a Program for Stating and Solving Combinatorial Problems, *Artificial Intelligence*, **10**, pp. 29–127. (Cited on pages 30 and 47.)

Lechner, F. J., (2009), *Globalization: The Making of World Society*, Wiley-Blackwell, Chichester. (Cited on page 1.)

Lecoutre, C., (2009), *Constraint Networks: Techniques and Algorithms*, John Wiley & Sons, London. (Cited on page 35.)

Lei, X. M., Xing, C. F., Wu, L., and Wen, Y. F., (2011), Multiple-Depot Vehicle Routing Problems as a Distributed Constraint Optimization Problem, *Applied Mechanics and Materials*, **66-68**, pp. 1033–1038. (Cited on pages 22, 25, and 27.)

Lenstra, J. K. and Kan, A. H. G. R., (1981), Complexity of Vehicle Routing and Scheduling Problems, *Networks*, **11**, pp. 221–227. (Cited on page 11.)

Levin, A. and Yovel, U., (2014), Local Search Algorithms for Multiple-Depot Vehicle Routing and for Multiple Traveling Salesman Problems with Proved Performance Guarantees, *Journal of Combinatorial Optimization*, **28**, pp. 726–747. (Cited on pages 21, 25, and 28.)

Li, C. M., Zhu, Z., Manyà, F., and Simon, L., (2012), Optimizing with Minimum Satisfiability, *Artificial Intelligence*, **190**, pp. 32–44. (Cited on page 35.)

Lim, A. and Wang, F., (2005), Multi-Depot Vehicle Routing Problem: A One-Stage Approach, *IEEE Transactions on Automation Science and Engineering*, **2**, pp. 397–402. (Cited on pages 19, 24, and 27.)

Lin, C., Choy, K. L., Ho, G., Chung, S. H., and Lam, H. Y., (2014), Survey of Green Vehicle Routing Problem: Past and Future Trends, *Expert Systems with Applications*, **41**, pp. 1118–1138. (Cited on page 15.)

Lin, S., (1965), Computer Solutions of the Traveling Salesman Problem, *The Bell System Technical Journal*, **44**, pp. 2245–2269. (Cited on page 18.)

Lin, S. and Kernighan, B. W., (1973), An Effective Heuristic Algorithm for the Traveling-Salesman Problem, *Operations Research*, **21**, pp. 498–516. (Cited on pages 63 and 64.)

Little, J. D. C., Murty, K. G., Sweeney, D. W., and Karel, C., (1963), An Algorithm for the Traveling Salesman Problem, *Operations Research*, **11**, pp. 972–989. (Cited on page 18.)

Liu, R., Jiang, Z., Fung, R. Y., Chen, F., and Liu, X., (2010), Two-Phase Heuristic Algorithms for Full Truckloads Multi-Depot Capacitated Vehicle Routing Problem in Carrier Collaboration, *Computers & Operations Research*, **37**, pp. 950–959. (Cited on pages 20, 25, and 27.)

Lombardi, M. and Schaus, P., (2014), Cost Impact Guided LNS, in: Simonis, H., ed., *Integration of AI and OR Techniques in Constraint Programming*, *Lecture Notes in Computer Science*, vol. 8451, Springer, Berlin, pp. 293–300. (Cited on pages 106 and 107.)

Lopéz-Ortiz, A., Quimper, C.-G., Tromp, J., and Beek, P. v., (2003), A Fast and Simple Algorithm for Bounds Consistency of the All Different Constraint, in: *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence*, pp. 245–250. (Cited on pages 46 and 48.)

Lübbecke, M. E. and Desrosiers, J., (2005), Selected Topics in Column Generation, *Operations Research*, **53**, pp. 1007–1023. (Cited on page 94.)

Luo, Z., Qin, H., and Lim, A., (2014), Branch-and-Price-and-Cut for the Multiple Traveling Repairman Problem with Distance Constraints, *European Journal of Operational Research*, **234**, pp. 49–60. (Cited on pages 12 and 107.)

Mackworth, A. K., (1977), Consistency in Networks of Relations, *Artificial Intelligence*, **8**, pp. 99–118. (Cited on pages 30, 36, 37, and 38.)

Mairy, J.-B., Schaus, P., and Deville, Y., (2010), Generic Adaptive Heuristics for Large Neighborhood Search, in: *Seventh International Workshop on Local Search Techniques in Constraint Satisfaction*. (Cited on page 95.)

Marriott, K. and Stuckey, P. J., (1999), *Programming with Constraints: An Introduction*, 2nd edn., MIT Press, Cambridge. (Cited on pages 29, 32, and 46.)

Mehlhorn, K. and Thiel, S., (2000), Faster Algorithms for Bound-Consistency of the Sortedness and the Alldifferent Constraint, in: Dechter, R., ed., *Principles and Practice of Constraint Programming*, *Lecture Notes in Computer Science*, vol. 1894, Springer, Berlin, pp. 306–319. (Cited on page 48.)

Meng, Q., Wang, S., Andersson, H., and Thun, K., (2014), Containership Routing and Scheduling in Liner Shipping: Overview and Future Research Directions, *Transportation Science*, **48**, pp. 265–280. (Cited on page 13.)

Mesquita, M. and Paixão, J., (1992), Multiple Depot Vehicle Scheduling Problem: A New Heuristic Based on Quasi-Assignment Algorithms, in: Desrochers, M. and Rousseau, J.-M., eds., *Computer-Aided Transit Scheduling*, *Lecture Notes in Economics and Mathematical Systems*, vol. 386, Springer, Berlin and New York, pp. 167–180. (Cited on page 15.)

Michel, L., Schulte, C., and Hentenryck, P. v., (2007), Constraint Programming Tools, in: Benhamou, F., Jussien, N., and O'Sullivan, B., eds., *Trends in Constraint Programming*, ISTE, London, pp. 41–57. (Cited on page 29.)

Milano, M. and Trick, M., (2004), Constraint and Integer Prorgamming: Basic Concepts, in: Milano, M., ed., *Constraint and Integer Programming*, Kluwer, Boston, pp. 1–31. (Cited on page 30.)

Milano, M. and Wallace, M. G., (2010), Integrating Operations Research in Constraint Programming, *Annals of Operations Research*, **175**, pp. 37–76. (Cited on page 95.)

Mohr, R. and Henderson, T. C., (1986), Arc and Path Consistency Revisited, *Artificial Intelligence*, **28**, pp. 225–233. (Cited on pages 39 and 40.)

Montanari, U., (1974), Networks of Constraints: Fundamental Properties and Applications to Picture Processing, *Information Sciences*, **7**, pp. 95–132. (Cited on pages 30 and 45.)

Montemanni, R., Gambardella, L. M., Rizzoli, A. E., and Donati, A. V., (2005), Ant Colony System for a Dynamic Vehicle Routing Problem, *Journal of Combinatorial Optimization*, **10**, pp. 327–343. (Cited on pages 14 and 82.)

Mourgaya, M. and Vanderbeck, F., (2006), The Periodic Vehicle Routing Problem: Classification and Heuristic, *RAIRO - Operations Research*, **40**, pp. 169–194. (Cited on pages 12 and 15.)

Mouthuy, S., Hentenryck, P. v., and Deville, Y., (2012), Constraint-Based Very Large-Scale Neighborhood Search, *Constraints*, **17**, pp. 87–122. (Cited on page 95.)

Muter, I., Cordeau, J.-F., and Laporte, G., (2014), A Branch-and-Price Algorithm for the Multidepot Vehicle Routing Problem with Interdepot Routes, *Transportation Science*, **48**, pp. 425–441. (Cited on pages 23, 25, and 28.)

Nadel, B. A., (1988), Tree Search and Arc Consistency in Constraint Satisfaction Algorithms, in: Kanal, L. N., ed., *Search in artificial intelligence*, Symbolic Computation, Springer, New York, pp. 287–342. (Cited on page 53.)

Nagy, G. and Salhi, S., (2005), Heuristic Algorithms for Single and Multiple Depot Vehicle Routing Problems with Pickups and Deliveries, *European Journal of Operational Research*, **162**, pp. 126–141. (Cited on pages 20, 24, and 27.)

Ninikas, G. and Minis, I., (2014), Reoptimization Strategies for a Dynamic Vehicle Routing Problem with Mixed Backhauls, *Networks*, **64**, pp. 214–231. (Cited on page 12.)

Onoyama, T., Maekawa, T., Sakurai, Y., Tsuruta, S., and Komoda, N., (2008), Selfish Constraint Satisfaction Genetic Algorithm for Planning a Long-Distance Transportation Network, *Journal of Computers*, **3**, pp. 77–85. (Cited on page 65.)

Pang, G. and Muyldermans, L., (2012), Vehicle Routing and the Value of Postponement, *Journal of the Operational Research Society*, **64**, pp. 1429–1440. (Cited on page 12.)

Pankratz, G., (2005), Dynamic Vehicle Routing by Means of a Genetic Algorithm, *International Journal of Physical Distribution & Logistics Management*, **35**, pp. 362–383. (Cited on pages 13 and 14.)

Pape, C. L., (2010), Constraint Programming, in: Paschos, V. T., ed., *Concepts of combinatorial optimization*, *Combinatorial optimization*, vol. 1, ISTE, London, pp. 325–338. (Cited on page 29.)

Parthanadee, P. and Logendran, R., (2006), Periodic Product Distribution from Multi-Depots Under Limited Supplies, *IIE Transactions*, **38**, pp. 1009–1026. (Cited on pages 22, 24, and 27.)

Pepin, A.-S., Desaulniers, G., Hertz, A., and Huisman, D., (2009), A Comparison of Five Heuristics for the Multiple Depot Vehicle Scheduling Problem, *Journal of Scheduling*, **12**, pp. 17–30. (Cited on page 15.)

Perl, J. and Daskin, M. S., (1985), A Warehouse Location-Routing Problem, *Transportation Research Part B: Methodological*, **19**, pp. 381–396. (Cited on pages 16, 24, and 26.)

Pesant, G. and Gendreau, M., (1996), A View of Local Search in Constraint Programming, in: Freuder, E. C., ed., *Principles and Practice of Constraint Programming (CP96), Lecture Notes in Computer Science*, vol. 1118, Springer, Berlin, pp. 353–366. (Cited on page 95.)

Pesant, G., Gendreau, M., Potvin, J.-Y., and Rousseau, J.-M., (1998), An Exact Constraint Logic Programming Algorithm for the Traveling Salesman Problem with Time Windows, *Transportation Science*, **32**, pp. 12–29. (Cited on pages 65, 74, and 75.)

Pesant, G., Gendreau, M., Potvin, J.-Y., and Rousseau, J.-M., (1999), On the Flexibility of Constraint Programming Models: From Single to Multiple Time Windows for the Traveling Salesman Problem, *European Journal of Operational Research*, **117**, pp. 253–263. (Cited on page 65.)

Pillac, V., Gendreau, M., Guéret, C., and Medaglia, A. L., (2013), A Review of Dynamic Vehicle Routing Problems, *European Journal of Operational Research*, **225**, pp. 1–11. (Cited on page 14.)

Pisinger, D. and Ropke, S., (2007), A General Heuristic for Vehicle Routing Problems, *Computers & Operations Research*, **34**, pp. 2403–2435. (Cited on pages 20, 25, and 27.)

Pisinger, D. and Ropke, S., (2010), Large Neighborhood Search, in: Gendreau, M. and Potvin, J.-Y., eds., *Handbook of Metaheuristics*, vol. 146, Springer, New York, pp. 399–419. (Cited on page 64.)

Polacek, M., Benkner, S., Doerner, K. F., and Hartl, R. F., (2008), A Cooperative and Adaptive Variable Neighborhood Search for the Multi Depot Vehicle Routing Problem with Time Windows, *BuR - Business Research*, **1**, pp. 207–218. (Cited on pages 20, 25, and 27.)

Polacek, M., Hartl, R. F., Doerner, K. F., and Reimann, M., (2004), A Variable Neighborhood Search for the Multi Depot Vehicle Routing Problem with Time Windows, *Journal of Heuristics*, **10**, pp. 613–627. (Cited on pages 20, 24, and 27.)

Porter, M. E., (2004), *Competitive Strategy: Techniques for Analyzing Industries and Competitors*, Free Press, New York. (Cited on page 1.)

Potvin, J.-Y. and Bengio, S., (1996), The Vehicle Routing Problem with Time Windows, Part II: Genetic Search, *INFORMS Journal on Computing*, **8**, pp. 165–172. (Cited on page 69.)

Prosser, P., (1993), Hybrid Algorithms for the Constraint Satisfaction Problem, *Computational Intelligence*, **9**, pp. 268–299. (Cited on page 56.)

Prud'homme, C., Lorca, X., and Jussien, N., (2014), Explanation-Based Large Neighborhood Search, *Constraints*, **19**, pp. 339–379. (Cited on page 95.)

Psaraftis, H. N., (1980), A Dynamic Programming Solution to the Single Vehicle Many-to-Many Immediate Request Dial-a-Ride Problem, *Transportation Science*, **14**, p. 130. (Cited on page 14.)

Psaraftis, H. N., (1991), Dynamic Vehicle Routing Problems, in: Golden, B. L., ed., *Vehicle Routing*, North-Holland, Amsterdam, pp. 223–248. (Cited on page 14.)

Psaraftis, H. N., (1995), Dynamic Vehicle Routing: Status and Prospects, *Annals of Operations Research*, **61**, pp. 143–164. (Cited on page 13.)

Puchinger, J., Stuckey, P. J., Wallace, M. G., and Brand, S., (2011), Dantzig-Wolfe Decomposition and Branch-and-Price Solving in G12, *Constraints*, **16**, pp. 77–99. (Cited on page 95.)

Rahimi-Vahed, A., Crainic, T. G., Gendreau, M., and Rei, W., (2013), A Path Relinking Algorithm for a Multi-Depot Periodic Vehicle Routing Problem, *Journal of Heuristics*, **19**, pp. 497–524. (Cited on pages 21, 25, and 28.)

Rechenberg, I., (1973), *Evolutionsstrategie*, Frommann-Holzboog, Stuttgart. (Cited on page 68.)

Régin, J.-C., (1994), A Filtering Algorithm for Constraints of Difference in CSPs, in: *Proceedings of the Twelfth National Conference on Artificial Intelligence*, vol. 1, AAAI Press, Menlo Park, pp. 362–367. (Cited on page 47.)

Régin, J.-C., (2011), Global Constraints: A Survey, in: Hentenryck, P. v. and Milano, M., eds., *Hybrid Optimization*, Springer, New York, pp. 63–134. (Cited on page 47.)

Renaud, J., Laporte, G., and Boctor, F. F., (1996), A Tabu Search Heuristic for the Multi-Depot Vehicle Routing Problem, *Computers & Operations Research*, **23**, pp. 229–235. (Cited on pages 19, 24, and 27.)

Righini, G. and Salani, M., (2008), New Dynamic Programming Algorithms for the Resource Constrained Elementary Shortest Path Problem, *Networks*, **51**, pp. 155–170. (Cited on page 94.)

Ropke, S. and Pisinger, D., (2006), An Adaptive Large Neighborhood Search Heuristic for the Pickup and Delivery Problem with Time Windows, *Transportation Science*, **40**, pp. 455–472. (Cited on page 20.)

Rosenfeld, A., Hummel, R. A., and Zucker, S. W., (1976), Scene Labeling by Relaxation Operations, *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, **6**, pp. 420–433. (Cited on page 37.)

Rousseau, L.-M., Gendreau, M., and Pesant, G., (2002a), Solving Small VRPTWs with Constraint Programming Based Column Generation, in: Jussien, N. and Laburthe, F., eds., *Proceedings of the Fourth International Workshop on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimisation Problems*, pp. 333–344. (Cited on pages 65, 95, 104, and 105.)

Rousseau, L.-M., Gendreau, M., and Pesant, G., (2002b), Using Constrained-Based Operators to Solve the Vehicle Routing Problem with Time Tindows, *Journal of Heuristics*, **8**, pp. 43–58. (Cited on page 65.)

Rousseau, L.-M., Gendreau, M., and Pesant, G., (2013), The Synchronized Dynamic Vehicle Dispatching Problem, *INFOR*, **51**, pp. 76–83. (Cited on page 14.)

Rousseau, L.-M., Gendreau, M., Pesant, G., and Focacci, F., (2004), Solving VRPTWs with Constraint Programming Based Column Generation, *Annals of Operations Research*, **130**, pp. 199–216. (Cited on pages 65, 95, 102, and 104.)

Sabin, D. and Freuder, E. C., (1994), Contradicting Conventional Wisdom in Constraint Satisfaction, in: Borning, A., ed., *Principles and Practice of Constraint Programming*, *Lecture Notes in Computer Science*, vol. 874, Springer, Berlin, pp. 10–20. (Cited on page 58.)

Sahin, F., Narayanan, A., and Robinson, E. P., (2013), Rolling Horizon Planning in Supply Chains: Review, Implications and Directions for Future Research, *International Journal of Production Research*, **51**, pp. 5413–5436. (Cited on page 14.)

Salani, M. and Vacca, I., (2011), Branch and Price for the Vehicle Routing Problem with Discrete Split Deliveries and Time Windows, *European Journal of Operational Research*, **213**, pp. 470–477. (Cited on page 95.)

Salhi, S., Imran, A., and Wassan, N. A., (2014), The Multi-Depot Vehicle Routing Problem with Heterogeneous Vehicle Fleet: Formulation and a Variable Neighborhood Search Implementation, *Computers & Operations Research*, **52**, pp. 315–325. (Cited on pages 21, 25, and 28.)

Salhi, S. and Nagy, G., (1999), A Cluster Insertion Heuristic for Single and Multiple Depot Vehicle Routing Problems with Backhauling, *The Journal of the Operational Research Society*, **50**, pp. 1034–1042. (Cited on pages 18, 24, and 27.)

Salhi, S. and Sari, M., (1997), A Multi-Level Composite Heuristic for the Multi-Depot Vehicle Fleet Mix Problem, *European Journal of Operational Research*, **103**, pp. 95–112. (Cited on pages 22, 24, and 27.)

Salhi, S., Thangiah, S. R., and Rahman, F., (1998), A Genetic Clustering Method for the Multi-Depot Vehicle Routing Problem, in: Smith, G. D., Steele, N. C., and Albrecht, R. F., eds., *Artificial Neural Nets and Genetic Algorithms*, Springer, Vienna, pp. 234–237. (Cited on pages 21, 24, and 27.)

Sariklis, D. and Powell, S., (2000), A Heuristic Method for the Open Vehicle Routing Problem, *The Journal of the Operational Research Society*, **51**, pp. 564–573. (Cited on page 71.)

Schiex, T., Fargier, H., and Verfaillie, G., (1995), Valued Constraint Satisfaction Problems: Hard and Easy Problems, in: Mellish, C. S., ed., *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, Kaufmann, pp. 631–637. (Cited on page 35.)

Schneider, M., Stenger, A., and Goeke, D., (2014), The Electric Vehicle-Routing Problem with Time Windows and Recharging Stations, *Transportation Science*, **48**, pp. 500–520. (Cited on page 15.)

Schönberger, J., (2011), *Model Based Control of Logistics Processes in Volatile Environments: Decision Support for Operations Planning in Supply Consortia*, Springer, New York. (Cited on page 6.)

Schulte, C. and Carlsson, M., (2006), Finite Domain Constraint Progamming Systems, in: Rossi, F., Beek, P. v., and Walsh, T., eds., *Handbook of Constraint Programming*, Elsevier, Amsterdam, pp. 495–526. (Cited on page 29.)

Schulte, C., Tack, G., and Lagerkvist, M. Z., eds., (2015), *Modeling and Programming with Gecode*. (Cited on pages 84 and 110.)

Schwefel, H.-P., (1977), *Numerische Optimierung von Computer-Modellen mittels der Evolutionsstrategie: Mit einer vergleichenden Einführung in die Hill-Climbing- und Zufallsstrategie*, Birkhäuser, Basel. (Cited on page 68.)

Sellmann, M., Zervoudakis, K., Stamatopoulos, P., and Fahle, T., (2002), Crew Assignment via Constraint Programming: Integrating Column Generation and Heuristic Tree Search, *Annals of Operations Research*, **115**, pp. 207–225. (Cited on page 95.)

Shaw, P., (1998), Using Constraint Programming and Local Search Methods to Solve Vehicle Routing Problems, in: Maher, M. and Puget, J.-F., eds., *Principles and Practice of Constraint Programming*, *Lecture Notes in Computer Science*, vol. 1520, Springer, Berlin, New York, pp. 417–431. (Cited on pages 64, 65, 95, and 105.)

Solnon, C., (2010), *Ant Colony Optimization and Constraint Programming*, ISTE, London. (Cited on page 64.)

Solomon, M. M., (1987), Algorithms for the Vehicle Routing and Scheduling Problems with Time Window Constraints, *Operations Research*, **35**, pp. 254–265. (Cited on page 83.)

Song, J. and Regan, A., (2001), Transition or Transformation? Emerging Freight Transportation Intermediaries, *Transportation Research Record*, **1763**, pp. 1–5. (Cited on page 12.)

Stefik, M., (1981), Planning with Constraints (MOLGEN: Part 1), *Artificial Intelligence*, **16**, pp. 111–139. (Cited on page 30.)

Subramanian, A., Uchoa, E., and Ochi, L. S., (2013), A Hybrid Algorithm for a Class of Vehicle Routing Problems, *Computers & Operations Research*, **40**, pp. 2519–2531. (Cited on pages 21, 25, and 28.)

Sumichras, R. T. and Markham, I. S., (1995), A Heuristic and Lower Bound for a Multi-Depot Routing Problem, *Computers & Operations Research*, **22**, pp. 1047–1056. (Cited on pages 18, 24, and 27.)

Sussman, G. J. and Steele, G. L., (1980), Constraints: A Language for Expressing Almost-hierarchical Descriptions, *Artificial Intelligence*, **14**, pp. 1–39. (Cited on page 30.)

Sutherland, I. E., (1963), Sketchpad: A Man-Machine Graphical Communication System, Ph.D. Thesis, Massachusetts Institute of Technology. (Cited on page 29.)

Taillard, É., Badeau, P., Gendreau, M., Guertin, F., and Potvin, J.-Y., (1997), A Tabu Search Heuristic for the Vehicle Routing Problem with Soft Time Windows, *Transportation Science*, **31**, pp. 170–186. (Cited on page 20.)

Tan, K. C., Lee, T. H., Ou, K., and Lee, L. H., (2001), A Messy Genetic Algorithm for the Vehicle Routing Problem with Time Window Constraints, in: *Proceedings of the 2001 Congress on Evolutionary Computation*, vol. 1, pp. 679–686. (Cited on page 69.)

Tansini, L. and Viera, O., (2006), New Measures of Proximity for the Assignment Algorithms in the MDVRPTW, *The Journal of the Operational Research Society*, **57**, pp. 241–249. (Cited on pages 19, 24, and 27.)

Tarantilis, C. D. and Kiranoudis, C. T., (2002), Distribution of Fresh Meat, *Journal of Food Engineering*, **51**, pp. 85–91. (Cited on pages 20, 24, and 27.)

Thangiah, S. R., Nygard, K., and Juell, P., (1991), GIDEON: A Genetic Algorithm System for Vehicle Routing with Time Windows, in: IEEE, ed., *The Seventh IEEE Conference on Artificial Intelligence Application*, pp. 322–328. (Cited on page 68.)

Thangiah, S. R., Osman, I. H., Vinayagamoorthy, R., and Sun, T., (1993), Algorithms for the Vehicle Routing Problems with Time Deadlines, *American Journal of Mathematical and Management Sciences*, **13**, pp. 323–355. (Cited on page 68.)

Thangiah, S. R. and Salhi, S., (2001), Genetic Clustering: An Adaptive Heuristic for the Multidepot Vehicle Routing Problem, *Applied Artificial Intelligence*, **15**, pp. 361–383. (Cited on pages 21, 24, and 27.)

Tillman, F. A., (1969), The Multiple Terminal Delivery Problem with Probabilistic Demands, *Transportation Science*, **3**, pp. 192–204. (Cited on pages 16, 24, and 26.)

Tillman, F. A. and Cain, T. M., (1972), An Upperbound Algorithm for the Single and Multiple Terminal Delivery Problem, *Management Science*, **18**, pp. 664–682. (Cited on pages 16, 24, and 26.)

Tirado, G., Hvattum, L. M., Fagerholt, K., and Cordeau, J.-F., (2013), Heuristics for Dynamic and Stochastic Routing in Industrial Shipping, *Computers & Operations Research*, **40**, pp. 253–263. (Cited on page 13.)

Toth, P. and Vigo, D., (2002a), An Overview of Vehicle Routing Problems, in: Toth, P. and Vigo, D., eds., *The Vehicle Routing Problem*, Society for Industrial and Applied Mathematics, Philadelphia, pp. 1–26. (Cited on page 11.)

Toth, P. and Vigo, D., eds., (2002b), *The Vehicle Routing Problem*, Society for Industrial and Applied Mathematics, Philadelphia. (Cited on page 4.)

Toth, P. and Vigo, D., (2003), The Granular Tabu Search and its Application to the Vehicle-Routing Problem, *INFORMS Journal on Computing*, **15**, pp. 333–346. (Cited on page 21.)

Vidal, T., Crainic, T. G., Gendreau, M., Lahrichi, N., and Rei, W., (2012), A Hybrid Genetic Algorithm for Multidepot and Periodic Vehicle Routing Problems, *Operations Research*, **60**, pp. 611–624. (Cited on pages 22, 25, and 28.)

Vidal, T., Crainic, T. G., Gendreau, M., and Prins, C., (2013a), Heuristics for Multi-Attribute Vehicle Routing Problems: A Survey and Synthesis, *European Journal of Operational Research*, **231**, pp. 1–21. (Cited on page 15.)

Vidal, T., Crainic, T. G., Gendreau, M., and Prins, C., (2013b), A Hybrid Genetic Algorithm with Adaptive Diversity Management for a Large Class of Vehicle Routing Problems with Time-Windows, *Computers & Operations Research*, **40**, pp. 475–489. (Cited on pages 22 and 69.)

Vidal, T., Crainic, T. G., Gendreau, M., and Prins, C., (2014), A Unified Solution Framework for Multi-Attribute Vehicle Routing Problems, *European Journal of Operational Research*, **234**, pp. 658–673. (Cited on page 15.)

Voudouris, C., Tsang, E. P. K., and Alsheddy, A., (2010), Guided Local Search, in: Gendreau, M. and Potvin, J.-Y., eds., *Handbook of Metaheuristics*, vol. 146, Springer, New York, pp. 321–361. (Cited on page 63.)

Waltz, D., (1975), Understanding Line Drawings of Scenes with Shadows, in: Winston, P. H., ed., *The Psychology of Computer Vision*, McGraw-Hill, New York, pp. 19–91. (Cited on page 30.)

Wang, X. and Kopfer, H., (2013), Dynamic Collaborative Transportation Planning: A Rolling Horizon Planning Approach, in: Pacino, D., Voss, S., and Jensen, R. M., eds., *Proceedings of the Fourth International Conference on Computational Logistics*, *Lecture Notes in Computer Science*, vol. 8197, pp. 128–142. (Cited on page 15.)

Wang, X. and Kopfer, H., (2014), Collaborative Transportation Planning of Less-Than-Truckload Freight, *OR Spectrum*, **36**, pp. 357–380. (Cited on page 15.)

Wang, X. and Kopfer, H., (2015), Rolling Horizon Planning for a Dynamic Collaborative Routing Problem with Full-Truckload Pickup and Delivery Requests, *Flexible Services and Manufacturing Journal*, to appear. (Cited on page 15.)

Wang, X., Kopfer, H., and Gendreau, M., (2014), Operational Transportation Planning of Freight Forwarding Companies in Horizontal Coalitions, *European Journal of Operational Research*, **237**, pp. 1133–1141. (Cited on page 15.)

Winston, W. L., (2005), *Operations Research*, 4^{th} edn., Thomson Brooks, Belmont. (Cited on page 61.)

Wren, A. and Holliday, A., (1972), Computer Scheduling of Vehicles from One or More Depots to a Number of Delivery Points, *Operational Research Quarterly*, **23**, pp. 333–344. (Cited on pages 18, 24, and 26.)

Xiong, G. and Wang, Y., (2014), Best Routes Selection in Multimodal Networks Using Multi-Objective Genetic Algorithm, *Journal of Combinatorial Optimization*, **28**, pp. 655–673. (Cited on pages 25 and 28.)

Yang, J., Jaillet, P., and Mahmassani, H., (2004), Real-Time Multivehicle Truckload Pickup and Delivery Problems, *Transportation Science*, **38**, pp. 135–148. (Cited on page 14.)

Yang, W.-T. and Chu, L.-C., (2000), A Heuristic Algorithm for the Multi-Depot Periodic Vehicle Routing Problem, *Journal of Information and Optimization Sciences*, **21**, pp. 359–367. (Cited on pages 19, 24, and 27.)

Yanik, S., Bozkaya, B., and deKervenoael, R., (2014), A New VRPPD Model and a Hybrid Heuristic Solution Approach for E-tailing, *European Journal of Operational Research*, **236**, pp. 879–890. (Cited on page 15.)

Yokoo, M., (1997), Why Adding More Constraints Makes a Problem Easier for Hill-Climbing Algorithms: Analyzing Landscapes of CSPs, in: Smolka, G., ed., *Principles and Practice of Constraint Programming*, *Lecture Notes in Computer Science*, vol. 1330, Springer, Berlin, pp. 356–370. (Cited on page 63.)

Yunes, T., Moura, A., and Souza, C. d., (2000), Solving Very Large Crew Scheduling Problems to Optimality, in: Carroll, J., Damiani, E., Haddad, H., and Oppenheim, D., eds., *Proceedings of the 2000 ACM Symposium on Applied Computing*, ACM, New York, pp. 446–451. (Cited on page 95.)

Yunes, T., Moura, A., and Souza, C. d., (2005), Hybrid Column Generation Approaches for Urban Transit Crew Management Problems, *Transportation Science*, **39**, pp. 273–288. (Cited on page 95.)

Zhang, J., Tang, J., and Fung, R. Y. K., (2011), A Scatter Search For Multi-Depot Vehicle Routing Problem with Weight-Related Cost, *Asia-Pacific Journal of Operational Research*, **28**, p. 323. (Cited on pages 22, 25, and 27.)

Zhang, Y. and Yap, H. C., (2001), Making AC-3 an Optimal Algorithm, in: Nebel, B., ed., *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence*, vol. 1, Kaufmann, San Francisco, pp. 316–321. (Cited on page 39.)

Zhu, K. Q., (2000), A New Genetic Algorithm For VRPTW, in: *Proceedings of the International Conference on Artificial Intelligence*. (Cited on page 69.)

Acknowledgement

This work was done with financial support from Deutsche Forschungsgemeinschaft (DFG) under grants KI 1272/5-1 and KI 1272/5-2 (”Kooperative Rundreiseprobleme bei rollierender Planung”).