On a Vehicle Routing Problem with Customer Costs and Multi Depots

D I S S E R T A T I O N

zur Erlangung des akademischen Grades
Doctor rerum naturalium
(Dr. rer. nat.)

vorgelegt

der Fakultät Mathematik und Naturwissenschaften
der Technischen Universität Dresden

von

Dipl. Math (FH) Franziska Theurich geb. Heinicke

geboren am 25.07.1987 in Altenburg

Tag der Einreichung:	7. Juli 2022
Gutachter:	Prof. Dr. Andreas Fischer, Technische Universität Dresden Prof. Dr. Thomas Kalinowski, Hochschule Mittweida
Tag der Verteidigung:	9. Dezember 2022

Danksagung

Schon Goethe soll gesagt haben, dass sich wahre Dankbarkeit nicht mit Worten ausdrücken lässt. Trotzdem möchte ich an dieser Stelle den Personen, ohne deren Anregungen und Unterstützung diese Promotionsschrift nicht zustande gekommen wäre, meinen besonderen Dank aussprechen.

Zunächst möchte ich meinen wissenschaftlichen Betreuern, Herrn Prof. Andreas Fischer und Herrn Dr. Guntram Scheithauer, dafür danken, dass sie meinem Themenvorschlag von Beginn an interessiert gegenüberstanden und mir die Möglichkeit gegeben haben, neben dem Beruf zu promovieren. Ihre vielfachen konstruktiven Anregungen und die fachlichen Diskussionen haben meinen Blick auf den mathematischen Teil des Themas gelenkt und damit sehr zum Gelingen der Arbeit beigetragen. Außerdem möchte ich Herrn Dr. Guntram Scheithauer für die zahlreichen Korrekturvorschläge danken.

Mein Dank gilt auch Herrn Prof. Peter Tittmann für die Betreuung im kooperativen Verfahren mit der Hochschule Mittweida und Herrn Prof. Thomas Kalinowski für sein Interesse am Thema und inspirierende Gespräche.

In besonderem Maße möchte ich meiner Kollegin Dr. Ute Gläser vom Fraunhofer IVI für ihre konstruktive Kritik und die zahlreichen Verbesserungsvorschläge danken. Sie stand mir – gerade auch in der finalen Phase der Promotion – stets mit ihrem Rat zur Seite. Mein Dank gilt auch meinen Kollegen am Fraunhofer IVI, vor allem Axel Simroth, der mich auf die Idee zur Promotion gebracht hat, sowie Denise Holfeld und Maximilian Rosner für ihre Rücksichtnahme und die freundschaftlichen Ermunterungen.

Schließlich möchte ich meinem Ehemann Jörg Theurich für den Rückhalt und die vielen kinderfreien Sonntage in den letzten Jahren danken.

Contents

Nomenclature

Acronyms

LP	linear program
MILP	mixed-integer linear program
TSP	traveling salesman problem
VRP	vehicle routing problem
CVRP	capacitated vehicle routing problem
VRPTW	vehicle routing problem with time windows
VRPCC	vehicle routing problem with customer costs
MST	minimal spanning tree
BPP	bin packing problem

Definitions

$\lfloor x \rfloor$	rounding to the largest integer less than or equal to x
$\lceil x \rceil$	rounding to the smallest integer greather than or equal to x
$\lfloor x \rceil$	rounding to the nearest integer, which is $\lfloor x + \frac{1}{2} \rfloor$

Variables

N	set of jobs $N = \{1, 2, \ldots, n\}$
M	set of vehicles $M = \{1, 2, \ldots, m\}$
N_s	set of start depots $N_s = \{s_k \mid k \in M\}$
N_z	set of end depots $N_z = \{z_k \mid k \in M\}$
N_a	set of all jobs and depots
$g^c(S)$	customer costs of a solution S
$g^d(S)$	travel costs of a solution S
$g(S)$	total costs of a solution S computed as $g(S) := g^c(S) + g^d(S)$
c_i	customer cost coefficient of job i
a_i	working duration of job i
d_{ij}	costs for travelling from job i to job j
r_{ij}	time for travelling from job i to job j
$t_i = (t_i^d, t_i^m)$	tuple of start day and minute of job i
u	working shift length
u_i	latest start minute of job i at any day

1. Introduction

The vehicle routing problem is a widely used approach in transportation planning to schedule deliveries. Roughly speaking, routes are defined to serve the demands of customers by a fleet. One of its first known applications was related to a truck dispatching problem investigated in [37], which aims in routing m gasoline delivery trucks between a bulk terminal and n gasoline stations that have to be supplied. Thereby, the gasoline stations have a certain demand for fuel and the trucks a limited capacity to load fuel. In Figure 1.1, the problem is illustrated.

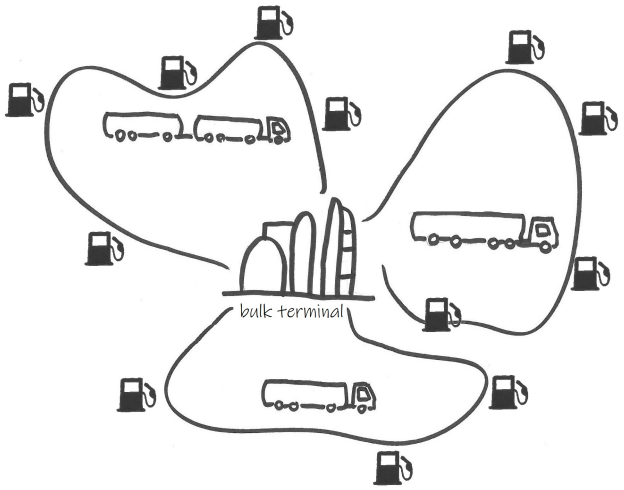


Figure 1.1.: The gasoline truck dispatching problem.

The resulting optimization problem is called capacitated vehicle routing problem. The gasoline stations are the customers $i \in \{1, 2, ..., n\}$ and each customer has a demand q_i. The trucks are the vehicles $k \in \{1, 2, ..., m\}$, each with a given capacity Q_k. The bulk terminal is the depot represented by 0. Between each two customers inclusive the depot $i, j \in \{0, 1, 2, ..., n\}$, a journey is computed in advance to determine the costs c_{ij} for traveling from customer i to customer j, which could be, e.g., proportional to the distance. Then, a feasible solution of the capacitated vehicle routing problem is a set of routes for the vehicles such that all customers are delivered by one stop of one vehicle according to their demands and the capacities of the vehicles are not exceeded. The aim is to find a feasible solution with minimal total travel costs. Several solution approaches for this optimization problem have been developed. For a comprehensive overview compare, e.g., [143].

In real-world transportation planning, there are often further requirements for the

solutions like time windows for delivery, daily traffic consideration or unknown customer demands. For this reason, several variants of the vehicle routing problem have been defined over time. Based on the respective application, additional constraints were added or the objective function was adapted, for example:

- In the vehicle routing problem with time windows, each customer must be delivered during a customer-specific time window, see, e.g., [47, 132]. Then, not only the costs to travel from customer to customer must be known, but also the time for traveling and the time for delivery. With it, for each job the service start time can be determined dependent on the defined routes.

- For urban transportation, it can be useful to consider daily traffic congestion to ensure compliance with the customers time windows. For this purpose, the vehicle routing problem with time-dependent travel costs and times was developed, as shown, e.g., in [88, 114].

- In some applications, a customer can be delivered in more than one stop which leads to a split delivery vehicle routing problem as investigated, e.g., in [4].

- To compute robust schedules for real-world application, the stochastic nature of customer demands or travel times can be taken into account, see, e.g., [65, 99] or [26, 138], respectively.

- Transportation problems, where goods are transported between pickup and delivery points, are represented by the class of vehicle routing problems with pickup and delivery. An overview of related problems is given in [121]. The relation of pick-up and delivery is either paired, which means that a good is transported from one customer to another as investigated, e.g., in [45], or unpaired, which means that a good is picked up by a customer and can be delivered to any customer with a delivery demand, compare, e.g., [43].

- Recently, energy minimization vehicle routing problems as introduced, e.g., in [56], are becoming more important. There, the objective is to minimize the total energy consumption of the fleet where the energy consumption to travel from customer i to customer j, with $i, j \in \{0, 1, 2, ..., n\}$, depends on the weight of the truck during the trip.

As it can be seen from the various examples, many papers have been written on the subject of vehicle routing in recent decades. Moreover, several (taxonomic) reviews have been published. An overview about existing literature is given, for example, in [46, 94, 137].

Beside transportation, another interesting application of the vehicle routing problem is the scheduling of geographically distributed maintenance works such as those for railway infrastructure [151]. Railway infrastructures consists of thousands of kilometers of railway track, which in turn consists of rails, sleepers, fasteners and ballast. Each component has to be in a suitable condition to ensure a safe and

reliable track service. Regular maintenance is therefore required, usually planned several months in advance to coordinate maintenance with railway operations. For this purpose, it must be defined in advance when maintenance is required. However, failures can occur unexpectedly. Reasons for such unforeseen defects can be, for example, severe weather like heavy rain or material defects in a track component resulting, e.g., from the manufacturing process, transportation or installation. These unexpected failures are maintained by corrective maintenance activities. Dependent on the severity of the failure, it can be necessary to reduce the top speed on the track section in order to avoid safety risks or a too fast deterioration [153]. For fatal failures, it can even be necessary to close the track section. The resulting limitations on railway service lead to penalty costs for the maintenance operator. These must be paid until the track is repaired and the restrictions are removed. By scheduling the maintenance tasks, these penalty costs can be reduced by resolving corresponding maintenance tasks earlier. However, this may in return lead to increased costs for travel of maintenance machine and crew.

Topic of this Thesis

This thesis introduces and studies the novel vehicle routing problem with customer costs (VRPCC) which is developed for the short-term planning of corrective maintenance jobs. Each job is characterized by its working duration, time and costs for traveling to other jobs, and a customer cost coefficient. The latter is a penalty for speed restrictions caused by worse track condition. These penalties have to be paid each day until the job is completed. The jobs have to be visited and maintained by a fleet of maintenance machines with its crew. It is assumed that the fleet is homogeneous, which means that all maintenance machines are equal, but each machine may have another start and end depot. A solution of the VRPCC is a set of routes, one for each machine, such that each job belongs to exactly one route. Based on these routes, for each job a start time is computed taking into account that maintenance is only possible during eight-hour working shifts in the night, but that the maintenance machine can be driven to the next maintenance location outside the working shifts. The objective function of the VRPCC is the sum of travel costs and customer costs, where the latter are computed for each job by the product of its start day and its customer cost coefficient.

The aim of this thesis is to develop and investigate solution approaches for this novel vehicle routing problem. In detail, several mixed-integer linear programs are formulated for the VRPCC. It turned out that the resulting problems are significantly harder to solve because of the customer costs. Therefore, the application of the branch-and-bound approach to the VRPCC is comprehensively exploited and studied. For this purpose, several novel lower bounds for the customer cost part of the objective function are developed and analyzed; and special branching strategies are designed that allow the usage of this new lower bounds.

Furthermore, construction heuristics and local search algorithms are designed to obtain feasible and more cost-efficient solutions for the VRPCC.

Outline

In Chapter 2, firstly, a brief introduction to graph theory is given and some problems on graphs, for example the vehicle routing problem, are formulated. Secondly, since optimization problems such as the vehicle routing problem and its variants are often formulated as a linear program with integrality constraints, linear programming is introduced and some common solution techniques for mixed-integer linear programs are presented. Thirdly, the application of these solution techniques to several vehicle routing problems is investigated. And finally, a review of literature regarding railway maintenance planning is presented.

In Chapter 3, a detailed description of the VRPCC is given that leads to a non-linear partition and permutation model. In this model, a feasible solution is defined by a set of permutations that represent the routes of the vehicles. For this purpose, the set of jobs is partitioned into subsets to allocate the jobs to routes. Then, a route is a permutation of such a subset that represents the order to process the jobs. Dependent on the job order, the start times of the jobs are computed taking into account that working is only possible in the night. The objective function is defined by the sum of the travel costs between consecutive processed jobs and the customer costs that depend on the start times of the jobs.

In Chapter 4, several formulations of the VRPCC as mixed-integer linear program are given. The chapter starts with a basic model, which is derived from a common formulation for the vehicle routing problem with time windows as presented in [143]. This model uses three-index binary variables to define the routes of the maintenance vehicles and two time variables, a start day and a start minute on the start day, to define the start times of the jobs. After that, several alternatives to formulate time constraints are provided. Furthermore, several variants to formulate route constraints by two-index binary variables are shown. Finally, the presented mixed-integer linear programs are compared in terms of computational time for solving them with the commercial solver CPLEX.

Chapter 5 provides some heuristics to achieve a feasible solution for the VRPCC: Firstly, some greedy heuristics are developed that build-up a schedule job by job appending in each iteration one job at the end of one route. Due to its low computational effort, these heuristics are applied in a rollout algorithm as introduced in [16]. Additionally, a local search algorithm is presented where iteratively improved solutions are searched. Finally, the heuristics and algorithms are compared in terms of solution quality.

In Chapter 6, the application of the branch-and-bound method to the non-linear formulation of the VRPCC is shown. For this purpose, two branching strategies were designed. Furthermore, suitable lower bounds for the two cost terms of the VRPCC are required: For the new customer cost part of the objective function, new bounds are developed. And for the travel cost part, known lower bounds from the traveling salesman problem are applied. Finally, computational experiments are made to investigate which branching strategy and which two lower bounds, one for each of the two cost parts, lead to the best performance. Furthermore, the branch-

and-bound algorithms are compared with applying CPLEX to a mixed-integer linear program of the VRPCC and also with heuristics.

In Chapter 7, this thesis is concluded by a summary of the most important contributions and an outlook concerning further research.

As an outcome of my research in collaboration with colleagues, four papers have already been published concerning the VRPCC or its variant with a single maintenance machine:

- F. Heinicke, A. Simroth, G. Scheithauer, and A. Fischer. A railway maintenance scheduling problem with customer costs. *EURO Journal on Transportation and Logistics*, 4:113–137, 2015

  [Chapter 4]

- F. Heinicke, A. Simroth, R. Tadei, and M. Baldi. Job order assignment at optimal costs in railway maintenance. In *ICORES 2013 - Proceedings of the 2nd International Conference on Operations Research and Enterprise Systems*, pages 304–309, 2013

  [Chapter 5.1]

- F. Heinicke and A. Simroth. Application of simulated annealing to railway routine maintenance scheduling. In *Proceedings of the 14th International Conference on Civil, Structure and Environmental Engineering Computers*. Civil-Comp Press, 2013

  [Chapter 5.3]

- F. Theurich, A. Fischer, and G. Scheithauer. A branch-and-bound approach for a vehicle routing problem with customer costs. *EURO Journal on Computational Optimization*, 9:100003, 2021

  [Chapter 6]

2. Background and Literature Review

In this section, at first, some basic notations of graph theory graphs and related optimization problems are introduced. After that, (mixed-integer) linear programming is defined and some general solution methods for it are briefly introduced. Hereinafter, the application of integer linear programming and some of its solution techniques to vehicle routing problems is investigated. Finally, an overview about literature regarding railway maintenance planning is provided.

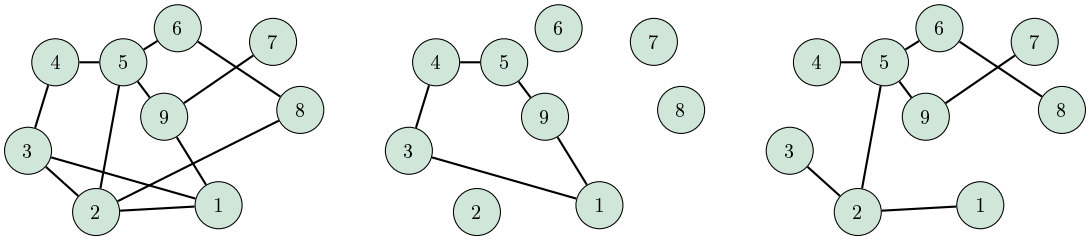
2.1. Graph Theory

The VRPCC, and some related problems, can be formulated as a problem in a graph. Informally, a graph is a set of points which are connected by some edges. As an example, a road map can be represented by a graph where cities are vertices and highways are edges. A mathematical definition of a graph is provided below in Definition 2.1. Hereinafter, some further required definitions are given needed to introduce some problems in graphs, e.g., the vehicle routing problem. A detailed introduction into graph theory can be found, e.g., in [80, 90, 141].

Definition 2.1. An (undirected) *graph* $G = (V, E)$ consists of a finite and non-empty set of vertices $V = \{1, 2, \ldots, n\}$ and a set of edges $E \subseteq \{\{i, j\} | \, i, j \in V\}$ where an edge $e = \{i, j\} \in E$ connects two vertices $i, j \in V$. The vertices $i, j \in V$ are named *adjacent* if an edge $\{i, j\} \in E$ exists; and the edge $\{i, j\} \in E$ is *incident* to vertices i and j. The *degree* of a vertex is the number of edges that are incident to it.

Definition 2.2. A *complete graph* is a graph where each vertex is connected to each other vertex. Consequently, $E = \{\{i, j\} | \, i, j \in V, i \neq j\}$ and each vertex has degree $n - 1$.
A graph $G' = (V', E')$ is a *subgraph* of graph $G = (V, E)$, if $V' \subset V$ and $E' \subset E$.

Definition 2.3. A *weighted graph* $G = (V, E, w)$ is a graph where each edge is associated with a weight defined by a weight function $w : E \to \mathbb{R}$. A weighted complete graph can be represented by a *weight matrix* $W = (w_{ij})_{i,j \in V}$ with $w_{ij} = w(\{i, j\})$.



(a) A graph G. (b) A cycle in G. (c) A tree in G.

Figure 2.1.: Examples for a graph, a cycle and a tree.

The weights of a graph $G = (V, E, w)$ satisfy the *triangle inequality*, if each vertex triple (i, j, k) of vertices $i, j, k \in V$ with $\{\{i, j\}, \{i, k\}, \{k, j\}\} \subseteq E$ satisfies

$$w(\{i, j\}) \leq w(\{i, k\}) + w(\{k, j\}).$$

Real-life applications like tour planning are often formulated as graph problem. Then, vertices are geographical points, e.g., customer addresses, and edges represent travel routes between two addresses. Normally, the weights correspond to driven distances, needed travel times or costs. If the travel routes between two customers are computed minimizing, e.g., travel costs, the travel costs will satisfy the triangle inequality because if the travel costs from customer i to j over k are smallest, then it is $w(\{i, j\}) = w(\{i, k\}) + w(\{k, j\})$.

Definition 2.4. A *path* is a vertex sequence $P = (v_1, v_2, \ldots, v_k)$ such that no vertex is repeated, thus $v_p \neq v_q$ for $1 \leq p \neq q \leq k$, and two consecutive vertices are connected by an edge, thus $\{v_{p-1}, v_p\} \in E$ for $2 \leq p \leq k$. A *cycle* is a vertex sequence $C = (v_1, v_2, \ldots, v_k, v_1)$, such that $(v_1, v_2, \ldots, v_k)$ is a path and $\{v_k, v_1\} \in E$. A graph is *connected*, if for each pair of vertices $i, j \in V$ with $i \neq j$, a path from i to j can be constructed. A graph is *acyclic*, if it does not contain a cycle.

Definition 2.5. A *tree* is a connected, acyclic graph. A *tour* or *Hamiltonian cycle* in a connected graph is a cycle $C = (v_1, v_2, \ldots, v_n, v_1)$ through all vertices of V.

Example To illustrate some of the above definitions, three example are provided in Figure 2.1:

- Subfigure 2.1(a) shows a graph G with vertex set $V = \{1, 2, \ldots, 9\}$ and edge set $E = \{\{1, 2\}, \{1, 3\}, \{1, 9\}, \{2, 3\}, \{2, 5\}, \{2, 8\}, \ldots, \{7, 9\}\}$.
- Subfigure 2.1(b) shows a cycle $C = (3, 1, 9, 5, 4, 3)$ in G.
- Subfigure 2.1(c) shows a tree in G.

Note that G does not contain a tour because vertex 7 is not connected by two edges to G.

It is known that in trees and cycles the number of edges is indicated by the number of vertices, see, e.g., [80, 90, 141].

Proposition 2.1. *Let $G = (V, E)$ be a connected graph with $|V| = n$ vertices. Then, any tree of G containing all vertices of V has $n - 1$ edges and any tour in G has n edges.*

After introducing several fundamental graph theoretical definitions, some graph problems can be presented. Let $G = (V, E, w)$ be a connected, weighted graph.

A *Minimum-Weight Spanning Tree (MST)* of G is a subgraph $T = (V, E', w)$ such that T is a tree, containing all n vertices, of minimum weight

$$W(T) = \sum_{e \in E'} w(e).$$

The *Traveling Salesman Problem (TSP)* is to find a tour $C = (v_1, v_2, \ldots, v_n, v_1)$ through all vertices of minimum weight

$$W(C) = \sum_{i=1}^{n-1} w(\{v_i, v_{i+1}\}) + w(\{v_n, v_1\}).$$

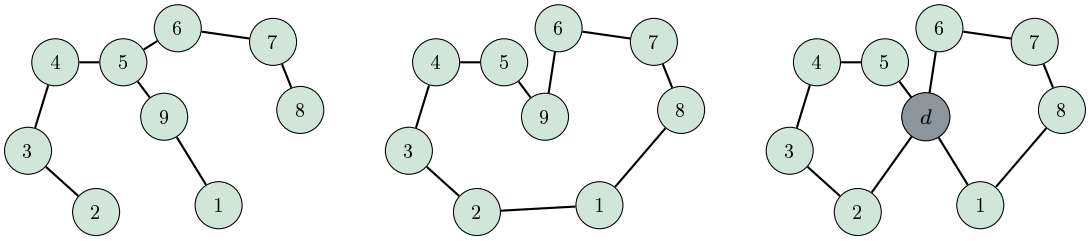
Vehicle Routing Problems (VRP) are combinatorial optimization problems that aim to find a set of routes in a graph. Let $G = (V, E, w)$ be a connected, weighted graph. Further, let one vertex $d \in V$ be the depot and let $M = \{1, 2, \ldots, m\}$ be a fleet of vehicles. A solution of a VRP is a set of cycles $C = \{C_1, C_2, \ldots C_m\}$ in G such that each vertex $i \in V \setminus \{d\}$ belongs to exactly one cycle and that the start and end vertex of each cycle is the depot d. In the context of the VRP, the cycles are called routes. Several variants of the VRP are defined by adding more constraints. Often, the objective is to find routes of minimal total weight

$$W(\{C_1, C_2, \ldots C_m\}) = \sum_{k=1}^{m} W(C_k).$$

A comprehensive overview about VRP variants can be found in, e.g., [38, 143, 144]. The two most common VRP variants are shortly introduced:

In the *Capacitated Vehicle Routing Problem (CVRP)*, each vertex $i \in V$ is afflicted with a demand q_i and each vehicle with a capacity Q. Then, for each vehicle the total demand of the associated vertices must not exceed the capacity, which implies $\sum_{i \in C_k} q_i \leq Q$ for each $k \in M$. The objective is either to minimize the total number of needed vehicles to serve all demands or to minimize the total weight of all routes. A detailed analysis of the CVRP can be found, e.g., in [143].

In the *Vehicle Routing Problem with Time Windows (VRPTW)*, each vertex $i \in V$ is afflicted with a time window $[a_i, b_i]$ and a service time s_i. Further, for each edge $\{i, j\} \in E$, a travel time $r_{ij} \geq 0$ is given. The aim is to find routes such that each



(a) A minimal spanning tree of G.

(b) A solution of the travelling salesman problem in G.

(c) A solution of a vehicle routing problem with two vehicles in G.

Figure 2.2.: Examples for the minimal spanning tree problem, the traveling salesman problem and a vehicle routing problem.

vertex is visited inside the time window. For this purpose, for each vertex $i \in V \setminus \{d\}$ with predecessor p_i in the corresponding route, the visit time t_i has to be computed based on the routes such that $t_i \geq t_{p_i} + s_{p_i} + r_{p_i i}$ and $a_i \leq t_i \leq b_i$. The objective is to minimize the total weight of all routes. A nice introduction into the VRPTW is given in [47]. One interesting variant is the *Vehicle Routing Problem with Soft Time Windows (VRPSTW)*, where visiting customers outside the time windows leads to additional costs which have to be minimized, as investigated, e.g., in [52, 135].

Example To illustrate some of the above definitions, Figure 2.2 shows an MST, an optimal tour and a solution of a VRP. For this purpose, let G be a complete graph with vertex set $V = \{1, 2, \ldots, 9\}$ and weight matrix

$$W = \begin{pmatrix} 0.0 & 18.0 & 29.1 & 31.9 & 25.2 & 26.7 & 25.3 & 18.4 & 15.3 \\ 18.0 & 0.0 & 13.5 & 22.8 & 22.4 & 29.5 & 36.1 & 33.5 & 17.2 \\ 29.1 & 13.5 & 0.0 & 13.6 & 19.1 & 28.4 & 39.4 & 40.4 & 20.6 \\ 31.9 & 22.8 & 13.6 & 0.0 & 10.0 & 18.7 & 32.1 & 36.7 & 17.9 \\ 25.2 & 22.4 & 19.1 & 10.0 & 0.0 & 9.4 & 22.2 & 26.9 & 10.0 \\ 26.7 & 29.5 & 28.4 & 18.7 & 9.4 & 0.0 & 14.1 & 21.6 & 13.2 \\ 25.3 & 36.1 & 39.4 & 32.1 & 22.2 & 14.1 & 0.0 & 10.8 & 19.4 \\ 18.4 & 33.5 & 40.4 & 36.7 & 26.9 & 21.6 & 10.8 & 0.0 & 20.0 \\ 15.3 & 17.2 & 20.6 & 17.9 & 10.0 & 13.2 & 19.4 & 20.0 & 0.0 \end{pmatrix}$$

obtained from the Euclidean distance of the vertices embedded into $\mathbb{R}^2$ as shown in Figure 2.2. Then,

- Subfigure 2.2(a) shows an MST of G with total weight 96.7.
- Subfigure 2.2(b) shows a tour of minimal weight in G which is a solution of the TSP on G. The total weight amounts to 127.8.
- Subfigure 2.2(c) shows a solution of the following VRP: Let $V = \{1, 2, \ldots, 8\}$ be the vertex set and let vertex 9 be the depot for two machines. In addition, each route must contain not more than six jobs. With this constraint, the shown two routes have minimal total weight which is 136.1.

2.2. (Mixed-Integer) Linear Programming

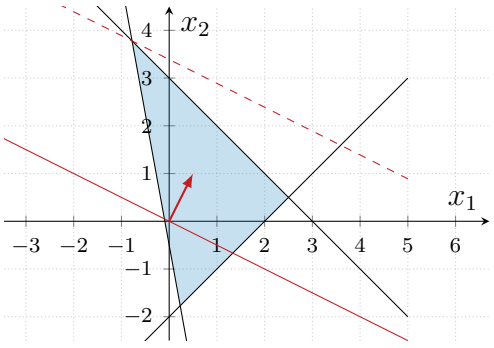
Optimization problems like the TSP or a VRP can also be modeled as mixed-integer linear programs. This subsection introduces the concept of linear programs and mixed-integer linear programs. For a detailed investigation, see for example [90, 116, 148, 149].

Definition 2.6. For a matrix $A \in \mathbb{R}^{m \times n}$, and two vectors $b \in \mathbb{R}^m$ and $c \in \mathbb{R}^n$, a *linear program (LP)* is to find a vector $x^* \in \mathbb{R}^n$ such that $Ax^* \leq b$ and $c^\top x^*$ is minimal (or maximal). Thereby, x is the vector of *decision variables* and $Ax \leq b$ is a set of linear *constraints*. A vector $x \in \mathbb{R}^n$ with $Ax \leq b$ is a *feasible solution* and x^* is called an (optimal) solution of the LP. The *objective function* is given by $c^\top x$ and $c^\top x^* = \min\{c^\top x |\, Ax \leq b,\ x \in \mathbb{R}^n\}$ is the *optimal value* of the linear program.

Minimizing the linear objective function $c^\top x$ is equal to shifting the hyperplane $\{x |\, c^\top x = 0\}$ in direction of the vector $-c$ until the boundary of the polyhedron is attained. Due to this, an optimal solution, if it exists, always belongs to a face of the polyhedron $P = \{x \in \mathbb{R}^n |\, Ax \leq b\}$. This is shown, e.g., in [90, Corollary 3.4]. Consequently, at least one optimal solution is a vertex of the polyhedron. But it is also possible, that an LP does not have an optimal value for two reasons:

- If the constraints contradict each other, the set of feasible solutions is empty. Then, the LP is infeasible and no solution exists.
- If the set of feasible solutions is unbounded in the direction of $-c$, the objective value is also unbounded and no solution can be attained.

The fact, that the optimal value, if it exists, is always attained in a vertex of P, is exploited by the simplex algorithm [90, 148], which is the most common way to solve an LP. The main idea of the simplex algorithm is to go from vertex to vertex until no further improvement is possible.



$$\begin{aligned} & \min && -x_1 - 2x_2 \\ \text{such that} \quad & && x_1 + x_2 \leq 3 \\ & && x_1 - x_2 \leq 2 \\ & && -11x_1 - 2x_2 \leq 1 \\ & && x_1, x_2 \in \mathbb{R} \end{aligned}$$

Figure 2.3.: Illustration of a linear program.

In Figure 2.3, the geometrical interpretation of an LP is illustrated in $\mathbb{R}^2$. Three constraints bound a two-dimensional polyhedron, which is highlighted by light blue.

To find an optimal solution, the line $-x_1 - 2x_2 = 0$, which is drawn red, is moved in direction $\binom{1}{2}$ until the boundary of the polygon is attained in point $(-\frac{7}{9}, \frac{34}{9})^\top$, which is shown by a red dot. The optimal value is $-\frac{61}{9}$.

From each LP, a *dual* problem can be derived. For example, the LP

$$z_{\text{LP}} = \min\{c^\top x |\, Ax \leq b,\ x \geq 0,\ x \in \mathbb{R}^n\},$$

has the dual problem

$$w_{\text{LP}} = \max\{b^\top y |\, A^\top y \leq c,\ y \leq 0,\ y \in \mathbb{R}^m\}.$$

Then, for two vectors x and y feasible to the primal and dual problem, respectively, it is true that

$$c^\top x \geq z_{\text{LP}} \geq w_{\text{LP}} \geq b^\top y.$$

This fact is known as weak duality, as proposed based on a primal maximization problem, e.g., in [116, Proposition 2.2]. Consequently, the dual problem gives lower bounds for the primal problem. Further, if an LP is neither unbounded nor infeasible, its objective value and the objective value of its dual problem are equal which is known as the strong duality theorem, compare, e.g., [116, Theorem 2.4]. For problems with a large number of constraints and a small number of decision variables, it can be easier to solve the dual problem by means of the simplex algorithm which provides at the same time a solution of the primal problem. Further, optimality of a feasible primal solution x can be shown by comparing $c^\top x$ with w_{LP} which is useful for some solution methods.

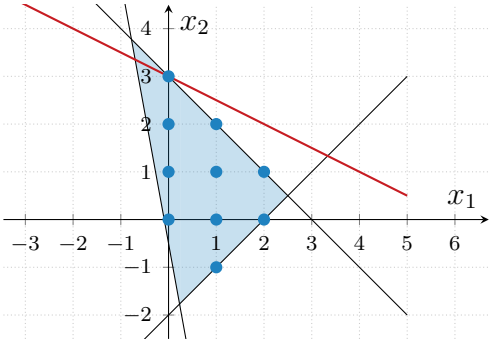
Definition 2.7. An *integer linear program (ILP)* is a linear program with the additional constraint $x \in \mathbb{Z}^n$. The set of feasible solutions for the ILP is

$$I(P) = P \cap \mathbb{Z}^n = \{x \in \mathbb{Z}^n |\, Ax \leq b\}.$$

A *mixed-integer linear program (MILP)* is a linear program where some decision variables x_i, $i \in T \subseteq \{1, 2, \ldots, n\}$, have to be integer.
The *linear programming relaxation (LP relaxation)* is a linear program that results from removing the integrality constraints of a (mixed-) integer linear program.

Note that in case of an LP where the objective function $c^\top x$ is minimized (or maximized), the LP relaxation provides a lower bound (or upper bound) because relaxing the integrality constraints enlarges the set of feasible solutions.

Example In Figure 2.4, the polyhedron P of the LP relaxation is highlighted by the light blue area between the boundary constraints. The feasible solutions of the ILP are the blue dots. The red line shows, where the optimal value of the ILP is attained.



$$\begin{aligned} & \min && -x_1 - 2x_2 \\ \text{such that} \quad & && x_1 + x_2 \leq 3 \\ & && x_1 - x_2 \leq 2 \\ & && -11x_1 - 2x_2 \leq 1 \\ & && x_1, x_2 \in \mathbb{Z} \end{aligned}$$

Figure 2.4.: Illustration of an integer linear program.

Due to the integrality constraints, the optimal value is not necessarily attained in a vertex of the polyhedron $P = \{x \in \mathbb{R}^n | \, Ax \leq b\}$ because often the vertices of P are fractional as in the example provided in Figure 2.4. One possibility to solve an ILP (or MILP) is *the cutting plane method*, see, e.g., [90], where fractional vertices of the polyhedron are cut off: For this purpose, an optimal solution over P is computed. If the obtained solution satisfies all integrality constraints, it solves also the ILP and the algorithm terminates. Otherwise, additional valid inequalities are added to cut off the obtained solution. With it, a new polyhedron $P' \subset P$ with $I(P') = I(P)$ is created. Then, with replacing P by P', the steps solving and cutting are repeated until the obtained solution satisfies all integrality constraints. It was shown that the cutting plane algorithm finds, if exists, an optimal solution for a ILP after a finite number of cuts, compare for example [116, Theorem 3.8]. But, the number of cuts required to attain an optimal solution can be large and pure cutting plane algorithms are rarely used to solve ILPs or MILPs.

Example For the ILP shown in Figure 2.4, solving the LP relaxation leads to the solution $(-\frac{7}{9}, \frac{34}{9})^\top$ which is not an integer solution. With the constraint $x_2 \leq 3$, a valid inequality for the ILP is found that cuts off the solution of the LP relaxation. The solution of the LP relaxation with this additional constraint is $\binom{0}{3}$ which is an integer solution. Consequently, it is also an optimal solution of the original ILP and the algorithm terminates. The objective value is –6.

Another strategy to solve an ILP is *branch-and-bound*, as described in Chapter 6 or, e.g., in [116]. The main idea is to successively break up the set of feasible solutions into certain subsets and to discard subsets that cannot contain an optimal solution. In the following, one possibility of a branch-and-bound method for ILPs, where an objective function is minimized, is introduced. At first, an upper bound for the optimal value of the ILP is computed, e.g., the objective value of an arbitrary feasible solution, or the upper bound is set sufficiently large. Then, the algorithm is initialized with the ILP as first subproblem. In each step, one not yet analyzed subproblem is analyzed as follows:

- Solve the LP relaxation of the subproblem.

- If the obtained value is not smaller than the upper bound, an optimal solution cannot belong to the set of feasible solutions. Consequently, the subproblem is removed and the next subproblem is selected.
- Otherwise and if this solution satisfies all integrality constraints, a better feasible solution of the ILP is found. Update the upper bound by the obtained value and store the solution. Select the next subproblem.
- Otherwise, if the obtained solution does not satisfy all integrality constraints, select one variable that has a fractional value and create two new subproblems. For the first new subproblem, add the constraint that the chosen variable has to be larger than the rounded up value. And for the second new subproblem, add the constraint that the chosen variable has to be less than the rounded down value. Select the next subproblem.

The algorithm terminates, when all subproblems are analyzed. Note that, in contrast to the cutting plane method, the added constraints must not be valid.

Example Solving an ILP with the branch-and-bound method is demonstrated based on the example shown in Figure 2.4. Firstly, an upper bound is computed: For this purpose, an arbitrary feasible solution can be used, for example $\binom{0}{0}$, and the upper bound is set to its objective value which is zero. The first analyzed subproblem, which is the LP relaxation, leads to the non-integer solution $(-\frac{7}{9}, \frac{34}{9})^\top$ with optimal value $-\frac{61}{9}$ which is smaller than the current upper bound. Consequently, branching is necessary, for example on variable x_1. Then, two new subproblems are generated: ILP_2 with the additional constraint $x_1 \geq 0$ and ILP_3 with the additional constraint $x_1 \leq -1$. Let ILP_2 be the second analyzed subproblem. A solution of the LP relaxation of ILP_2 is $\binom{0}{3}$. Its objective value is –6 and with it below the upper bound. Further, this solution satisfy the integrality constraints. Consequently, $\binom{0}{3}$ is the best solution so far and –6 is the new upper bound. New subproblems are not generated. It remains to analyze subproblem ILP_3. Obviously, ILP_3 is infeasible because no feasible solution with $x_1 \leq -1$ exists. Hence, the algorithm terminates because all subproblems are analyzed. The obtained optimal solution is $\binom{0}{3}$ and the optimal value is -6.

Another possibility to obtain lower bounds for a branch-and-bound application is the usage of Lagrangean relaxations as introduced, e.g., in [90]. Assuming an ILP

$$z_{\text{LP}} = \min\{c^\top x |\, Ax \leq b, A'x \leq b', x \in \mathbb{Z}^n\}$$

where the problem without constraints $A'x \leq b'$ can be efficiently solved. Then, the Lagrangean relaxation is given by

$$LR(\lambda) = \min\{c^\top x - \lambda^\top (b' - A'x) |\, Ax \leq b, x \in \mathbb{Z}^n\},$$

which is a lower bound for z_{LP} for each $\lambda \geq 0$. Consequently, $\max\{LR(\lambda) |\, \lambda \geq 0\}$ is the best lower bound which might be tighter than the bound obtained from the LP relaxation.

There are two common extensions of the branch-and-bound approach: branch-and-cut and branch-and-price. The *branch-and-cut method* is a combination of branch-and-bound and cutting plane method, see for example [106, 120]. In this solution approach, initially some constraints are relaxed to obtain a problem that can be solved easier, e.g., the integrality constraints. To compute tight bounds for the branch-and-bound approach, valid constraints are successively added to the subproblem until a feasible solution is obtained or no further violated constraints can be detected. Also commercial solvers for MILPs like CPLEX apply a branch-and-cut algorithm [76]. The subproblems are LPs which are often solved with a simplex algorithm. If some variables of the obtained solution are fractional, cuts are added to find an integer solution. If this approach fails, the branch-and-cut algorithm implemented in CPLEX branches on a fractional variable to generate two new subproblems. More precisely, 14 types of cuts are added, for example mixed-integer rounding cuts obtained by changing the coefficients of integer variables and the constant in a constraint, or generalized upper bound cover cuts, where it is claimed that at most one of a set of binary variables can be one in a solution.

The *branch-and-price method* combines branch-and-bound and column generation, compare, e.g,. [75, 130]. This solution method is applied to problems with a large number of variables from which the most will be zero in an optimal solution. Then, initially only a small part of the variables is taken into account, which results in tighter LP relaxation bounds. By means of its dual problem, it can be proven whether the obtained solution is also optimal for the problem with all variables. Otherwise, further variables have to be considered. A detailed description of applying the branch-and-price method to solve VRPs is given in Section 2.3.3.

2.3. Overview of Solution Methods for Vehicle Routing Problems

In this section, an overview about common solution techniques to solve a VRP is presented. At first, a review about the formulation of VRPs as MILP is given. After that, some heuristics to solve VRPs approximately are listed. Finally, exact solution methods are investigated. A deeper insight in the VRP and its variants can be found, e.g., in [62, 95, 143, 144].

2.3.1. Formulations as Mixed-Integer Linear Program

The core of VRPs is the definition of routes such that each customer is visited exactly once and that each vehicle starts and ends in a depot. For this purpose, there are mainly three approaches:

- In two-index vehicle flow formulations, binary variables x_{ij} define whether customer $i \in V$ is the predecessor of customer $j \in V$ in any route. The depot

is visited $|M|$-times, compare, e.g., [143, p. 12 (VRP1)], or $|M|-1$ copies of the depot are generated, as shown, e.g., in [40, 95].

- In three-index vehicle flow formulations, binary variables x_{ij}^k define whether vehicle $k \in M$ travels from customer $i \in V$ to customer $j \in V$, compare, e.g., [143, p. 15 (VRP4)]. The advantage of this formulation is that vehicle-dependent constraints or costs can be integrated easily. However, the number of variables increases to $|M||V|^2$.

- In set partitioning formulations, binary variables θ_r define whether route $r \in \Omega$ is selected or not where Ω is the set of feasible routes, see, e.g., [143, p. 21 (VRP8)]. To ensure that each customer is visited once, binary variables a_{ir} denotes whether customer $i \in V$ belongs to route $r \in \Omega$. Note, that it is not practicable to generate all feasible routes because the size of Ω increases exponentially with $|V|$. Instead, this formulation is mainly used to solve a VRP with the branch-and-price method as shown later in Section 2.3.3. Thereby, initially a small set of feasible routes Ω' is generated and promising routes are added iteratively until an optimal solution is found.

In several variants of VRPs, not only the order to visit the customers has to be known, but also the times when servicing a customer starts. Then, additional constraints are necessary to determine correct start times. For this purpose, let r_{ij} be the travel time from customer i to customer j with $i, j \in V \cup \{d\}$ (including the service time s_i on customer $i \in V$). To ensure correctly defined start times, the most common formulation is based on a big-M term, see for example [40, 83, 108, 127]. In detail, the time constraints with a big-M term can be formulated as

$$t_j + \mathcal{M}(1 - x_{ij}) \geq t_i + r_{ij}$$

where t_i is the visit time on customer $i \in V$ and $\mathcal{M}$ is sufficiently large, compare [143, p. 158 (VRPTW)]. Then, only if x_{ij} is equal to one, the start time of customer j must be larger than the sum of the start time of customer i and the time required for visiting customer i and traveling to customer j.

An alternative formulation is presented by [109, 146]: To avoid the big-M term, two-index time variables t_{ij} are introduced which are equal to the start time of customer i if and only if $x_{ij} = 1$. Then, constraints

$$\sum_{i \in V} t_{ji} \geq \sum_{i \in V} (t_{ij} + r_{ij} x_{ij})$$

ensure that customer $j \in V$ is not visited before the vehicle has traveled from the predecessor of job j to job j itself.

2.3.2. Heuristics and Local Search Approaches

In this subsection, the most common heuristics and local search approaches for VRPs are listed. Due to the fact that there exists a wide range of different VRP variants, mainly heuristics regarding to VRPs which are similar to the investigated VRPCC are taken into account. These are namely the VRPTW or other in some way time-dependent VRPs. An overview about existing heuristic solution procedures can be found, e.g., in [21, 47, 96, 136].

The first heuristics developed for VRPs are the savings heuristic [29], the sweep algorithms [60] and the generalized assignment heuristic [54], which are often used as inspiration to develop heuristics for the VRPTW.

One of the first papers dealing with heuristics for the VRPTW is [132]. There, the objective is to minimize a weighted sum of travel costs and schedule time. Common heuristics to solve the CVRP are adapted in order to consider time windows and to minimize waiting times. In detail, the presented heuristics are a savings heuristic with restricted waiting time, a time-oriented nearest neighbor heuristic, an insertion heuristic and a time-oriented sweep heuristic. The comprehensive analysis of [132] showed that the insertion heuristic led to the best heuristic solutions. The authors also proved that these heuristics have a worst-case ratio not better than $\Omega(n)$ for an instance with n customers, see [131]. Similar heuristics are presented, e.g., in [7, 9, 21, 110].

In more recent papers, heuristics are used to initialize some kind of local search like improvement heuristics, simulated annealing or tabu search, compare, e.g., [22, 52, 59, 135, 138]. An overview about common neighborhood structures for VRPs can be found in [7, 21]. Also genetic or memetic algorithms are applied to VRPs, see, e.g., [127, 136, 139].

2.3.3. Exact Solution Approaches

The most common exact solution methods are branch-and-bound or the further developed variants branch-and-cut and branch-and-price. An overview can be found, e.g., in [13].

Application of Branch-and-Bound and Branch-and-Cut Methods

Several branch-and-bound and branch-and-cut algorithms were developed for the VRP and its variants. They differ mainly in the branching strategy and the applied lower bounds to prune on nodes that did not contain an optimal solution. For an overview of their application to the CVRP, see, e.g., [143].

The most common approach to create new branches is to include or exclude edges of the underlying graph:

- Often, branching is done on a binary variable that describes, whether the jobs i and j are visited consecutively which implies that edge $\{i, j\} \in E$ is used by a route, see, e.g., [44, 97]. In doing so, one variable, which is fractional in

the current node is selected and two new branches are generated: One branch where the variable is set to zero, which means the edge is excluded, and one branch where the variable is one, which means the edge belongs in each case to one route.

- A similar approach is to add edges to a list of included or excluded edges as shown, e.g., in [2, 98, 142]. More precisely, if bounds for subproblems are not computed by the LP relaxation but, e.g., by relaxing subtour elimination constraints, the solutions of the subproblems are integer solutions but can contain infeasible routes which can be avoided by excluding an edge in further analyzed branches.

Another approach for the CVRP is to branch on sets of flow-variables for the capacity constraints, as shown in [8, 12].

For certain problem formulations, special branching strategies were defined. For example, in [107] a model for the time-dependent TSP is described where binary variables are defined to allocate jobs to a position in the route. In each branching step, a city is selected which is not allocated to a position. Then, for each open position of the route, a branch is created where the city is placed on this position.

In [89], branching leads to a successive construction of the route. Each node in the search tree has a set of fixed routes, a partial route and a set of customers forbidden to be the next one in the partial route. To create two new branches, a customer i is selected that does not belong to any route or to the set of forbidden customer. For the first created branch, customer i is appended to the partial route and the set of forbidden customers is emptied. And for the second created branch, customer i is forbidden to be the next visited customer and added to the corresponding set. If no customer i is found because each not yet visited customer is forbidden, the partial route is fixed, the set of forbidden customers is emptied and a new partial route is started.

For the most problems, the objective is to minimize the travel costs. The following methods are applied to obtain lower bounds for this objective:

- A popular lower bound is derived by relaxing subtour elimination constraints which leads, if for each vehicle an own depot is defined, to an assignment problem, see, e.g., [2].

- Further common bounds result from relaxing the degree constraints, which ensure that each job is visited exactly once. Then, a lower bound can be obtained from a minimum spanning tree or similar problems which can be efficiently computed. To obtain tighter lower bounds, the Lagrangean technique is applied, as shown, e.g., in [28, 58, 74], where the lower bound is iteratively improved by updating Lagrange multipliers for the relaxed degree constraints.

- For the VRPTW, a Lagrangean relaxation can be applied where the constraint that each customer has to be visited exactly once is omitted. Then, lower

bounds are computed solving a shortest path problem with time windows and capacity constraint, which permits to visit customers more than once. For this purpose, dynamic programming is applied, see, e.g., [82, 87].

- Also the LP relaxation of the MILP formulation can be used as lower bound, as shown, e.g., in [44]. To tighten the LP relaxation, several valid inequalities can be added. An application of such a branch-and-cut method can be found for example in [4, 12, 36].

For specific problems, specific lower bounds were developed. For example in [107] a lower bound for a TSP with time-dependent costs is provided where the constraint to visit each city is relaxed. In [142], a branch-and-bound method for a VRP with backhauls is presented where a special Lagrangian lower bound is applied.

Application of Column Generation with the Branch-and-Price Method

A widely used solution technique for VRPs is the branch-and-price method, which is a combination of column generation and the branch-and-bound method. In this paragraph, it is shown how branch-and-price can be applied to VRPs. After that, various branch-and-price methods for VRPs are presented. Finally, it is discussed why the branch-and-price method is not used for the VRPCC in this thesis.

As mentioned above, VRPs can be formulated by a set partition model. Then, a solution of the CVRP is a selection of m routes, such that each route does not exceed the capacity constraint and each vertex belongs to at least one selected route, see, e.g., [143, p. 21 (VRP8)]. For each route, one binary variable is used to define whether the route is selected or not. There will be a lot of feasible routes, but most of them are not selected and the corresponding variables are zero. Consequently, an optimal solution remains optimal even if not selected routes are removed from the set of feasible routes. For this reason, not all feasible routes must be generated at the beginning, but only a small selection. For this set of routes, the problem "Select at most m routes with minimal costs, such that each vertex belongs to at least one route" is solved. In detail, the dual problem of its LP relaxation is solved because with help of the dual variables, new routes can be generated that may reduce the costs of the solution. Then, the problem is solved again with an enlarged set of routes. These two steps are repeated until no more routes can be found that can improve the current solution. This approach is called column generation method, because new columns of the matrix of the LP are generated by adding new decision variables, which are in case of the CVRP new routes. Note that the column generation method solves only an LP. To obtain a solution of the CVRP, it has been integrated into a branch-and-bound method, which leads to a branch-and-price algorithm.

As detailed explored in [49], applying column generation and branch-and-price to VRPs works as follows: Let V be the vertex set of the customers, which has to be served by m machines, and let $d \in V$ be the depot. Further, let $\Omega = \{r_1, r_2, \ldots\}$ be the set of all feasible routes of the VRP with route r_k having costs c_k. Further,

a_{ik} denotes how often vertex i is visited by route r_k. With the decision variable θ_k, that defines how often route r_k is selected, the VRP can be formulated as

$$
\begin{aligned}
(\mathrm{MP}(\Omega)) \qquad \min \quad & \sum_{r_k \in \Omega} c_k \theta_k & & \\
\text{s.t.} \quad & \sum_{r_k \in \Omega} a_{ik} \theta_k \geq 1 & & i \in V \setminus \{d\}, \\
& \sum_{r_k \in \Omega} \theta_k \leq m, & & \\
& \theta_k \in \mathbb{N} & & r_k \in \Omega.
\end{aligned}
$$

The formulation (MP(Ω)) has a better LP relaxation than the common MILP to model VRPs. But the set of feasible routes Ω is very large. Thus, by means of branch-and-price it is aimed to find an optimal solution for the master problem (MP(Ω)) with help of a subset of feasible routes $\Omega' \subset \Omega$.

In the branch-and-price algorithm, in each search tree node an LP relaxation of (MP(Ω')) is solved which will be denoted as (LMP(Ω')). Clearly, a solution of (LMP(Ω')) does not have to be optimal for (LMP(Ω)) since not all feasible routes are considered. But applying the theorem of strong duality, see [116, Theorem 2.4], optimality can be shown by considering the dual problem of (LMP(Ω)) which is

$$
\begin{aligned}
(\mathrm{DMP}(\Omega)) \quad \max \quad & \sum_{i \in V \setminus \{d\}} \lambda_i + m\lambda_0 & & & (2.1) \\
\text{s.t.} \quad & \sum_{i \in V \setminus \{d\}} a_{ik}\lambda_i + \lambda_0 \leq c_k & & r_k \in \Omega, & (2.2) \\
& \lambda_0 \leq 0, \ \lambda_i \geq 0 & & i \in V \setminus \{d\}. & (2.3)
\end{aligned}
$$

Let λ^* be an optimal solution of (DMP(Ω')). Since in the dual problem the objective function is independent from the set of routes Ω, λ^* is also optimal for (DMP(Ω)), if it can be shown that constraint (2.2) is valid for each feasible route $r_k \in \Omega$. For this purpose, the column generation problem

$$\min\{c_k - a_{ik}\lambda_i - \lambda_0 | \, r_k \in \Omega\} \tag{2.4}$$

has to be solved. A route with $c_k - a_{ik}\lambda_i - \lambda_0 < 0$ leads to a violation of constraint (2.2) and has to be added to Ω'. Note, such a route is called a route with negative reduced costs.

To find routes with negative reduced costs, the column generation problem (2.4) is reformulated. Let the binary variable x_{ij} be one, if and only if customer i is the predecessor of customer j in the route. Consequently, $c_k = \sum_{i,j \in V} c_{ij} x_{ij}$ and $a_{ik} = \sum_{j \in V} x_{ij}$. With it, the column generation problem (2.4) of any dual solution λ can also be formulated as

$$
\begin{aligned}
(\mathrm{CGP}(\lambda)) \qquad \min \quad & \sum_{i,j \in V} x_{ij}(c_{ij} - \lambda_i), & \\
\text{s.t.} \quad & \sum_{j \in V} x_{ij} - \sum_{j \in V} x_{ji} = 0 & i \in V \setminus \{d\}, \\
& \sum_{j \in V} x_{dj} = 1, & \\
& \sum_{i \in V} x_{id} = 1, & \\
& \sum_{j \in V} x_{ij} \leq 1 & i \in V \setminus \{d\}, \\
& x_{ij} \in \{0,1\} & i,j \in V, \\
& \text{and further restrictions from the underlying VRP.} &
\end{aligned}
$$

Then, (CGP(λ)) is an elementary shortest path problem with additional constraints. This problem can be solved by a label correcting algorithm as presented in [50].

If routes with negative reduced costs are found, they are added to the set of routes Ω' and (DMP(Ω')) is solved again. Otherwise, the obtained solution is also optimal for (DMP(Ω)) and an optimal solution θ^* of (LMP(Ω)) can be deduced. Then, if not all variables θ_k^* are integers, branching is necessary to obtain an integer solution. As mentioned in [49], it is not recommendable to branch on the binary variables θ_k. Indeed, branching is performed at an edge (i,j) which is fractional included by the routes selected by θ^*. Two subproblems are generated, one where the edge cannot belong to the solution and one where the edge is enforced in the solution. The latter is equal to excluding all edges (i,k) with $k \neq j$ and (k,j) with $k \neq i$, which implies that the only remaining possibility to visit customer j is via visiting customer i directly before. Furthermore, subproblems can be generated by branching on the fleet size constraint, if the sum of route selection variables θ_k is fractional. Then, for one branch the number of vehicles is restricted by the rounded down current sum value and for a second branch it is claimed that the number of selected routes is not less than the rounded up current sum value.

Applications of the branch-and-price method to VRPTWs and CVRPs can be found, e.g., in [127, 129, 134] and [11, 56, 122], respectively. For the VRPSTW, where service costs have to be paid if time windows are not hold, applications of the branch-and-price method are provided in [134, 138]. Then, the elementary shortest path problem to obtain routes with negative reduced costs is time-dependent. To solve a multi-depot VRP with capacity and route length constraints, a branch-and-price method is developed in [32]. Since multiple depots have to be considered, for each of them a subproblem has to be solved to find routes with negative reduced costs. VRPTWs with time-dependent travel times are considered, e.g., in [34, 100].

In this thesis, the branch-and-price method is not applied to the VRPCC for two reasons: Firstly, the branch-and-price method is mainly applied to VRPTW, where only a small part of all possible routes is feasible with respect to the time windows. In the last years, some successful applications of branch-and-price to the CVRP were

developed, see, e.g., [56, 122]. In contrast, in the VRPCC routes must not meet time windows or capacity constraints, which would reduce the number of feasible routes. Furthermore, because each vehicle has its own start and end depot, more routes are possible. Secondly, the objective function of the VRPCC is a sum of travel costs and time-dependent customer costs. Consequently, also the cost function in the column generation problem will be time-dependent. It is expected that this leads to a column generation problem which is harder to solve.

2.4. Railway Maintenance Planning

Maintenance of railway infrastructure is essential to ensure a safe and reliable railway service. In the last decade, I was involved in several research projects dealing with the improvement of railway infrastructure maintenance: ACEM-Rail [78], INFRALERT [79] and In2Smart [31]. In these projects, new measurements systems, inspection techniques and data analysis tools were developed to obtain detailed information about the current track condition on the one hand and to predict the future development of the track condition on the other hand. Furthermore, decision-support tools for maintenance planning have been developed that benefit from this new information.

Railway maintenance management is complex. Because of that, it is separated into several planning steps with different planning horizon and level of detail. The resulting planning problems can be distinguished into three levels, compare for example [79, 102]:

- strategic (long-term) planning, which enfolds, e.g., determining the budget for maintenance activities, identification of target quality standards and key performance indicators or the definition of maintenance and renewal policies;
- tactical (mid-term) planning, which enfolds, e.g., scheduling of large and complex maintenance tasks and renewal projects;
- operational (short-term) planning, which enfolds, e.g., daily planning and scheduling of unexpected occurring failures.

In the following, the planning of maintenance activities or tasks will be addressed. Most papers deal with maintenance activity planning on a tactical level, where processing of a maintenance task usually requires several days or weeks. The resulting scheduling problems are focused on assigning (larger) maintenance activities to time intervals like days or weeks and to maintenance crews and/or machines. If different kinds of maintenance tasks are planned together, it has to be considered that maintenance crews have different skills and can only process appropriate maintenance actions. The maintenance tasks correspond to a certain track section or segment. Accordingly, crew and machines have to be moved from tasks to tasks which takes time and causes costs. For the movement of crews and machines, the term traveling is used. There are some papers, where traveling is neglected because travel times are

small compared to the time intervals for planning, e.g., [24, 66]. In other papers, constraints are applied that consider travel time between two consecutive maintenance actions to ensure a realizable maintenance plan, see, e.g., [17, 113, 119, 145, 147]. For scheduling problems with a larger time horizon, often the future development of the infrastructure deterioration is integrated. Thereby, deterioration models are either deterministic, see, e.g., [66, 113, 119, 145], or stochastic, for example [152] or, for a road infrastructure, [84].

Maintenance Scheduling with Time-Space Network

Some research is done on maintenance planning problems, where (among others) travel costs have to be minimized. One common approach is based on time-space networks, see, e.g., [20, 63, 101, 124]. In a time-space network, vertices are defined by the location of a maintenance task and a certain time interval. Directed edges, or arcs, between vertices describe a possible activity like working or traveling considering the time required for maintenance. For example, assuming a set of maintenance tasks that have to be scheduled to weeks. Then, for each maintenance activity and each possible week to start the maintenance activity, a vertex is defined. The outgoing arcs link the maintenance activity to possible successors in the schedule considering the working duration. For example, if a maintenance task has a duration of two weeks, the outgoing arcs link the maintenance task to other tasks where the start time is two weeks later. Further, a source and a sink are defined that represent the start and end of the schedule. In such a time-space network, a schedule is described by one flow per maintenance crew where the vertices of a flow provide the maintenance taks and the start week.

As an example consider [124], where a time-space network is constructed on a weekly base. Two types of arcs are defined: travel arcs for traveling from one location to the next within one week and working arcs for performing a project on a location over several weeks. In this time-space network, flows are defined such that for each project one working arc is selected. The objective is to minimize a weighted sum of travel costs and penalties for violated soft constraints like compliance with time windows of projects, mutually exclusive constraints and precedence constraints. A similar approach can be found in [63], where also idle times between projects are taken into account. Additionally, an alternative non-linear formulation as job scheduling problem is given, which consists of allocating maintenance activities to teams and defining start times for the maintenance activities. Two solution variants for this formulation are suggested: applying a commercial solver for constraint programming and using a genetic algorithm. Concluding, it is stated that the job scheduling formulation is closer to the real problem, but was more difficult to solve than the time-space model which was observed in preliminary computational tests.

If the time required for maintenance varies considerably from job to job, it can be difficult to find a suitable discretization of the time horizon for the time-space network. One possibility to overcome this obstacle is to merge jobs to larger projects.

For example, in [123], beside a time-space network model for larger projects, a VRP is presented in order to merge small jobs to larger projects that can be allocated to one or more weeks. The aim is to find a set of projects, each represented by one route, with minimal total number of required weeks which is equal to minimize the idle times of the projects if they are planned on a weekly base. Thereby, several additional constraints like required crew skills, mutually exclusion of jobs or project duration constraints are considered. The planning horizon amounts to one year and more than 2300 jobs have to be merged into projects. Due to the resulting complexity, a solution for this problem is computed by local search.

Maintenance Scheduling as Vehicle Routing

The review of literature has shown that maintenance scheduling problems can also be formulated as VRPs, compare for example [125, 151]. In doing so, the maintenance tasks are not assigned to time intervals and crew/machine, but for each crew/machine, a route is constructed with the maintenance tasks to be processed. It should be mentioned that this approach is rarely used.

In [151], maintenance jobs are represented by vertices in a graph. Thereby, each job equals to maintaining a track segment of a non-negligible length. Consequently, the direction in which a job is executed has to be considered. For this purpose, between two vertices four edges are constructed: one for each direction combination. Several side constraints are considered in the resulting VRP: periodicity of jobs, preferred or forbidden time windows for execution, mutually exclusion of jobs which cannot be performed simultaneously or precedence constraints. The objective is to minimize a sum of travel costs, penalties for not performed jobs and penalties for violated constraints. Due to the periodicity of the jobs, a rolling horizon is used to decompose the problem into separate subproblems. However, the resulting subproblems are still too large to solve them exactly. Because of that, the subproblems are solved using a heuristic: Important jobs are scheduled by applying a commercial solver to the underlying MILP. After that, the remaining jobs are inserted heuristically. Finally, a local search procedure is applied to improve the obtained solution.

A scheduling problem for maintenance of railway signals is investigated in [125], where some tasks cannot be handled by a single crew member. The problem is formulated as VRPTW with multiple depots and synchronization constraints where the latter ensure that tasks requiring two crew members are visited by two crew members simultaneously. The aim of the resulting VRPTW is to define one route per crew member and day such that the total travel costs are minimal. Since the planning horizon is large and up to 1000 maintenance tasks have to be planned, the problem is solved heuristically. Thereby, the two major steps are firstly, the clustering of maintenance tasks into a set of tasks that can be performed by a crew member and secondly, the scheduling of each cluster taking into account dependencies between the schedules of the different crew members resulting from the simultaneous processing of some tasks.

Scheduling of larger track maintenance activities can also be formulated as an arc

routing problem, since the direction of track maintenance can have a major impact on travel costs. In [133], a track maintenance scheduling problem is introduced as capacitated arc routing problem which is transformed into a CVRP. In both problems, the objective is to find a set of routes where travel costs and the cost of hired maintenance staff are minimal, while ensuring that each route does not exceed a certain time limit and that each maintenance activity belongs to a route.

Maintenance Scheduling with Time-Dependent Costs

The VRPCC was developed within the European project ACEM-Rail, which is presented in [78]. In this project, developments were achieved in five areas: new inspection and measurement techniques, models to estimate track defect evolution, models and algorithms for maintenance planning, management system, and monitoring of executed maintenance. With respect to maintenance planning, two problems were investigated: one on tactical level and one on operational level. In the tactical maintenance problem, predicted maintenance activities are allocated to time periods considering a stochastic demand of resources caused by the uncertain development of the track condition, see [14, 72]. And the operational planning problem was developed for scheduling of small tamping tasks by a single tamping team taking into account that sections that do not meet the track condition requirements cause penalty costs until they are repaired, see [70, 73]. The reasons for these penalty costs are that in case of insufficient track condition, the infrastructure manager can declare speed limitations for the track, compare, e.g., [3, 117, 153], or the infrastructure manager may demand a compensation from the contracted maintenance companies if this has been agreed in the maintenance contract [64, 118]. This operational planning problem is the origin of the VRPCC with the difference that in the VRPCC several tamping machines are available. Consequently, in the VRPCC, some maintenance tasks are afflicted with a cost value that have to be paid every day until maintenance is executed.

In the literature, some papers considering costs for insufficient track condition can be found. In [152], costs occur for each day on which the track condition is predicted to be unsafe. These costs are minimized together with costs for losing lifetime due to premature maintenance as well as costs for traveling and maintenance. The corresponding maintenance problem is to assign jobs to days and teams considering travel times.

In [69], a derailment risk is predicted, which increases if maintenance is executed later. It is also considered that delayed maintenance can lead to defects and with it to increased maintenance costs. The analyzed maintenance problem consist of clustering track sections, that require maintenance, and assigning these clusters to days in a time horizon of up to one month. Thereby, travel costs are approximated by the costs to travel to the center of the cluster. The objective is either minimizing the costs for derailment and maintenance or minimizing the maximal derailment risk. The problem is solved by a commercial MILP solver.

Also in other maintenance areas, problems occur where delayed maintenance

causes additional costs: For example, maintenance scheduling problems of offshore wind farms often take into account that delayed maintenance leads to a reduction of the turbine productivity which causes losses in energy production, see, e.g., [35, 51, 91]. The resulting problem is a VRP with pickup and delivery, where a fleet of vessels is routed that bring technicians to platforms and pick them up when maintenance is completed.

A more general problem with time-dependent costs is the traveling maintainer problem introduced in [25]. This problem is close to the TSP, but the objective function is a sum of latency costs. Thereby, the latency of city i equals to the sum of travel times to reach city i. After introducing the general traveling maintainer problem, some examples for latency functions of cities are given, e.g., a sum of time-dependent expected failure costs, maintenance costs and travel costs. Since the latency functions can be non-linear, it is proposed to apply a genetic algorithm or particle swarm optimization to solve the traveling maintainer problem.

Maintenance Scheduling and Railway Service

In the so far presented papers, it is assumed that there is a certain amount of track possession time per time period where the track can be closed for maintenance. But, recent papers have explored how to coordinate maintenance on railway infrastructure and train traffic, either by defining possession times for maintenance [81, 103, 104] or by considering track disturbances directly [18, 19].

In [103, 104], a train schedule is defined such that a demand of maintenance windows is satisfied. For each train, a set of feasible routes and a range for a feasible service time is given. Further, for links in the railway network, the demand of maintenance windows is given by time window options (for example, a single window of three hours or two windows of two hours), from which one has to be chosen and scheduled. The resulting optimization problem consists of train scheduling and definition of maintenance windows. Thereby, train scheduling means that for each train a route and a start time are chosen such that the resulting train schedule is feasible which means, e.g., that travel and dwell times are considered. And the definition of maintenance windows consists of selecting a maintenance option and an according set of time periods for each link, such that the demand on maintenance is satisfied. Furthermore, in [104] a crew assignment is integrated to ensure that the time windows for maintenance can be efficiently used.

Annual track possession scheduling is addressed in [81] for a mining supply chain rail network. The aim is to define a set of track possession times such that capacity reduction is minimized, the given demand on track possession is satisfied and at no time more resources are required than are available. The problem is formulated as MILP. Due to the large planning horizon, a iteratively metaheuristic is developed to obtain a feasible solution. For this purpose, the MILP is split into subproblems by fixing some of the variables and optimizing over the rest.

In [19], a maintenance rescheduling problem is analyzed which aims to reschedule maintenance activities of an initial schedule in order to maximize the total through-

put in the planning horizon of one year. The problem is modeled as MILP. To solve it by a commercial solver for a planning horizon of a year, the number of variables is reduced by limiting the potential start times of rescheduled jobs and by dividing the planning horizon to obtain smaller subproblems. Computational experiments on real world data sets have shown that with this approach a significant improvement can be achieved.

In [18], start times of maintenance jobs are planned taking into account that during maintenance (freight) trains cannot pass the track. The objective is to find a schedule such that the total throughput during the planning horizon is maximal. The resulting optimization problem is modeled as MILP which have to find a) a segmentation of the time horizon in separated time intervals, b) an allocation of maintenance activities to consecutive time intervals complying release time and deadline and c) a maximal flow through the network considering down times of arcs due to maintenance.

3. The Vehicle Routing Problem with Customer Costs

As explained in detail in the introduction, in this thesis a novel vehicle routing problem is analyzed that results from operational planning of small maintenance tasks to correct unexpected failures. Thereby, two kinds of costs need to be considered: Firstly, travel costs for machinery and crew; and secondly, penalty costs for an unsafe track condition that have to be paid for each day from failure detection to maintenance completion. For the latter, customer cost coefficients are defined for each maintenance activity. The objective function of this problem is defined by the sum of travel costs and time-dependent customer costs. With it, the priority of customers can be taken into account without losing the sight on travel costs. This new vehicle routing problem is called *vehicle routing problem with customer costs* or shortly **VRPCC**.

This chapter gives a detailed description of the VRPCC. In detail, Section 3.1 introduced the VRPCC and defined the used notation. Because of the technical background, the customers are named as jobs. Then, Section 3.2 provides a non-linear partition and permutation model, called (PP), which allows to define each solution by a partition of the jobs into subsets for each vehicle; and a permutation of each partition to define the order to visit the jobs. Finally, in Section 3.3 some extensions of the VRPCC are mentioned.

3.1. Problem Data and Variables

Let $N = \{1, 2, \ldots, n\}$ be a set of jobs that have to be scheduled to a set of vehicles $M = \{1, 2, \ldots, m\}$. Each vehicle $k \in M$ has a start point s_k and an end point z_k. According to the common terminology, the start and end points will be called depots. The depot sets are given by $N_s := \{s_k : k \in M\}$ and $N_z := \{z_k : k \in M\}$. Furthermore, let $N_a := N \cup N_s \cup N_z$ be the set of all jobs and depots with cardinality $n_a := |N_a| = n + 2m$. Each job $i \in N_a$ is characterized by

- its customer cost coefficient c_i,
- its working duration a_i,
- the travel costs d_{ij} to job $j \in N_a$, $j \neq i$, and
- the travel time r_{ij} to job $j \in N_a$, $j \neq i$.

All values are assumed to be non-negative integers. Furthermore, it is assumed that

- $a_i > 0$ for all jobs $i \in N$,
- $a_{s_k} = a_{z_k} = 0$ and $c_{s_k} = c_{z_k} = 0$ for all $k \in M$, and
- both the travel costs and the travel times satisfy the triangle inequality as described in Definition 2.3.

Definition 3.1. A *route* is an order to visit a set of jobs N_k with a vehicle k. A route is represented by a permutation $\Pi^k(N_k) = (\pi_1^k, \pi_2^k, \ldots, \pi_{|N_k|}^k)$ with $\pi_i^k \in N_k$. Each route starts in the start depot s_k of vehicle k, passes the jobs of N_k in the defined order and ends in the end depot z_k:

$$\pi_0^k := s_k \mapsto \pi_1^k \mapsto \pi_2^k \mapsto \ldots \mapsto \pi_{|N_k|}^k \mapsto \pi_{|N_k|+1}^k := z_k.$$

Definition 3.2. A *schedule* $S := (N_k,\ \Pi^k(N_k))_{k \in M}$ is an order to complete a set of jobs N with a set of vehicles M. It consists of a partition $(N_k)_{k \in M}$ of N into m non-empty subsets and, for each vehicle $k \in M$, a permutation $\Pi^k(N_k)$ defining its route.

In the following the two parts of the objective function are investigated in detail. As mentioned previously, the objective function consists of travel costs and time-dependent customer costs. The travel cost value is computed from the routes as

$$\sum_{k \in M} (d_{s_k \pi_1^k} + \sum_{i=1}^{|N_k|-1} d_{\pi_i^k \pi_{i+1}^k} + d_{\pi_{|N_k|}^k z_k}).$$

And the customer costs are calculated as

$$\sum_{i \in N} c_i t_i^d,$$

where t_i^d is the day where job $i \in N$ is started. Thus, for each job the start time has to be determined. For that, a special property of the VRPCC has to be considered: job execution is only possible during working shifts in the nights with length u, but traveling from job to job is possible all day long. To model the working shifts, each day any work can start at minute 0 or later, but has to be finished at latest when the working shift ends at minute u. This implies that a job i cannot start if its location is reached after minute $u_i := u - a_i$. In this case, the job must be postponed to the next day. The working shift length u is naturally bounded by 1440 minutes, i.e., by 24 hours per day. To avoid infeasibility of an instance, the working duration of any job $i \in N$ is bounded by $a_i \leq u$. For a correct scheduling, the time t_i, when the execution of job i starts, is given as provided in Definition 3.3.

Definition 3.3. The *start time* of a job i is defined as $t_i = (t_i^d, t_i^m)$, where $t_i^d \in \mathbb{N}$ represents the start day and $t_i^m \geq 0$ the start minute on day t_i^d.

The start day t_i^d is the basis for the calculation of the customer costs. The start minute is restricted by the working shift, see condition (3.3) below. Since travel time and working duration are given in minutes, a start time $t = (t^d, t^m)$ is converted into minutes by the function $\xi : \mathbb{N} \times \mathbb{R} \to \mathbb{R}$ with

$$\xi(t) := ht^d + t^m, \tag{3.1}$$

where $h := 1440$ are the minutes of a day. A start time pair t_i of any job $i \in N$, with job $p_i \in N \cup N_s$ as predecessor in the schedule, has to fulfill:

$$\xi(t_i) \geq \xi(t_{p_i}) + a_{p_i} + r_{p_i i} \tag{3.2}$$

and

$$0 \leq t_i^m \leq u_i. \tag{3.3}$$

Condition (3.2) ensures that the job i does not start before its predecessor p_i is completed and the vehicle has traveled to job i. By restriction (3.3), the compliance with the working shift is guaranteed.

Theorem 3.1. *If the customer cost function of the objective function is monotonically increasing, the minimally allowed start times lead to minimal customer costs.*

Proof. In case of a monotonically increasing customer cost function, later execution of a job cannot result in a smaller customer cost value for this job. Furthermore, since the travel times are constant, a later execution of a job cannot lead to an earlier execution of its successors and consequently also the customer cost value of the successors cannot become smaller. □

Due to Theorem 3.1 in the following the minimally allowed start times are computed. Therefore, firstly the arrival time t_i^a of any job $i \in N$, which is the time in minutes of reaching the location of job i, is calculated as

$$t_i^a := \xi(t_{p_i}) + a_{p_i} + r_{p_i i},$$

where t_{p_i} is the start time of the predecessor which needs to be already computed. Then, the arrival time has to be converted into a start time (t_i^d, t_i^m) so that constraint (3.3) applies. This is achieved by the function $\zeta : \mathbb{R} \times \mathbb{N} \to \mathbb{N} \times \mathbb{R}$ with

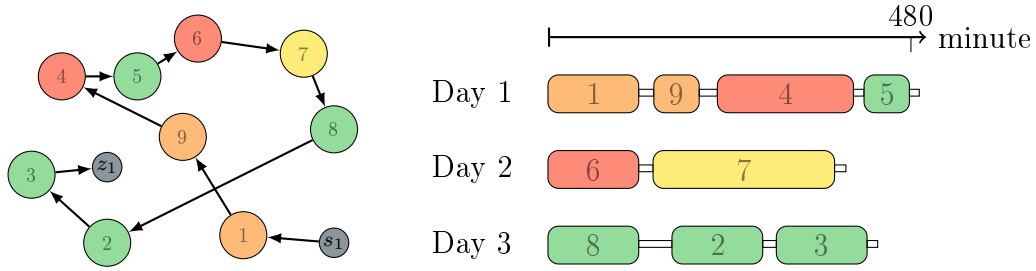
$$\zeta(t, u_i) := \begin{cases} (\lfloor \frac{t}{h} \rfloor,\ t - \lfloor \frac{t}{h} \rfloor h) & \text{if } t - \lfloor \frac{t}{h} \rfloor h \leq u_i, \\ (\lfloor \frac{t}{h} \rfloor + 1,\ 0) & \text{otherwise.} \end{cases}$$

Then, the time pair $(t^d, t^m) = \zeta(t, u_i)$ is the earliest with $\xi((t^d, t^m)) \geq t$ that fulfills condition (3.3). With it, the earliest possible start times of the jobs can be calculated based on the schedule $S = (N_k, \Pi^k(N_k))_{k \in M}$. The start time of the depots is given by t_0:

$$t_{\pi_0^k} := t_{s_k} = t_0, \quad k \in M.$$

The earliest possible start times of the jobs are calculated in the order defined by the permutations based on the arrival time as

$$t_{\pi_i^k} := \zeta(t^a_{\pi_i^k}, u_{\pi_i^k}) = \zeta(\xi(t_{\pi_{i-1}^k}) + a_{\pi_{i-1}^k} + r_{\pi_{i-1}^k \pi_i^k}, u_{\pi_i^k}), \quad i = 1, 2, \ldots, |N_k|,\ k \in M.$$



(a) Solution of the VRPCC.

(b) Schedule of the solution.

Figure 3.1.: Example for a solution of the vehicle routing problem with customer costs.

i		s_1	z_1	1	2	3	4	5	6	7	8	9
c_i		0	0	100	0	0	150	0	150	50	0	100
a_i		0	0	120	120	120	180	60	120	240	120	60
						travel time r_{ij}						
s_1		-	42	16	40	54	55	45	43	33	20	32
z_1		317	-	27	14	14	18	17	28	40	40	15
1		121	202	-	24	38	42	33	35	33	24	20
2	travel costs d_{ij}	301	101	181	-	18	30	30	39	47	44	23
3		411	101	292	135	-	18	25	37	52	53	27
4		422	135	319	229	137	-	14	25	42	48	24
5		341	127	253	224	192	101	-	13	29	36	14
6		325	209	267	296	285	187	95	-	19	29	18
7		254	301	253	361	394	322	223	142	-	15	26
8		151	305	185	336	405	367	270	217	108	-	27
9		245	108	153	173	207	179	101	132	195	201	-

Table 3.1.: Costs of the example instance.

Example In Figure 3.1, an example for the VRPCC is given. Assuming nine jobs and two depots located in $\mathbb{R}^2$ as shown in Subfigure 3.1(a). Each job $i \in \{1, 2, \ldots, 9\}$ is drawn by a dot and the color of the dot represents its customer cost coefficient where green stands for $c_i = 0$ and red stands for the highest value $c_i = 150$. The travel costs and times are derived from the Euclidean distance between the jobs. Further, Table 3.1 provides in the first two rows the customer cost coefficient and the working duration of each job, and in the rows below the travel costs and times. For this purpose, this part of Table 3.1 is designed as an upper triangular matrix that contains the travel times between the jobs and a lower triangular matrix with the travel costs. All jobs have to be served by one vehicle.

For this instance, an optimal solution is the route shown in Subfigure 3.1(a). The corresponding schedule, which describes the start time of each job, is given in Subfigure 3.1(b). The y-axis provides the start day and the x-axis the start minute on that day. The length of the box for a job shows the working duration. As it can

be seen, on the first day, the jobs 1, 9, 4 and 5 are executed. After that, the vehicle is moved to the next job 6 over the day such that job 6 can be started at the begin of the working shift of day two. Due to the long working duration of job 7, the jobs 8, 2 and 3 are done on day three.

Note that this route is not optimal regarding travel costs. Some detours are necessary to visit jobs with higher customer cost coefficient first. It is also not optimal regarding customer costs. However, the route has a minimal total cost value of 2221 whereby the travel cost value is 1471 and the customer cost value is 750.

A travel-cost-optimal solution is given by the permutation $(1, 8, 7, 6, 9, 5, 4, 3, 2)$ with travel costs value 1263; and the permutation $(4, 6, 1, 9, 7, 3, 5, 8, 2)$ is a customer-cost-optimal route with a total customer cost value of 700.

3.2. The Non-Linear Partition and Permutation Model (PP)

The VRPCC can be formulated by the following non-linear partition and permutation model (PP).

$$\text{(PP)} \quad \min \quad g(S) = g^d(S) + g^c(S) \tag{3.4}$$

$$\text{s.t.} \quad g^d(S) := \sum_{k \in M} (d_{s_k \pi_1^k} + \sum_{i=1}^{|N_k|-1} d_{\pi_i^k \pi_{i+1}^k} + d_{\pi_{|N_k|}^k z_k}) \tag{3.5}$$

$$g^c(S) := \sum_{i \in N} c_i t_i^d \tag{3.6}$$

$$S := (N_k, \Pi^k(N_k))_{k \in M}$$

$(N_k)_{k \in M}$ is a partition of N into exactly m non-empty subsets

$\Pi^k(N_k)$ is a permutation of the elements of N_k, $k \in M$

$$t_{\pi_0^k} := t_{s_k} = t_0 \qquad k \in M \tag{3.7}$$

$$t_{\pi_i^k} := \zeta(t^a_{\pi_i^k}, u_{\pi_i^k}) = \zeta(\xi(t_{\pi_{i-1}^k}) + a_{\pi_{i-1}^k} + r_{\pi_{i-1}^k \pi_i^k}, u_{\pi_i^k}) \tag{3.8}$$
$$i = 1, 2, \ldots, |N_k|, \ k \in M$$

A schedule $(N_k, \Pi^k(N_k))_{k \in M}$ consists of the assignment of jobs to vehicles and the orders to visit the jobs. The allocation is stored by means of a partition of the set N into m non-empty subsets N_k, $k \in M$. For each subset N_k, the permutation $\Pi(N_k) = (\pi_1^k, \pi_2^k, \ldots, \pi_{|N_k|}^k)$ gives the execution order of the corresponding jobs. The

execution order $\Pi(N_k)$ is also called route k. Note, that there is no constraint to equally distribute the jobs to the routes. Thus, in a feasible schedule, one route can contain $n - m + 1$ jobs and the other routes include only one job. The objective function $g(S)$, defined by equation (3.4), consists of travel costs and time-dependent customer costs and has to be minimized. The travel costs are calculated based on the particular order of the jobs, see equation (3.5). To determine the customer cost value of a schedule with equation (3.6), the minimally allowed start time of each job has to be determined. For that, the start time of each start depot is set to t_0 by means of equations (3.7). Then, considering the processing order of the jobs given by the permutation $\Pi^k(N_k)$, the start time of each job $i \in N$ can be computed with equation (3.8). As it can be seen in (3.6), the customer cost value of any job increases strictly with the start day. Thus, according to Theorem 3.1, minimally allowed start times lead to the best objective value of a certain schedule.

The main advantage of this model is that the start times are calculated directly from the schedule. Because of that, this model is the base for designing heuristics as shown in Chapter 5, and for developing two branch-and-bound algorithms for the VRPCC, see Chapter 6. Furthermore, this model has low memory requirements: a solution can be stored in an array of length $n + 2m$.

To solve the VRPCC with a commercial optimizer like CPLEX, a collection of various formulations of the VRPCC as mixed-integer linear program is given in Chapter 4. There, the start times become decision variables and the routes are defined by binary variables.

3.3. Extensions of the VRPCC

For practical applications, some additional requirements can be made on the schedule. For example, it can be necessary that all jobs have to be completed within a given time horizon. This further property of the VRPCC is formulated by an upper bound for the start day: $t_i^d \leq d_{\max}$ for each job $i \in N$. An instance can be infeasible, if $d_{\max}$ is not sufficiently large to enable a scheduling of all jobs. This implies that no solution exists.

Definition 3.4. In the *time-constrained VRPCC*, all jobs have to be served within a given time horizon $d_{\max}$. Then, a schedule S is feasible, if $t_i^d \leq d_{\max}$ for each job $i \in N$.

It is also possible, that the maintenance manager demands a plan where the jobs are distributed evenly among all vehicles.

For some applications, e.g., if there is a restriction that high priority jobs have to be resolved within a certain time, it can be necessary to define for each job an individual time windows for its execution. Then, for each job an earliest start time e_i and a latest start time l_i is provided.

Definition 3.5. In the *VRPCC with time windows*, for each job a time window $[e_i, l_i]$ for its execution is defined. Then, a schedule S is feasible, if $e_i \leq t_i^d \leq l_i$ for each job $i \in N$.

In the VRPCC, it is assumed that the track can be booked for maintenance every night. But in practice, the time for maintenance can be further restricted due to planned night trains or other maintenance activities. Then, for each job a list of days can be provided where the track is already booked by traffic or maintenance.

Definition 3.6. In the *VRPCC with closed time windows*, each job $i \in N$ has a list of days $\overline{T}_i \subset T_i$, where maintenance is not possible. Then, a schedule is feasible if $t_i^d \notin \overline{T}_i$ for each job $i \in N$.

It is also possible that only a part of the working shift is closed in some nights because, e.g., a night train will pass the track section which will lead to a more complex computation of the start times of the jobs.

In the VRPCC it is assumed that all jobs can be processed by all machines because all jobs regard to the same category of maintenance. But in practice, it can be necessary to plan jobs of different maintenance categories together. Then, it has to be ensured that each job is processed by an appropriate vehicle, e.g., tamping jobs are executed by tamping machines and grinding jobs are handled by grinding machines.

4. Formulations as Mixed-Integer Linear Program

In Chapter 3, a non-linear formulation of the VRPCC was introduced that describes a schedule by partitioning the jobs into m subsets to allocate jobs to routes and defining a permutation of the jobs of each subset to describe its order in the route. To complete the schedule, the start times of the jobs are computed dependent on the defined routes. But the VRPCC can also be formulated as MILP which allows the usage of commercial software to solve MILPs like CPLEX or Gurobi. Then, the start times (t^d, t^m) cannot be calculated directly from the schedule, but become decision variables. With it, not only the assignment to routes and the order of the jobs have to be determined, but also the start times are part of the decision process. This leads to a strong increase of the solution space which is the main drawback of solving the VRPCC via solving a MILP of it. However, one can profit from years of research of solving MILPs which have found their way in commercial solvers like CPLEX that "uses techniques from branching and cutting together with a "bag of tricks" [...] e.g., strategies for the pre-processing of optimization problems, various ways for using cutting planes, and different heuristics used during the search" [67, p. 154]. With it, the computational times to solve MILPs were improved significantly in the last years, see [86]. Furthermore, a formulation as MILP allows to add straightforwardly several additional constraints, e.g., a limited time horizon for maintenance, the compliance with time windows or restrictions on the choice of the maintenance machine for some jobs.

As introduced in Section 2.3.1, there are different possibilities to formulate a VRP with time constraints as MILP. Thereby, the formulations can be split into a part to define the routes of the vehicles and a part to define start times for the jobs. For the VRPCC, at first a basic MILP with a three-index formulation to define routes and a formulation of the time constraints based on a big-M linearization is presented in Section 4.1. However, computational experiments showed that this first MILP was hard to solve. Because, as proposed in [116, p. 14], "in integer programming, formulation [of] a 'good' model is of crucial importance", some alternatives formulations of the time and route constraints are presented. In detail, Section 4.2 provides several variants to formulate time constraints which are the crucial point so solve the VRPCC. Three alternative ideas to the initial big-M formulation are investigated: Firstly, another definition of the start times is introduced. Secondly, the formulation presented in [146], that uses two-index time variables, is applied. And thirdly, binary time variables are used for the start days that allow to integrate additional valid

constraints on the number of jobs executed per day. In Section 4.3, it is shown how the route constraints can be formulated by means of two-index binary variables. As it will be observed, additional constraints are necessary to ensure the correct order of the depots. For this purpose, four alternative formulations are presented. Finally, in Section 4.4, the presented MILPs for the VRPCC are compared with respect to solution performance with CPLEX based on computational experiments.

Since the start times are non-negative decision variables and there is no constraint that stipulates to use the minimally allowed start time, the solution space can be unbounded. In detail, for jobs with zero customer costs, several values for the start day may lead to an optimal solution. To restrict the solution space without losing the optimal value, an appropriate upper bound for the start day $d_{\max}$ should be defined. In case of the time-constrained VRPCC, $d_{\max}$ is a given value. Otherwise, the upper bound has to be chosen sufficiently large to ensure that an optimal schedule with minimally allowed start times does not exceed it. In this chapter, it is assumed that an upper bound for the start day $d_{\max}$ is given.

4.1. A Basic Formulation (R1T1)

In this section, an initial formulation of the VRPCC as MILP is presented which is based on a common formulation of the VRPTW as presented in [143, p. 158f (VRPTW)]. According to this, the route constraints are formulated by binary three-index variables x_{ij}^k which equals to one if and only if jobs $i \in N_a$ and $j \in N_a$ with $i \neq j$ are assigned to route $k \in M$ and job i is executed directly before job j. To formulate the time constraint, the start time of job $i \in N \cup N_s$ equals to $ht_i^d + t_i^m$ with h are the minutes per day. In contrast to the formulation in [143], the time variables are defined independently from the routes.

The basic MILP for the VRPCC is formulated as:

$$\text{(R1T1)} \quad \min \sum_{k \in M} \sum_{i \in N_a} \sum_{j \in N_a \setminus \{i\}} d_{ij} x_{ij}^k + \sum_{i \in N} c_i t_i^d \tag{4.1}$$

$$\text{s.t.} \sum_{k \in M} \sum_{j \in N_a \setminus \{i\}} x_{ij}^k = 1 \qquad i \in N \cup N_s, \tag{4.2}$$

$$\sum_{j \in N_a \setminus \{i\}} x_{ji}^k - \sum_{j \in N_a \setminus \{i\}} x_{ij}^k = \begin{cases} -1 & \text{if } i = s_k, \\ 1 & \text{if } i = z_k, \\ 0 & \text{otherwise,} \end{cases} \qquad i \in N_a,\ k \in M, \tag{4.3}$$

$$x_{s_k z_k}^k = 0 \qquad k \in M, \tag{4.4}$$

$$x_{ij}^k \in \{0, 1\} \qquad i, j \in N_a,\ k \in M, \tag{4.5}$$

$$(t^d_{s_k}, t^m_{s_k}) = (t^d_0, t^m_0) \qquad k \in M, \tag{4.6}$$

$$ht^d_i + t^m_i + a_i + r_{ij} - ht^d_j - t^m_j \leq (1 - \sum_{k \in M} x^k_{ij})\mathcal{M} \qquad i \in N \cup N_s,\ j \in N,\ i \neq j, \tag{4.7}$$

$$ht^d_i + t^m_i \geq ht^d_0 + t^m_0 + \min_{k \in M}\{r_{s_k i}\} \qquad i \in N, \tag{4.8}$$

$$t^d_0 \leq t^d_i \leq d_{\max}\,, \quad t^d_i \in \mathbb{N} \qquad i \in N, \tag{4.9}$$

$$0 \leq t^m_i \leq u_i\,, \quad t^m_i \in \mathbb{R}_+ \qquad i \in N. \tag{4.10}$$

The objective function is given by formula (4.1) that minimizes the total costs, which are the sum of travel costs and customer costs. The travel costs depend on the predecessor-successor-relations, which are given by the binary variables x^k_{ij}, and the customer costs depend on the start days t^d_i.

The constraints (4.2)–(4.5) represent a three-index route formulation with binary variables x^k_{ij}. In detail, equations (4.2) ensure that each job $i \in N$ is visited once by one vehicle. Equations (4.3) impose that a) each vehicle $k \in M$ starts the route in its start depot, b) each vehicle $k \in M$ finishes its route in its end depot, and c) each job $i \in N$ is entered and left by the same vehicle. Constraints (4.4) guarantee that a vehicle does not travel directly from its start depot to its end depot. With it, empty routes cannot occur. And with constraint (4.5), the variables x^k_{ij} are defined as binary variables.

The definition of start times is regulated by constraints (4.6)–(4.10). To be more precise, equations (4.6) set the start time of each start depot to $t_0 = (t^d_0, t^m_0)$. Constraints (4.7) realize that job $j \in N$ does not start before the previous job i is finished and the vehicle has traveled to the location of job j: If $\sum_{k \in M} x^k_{ij} = 1$, the time variables have to fulfill $ht^d_i + t^m_i + a_i + r_{ij} \leq ht^d_j + t^m_j$. Contrariwise, $\sum_{k \in M} x^k_{ij} = 0$ leads to $ht^d_i + t^m_i + a_i + r_{ij} - ht^d_j - t^m_j \leq \mathcal{M}$ which does not restrict the start times if $\mathcal{M}$ is chosen sufficiently large. For any job $i \in N$, equation (4.8) defines an earliest start time taking into account that at least one start depot has to be scheduled before. Note that the travel times must satisfy the triangle constraint, see Definition 2.3, to ensure that (4.8) are valid constraints. Conditions (4.9) indicate that the start days are integers not larger than the given time horizon $d_{\max}$. Finally, for any job $i \in N$, constraint (4.10) bounds the start minute by u_i to ensure that job i can be finished during the working shift.

Note that constraints (4.7) are a big-M linearization of the non-linear time constraints (3.2). A sufficiently large value for $\mathcal{M}$ is $\mathcal{M} := h(d_{\max} + 1)$. This is an upper bound for the start time in minutes $ht^d_i + t^m_i$, because t^d_i is bounded by $d_{\max}$ for any job $i \in N$ and t^m_i is bounded by $u_i \leq h$.

In modeling TSPs and VRPs, route models similar to (4.2)–(4.5) are normally

completed by subtour elimination constraints. Otherwise, a solution feasible to (4.2)–(4.5) can contain subtours which are cycles of jobs $C = (i_1, i_2, \ldots, i_l)$, $l < |N|$ with $x^k_{i_p i_{p+1}} = 1$, for $p = 1, 2 \ldots, l-1$, and $x^k_{i_l i_1} = 1$. Theorem 4.1 shows that for (R1T1) additional subtour elimination constraints are not necessary since for jobs of a cycle it is not possible to assign time variables that fulfill (4.7).

Theorem 4.1. *All solutions feasible to* (4.2)–(4.10) *cannot contain subtours.*

Proof. Assuming $C = (i_1, i_2, \ldots, i_p)$ is a cycle defined by (4.2)–(4.5). Thus, it exists a route $k \in M$ with $x^k_{i_l i_{l+1}} = 1$, for $l = 1, 2 \ldots, p-1$, and $x^k_{i_p i_1} = 1$. According to (4.3), C cannot contain depots.

From (4.7) results that $\xi(t_{i_{l+1}}) \geq \xi(t_{i_l}) + a_{i_l} + r_{i_l i_{l+1}}$, for $l = 1, 2 \ldots, p-1$, and $\xi(t_{i_1}) \geq \xi(t_{i_p}) + a_{i_p} + r_{i_p i_1}$ with the linear function $\xi : \mathbb{N} \times \mathbb{R} \to \mathbb{R}$ that transforms the start time pair $t = (t^d, t^m)$ into a scalar time value as defined in (3.1). Since for each job $i \in N$, the working duration a_i is larger than zero, the following contradiction is observed

$$\xi(t_{i_1}) < \xi(t_{i_2}) < \ldots < \xi(t_{i_p}) < \xi(t_{i_1}).$$

Consequently, a feasible solution of (R1T1) cannot contain a cycle. □

4.2. Improvements of the Time Constraints

The computational experiments provided in Section 4.4.1 showed that the LP relaxation value of formulation (R1T1) is small compared to the optimal value. This results from small values for the start days in an optimal LP solution. Furthermore, constraints (4.7) are a big-M linearization of the non-linear constraints

$$ht^d_j + t^m_j \geq ht^d_i + t^m_i + a_i + r_{ij}, \quad \text{if } \sum_{k \in M} x^k_{ij} = 1, \quad i, j \in N_a$$

which is known to lead to weak LP relaxations, see [30], and also to cause numerical problems, compare [6] for more information. To overcome these difficulties, in this section some alternative time formulations are presented that will have a tighter LP relaxation. At first, Section 4.2.1 provides a second time formulation based on big-M linearization but with another definition of the time variables. Based on this, Section 4.2.2 provides a flow formulation of the start times which is known to lead to a more tight LP relaxation. Finally, in Section 4.2.3, a model with binary variables for the start days is presented where additional valid constraints can be formulated.

4.2.1. Another Splitting of the Start Times (R1T2)

The first alternative time model is based on a start time t_i in minutes for each job $i \in N \cup N_s$. The model is derived from (R1T1) by replacing $ht^d_i + t^m_i$ by t_i and

adding constraints to define the start day t_i^d as the greatest integer not larger than the start time in minutes divided by the minutes per day.

To obtain the model (R1T2), the constraints (4.6)–(4.10) of (R1T1) are replaced by the constraints (4.11)–(4.16).

$$\text{(R1T2)} \quad \min \sum_{k\in M}\sum_{i\in N_a}\sum_{j\in N_a\setminus\{i\}} d_{ij}x_{ij}^k + \sum_{i\in N} c_i t_i^d$$

$$\text{s.t.} \quad (4.2)\text{–}(4.5)$$

$$t_{s_k} = ht_0^d + t_0^m \qquad k \in M, \qquad (4.11)$$

$$t_i - t_j + a_i + r_{ij} \le (1 - \sum_{k\in M} x_{ij}^k)\mathcal{M} \qquad i \ne j,\ i \in N \cup N_s,\ j \in N, \qquad (4.12)$$

$$ht_0^d + t_0^m + \min_{k\in M}\{r_{s_k i}\} \le t_i \le hd_{\max} + u_i \qquad i \in N, \qquad (4.13)$$

$$t_i \in \mathbb{R}_+ \qquad i \in N, \qquad (4.14)$$

$$0 \le t_i - ht_i^d \le u_i \qquad i \in N, \qquad (4.15)$$

$$t_0^d \le t_i^d \le d_{\max}\,, \quad t_i^d \in \mathbb{N} \qquad i \in N. \qquad (4.16)$$

Constraints (4.11)–(4.14) result from constraints (4.6)–(4.10) by replacing $ht_i^d + t_i^m$ with t_i. Inequalities (4.15) ensure the correct definition of the start days and the compliance with the working shift: If job i is executed on day t_i^d, the start minute on this day is equal to $t_i - ht_i^d$ which has to be in $[0, u_i]$. And constraints (4.16) bound the start days of the jobs.

The following two small changes in the model can significantly improve the LP relaxation:

- In the LP relaxation, the whole time between two workings shifts can be used because the working shift constraints are neglected when the start days become non-integer values. The reason is that constraint (4.15) can always be satisfied, e.g., by $t_i^d = \frac{t_i}{h}$. To minimize the falsely usable time between the working shifts, it can be defined that

$$h = \min\left\{1440,\ u + \max_{i,j\in N_a} r_{ij}\right\}. \qquad (4.17)$$

  If the longest travel time between two jobs $i, j \in N_a$ is less than the time between two working shifts, traveling between two jobs can always be completed before the next working shift begins. This remains true, if h is redefined by

(4.17). Otherwise, equation (4.17) ensures that h is not larger than the number of minutes per day.

- If $t_0^m + \min_{k \in M}\{r_{s_k i}\} > u_i$, job i cannot start at day t_0^d. Consequently, for each job $i \in N$, the lower bound of the start day in constraint (4.16) can be replaced by

$$t_{0i}^d := t_0^d + \max\left\{0, \left\lceil \frac{1}{h}\left(t_0^m + \min_{k \in M}\{r_{s_k i}\} - u_i\right)\right\rceil\right\}. \tag{4.18}$$

The computational results showed that these small changes significantly increase the quality of the LP relaxation, compare Section 4.4.1. Nevertheless, also with these improvements, in an LP solution of (R1T2), all start days can be equal to t_{0i}^d because if the binary variables x_{ij}^k become real numbers, subtour-like structures can be defined and start times can be chosen small.

Note that both improvements can also be applied to (R1T1). The resulting formulation is referred to as (R1T1i).

4.2.2. Two-Index Variables for the Start Times (R1T3)

It is known, that the LP relaxation of the big-M formulation used in (R1T1) and (R1T2) is weak, see [143]. The computational experiments, provided in Section 4.4.1, showed that in the LP relaxation solutions, the start times are close to its lower bounds. This is caused by the fact that constraints (4.12) are satisfied for (partly) consecutive executed jobs $i, j \in N$, which means $x_{ij}^k > 0$ for any $k \in M$, with start times t_i and $t_j = t_i + \Delta$, if

$$\Delta \geq a_i + r_{ij} - (1 - \sum_{k \in M} x_{ij}^k)\mathcal{M}.$$

Because $\mathcal{M}$ is large in comparison to $a_i + r_{ij}$, the time difference Δ between the start times t_j and t_i can be small, even if the sum $\sum_{k \in M} x_{ij}^k$ is close to one. If $(1 - \sum_{k \in M} x_{ij}^k)$ is larger than $\frac{1}{\mathcal{M}}(a_i + r_{ij})$, then the time difference Δ can even be negative which means that job j can have a start time smaller than the start time of its (fractional) predecessor i. Consequently, a solution of the LP relaxation of (R1T2) can be found in which all jobs start on the first day which minimizes the customer cost value.

To obtain a better LP relaxation, for the next time formulation the time variables t_i of (R1T2) are replaced by the two-index flow variables t_{ij} with $i \in N \cup N_s$ and $j \in N \cup N_z$ as proposed in [146]. These variables are zero if $x_{ij}^k = 0$ for each $k \in M$. Contrariwise, if any vehicle travels from job i to job j, the start time of job i is given by t_{ij}. Since job i is left once by one vehicle, only one variable t_{ij} is larger than zero. To the resulting MILP formulation is referred by (R1T3). In the following, firstly the formulation is presented and secondly, it is discussed whether two-index time variables lead to larger start times in the LP relaxation solutions.

For a job $i \in N$, let $\delta_i^{\text{out}} = N \setminus \{i\} \cup N_z$ be the set of all possible successors and $\delta_i^{\text{in}} = N_s \cup N \setminus \{i\}$ be the set of all possible predecessors. For the start depots s_k with $k \in M$, the set of possible successors is $\delta_{s_k}^{\text{out}} = N$. Then, the constraints (4.6)–(4.10) of (R1T1) are replaced by (4.19)–(4.24) to obtain (R1T3).

$$\text{(R1T3)} \quad \min \sum_{k \in M} \sum_{i \in N_a} \sum_{j \in N_a \setminus \{i\}} d_{ij} x_{ij}^k + \sum_{i \in N} c_i t_i^d$$

$$\text{s.t.} \quad (4.2)\text{–}(4.5)$$

$$\sum_{i \in \delta_{s_k}^{\text{out}}} t_{s_k i} = h t_0^d + t_0^m \qquad k \in M, \qquad (4.19)$$

$$\left(h t_0^d + t_0^m + \min_{k \in M} \{ r_{s_k i} \} \right) \sum_{k \in M} x_{ij}^k \le t_{ij} \le (h d_{\max} + u_i) \sum_{k \in M} x_{ij}^k \qquad i \in N \cup N_s,\ j \in \delta_i^{\text{out}}, \qquad (4.20)$$

$$\sum_{j \in \delta_i^{\text{out}}} t_{ij} \ge \sum_{j \in \delta_i^{\text{in}}} \left(t_{ji} + (a_j + r_{ji}) \sum_{k \in M} x_{ji}^k \right) \qquad i \in N, \qquad (4.21)$$

$$t_{ij} \in \mathbb{R}_+ \qquad i \in N \cup N_s,\ j \in \delta_i^{\text{out}}, \qquad (4.22)$$

$$0 \le \sum_{j \in \delta_i^{\text{out}}} t_{ij} - h t_i^d \le u_i \qquad i \in N, \qquad (4.23)$$

$$t_{0i}^d \le t_i^d \le d_{\max}\,, \quad t_i^d \in \mathbb{N} \qquad i \in N. \qquad (4.24)$$

Equations (4.19) define the start times of the start depots. Inequalities (4.20) ensure that the flow variable t_{ij} is zero, if $\sum_{k \in m} x_{ij}^k = 0$. Otherwise, t_{ij} must be not smaller than the minimal feasible start time and not larger than the maximal feasible start time. Consequently, together with the route constraints (4.2) and (4.3), it is guaranteed that for each job $i \in N$, a single time variable t_{ij} with $j \in \delta_i^{\text{out}}$ is larger than zero, which is the start time of job i. Further, there is a single time variable t_{ji} with $j \in \delta_i^{\text{in}}$ larger than zero, which is the start time of the predecessor of job i. Constraints (4.21) ensure the correct scheduling of the jobs which means that job $i \in N$ must not start before its predecessor, which belongs to the set δ_i^{in}, is finished and the vehicle has traveled to the location of job i. Constraints (4.22) define the start times in minutes as continuous numbers. Inequalities (4.23) ensure the correct definition of start days and the compliance with the working shift. Finally, constraints (4.24) specify the start days as integers between the minimal feasible start day t_{0i}^d and the time horizon $d_{\max}$.

Note that the time formulation of (R1T3) has similarities to flow problems where t_{ij} represents the time flow on edge $\{i, j\}$. The inequalities (4.21) can be interpreted

as a kind of the flow conservation rule, see, e.g., [90, p. 153]. Because of that, the right side of inequality (4.21) is called the time flow entering job i and the left side of inequality (4.21) is the time flow leaving job i.

The comparison of the LP relaxation solutions of (R1T3) and (R1T2) shows that in the LP relaxation of (R1T3), the start time of a job i cannot be smaller than

$$\sum_{j \in \delta^{\text{in}}} t_{ji} + (a_j + r_{ji}) \sum_{k \in M} x_{ji}^k.$$

This means that from each predecessor the working duration and travel time to job i are partly taken into account and summarized. This is an essential difference to the big-M formulation, where each pair of jobs is separately analyzed. Consequently, in the LP relaxation of (R1T2), the start time of job i cannot be smaller than

$$\max_{j \in \delta^{\text{in}}} \; t_j + a_j + r_{ji} - (1 - \sum_{k \in M} x_{ji}^k)\mathcal{M}.$$

Recap, since $(1 - \sum_{k \in M} x_{ji}^k) \geq 0$ and $\mathcal{M}$ is significantly larger than $a_j + r_{ji}$, it is also possible to obtain $t_i < t_j$ even job j is fractionally scheduled before job i.

The computational experiments confirmed that the LP relaxation of (R1T3) is not worse than the LP relaxation of (R1T2), but also not much better, see Section 4.4.1. It turned out that also the LP relaxation of the time constraints (R1T3) led to start times close to its lower bound. The reason for this behavior is that in the LP relaxation, a part of the entering flow can be moved to other jobs to reduce the start time of the main successor, which is the job $j \in \delta_i^{\text{out}}$ for which $\sum_{k \in M} x_{ji}^k$ is close to one. In detail, for a job $i \in N$ with job $\hat{\jmath}$ as main predecessor (thus $x_{\hat{\jmath}i}$ is close to one), it is

$$\sum_{j \in \delta_i^{\text{out}}} t_{ij} \geq t_{\hat{\jmath}i} + (a_{\hat{\jmath}} + r_{\hat{\jmath}i}) \sum_{k \in M} x_{\hat{\jmath}i}^k \; + \sum_{j \in \delta_i^{\text{in}} \setminus \{\hat{\jmath}\}} \left(t_{ji} + (a_j + r_{ji}) \sum_{k \in M} x_{ji}^k \right).$$

To reduce the start time of job i, either $\sum_{k \in M} x_{\hat{\jmath}i}^k$ or $t_{\hat{\jmath}i}$ itself has to be reduced. In detail, the time flow from $\hat{\jmath}$ to i must satisfy

$$t_{\hat{\jmath}i} \geq \sum_{j \in \sigma_{\hat{\jmath}}^{\text{in}}} \left(t_{j\hat{\jmath}} + (a_j + r_{j\hat{\jmath}}) \sum_{k \in M} x_{j\hat{\jmath}}^k \right) - \sum_{j \in \sigma_{\hat{\jmath}}^{\text{out}} \setminus \{i\}} t_{\hat{\jmath}j}.$$

Consequently, to reduce the start time of job i, a part of the time flow can be shifted to other jobs, e.g., jobs with a zero customer cost coefficient, by choosing $x_{\hat{\jmath}j} > 0$, $j \in \sigma_{\hat{\jmath}}^{\text{out}} \setminus \{i\}$. The following example illustrates this effect.

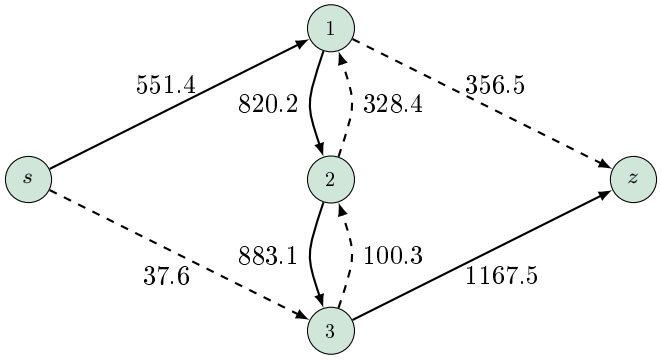


Figure 4.1.: Example for the time flow in an LP relaxation solution of formulation (R1T3).

Example Assuming an instance with three jobs $\{1, 2, 3\}$ and two depots s and z, which are located as shown in Figure 4.1. Each job $i \in \{1, 2, 3\}$ has a customer cost coefficient $c_i = 100$ and working duration $a_i = 240$. The travel costs and times correspond to the Euclidean distances and are given by

$$(d_{ij})_{i,j\in\{s,1,2,3,z\}} = (r_{ij})_{i,j\in\{s,1,2,3,z\}} = \begin{pmatrix} 0 & 112 & 101 & 112 & 201 \\ 112 & 0 & 51 & 101 & 112 \\ 101 & 51 & 0 & 51 & 101 \\ 112 & 101 & 51 & 0 & 112 \\ 201 & 112 & 101 & 112 & 0 \end{pmatrix}.$$

The working shift has a length of 480 minutes and the start time in the start depot is $t_0 = (0, 480)$. Consequently, if a single vehicle is available, on each day only one job can be finished.

With $h = 682$, the two matrices

$$X = (x^1_{ij})_{i,j\in\{s,1,2,3,z\}} = \begin{pmatrix} 0 & 0.94 & 0 & 0.06 & 0 \\ 0 & 0 & 0.93 & 0 & 0.07 \\ 0 & 0.06 & 0 & 0.94 & 0 \\ 0 & 0 & 0.07 & 0 & 0.93 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

and

$$T = (t_{ij})_{i\in\{s,1,2,3\};j\in\{s,1,2,3,z\}} = \begin{pmatrix} 0 & 448 & 0 & 31 & 0 \\ 0 & 0 & 550.5 & 0 & 332 \\ 0 & 311 & 0 & 610.5 & 0 \\ 0 & 0 & 80 & 0 & 842 \end{pmatrix}$$

represent a solution of the LP relaxation. In this solution, each job $i \in \{1, 2, 3\}$ has $\sum_{j\in\{s,1,2,3,z\}} t_{ij} \leq 1162$ which is the end of the working shift of day one. Consequently, $t^d_i = 1.0$ for each job $i \in \{1, 2, 3\}$. Recap, in a feasible solution, each day a single job can be executed. The reason for this behavior is that due to non-integer route variables x^1_{ij}, a part of the time flow is shifted to other jobs. This is illustrated

in Figure 4.1 where an arrow between two vertices $i, j \in V$ represent that $x_{ij}^1 > 0$. If the arrow has a dashed line, then $x_{ij}^1 < 0.1$. The labels on the arrows show the value of corresponding time flow from job i to job j which is $t_{ij} + (a_i + r_{ij})x_{ij}^1$. It can be seen in Figure 4.1 that there is a main flow where the jobs are visited in order $1 \to 2 \to 3$, and a second flow where the jobs are visited in reverse order. With this second flow, a part of the entering time flow is shifted to other jobs to reduce the start time of each job.

4.2.3. Binary Variables for Start Days (R1T4) and (R1T5)

Comparing the LP relaxation of (R1T3) with (R1T2) showed that the improvements are small and that nevertheless all job with non-zero customer cost coefficient had the first day as start day in the LP relaxation. Furthermore, the computational experiments showed that (R1T3) was harder to solve than (R1T1) and (R1T2), compare Section 4.4.1.

Another possibility to improve the LP relaxation is to add further valid constraints. For example, it can be assumed that at most η jobs can be visited per day which is valid if η is not too small. To add an according constraint, the start days are modeled by binary variables. Then, $y_{i\tau}$ is one if and only if job i is started at day $\tau \in T = \{t_0^d, t_0^d + 1, \ldots, d_{\max}\}$. Thus, the start day of job i equals to $\sum_{\tau \in T} \tau y_{i\tau}$. The model derived from (R1T2) by using binary variables for start days is given as

$$\text{(R1T4)} \quad \min \sum_{k \in M} \sum_{i \in N_a} \sum_{j \in N_a \setminus \{i\}} d_{ij} x_{ij}^k + \sum_{i \in N} c_i \left(\sum_{\tau \in T_i} \tau y_{i\tau} \right) \tag{4.25}$$

$$\text{s.t.} \quad \text{(4.2)–(4.5), (4.11)–(4.14),}$$

$$0 \le t_i - h \sum_{\tau \in T_i} \tau y_{i\tau} \le u_i \qquad i \in N, \tag{4.26}$$

$$\sum_{\tau \in T_i} y_{i\tau} = 1 \qquad i \in N, \tag{4.27}$$

$$\sum_{i \in N} y_{i\tau} \le \eta \qquad \tau \in T, \tag{4.28}$$

$$y_{i\tau} \in \{0, 1\} \qquad i \in N,\ \tau \in T. \tag{4.29}$$

In the objective function (4.25), the sum of travel cost value and customer cost value is minimized. The binary variables x_{ij}^k and the start times in minutes t_i have to satisfy the constraints (4.2)–(4.5) of (R1T1) and (4.11)–(4.14) of (R1T2). Constraints (4.26) ensure that the start day is the largest integer less than $\frac{t_i}{h}$ and that the job is processed within the working shift. Equalities (4.27) indicate that exactly one start day is allocated to each job. Note that $T_i := \{t_{0i}^d, t_{0i}^d + 1, \ldots, d_{\max}\} \subseteq T$ is the set of feasible start days for job i with t_{0i}^d is the smallest feasible start day as defined in equation (4.18). By means of constraints (4.28), it is realized that at

most η jobs are allocated to any day $\tau \in T$. Some possibilities to compute η are presented later in Section 6.2.1.1. For short, η is computed by finding a subset of jobs with maximal cardinality such that the sum of the job's working durations and the smallest possible travel times does not exceed the time available per day for working and traveling. Finally, constraints (4.29) defines $y_{i\tau}$ as binary variables.

The computational experiments showed that the upper bound of the jobs per day defined with constraint (4.28) leads to a significant improvement of the LP relaxation value, see Section 4.4.1.

Clearly, the binary start days can also be applied to the basic formulation (R1T1) which leads to the MILP called (R1T5). For this purpose, the variables t_i^d with $i \in N \cup N_s$ are replaced by $\sum_{\tau \in T} \tau y_{i\tau}$ in the base formulation (R1T1). Then, the objective function (4.25), the route constraints (4.2)–(4.5), the constraint (4.10) and the constraints on the binary variables for the start day (4.27)–(4.29) are combined with (4.30)–(4.33).

$$\text{(R1T5)} \quad \min \sum_{k \in M} \sum_{i \in N_a} \sum_{j \in N_a \setminus \{i\}} d_{ij} x_{ij}^k + \sum_{i \in N} c_i \left(\sum_{\tau \in T_i} \tau y_{i\tau} \right)$$

$$\text{s.t.} \quad \text{(4.2)–(4.5), (4.10), (4.27)–(4.29)}$$

$$y_{s_k,\tau} = \begin{cases} 1, & \text{if } \tau = t_0^d, \\ 0, & \text{otherwise,} \end{cases} \qquad k \in M,\ \tau \in T, \tag{4.30}$$

$$t_{s_k}^m = t_0^m \qquad k \in M, \tag{4.31}$$

$$h \sum_{\tau \in T} \tau y_{i\tau} + t_i^m + a_i + r_{ij} - h \sum_{\tau \in T} \tau y_{j\tau} - t_j^m \leq (1 - \sum_{k \in M} x_{ij}^k) \mathcal{M} \qquad i \in N \cup N_s,\ j \in N,\ i \neq j, \tag{4.32}$$

$$h \sum_{\tau \in T} \tau y_{i\tau} + t_i^m \geq h t_0^d + t_0^m + \min_{k \in M} \{ r_{s_k i} \} \qquad i \in N. \tag{4.33}$$

The start times of the start depots are defined by the constraints (4.30) and (4.31) as (t_0^d, t_0^m). For job $i \in N \cup N_s$ and $j \in N$ with $\sum_{k \in M} x_{ij}^k = 1$, the constraints (4.32) ensure that job j starts not before job i is finished and the vehicle has traveled to the location of job j. This constraint is similar to (4.7) with t_i^d is replaced by $\sum_{\tau \in T} \tau y_{i\tau}$. Finally, for any job $i \in N$, constraint (4.33) bounds the start time converted into minutes from below by the minimal feasible start time. Note, that h is computed by equation (4.17) to strengthen the LP relaxation.

Remark With binary variables for the day of job execution, additional constraints for the VRPCC with closed time windows can be easily integrated, e.g., by adapting T_i.

4.3. Route Constraints with Two-Index Variables

In this section, alternative formulations for the route constraints (4.2)–(4.5) of (R1T1) are presented. Recap, in (R1T1) the binary variables x_{ij}^k denote whether vehicle $k \in M$ travels from job $i \in N \cup N_s$ to job $j \in N \cup N_z$. With it, m routes are defined. To reduce the number of variables, the m routes of the m vehicles can be formulated as one large route through all depots using two-index binary variables. Then, decision variables x_{ij} encode whether a vehicle travels from job i to job j. Similar formulations can be found, e.g., in [40, 95, 143]. Additionally, there are artificial travels from end depot z_k to start depot s_{k+1}, for all $k = 1, 2, \ldots, m-1$, and from z_m to s_1. To the formulation of route constraints with two-index variables is referred as (R2).

$$\text{(R2)} \qquad \min \sum_{i \in N \cup N_s} \sum_{j \in N \cup N_z \setminus \{i\}} d_{ij} x_{ij} + \sum_{i \in N} c_i t_i^d \tag{4.34}$$

$$\text{s.t.} \sum_{j \in N_a \setminus \{i\}} x_{ij} = 1 \qquad i \in N_a, \tag{4.35}$$

$$\sum_{j \in N_a \setminus \{i\}} x_{ji} = 1 \qquad i \in N_a, \tag{4.36}$$

$$x_{z_m s_1} = 1, \tag{4.37}$$

$$x_{z_k s_{k+1}} = 1 \qquad k \in M \setminus \{m\}, \tag{4.38}$$

$$x_{s_k z_l} = 0 \qquad k, l \in M, \tag{4.39}$$

$$x_{ij} \in \{0, 1\} \qquad i, j \in N_a,\ k \in M, \tag{4.40}$$

$$t_i^d \in \mathbb{N} \text{ obtained from time constraints} \qquad i \in N.$$

The objective function, given by equation (4.34), is the sum of travel costs and customer costs which is minimized. Note that the artificial travels from the end depots to the start depots are not considered by calculating the travel costs. Constraints (4.35) and (4.36) ensure that each job and each depot is visited and left exactly once. Equations (4.37) and (4.38) define the artificial travelings between the depots. Equations (4.39) forbid empty routes.

To define the start times t_i^d of the job $i \in N$, the time constraints of the previous formulated models can be used by replacing $\sum_{k \in M} x_{ij}^k$ with x_{ij}: For this purpose,

- (T1) denotes constraints (4.6)–(4.10) and (T1i) its improvement,
- (T2) denotes constraints (4.11)–(4.16),
- (T3) denotes constraints (4.19)–(4.24),
- (T4) denotes objective function (4.25) and constraints (4.11)–(4.14), (4.26)–(4.29), and

- (T5) denotes objective function (4.25) and constraints (4.10), (4.27)–(4.29), (4.30)–(4.33),

either applying three-index variables x_{ij}^k or two-index variables x_{ij}.

However, (R2) in combination with one of the time constraint variants is not sufficient to define a feasible schedule of the VRPCC. Two kinds of infeasibilities can occur:

- In (R2), it is not ensured that behind the start depot s_k the first visited depot is the end depot z_k. For example, for an instance with $m = 3$, a large route with the depot order $s_1 \to \ldots \to z_2 \to s_3 \to \ldots \to z_1 \to s_2 \to \ldots \to z_3 \to s_1$ would be feasible to (R2), but is not a correct schedule because the depots have to be visited in the order $s_1 \to \ldots \to z_1 \to s_2 \to \ldots \to z_2 \to s_3 \to \ldots \to z_3 \to s_1$.
- In difference to (R1), subtours are not eliminated in (R2). Due to the fact that travels are also defined between the depots, a cycle $C = \{i_1, i_2, \ldots, i_l\}$ can contain depots. And since the time constraints must not be met for depots, the contradiction of Theorem 4.1 is resolved there. Note that at most m subtours can occur which would lead to m routes with an incorrect depot order.

So, additional constraints are necessary to guarantee the correct order of the depots and exclude subtours in the large route. Four variants of such constraints are introduced next: in Section 4.3.1 and Section 4.3.2, two variants that use a certain class of subtour elimination constraints are presented; and Section 4.3.3 and Section 4.3.4 provide two variants with additional variables to allocate jobs to routes.

4.3.1. Application of MTZ-Constraints (R2a)

One possibility for a well-defined large route with two-index variables is the usage of MTZ-constraints, as introduced in [112], which are subtour elimination constraints named after their creators Miller, Tucker and Zemlin. Then, an additional decision variable v_i represents the position of job $i \in N_a$ in the large route beginning in depot s_1. The resulting model is given by

$$\text{(R2a)} \qquad \min \sum_{i \in N \cup N_s} \sum_{j \in N \cup N_z \setminus \{i\}} d_{ij} x_{ij} + \sum_{i \in N} c_i t_i^d$$

$$\text{s.t.} \quad \text{(4.35)–(4.40)},$$

$$v_{s_1} = 0, \tag{4.41}$$

$$0 \leq v_i \leq n + 2m - 1 \qquad i \in N_a, \tag{4.42}$$

$$v_i - v_j + (n+2m-1) x_{ij} \leq n+2m-2 \; i, j \in N_a, \; j \neq s_1, \tag{4.43}$$

$$v_{s_k} \geq v_{s_{k-1}} + 3 \qquad k \in M \setminus \{1\}. \tag{4.44}$$

The model (R2a) is an extension of (R2). The objective function (4.34) and the constraints on the binary route variables (4.35)–(4.40) are completed by the MTZ-constraints (4.41)–(4.44). In detail, equation (4.41) sets the position of the start depot s_1 to zero. Constraints (4.42) bound the position value from below by zero and from above by $n+2m-1$. Inequalities (4.43) ensure that v_j is not smaller than v_i+1, if job i is the predecessor of job j which means $x_{ij}=1$. Otherwise, inequality (4.43) is trivially true because v_i-v_j cannot exceed $n+2m-2$. Constraints (4.44) guarantee the correct order of the start depots in the large route.

Remark In [39], an improved formulation of the MTZ-constraint is given as

$$v_i - v_j + (n+2m-1)x_{ij} + (n+2m-3)x_{ji} \leq n+2m-2 \qquad i,j \in N_a,\ j \neq s_1. \tag{4.45}$$

In case of $x_{ij}=1$, two inequalities have to be considered, which are $v_i - v_j \leq -1$ and, with interchanged indexes, $v_j - v_i \leq 1$. Consequently, $v_j = v_i + 1$, if $x_{ij}=1$. The comparison of the LP relaxation showed that with constraints (4.45) instead of (4.43), larger LP relaxation values were obtained, see Section 4.4.1. In the following, this variant is referred to as (R2as).

4.3.2. A Flow Formulation of the MTZ-Constraints (R2b)

In [57], a stronger formulation of the MTZ-constraints is presented, where the position of each job in the route is modeled by a flow formulation. For this purpose, the variables v_i are replaced by the flow variables v_{ij} with

$$v_{ij} = \begin{cases} \text{the position of job } i, & \text{if } x_{ij}=1, \\ 0, & \text{otherwise.} \end{cases}$$

The resulting mixed-integer program is

$$\text{(R2b)} \qquad \min \sum_{i \in N \cup N_s} \sum_{j \in N \cup N_z \setminus \{i\}} d_{ij}x_{ij} + \sum_{i \in N} c_i t_i^d$$

s.t. (4.35)–(4.40)

$$v_{s_1 i} = 0 \qquad i \in N_a, \tag{4.46}$$

$$v_{ij} \leq (n+2m-1)x_{ij} \qquad i,j \in N_a,\ i \neq s_1, \tag{4.47}$$

$$\sum_{j \in N_a \setminus \{i\}} v_{ij} - \sum_{j \in N_a \setminus \{i\}} v_{ji} = 1 \qquad i \in N_a \setminus \{s_1\}, \tag{4.48}$$

$$\sum_{i \in N_a} v_{s_k i} \geq \sum_{i \in N_a} v_{s_{k-1} i} + 3 \qquad k \in M \setminus \{1\}, \tag{4.49}$$

$$v_{ij} \geq 0 \qquad i,j \in N_a. \tag{4.50}$$

The constraints (4.46)–(4.50) complete (R2) by defining the position of each job in the long route. To ensure that the start depot s_1 is on position zero, equations (4.46) set all flow variables corresponding to start depot s_1 to zero. Constraints (4.47) ensure that v_{ij} is zero if $x_{ij} = 0$ and that otherwise v_{ij} is bounded by $n + 2m - 1$. Equations (4.48) guarantee that the position of any job or depot is by one larger than the position of the predecessor in the route. With it, again the successor of s_1 gets position 1 and the end depot z_m is on position $n + 2m - 1$. Constraints (4.49) lead to a correct order of the start depots. Finally, the variables v_{ij} are defined as numbers not smaller zero.

As shown in [150], the advantage of the flow formulation is that it leads to a stronger LP relaxation than the original MTZ-constraints. This effect was also obtained in the computational experiments. With route model (R2b), smaller gaps between the LP relaxation value and the optimal value were obtained. But the improved variant of (R2a) led to a tighter LP relaxation than (R2b), see Section 4.4.2.

4.3.3. Binary Route Assignment (R2c)

For some variants of the VRP, it is necessary to know which vehicle visits a job. Then, a binary variables y_{ik} can be introduced that represent the assignment of jobs to routes by

$$y_{ik} = \begin{cases} 1, & \text{if job } i \in N_a \text{ belongs to route } k \in M, \\ 0, & \text{otherwise,} \end{cases}$$

see [55, 143]. As shown later in Theorem 4.2, the assignment of jobs to routes is also able to solve both problems of the two-index formulation: the depots will be visited in the correct order and subtours cannot occur. The resulting MILP is called (R2c).

$$\begin{aligned}
\text{(R2c)} \quad \min \quad & \sum_{i \in N \cup N_s} \sum_{j \in N \cup N_z \setminus \{i\}} d_{ij} x_{ij} + \sum_{i \in N} c_i t_i^d \\
\text{s.t.} \quad & (4.35)\text{–}(4.40) \\
& \sum_{k \in M} y_{ik} = 1 && i \in N_a && (4.51) \\
& y_{s_k k} = 1 && k \in M && (4.52) \\
& y_{z_k k} = 1 && k \in M && (4.53) \\
& y_{ik} - y_{jk} \leq 1 - x_{ij} - x_{ji} && k \in M,\ i, j \in N && (4.54) \\
& y_{s_k k} - y_{jk} \leq 1 - x_{s_k j} && k \in M,\ j \in N && (4.55) \\
& y_{ik} - y_{z_k k} \leq 1 - x_{i z_k} && k \in M,\ i \in N && (4.56) \\
& y_{ik} \in \{0, 1\} && i \in N_a,\ k \in M && (4.57)
\end{aligned}$$

The constraints (4.51)–(4.57) complete (R2). To be more precise, equations (4.51) ensure that each job is assigned to one route. Equations (4.52) and (4.53) allocate the depots to the corresponding routes. Constraints (4.54) impose that two successively visited jobs are assigned to the same route: If x_{ij} or x_{ji} is one, two inequalities, namely $y_{ik} - y_{jk} \leq 0$ and $y_{jk} - y_{ik} \leq 0$, have to be considered which leads to $y_{ik} = y_{jk}$. Otherwise, constraints (4.54) are trivially hold because y_{ik} and y_{ik} are binary and consequently, $y_i - y_j$ cannot be larger than one. Constraints (4.55) and (4.56) ensure that the successor of start depot s_k and the predecessor of end depot z_k are assigned to route $k \in M$. All constraints together lead to $y_{ik} = 1$, if and only if job i is visited between the start depot s_k and the end depot z_k in the long route through all jobs and depots.

Theorem 4.2. *A solution of (R2c) in combination with time constraints cannot contain subtours and the depots are visited in the correct order.*

Proof. Firstly, the correct order of the depots is proven via contradiction. Assuming a solution where the first depot visited after s_k is z_l with $l \neq k \in M$. Constraint (4.52) leads to $y_{s_k k} = 1$. Let $i \in N$ be the successor of s_k, then from (4.55) follows that $y_{ik} = 1$. Further, inequalities (4.54) lead to $y_{jk} = 1$ for all $j \in N$ visited after job i but before the next end depot. In case of z_l is the first end depot visited after s_k with $l \in M$ and $l \neq k$, constraint (4.56) indicates that also $y_{z_l k} = 1$. Further, from equation (4.53) follows that $y_{z_l l} = 1$. Together, this leads to an infeasibility in constraint (4.51) for the end depot z_l.

Secondly, it is shown that the solution cannot contain subtours. Assuming a solution with a cycle C. To allow a feasible definition of the start times, C must contain depots as shown in Theorem 4.1. Without loss of generality, it is assumed that s_1 belongs to C (contrariwise, there is another cycle in $N_a \setminus C$ that contains s_1). As shown above, the next visited end depot has to be z_1. With constraint (4.38) of (R2) is ensured that s_2 is the successor of z_1. Then, the next visited end depot has to be z_2, which is the predecessor of s_3. A continuation of this process leads to the conclusion that all depots must be part of the cycle. Then, there has to be another cycle without depots. But, as shown in Theorem 4.1, for such a cycle, it is not possible to define feasible start times. Consequently, a feasible solution of (R2c) in combination with time constraints cannot contain cycles. □

4.3.4. Integer Route Assignment (R2d)

For the formulation (R2d), the binary variables y_{ik} of (R2c) are replaced by integer variables y_i with $y_i = k$ if and only if job $i \in N_a$ is assigned to route $k \in M$. With it, the number of variables is reduced.

$$\text{(R2d)} \quad \min \sum_{i \in N \cup N_s} \sum_{j \in N \cup N_z \setminus \{i\}} d_{ij} x_{ij} + \sum_{i \in N} c_i t_i^d$$

s.t. (4.35)–(4.40)

$$y_{s_k} = k \qquad k \in M \qquad (4.58)$$

$$y_{z_k} = k \qquad k \in M \qquad (4.59)$$

$$y_i - y_j \leq (1 - x_{ij} - x_{ji})m \qquad i, j \in N \qquad (4.60)$$

$$y_{s_k} - y_i \leq (1 - x_{s_k i})m \qquad k \in M,\ i \in N \qquad (4.61)$$

$$y_i - y_{z_k} \leq (1 - x_{i z_k})m \qquad k \in M,\ i \in N \qquad (4.62)$$

$$y_i \in \{1, 2, \ldots, m\} \qquad i \in N_a \qquad (4.63)$$

By means of the constraints (4.58)–(4.63), the route formulation (R2) is completed. With equations (4.58) and (4.59) the depots are assigned to the corresponding routes. Constraints (4.60) ensure that two consecutive executed jobs $i, j \in N$ are assigned to the same route. In detail, if x_{ij} or x_{ji} is one, two inequalities are considered, which are $y_i - y_j \leq 0$ and $y_j - y_i \leq 0$. Consequently, $y_j = y_i$. Otherwise, if $x_{ij} = x_{ji} = 0$, constraint (4.60) is trivially hold. The reason is that $y_i - y_j$ cannot exceed m, because y_i and y_i are defined as positive integers not larger than m. Constraints (4.61) guarantee that $y_i \geq k$, if job i is the successor of the start depot s_k. Similar to that, constraint (4.62) impose that $y_i \leq k$, if job i is the predecessor of the end depot z_k. Finally, constraint (4.63) defines that y_i is an integer not larger than m. All constraints together ensure that $y_i = k$, if job i is visted after s_k but before z_k in the long route through all jobs and depots.

Theorem 4.3. *A solution of (R2d) in combination with time constraints cannot contain subtours and the depots are visited in the correct order.*

Proof. Initially, the correct order of the depots is shown. Let s_k be a start depot and let z_l be the first end depot visited after s_k with $l, k \in M$. Let further $j \in N$ be the successor of s_k and $i \in N$ the predecessor of z_l. Then, it follows from the constraints (4.58) and (4.61) that $y_j \geq y_{s_k} = k$. Further, the constraints (4.59) and (4.62) lead to $y_i \leq y_{z_l} = l$. Additional, the constraints (4.60) stipulate that all jobs between j and i, including j and i, are assigned to the same machine. Thus,

$$k \leq y_j = y_i \leq l. \qquad (4.64)$$

With this observation, the proof can be given via backwards induction:

- If $k = m$, then also $l = m$, because this is the only end depot z_l with $m \leq l$. Thus, the first depot visited after s_m is z_m.

- If $k = m - 1$, then an end depot z_l, with $m - 1 \leq l$, has to be the first end depot visited behind s_{m-1}. As previously shown, the start depot visited before z_m is s_m. Consequently, the first depot reached after s_{m-1} can only be z_{m-1}.
- Assuming the depots are visited in the correct order for all $k = m, m - 1, \ldots, m - p$ and $1 \leq p \leq m - 2$. For $k = m - p - 1$, the first visited depot after s_{m-p-1} has to be z_l with $l \geq m - p - 1$ to satisfy inequality (4.64). Due to the fact that the depots s_k and z_k with $k \geq m - p$ are visited in the correct order, z_{m-p-1} has to be the first depot reached after s_{m-p-1}.

Consequently, only the correct order of the depots allow a feasible route assignment.

Analogous to the proof of Theorem 4.2, the correct order of the depots ensures that the solution does not contain subtours. □

Compared to (R2c), the number of variables and constraints is reduced. The results of the computational experiments in Section 4.4.2 suggest that using integer instead of binary variables leads to a better solvable MILP.

4.4. Computational Results

In this section, the MILPs are compared in terms of computational effort to solve them with CPLEX. For this purpose, the instances of benchmark `S` (as described in Appendix A) were modeled as different MILPs and solved with CPLEX version 12.8 using the default settings and limiting the computational time to ten minutes.

At first, the major finding is outlined. After that, in Section 4.4.1, the presented formulations for time constraints are analyzed and in Section 4.4.2, the provided formulations for route constraints are examined.

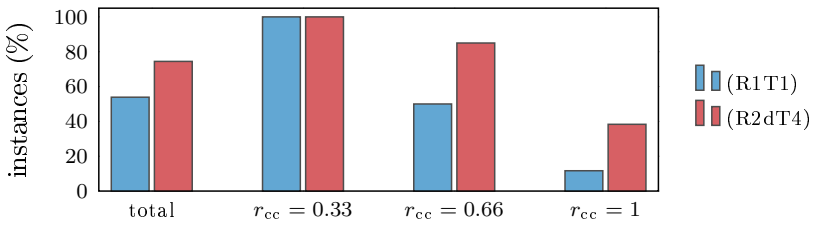


Figure 4.2.: Percentage of optimally solved instances for two selected MILPs.

The major finding of the computational results is that the VRPCC was hard to solve. In Figure 4.2, for two selected MILPs of the VRPCC, the percentage of optimally solved instances of benchmark `S` is shown dependent on the ratio of jobs with non-zero customer cost coefficient r_{cc}. In detail, the basic formulation (R1T1) is compared with the formulation (R2dT4) which turns out to be the best of the presented MILPs. Even the instances are small—with only fifteen jobs and two vehicles—a lot of instances were not optimally solved within the time limit of ten

minutes. In total, 46.1% of the instances were not optimally solved if they were formulated by the base model and 25.6% of the instances if (R2dT4) was applied. The instance of benchmark `S`, where only five of fifteen jobs have a non-zero customer cost coefficient, were always solved in less than ten minutes as shown in Figure 4.2 by the bars for $r_{cc} = 0.33$. In case of afflicting two third of the jobs with a non-zero customer cost coefficient, the percentage of within ten minutes optimally solved instances formulated according (R1T1) and (R2dT4) was drastically reduced to 50% and 85%, respectively. And of the instance where all jobs are afflicted with a non-zero customer cost coefficient, only 11.7% and 38.3%, respectively, of the instances were optimally solved within the time limit. This shows that the number of jobs with non-zero customer cost coefficient has a strong influence on the hardness to solve an instance.

4.4.1. Comparison of Time Formulations

In this subsection, the presented time formulations are compared in terms of LP relaxation quality and computational performance when solving them with CPLEX. Recap, the presented time formulations are

- (T1) and its improvement (T1i), where the start time equals to the sum $ht_i^d + t_i^m$ of the start day t_i^d and the start minute on this day t_i^m for job $i \in N$,
- (T2), where the start time equals to t_i and the start day t_i^d of job $i \in N$ is computed from t_i,
- (T3), where two-index variables t_{ij} are applied to define start times in minutes and to compute the start day,
- (T4), where the start time equals to t_i and the start day is defined by binary variables $y_{i\tau}$ which enables the formulation of additional constraints to limit the number of jobs visited per day, and
- (T5), which is similar to (T4) but with $h\left(\sum_{\tau \in T} \tau y_{i\tau}\right) + t_i^m$ as start time.

A detailed definition of the constraints is given in Sections 4.1 and 4.2.

At first, the LP relaxations of the formulated MILPs are compared because the quality of the LP relaxation has a strong influence to the performance of solving a MILP with CPLEX. After that, performance analysis for the computational time and the gap to an optimal solution are done. In conclusion, it will be seen that (T4) showed the best performance in terms of computational time, gap to an optimal value and also quality of the LP relaxation.

Figure 4.3 shows on the left the cumulative distribution of the start days obtained with the LP relaxations of the presented time models compared to the cumulative distribution of the start days in an optimal solution S^*. Note that the start time in the depots is $(0, 480)$ which is the end of the working shift on day zero, compare the benchmark definition given in Appendix A. Consequently, job processing cannot

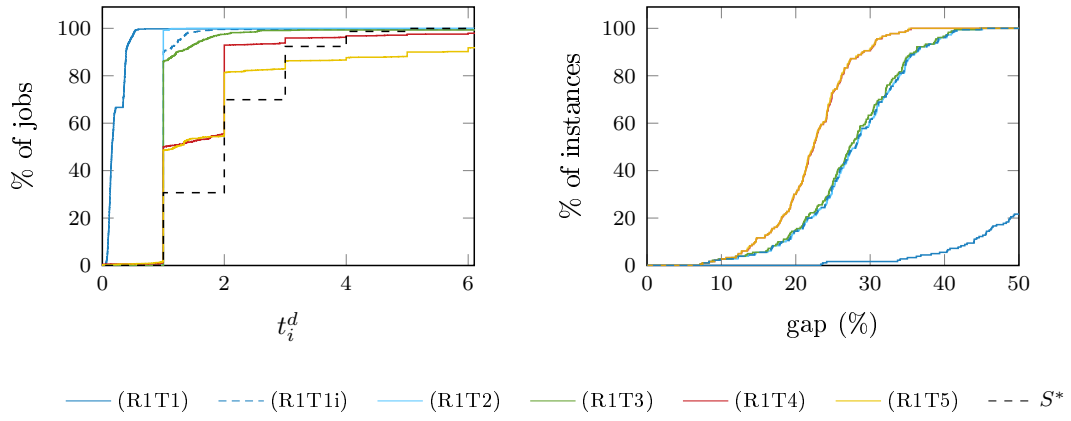


Figure 4.3.: Cumulative distribution for the start days and performance profiles for the gap between LP relaxation value and optimal value for the time models on benchmark S.

start before day one. On the right of Figure 4.3, performance profiles for the gap between LP relaxation value and the optimal value are shown.

With (R1T1) as presented in Section 4.1, the start day of each job was less than one which results from insufficient lower bounds for the start days and the big-M linearization. Consequently, the customer cost value of the LP relaxation was small and the gap to the optimal value was large: For each instance, the LP relaxation was more than 20% smaller than the optimal value and for more than 75% of the instances, the LP relaxation value was less then half of the optimal costs.

The formulation (R1T2) led to significantly better LP relaxation values. This is mainly caused by tighter lower bounds for the start day of the jobs: For each job $i \in N$, the start day t_i^d must greater than or equal to one. As it can be seen in the left plot of Figure 4.3, in the obtained LP relaxation solutions of (R1T2), nearly all jobs had start day one. With the improved variant of (R1T1i), where the conversion factor h and minimal feasible start day t_{0i}^d are defined as proposed for model (R1T2), for some jobs a larger start day was determined. But nevertheless, both formulations had the same LP relaxation values because for all jobs with a non-zero customer cost coefficient, the start day was one. The reason is that the big-M formulation leads to a weak LP relaxation. As mentioned in Section 4.2.2, due to $\mathcal{M}$ is large in comparison to the time difference between two consecutive executed jobs, even with x_{ij}^k close to one the start time of job j can be equal to the start time of its predecessor i.

The application of two-index variables for the start times in formulation (R1T3) resulted in some larger start days, but the LP relaxation was not significantly improved. This can be seen in the right plot of Figure 4.3 on the fact that the performance profile of (R1T3) is only slightly above the one of (R1T1i) and (R1T2). It was observed that also the formulation with two-index time variables allows to determine the start days minimal, but the binary route variables cannot be as close to one as with the big-M linearization. Consequently, the LP relaxations of the

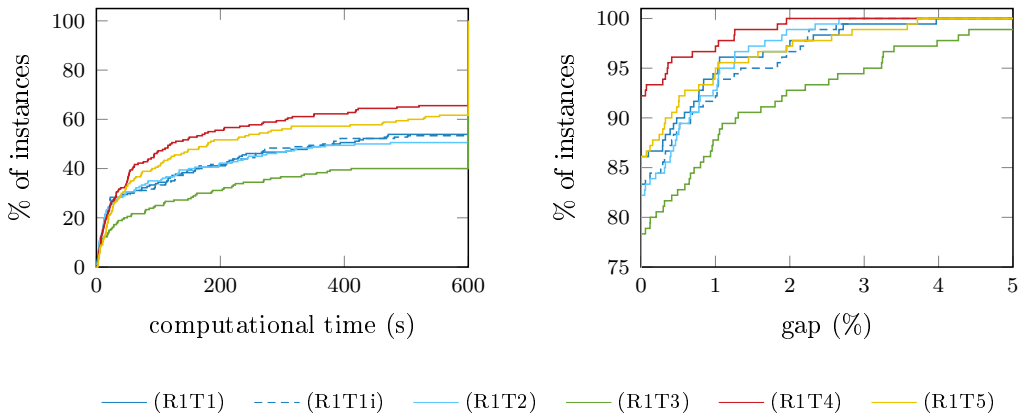


Figure 4.4.: Comparison of the formulations of time constraints on benchmark S.

	min	Q1	Q2	Q3	max
(R1T1)	0.01	0.18	2.41	4.36	5.79
(R1T1i)	<0.01	0.16	2.30	4.43	6.29
(R1T2)	<0.01	0.17	2.26	3.80	4.81
(R1T3)	<0.01	0.27	0.64	0.80	2.04
(R1T4)	<0.01	0.15	0.81	2.62	3.75
(R1T5)	0.01	0.19	1.06	3.13	4.56

Table 4.1.: Number of nodes (in millions), analyzed during the solution process with CPLEX, for different time formulations in benchmark S.

formulations (R1T2) and (R1T3) led to the same customer cost value, but with (R1T3) a larger travel cost value was determined.

The MILPs (R1T4) and (R1T5) led to almost the same LP relaxation value. Due to the additional constraints, that limit the number of jobs per day, the computed start times were larger than in the other variants. Consequently, a lot of jobs had a start day larger or equal to day two. But, in comparison to an optimal solution S^*, the start days were still too small. Comparing the performance profiles of the LP relaxation value in the right plot of Figure 4.3 shows that the LP relaxation of (R1T4) and (R1T5) outperformed the other models. Nevertheless, the gap between LP relaxation value and optimal value is large: the minimal measured gap was 7.1% and for half of instances, the gap exceeded 22%.

After comparing the quality of the LP relaxation of the different formulations, the performance of solving them with CPLEX is analyzed. For this purpose, Figure 4.4 shows performance profiles for computational time and gap to the optimal value of the time formulations combined with the basic route formulation (R1). Table 4.1 provides statistic values for the number of analyzed nodes during the solution process with CPLEX. In detail, the minimal value, the first, second and third quartile as

	min	Q1	Q2	Q3	max
$r_{cc} = 0.33$	0.7	7.1	17.0	36.9	350.9
$r_{cc} = 0.66$	4.2	50.9	181.2	600.0	600.0
$r_{cc} = 1.00$	10.4	415.0	600.0	600.0	600.0

Table 4.2.: Statistic values for computational times of the formulation (R1T4) for different groups of benchmark `S`.

well as the maximum value are given in millions. Note, that CPLEX solves MILPs with a branch-and-cut approach [105]. Thus, on each node of the search tree, the LP relaxation of the subproblem is solved with some iterations of an optimizer, such as the dual simplex algorithm.

As it can be seen in Figure 4.4, the MILP (R1T3), which applies two-index variables to formulate time constraints, was harder to solve. A reason could be that the significantly increased number of variables results in LP relaxations harder to solve. This impression is underpinned by the number of analyzed nodes. As it can be seen in Table 4.1, less nodes were analyzed than with the other approaches even more time was spend to the solution process.

The variants of the big-M formulation with integer variables for the start days, which are (R1T1), (R1T1i) which applies the tightened conversion factor h and minimal start day t^d_{0i}, and (R1T2), showed a similar performance. The computational times and the number of analyzed nodes were similar. This is an unexpected observation because (R1T1) in the variant presented in Section 4.1 had a worse LP relaxation in comparison to (R1T1i). But a comparison of the log-files of CPLEX showed that in both variants the same values in the root node were obtained which means that CPLEX adds appropriate cuts to exclude start days less than t^d_{0i} which cannot lead to a feasible solution.

As expected, the model (R1T4) and (R1T5), where additional constraints limit the number of jobs visited per day, outperformed the other models. The best of the presented time constraint formulations was (R1T4) because with this formulation, less time was required to solve the instances and, in case of exceeding the time limit of ten minutes, better solutions were obtained than with the other time constraint formulations. This can be seen in Figure 4.4 on the fact that the performance profiles for computational time and gap of (R1T4) are both well above the performance profiles of the others. Also comparing the number of analyzed nodes of (R1T4) and (R1T5) suggests that branching was more efficient if the start day is computed from an overall start time in minutes, as in formulation (R1T4), instead of defining the start time as sum of start day and start minute at a day, as in (R1T5).

Concluding, the formulation (R1T4), where the start times are defined in minutes and binary variables are used to define the start days and restrict the number of jobs visited per day by a valid upper bound η, led to the best performance in solving it with CPLEX.

Finally, Table 4.2 provides the statistic values minimum, 25th-percentile, median, 75th-percentile and maximum for computational times of the formulation (R1T4) for instances of benchmark S grouped by r_{cc}, which is the percentage of jobs afflicted with non-zero customer cost coefficient. As it can be seen, the computational effort was significantly higher if more jobs were afflicted with a non-zero customer cost coefficient. For instances generated with $r_{cc} = 0.33$, which means five of fifteen jobs have a non-zero customer cost coefficient, half of instances were solved within less than 17 seconds. Doubling the number of jobs with non-zero customer cost coefficient resulted in an increase of the median by more than factor ten, see Table 4.2 column Q2. From the instances with $r_{cc} = 1$, where each job has a non-zero customer cost coefficient, less than a half was solved within the time limit of 10 minutes (which are 600 seconds).

4.4.2. Comparison of Route models

In this subsection, the five different route models, introduced in Section 4.1 and Section 4.3, are compared. These are

- (R1), which is the three-index formulation,
- (R2a) and (R2as), which are the two-index formulations with classical MTZ-constraints and its strengthened variant, respectively,
- (R2b), which is the two-index formulation with a flow-formulation of the MTZ-constraints,
- (R2c), which is the two-index formulation with binary variables to allocate jobs to routes, and
- (R2d), which is the two-index formulation with integer variables to allocate jobs to routes.

The route formulations were combined with the time constraints (T4) because it showed the best performance, as observed in the computational experiments provided in Section 4.4.1

Again, firstly the quality of the LP relaxation is analyzed. After that, the performance of solving the formulations with CPLEX is compared with respect to computational time and gap to an optimal solution. In conclusion, it will be seen that with (R2d) the best performance in terms of computational time, gap to an optimal value and also quality of the LP relaxation was observed.

Figure 4.5 shows performance profiles for the gap of the LP relaxation value to the optimal value for the presented route formulations. As it can be seen, the formulations (R2asT4), (R2cT4) and (R2dT4) had the best LP relaxation values. Note that the LP relaxation values of (R2cT4) and (R2dT4) were equal. With the other three formulations, significantly smaller LP relaxation values were obtained.

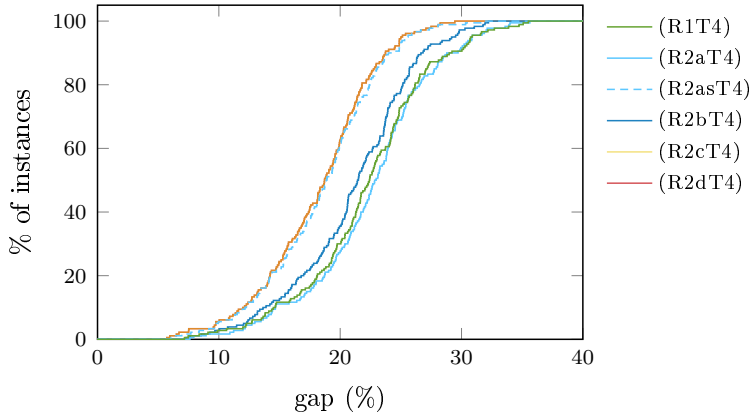


Figure 4.5.: Performance profiles for the gap between LP relaxation value and optimal value for the route formulations on benchmark `S`.

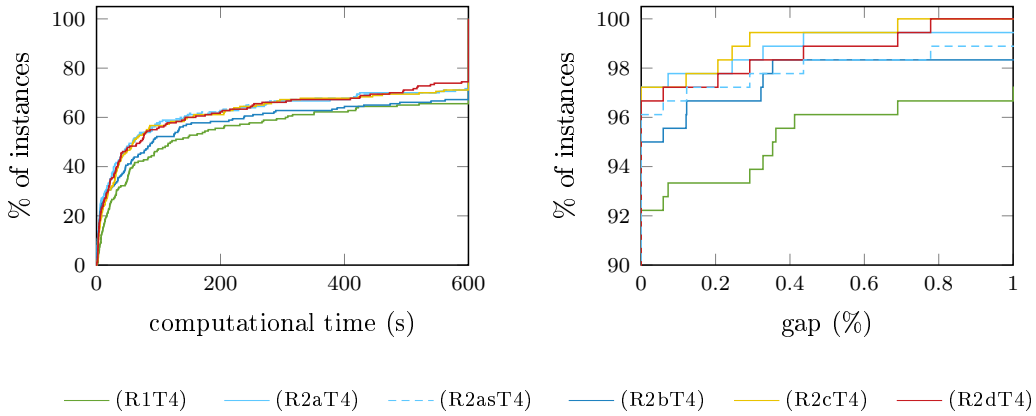


Figure 4.6.: Comparison of formulations of route constraints on benchmark `S`.

	min	Q1	Q2	Q3	max
(R1T4)	<0.01	0.15	0.81	2.62	3.75
(R2aT4)	<0.01	0.03	0.30	2.02	3.00
(R2asT4)	<0.01	0.05	0.31	1.91	2.84
(R2bT4)	<0.01	0.03	0.36	0.99	2.10
(R2cT4)	<0.01	0.09	0.46	2.57	5.67
(R2dT4)	<0.01	0.08	0.46	2.61	4.73

Table 4.3.: Number of nodes (in millions), analyzed during the solution process with CPLEX, for route formulation on benchmark `S`.

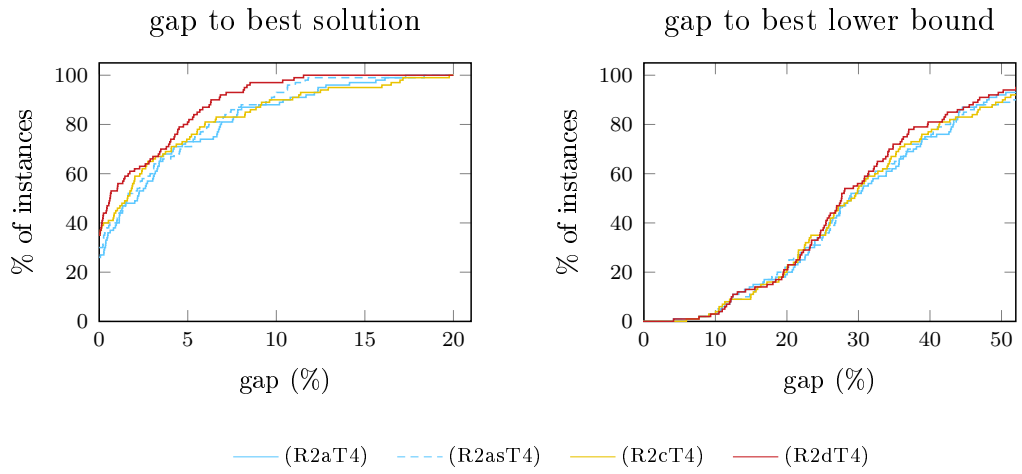


Figure 4.7.: Comparison of selected formulations of route constraints on benchmark `M`.

To compare the performance of solving the formulations with CPLEX, Figure 4.6 shows performance profiles for computational time and gap between the obtained objective value and the optimal value of the route formulations combined with the best time formulation (T4). Further, Table 4.3 provides the statistic values minimum, 25th-percentile, median, 75th-percentile and maximum for the number of nodes analyzed during the solution process. Recap, in each node an LP relaxation is solved where some variables are fixed.

As it can be seen in Figure 4.6, the formulations (R1T4) and (R2bT4) were outperformed by the other variants. Unexpectedly, (R2aT4) and its strengthened variant showed a similar performance even the LP relaxation of (R2asT4) was significantly better. For both, almost the same computational times were required and the number of analyzed nodes was similar. For the instances, were the time limit of ten minutes was exceeded, (R2aT4) led to slightly better solution than its strengthened variant.

Also (R2cT4) and (R2dT4), where the jobs are allocated to routes by additional constraints and variables, were solved fast. (R2dT4) led to the highest number of instances optimally solved within ten minutes. But, in case of exceeding the time limit of ten minutes, with (R2cT4) the gap to an optimal solution was smaller as shown in the right plot of Figure 4.6. For both variants, the number of analyzed nodes during the solution process was approximately equal but higher than for formulations (R2aT4) and (R2bT4), compare Table 4.3.

Due to the models (R2aT4), (R2cT4) and (R2dT4) showed a comparable performance on benchmark `S`, they are further analyzed on benchmark `M`, which consists of 100 instances with 30 jobs. For a detailed description of benchmark `M`, compare Appendix A. None of the instances was solved within the time limit of ten minutes. Because of that, only the quality of the best integer solution obtained after ten minutes is analyzed. Figure 4.7 shows on the left performance profiles for the gap between the obtained objective value and the best obtained solution with one

	min	Q1	Q2	Q3	max
$r_{cc} = 0.33$	0.4	2.9	6.0	20.3	143.0
$r_{cc} = 0.66$	1.9	28.7	72.7	424.8	600.0
$r_{cc} = 1.00$	6.7	247.1	600.0	600.0	600.0

Table 4.4.: Statistic values for computational times of formulation (R2dT4) for different groups of benchmark `S`.

of the presented MILPs, and on the right performance profiles for the gap between the obtained objective value and the best lower bound of CPLEX after ten minutes. Note that the lower bound provided by CPLEX is the minimum LP relaxation value of all unexplored nodes. It can be observed that better solutions were obtained with (R2dT4) because its performance profile is mostly above the other ones. But it can also be seen that after ten minutes the gap between the best found integer and the best lower bound was large. Thus, it may be that the quality of the obtained solution was worse compared to an optimal solution, or that the lower bounds were worse.

In closing, for the model (R2dT4), the best performance of CPLEX was reached. For this MILP formulation, Table 4.4 shows statistic values for the computational times of the instances of benchmark `S` grouped by the percentage of jobs with non-zero customer cost coefficient. As it can be seen, the computational effort was significantly higher if more jobs were afflicted with a non-zero customer cost coefficient. In detail, for instances, where five of fifteen jobs have a non-zero customer cost coefficient, the median of the computational time was 6.0 seconds. Doubling r_{cc} led to a more than ten times higher median. And from the instances, where all jobs have a non-zero customer cost coefficient, more than a half was not solved within ten minutes. But, in comparison to the results of (R1T4) provided in Table 4.2, the computational times were significantly improved in all three groups.

4.5. Conclusion

In this chapter, different MILP formulations for the VRPCC were presented. The constraints of each model can be divided into a part to model the routes of the vehicles and a part to control the time variables. For each part, several formulations were presented that can be combined without restrictions.

In the Sections 4.1 and 4.2, several variants to ensure correct start times were presented. The first model, given in Section 4.1, was based on the time pair $t = (t^d, t^m)$ with the start day t^d and the start minute t^m on day t^d as introduced in Chapter 3. The big-M linearization technique was used to linearize the time constraints (3.2) which ensure that each job does not start before its predecessor is finished and the vehicle has traveled to its location. Due to worse LP relaxation values, several alternative formulations for the time constraints were developed in Section

4.2. At first, a start time in minutes was used instead of the time tuple which led to some observations reasoning the worse LP relaxation of the initial model and leading to a significant improvement of it. After that, a linearization based on two-index time variables was applied which is known to lead to better LP relaxation values. Finally, binary variables for the start day were introduced in order to add valid constraints that restrict the number of jobs visited per day without eliminating feasible solutions.

After formulating several variants of time constraints, alternative route formulations were investigated. With it, five possibilities to ensure correct routes were stated. The first route model, presented in Section 4.1, is based on binary three-index variables. In Section 4.3, four variants were shown with binary two-index variables to model the routes of the vehicles by one large route. There, mainly two different approaches to ensure feasible routes were presented: On the one hand, the usage of well-known subtour elimination constraints that assign each job to a position in the large route; and on the other hand, an explicit assignment of jobs to vehicles.

The computational experiments provided in Section 4.4 showed that the formulation (R2dT4) led to the best performance in solving the VRPCC with CPLEX. Recap, (R2dT4) is a two-index route formulation with an assignment of jobs to routes combined with time constraints where the start time is defined in minutes and the start day is modeled by binary variables. Further, it was observed that instances were harder to solve, if more jobs are afflicted with a non-zero customer cost coefficient.

5. Heuristics and Local Search Approaches

The computational experiments of solving MILP formulations of the VRPCC showed that they are hard to solve. Even the test instances were small—with only fifteen jobs and two vehicles—a lot of instances could not be optimally solved within a time limit of ten minutes. In detail, for the best model (R2dT4) only 74.4% of the instances were solved in less than ten minutes. Furthermore, it was observed that the number of jobs afflicted with a non-zero customer cost coefficient had a significantly impact on the hardness to solve the VRPCC. The more jobs have a non-zero customer cost coefficient, the higher is the expected computational time. For larger instances, exact solving will be very time consuming and sometimes not even possible. Computational tests on instances with 30 jobs showed, that none of the hundred instances could be solved within ten minutes. Furthermore, after ten minutes the gap between the best lower bound and the best found integer solution was large.

In order to find good solutions for larger instances in reasonable time, in this chapter some heuristics and metaheuristics to obtain a feasible solution are presented: In the first section of this chapter, different greedy heuristics for the VRPCC are developed. Greedy heuristics are a common approach to get an approximate solution for a complex optimization problem that can be divided into single decisions. In each step, the decision is selected which seems to be the best. Three different criteria to decide which is the best decision are developed. Section 5.2 provides a rollout algorithm to obtain approximate solutions for the VRPCC. This is an algorithm that evaluates the decisions of each step by means of a heuristic to see the effects on the cost value. In each step, the decision with the smallest cost evaluation is taken. In Section 5.3, a local search procedure for the VRPCC is presented which improves an initial solution iteratively. For this purpose, more cost-efficient solutions are searched by moving jobs to another position or route. Finally in Section 5.4, the results of computational experiments are presented, where the performance of the heuristics in terms of solution quality is analyzed.

In the following, a solution will be a feasible solution, which is in general not optimal. Note that, if the heuristics are applied to an extension of the VRPCC, it is not ensured to find a feasible solution. For example, in case of the time-constrained VRPCC or the VRPCC with time windows, some jobs can remain unplanned which leads to an uncompleted schedule.

In this chapter, the following notation is used: A *partial schedule* consists of a set of

unplanned jobs $\widetilde{N}$ and a schedule of the jobs of $N \setminus \widetilde{N}$ given by $\widetilde{S} = (\widetilde{N}_k, \Pi^k(N_k))_{k \in M}$. An empty schedule is given by $(\emptyset, ())_{k \in M}$ and $\widetilde{N} = N$ with infinite costs.

5.1. Greedy Heuristics

Greedy heuristics are a special class of algorithms developed for optimization problems. A greedy solution is built up step-by-step, selecting in each step the decision that maximize or minimize an evaluation function, see [33].

For the VRPCC, the developed greedy algorithms are designed as follows: The solving process starts with an empty schedule. By means of n decision steps, the n jobs are appended to a certain route. In each step, locally the best choice to append one currently unplanned job to one route is selected and carried out. For short, the job and the route will be called job-route-pair.

Definition 5.1. A job-route-pair is feasible if and only if appending the job at the end of the route does not lead to constraint violations.

To take the locally best choice of all feasible job-route-pairs, a selection criteria is used. The selection step is repeated until all jobs are appended to a route or no more feasible job-route-pairs exist. In the latter case, the heuristic ends with an uncompleted schedule and another try to find a feasible solution is needed.

In this dissertation, three new selection criteria are developed for the VRPCC:

- nearest neighbor, that is based on the well-known nearest neighbor heuristic of TSPs [15] to minimize travel costs, see Section 5.1.1;
- most expensive neighbor, that is the contrary of the nearest neighbor idea and is focused on customer costs, see Section 5.1.2; and
- cost-balanced neighbor, that considers both cost parts, see Section 5.1.3.

In each case, the greedy algorithm for the VRPCC has a computational complexity of $\mathcal{O}(n^2 m)$: the algorithm requires n steps to append all jobs; and for each step, the effort to select the best pair of job and route is of order $\mathcal{O}(nm)$.

As defined in Chapter 3, a feasible solution of the VRPCC must not contain empty routes. Since the designed greedy heuristics do not ensure that each route contains at least one job, finally a simple algorithm to fill empty routes, called FILL algorithm, is applied: Firstly, for each route $k \in M$ it is checked whether it is empty. In this case, a job is selected from another route and added to the empty route. The route $l \in M$, from which the job is selected, must contain at least two jobs and the total costs to shift the job to route k must be minimal. The pseudocode of the FILL algorithm is shown in Appendix B as Algorithm 1.

5.1.1. Nearest Neighbor Heuristic

Due to its simplicity, the nearest neighbor heuristic [15] is an in practice widely used heuristic for the TSP: Starting with an arbitrary city, in each step the nearest city not visited so far is added to the tour until the tour is completed. This is a very simple approach because it is easy to apply and has a low computational effort, but it can produce solutions of poor quality. It can be shown that there is no constant worst case performance, see [128]. An application to a VRP can be found, e.g., in [23].

To apply the nearest neighbor heuristic (NN) to the VRPCC, the approach is modified such that from the unplanned jobs with small travel costs the one with the highest customer cost coefficient is selected. In more detail, the algorithm starts with an empty solution $\widetilde{S} := (\emptyset, ())_{k \in M}$ and the set of unplanned job $\widetilde{N}$ contains all jobs. A given factor $f \geq 0$ represents the allowed variation from the minimal travel cost value. The schedule is build up stepwise by appending one job after the other to a route. In each step, the best feasible job-route-pair is searched, which consists of four substeps:

1. If no feasible job-route-pair exists, return the uncompleted schedule. In this case, the NN heuristic was not successful.

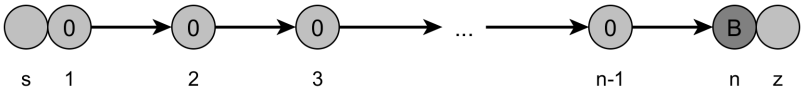
2. Determine the minimal travel cost value $d_{\min}$ of all feasible job-route-pairs, which is
$$d_{\min} := \min_{k \in M,\ j \in \widetilde{N}} \{d_{l_k j} |\, (j,k) \text{ feasible}\}$$
whereby l_k is the current last job of route $k \in M$. Note, if a route is empty, l_k equals to the start depot s_k.

3. Identify the maximal customer cost coefficient c_* of all jobs with travel costs not larger than $d_{\min}(1+f)$ to any route. This is computed by
$$c_* := \max_{k \in M,\ j \in \widetilde{N}} \{c_j |\, (j,k) \text{ feasible},\ d_{l_k j} \leq d_{\min}(1+f)\}.$$

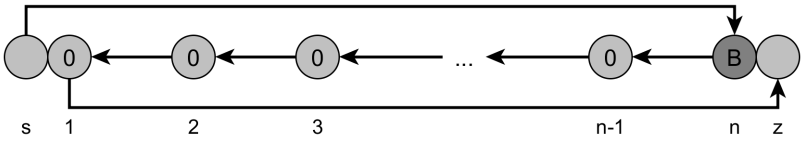
4. Select from the job-route-pairs, where the customer cost coefficient of the job is equal to c_*, the one which leads to the smallest travel costs. Then, append the job to the corresponding route and remove it from the set of unplanned jobs $\widetilde{N}$.

Finally, if all jobs are appended but one or more routes are empty, the FILL algorithm is applied to shift one job to each empty route. The pseudocode of the NN heuristic is shown in Appendix B as Algorithm 2.

Since the focus is mainly on travel costs, the customer cost value and the total costs can be large compared to an optimal solution. It is shown by the following theorem that the approximation ratio of the NN heuristic is unbounded.



(a) solution of NN heuristic



(b) an (in some cases optimal) solution

Figure 5.1.: Example of an instance where the nearest neighbor heuristic has an unbounded approximation ratio.

Theorem 5.1. *The approximation ratio of the NN heuristic is unbounded.*

Proof. The proof for a given algorithm parameter f is done by an example. Imagine the following instance with n jobs and one vehicle: Between the start depot s and the finish depot z, the jobs $\{1, 2 \dots n\}$ are arranged in a line in consecutive order. For job $i \in N$, the travel costs to job $j \in N$ are $d_{ij} = \frac{D}{n}|i - j|$ with $D > (1 + f)n$, and the costs to travel to a depot are $d_{si} = \frac{D}{n}(i - 1)$ and $d_{iz} = \frac{D}{n}(n - i)$. The customer cost coefficients are zero for all jobs except job n which has $c_n = B$. The working time of each job is set to $\frac{u}{2}$ and the travel times hold $1 \leq r_{ij} \leq h - \frac{u}{2}$ for all jobs $i, j \in N_a$. With it, every day one and only one job can be visited. The start time of the start depot is set to $t_0 = (0, u)$.

In this instance, because of $D > (1 + f)n$, only the nearest neighbor is taken into account when searching for the highest customer cost coefficient c_* of the nearest neighbors. Consequently, independent from the customer costs coefficients, the NN heuristic leads to the solution $S_{\text{NN}} = ((N, (1, 2, 3 \dots n)))$ with travel costs $g^d(S_{\text{NN}}) = (n - 1)\frac{D}{n}$ and customer costs $g^c(S_{\text{NN}}) = nB$ since job n is executed as last job on day n. This solution is shown in Figure 5.1 on the top. The circles represent the jobs and their customer cost coefficients are written inside. The route is illustrated by arrows.

Another feasible solution is $S = ((N, (n, n - 1, n - 2, \dots, 1)))$, which is shown in Figure 5.1 below. For this solution, the travel costs are $g^d(S) = 3(n - 1)\frac{D}{n}$; and the customer costs amount to $g^c(S) = B$ because job n is done as first job. Note that this solution is optimal for instances with B sufficient high. This leads to the approximation ratio

$$\frac{g(S_{\text{NN}})}{g(S^*)} \geq \frac{g(S_{\text{NN}})}{g(S)} = \frac{(n - 1)\frac{D}{n} + nB}{3(n - 1)\frac{D}{n} + B} \in \Theta(n).$$

Thus, in this example the approximation ratio increases with the number of jobs. Consequently, it exists no constant upper bound on the approximation ratio. □

5.1.2. Most-Expensive Neighbor Heuristic

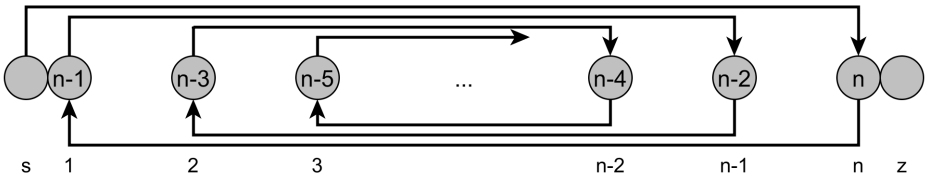
In the most-expensive neighbor heuristic (MEN), the selection of jobs mainly depends on customer costs. Due to choosing in each step the job with highest customer cost coefficient would lead to a too high travel effort, another approach is used: From the jobs with a high customer cost coefficient, a feasible job-route-pair with smallest total cost evaluation is selected.

To be more precise, the MEN heuristic starts with an empty solution and the set of unplanned job $\widetilde{N}$ contains all jobs. A given factor $f \geq 0$ represents the allowed variation from the most-expensive job, which is the unplanned job with highest customer cost coefficient. The schedule is build up stepwise by appending one job after the other to a route. Thus, in each step a feasible job-route-pair has to be chosen. Selecting the best feasible job-route-pair according to the most-expensive selection criterion consists of four substeps:

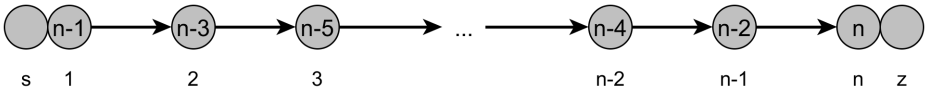
1. Determine the highest customer cost coefficient of the unplanned jobs
$$c_{\max} := \max\{c_j | \, j \in \widetilde{N}\}.$$
2. If no feasible job-route-pair (j,k) with $c_j \geq f c_{\max}$ exists, return the uncompleted schedule. In this case, the MEN heuristic was not successful.
3. Compute for each feasible job-route-pair (j,k) with $c_j \geq f c_{\max}$ the evaluation value
$$e(j,k) := d_{l_k j} + t^d c_{\max}$$
with l_k is the currently last job of route $k \in M$ and t^d is the start day resulting from appending job j to route k. Note that the evaluation is not computed with the customer cost coefficient of job j, but with $c_{\max}$. This avoids that jobs with a smaller customer cost coefficient are preferred. Otherwise, the jobs with smaller customer cost coefficient can have a better evaluation even when the travel costs are higher.
4. Select the job-route-pair with minimal evaluation value and append the job to the corresponding route.

These steps are repeated until all jobs are planned. Finally, it is checked whether the schedule contains empty routes that need to be filled by the FILL algorithm. The pseudocode of the MEN heuristic is shown in Algorithm 3, Appendix B.

The MEN heuristic is mainly focused on the customer costs and there are a lot of cases were the solution of the MEN heuristic will be far away from an optimal solution. In the proof of the following theorem, an example is designed which shows that there is no constant upper bound for the approximation ratio of the MEN heuristic.



(a) solution of MEN heuristic



(b) optimal solution

Figure 5.2.: Example of an instance where the most-expensive neighbor heuristic has an unbounded approximation ratio.

Theorem 5.2. *The approximation ratio of the MEN heuristic is unbounded.*

Proof. Assuming an instance with n jobs and one vehicle, where the n jobs are arranged in consecutive order between the start depot s and the end depot z. The travel costs between two jobs i and j are $d_{ij} = D|i - j|$ with a constant D, the distances to the depots are $d_{si} = D(i-1)$ and $d_{iz} = D(n-i)$. In order to ensure that all jobs can be executed within one day, the working times and travel times are very small, and u is large. The start time of the start depot is $t_0 = (0, u)$. With it, the first job is visited on day 1. The customer cost coefficient of job $i \in N$ is defined as follows:

$$c_i = \begin{cases} n - (2i - 1) & i < \frac{n}{2} \\ n - 2(n - i) & i > \frac{n}{2} \\ 0 & \text{otherwise.} \end{cases}$$

Consequently, for the customer cost coefficients it holds that $c_n > c_1 > c_{n-1} > c_2 > \cdots > c_{\lceil n/2 \rceil}$.

For this instance, the MEN heuristic applied with $f > \frac{n-1}{n}$ leads to the solution S_{MEN} with the route $\Pi_{\text{MEN}} = (n, 1, n-1, 2, n-2, 3, \ldots, \lceil \frac{n}{2} \rceil)$. This solution is illustrated in Figure 5.2 on the top. The travel costs of S_{MEN} are

$$\begin{aligned} g^d(S_{\text{MEN}}) &= \left((n-1) + \sum_{k=1}^{n-1} (n-k) + \left\lfloor \frac{n}{2} \right\rfloor \right) D \\ &= \left((n-1) + \frac{1}{2} n(n-1) + \left\lfloor \frac{n}{2} \right\rfloor \right) D \\ &\geq \frac{1}{2}(n^2 + n - 1)D \end{aligned}$$

and the customer costs are $g^c(S_{\text{MEN}}) = \sum_{i \in N} c_i$ independent from the route because all jobs are executed on the same day.

An optimal solution is $S^* = ((N, (1, 2, 3 \dots n)))$, also illustrated in Figure 5.2, with travel costs value $g^d(S^*) = (n-1)D$ and customer costs $g^c(S^*) = \sum_{i \in N} c_i$. Then, the approximation ratio of the MEN heuristic of this instance is

$$\frac{g(S_{\text{MEN}})}{g(S^*)} \geq \frac{(n^2 + n - 1)\,\frac{D}{2} + \sum_{i \in N} c_i}{(n-1)D + \sum_{i \in N} c_i} \in \Theta(n).$$

Thus, the approximation ratio increases with n, such that it exists no constant upper bound for it.

It should be noted that the proof so far applied only for $f > \frac{n-1}{n}$. In case of $f \leq \frac{n-1}{n}$, other customer cost coefficients $\tilde{c}_i,\ i \in N$, have to be used with $\tilde{c}_1 < f\tilde{c}_n$, e.g., $\tilde{c}_i = \phi^{-c_i}$ with $\phi < f$. □

5.1.3. Cost-Balanced Neighbor Heuristic

A third variant of the greedy heuristic is developed to consider both cost parts of the objective function together. Based on the customer cost coefficient and the travel costs, a balanced cost value is calculated. The job with the smallest balanced costs is appended to the corresponding route. This variant is called cost-balanced neighbor heuristic (CBN).

To compute the balanced costs, an estimation for the number of jobs that can be visited by one vehicle in one day is necessary, which is given by the quotient of the time available per day and the average time needed for one job:

$$n_{\text{day}} := \frac{u + \frac{1}{n}\sum_{i \in N} \min\{r_{ij} | \, j \in N,\ j \neq i\}}{\frac{1}{n}\sum_{i \in N}(a_i + \min\{r_{ij} | \, j \in N,\ j \neq i\})}. \tag{5.1}$$

The time available per day is estimated by the length of the working shift u plus the average travel time to the nearest neighbor; and the average time per job $i \in N$ is estimated by the average of the working duration a_i plus the travel time to the nearest neighbor. Then, the balances costs of a job-route-pair are computed as

$$c_b(j, k) = \beta d_{l_k j} - (1 - \beta)\frac{\frac{n}{m} - |N_k|}{n_{\text{day}}} c_j \tag{5.2}$$

with β is a factor to give more weight on travel costs or on customer costs. This cost evaluation consists of the travel costs $d_{l_k j}$ and an estimation for the saving in customer costs. Because the saving should be maximized, it is multiplied by minus one. To calculate the saving in customer costs, it is assumed that a job is visited at day $\frac{n}{m\, n_{\text{day}}}$, if it is not appended in the current iteration. Note that this is an estimation for the number of days needed to visit all jobs assuming near neighbors are scheduled consecutive and the jobs are equally distributed among all vehicles. In this case, the customer cost value of a job j equals to $\frac{n}{m\, n_{\text{day}}} c_j$. Otherwise, if job

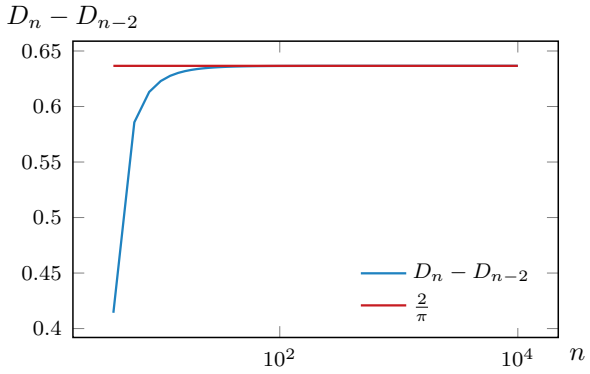


Figure 5.3.: Increment of the length of the largest diagonal in n-sided regular polygons with side length $D = 1$ and n even.

j is appended to route k in this iteration, the customer cost value is assumed to be $\frac{|N_k|}{n_{\text{day}}} c_j$. This estimation is used instead of the real start day to have the same basis as in the estimation of the customer cost value that occurs when j is not appended now. The difference between both are the saving in customer cost. Note that if route k already contains more than $\frac{n}{m}$ jobs, the savings are negative which means it could be better to add job j to a shorter route.

In more detail the CBN heuristic works as follows: Given is a factor $\beta \in [0, 1]$, which is used to have more weight on travel costs (β near 1) or on customer costs (β near 0). The algorithm is initialized by an empty schedule $\widetilde{S} := (\emptyset, ())_{k \in M}$ and the set of unplanned jobs $\widetilde{N}$ contains all jobs. While the set of unplanned jobs is not empty, in each step the best job-route-pair is determined from all feasible job-route-pairs (j, k) with $j \in \widetilde{N}$ and $k \in M$. This consist of three substeps:

1. If no feasible job-route-pair exists, return the uncompleted schedule. In this case, the CBN heuristic was not successful.

2. Compute for each feasible job-route-pair (j, k) the balanced costs $c_b(j, k)$ by equation (5.2).

3. Select the job-route-pair with minimal balanced cost value and append the job to the corresponding route.

Finally, if the CBN heuristic was successful, the schedule is checked for empty routes which have to be filled applying the FILL algorithm. The pseudocode of the CBN heuristic is given by Algorithm 4 in Appendix B.

Despite the fact that the CBN heuristic takes travel and customer costs into account, an instance can be designed which shows that the approximation ratio of the CBN heuristic is also unbounded. This instance is based on an n-sided regular polygon with edge length D. The travel costs and customer cost coefficients are chosen such that the CBN heuristic selects jobs in an order that leads to large travel costs and times. For this purpose, the distances in such a regular polygon are

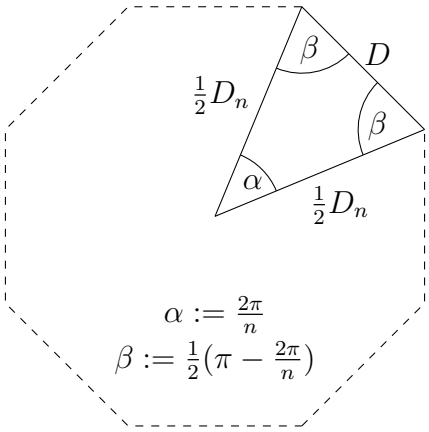


Figure 5.4.: Triangle to calculate the length D of the largest diagonal in an n-sided regular polygon for $n \in \mathbb{N}$ even.

analyzed. For even $n \in \mathbb{N}$, let D_n be the length of the largest diagonal in an n-sided regular polygon with edge length D. As shown in the proof of Lemma 5.3 below, D_n and the increment $D_n - D_{n-2}$ increases with n, see also Figure 5.3.

Lemma 5.3. *For any even numbers n, $n_0 \in \mathbb{N}$ with $n > n_0 \geq 4$, it holds for the largest diagonal length D_n in an n-sided regular polygon with edge length D that $D_n > D_{n_0} + (n - n_0)d_{n_0}$ with $d_{n_0} := \frac{1}{2}(D_{n_0} - D_{n_0-2})$.*

Proof. It will be shown firstly that $D_n - D_{n-2}$ increases with n. Based on this, the assumption of the lemma will be proved.

The length of the largest diagonal D_n can be calculated with the triangle illustrated in Figure 5.4. It is formed by two consecutive points of the polygon and the midpoint of the polygon. The angle at the polygon center point is $\frac{2\pi}{n}$, which results from dividing the full circle into n equal angles. The two angles at the polygon points are both $\frac{1}{2}(\pi - \frac{2\pi}{n})$, because the triangle is isosceles. The length of the basis and the two legs amounts to D and $\frac{1}{2}D_n$, respectively. With the law of sines, it turns out that

$$\frac{D}{\sin(\frac{2\pi}{n})} = \frac{D_n}{2\sin(\frac{\pi}{2} - \frac{\pi}{n})}.$$

Applying some trigonometric functions and solving the equation for D_n leads to

$$D_n = \frac{D}{\sin\frac{\pi}{n}} = D\csc\frac{\pi}{n}.$$

To show that the difference $D_{n+2} - D_n$ increases with n, the two differences $D_n - D_{n-2}$ and $D_{n+2} - D_n$ are compared. For this purpose, a series expansion of the cosecant for $0 < x < \pi$ is used, see [92],

$$\csc x = \sum_{k=0}^{\infty} \frac{(-1)^{k+1}2(2^{2k-1} - 1)B_{2k}}{(2k)!} x^{2k-1}$$

with B_{2k} is the $2k$-th Bernoulli number. To show that $D_n - D_{n-2} \overset{!}{<} D_{n+2} - D_n$, it has to be proven that

$$\begin{aligned} 0 &\overset{!}{<} D_{n+2} - 2D_n + D_{n-2} \\ &= D\csc\frac{\pi}{n+2} - 2D\csc\frac{\pi}{n} + D\csc\frac{\pi}{n-2} \\ &= D\sum_{k=0}^{\infty}\frac{(-1)^{k+1}2(2^{2k-1}-1)B_{2k}}{(2k)!}\left[\left(\frac{\pi}{n+2}\right)^{2k-1} - 2\left(\frac{\pi}{n}\right)^{2k-1} + \left(\frac{\pi}{n-2}\right)^{2k-1}\right] \end{aligned} \tag{5.3}$$

In the following, the obtained sum will be analyzed. For $k = 0$, it follows that $\frac{1}{\pi}(n+2-2n+n-2) = 0$. To investigate the remaining summands with $k \geq 1$, the Bernoulli numbers are replaced by a formula using the zeta function, see [5],

$$B_{2k} = \frac{(-1)^{k+1}2(2k)!}{(2\pi)^{2k}}\zeta(2k),$$

taking into account that

$$\zeta(s) = \sum_{j}^{\infty}\frac{1}{j^s}.$$

Then, for the first factor of each summand it is

$$\frac{(-1)^{k+1}2(2^{2k-1}-1)B_{2k}}{(2k)!} = \frac{4(2^{2k-1}-1)}{(2\pi)^{2k}}\sum_{j=1}^{\infty}\frac{1}{j^{2k}} > 0.$$

To obtain inequality (5.3), it remains to show that the second factor of each summand is not less than zero:

$$\begin{aligned} 0 &\overset{!}{<} \left(\frac{\pi}{n+2}\right)^k - 2\left(\frac{\pi}{n}\right)^k + \left(\frac{\pi}{n-2}\right)^k \\ &= \pi^k\frac{n^k(n-2)^k + n^k(n+2)^k - 2(n-2)^k(n+2)^k}{n^k(n-2)^k(n+2)^k} \\ &= \pi^k\frac{(n^2-2n)^k + (n^2+2n)^k - 2(n^2-4)^k}{n^k(n-2)^k(n+2)^k}. \end{aligned}$$

Obviously, for $n > 4$ the denominator is positive. For the numerator of the fraction,

the binomial formula yields

$$
\begin{aligned}
&(n^2-2n)^k + (n^2+2n)^k - 2(n^2-4)^k \\
&\quad = \sum_{i=0}^{k} \binom{k}{i} n^{2(k-i)}(-2n)^i + \sum_{i=0}^{k} \binom{k}{i} n^{2(k-i)}(2n)^i - 2\sum_{i=0}^{k} \binom{k}{i} n^{2(k-i)}(-4)^i \\
&\quad = 2\sum_{i=0}^{\lfloor \frac{k}{2} \rfloor} \binom{k}{2i} n^{2(k-2i)}(2n)^{2i} - 2\sum_{i=0}^{\lfloor \frac{k}{2} \rfloor} \binom{k}{2i} n^{2(k-2i)}4^{2i} + 2\sum_{i=0}^{\lfloor \frac{k-1}{2} \rfloor} \binom{k}{2i+1} n^{2(k-2i-1)}4^{2i+1} \\
&\quad = 2\sum_{i=0}^{\lfloor \frac{k}{2} \rfloor} \binom{k}{2i} n^{2(k-2i)}[(2n)^{2i} - 4^{2i}] + 2\sum_{i=0}^{\lfloor \frac{k-1}{2} \rfloor} \binom{k}{2i+1} n^{2(k-2i-1)}4^{2i+1} \\
&\quad > 0.
\end{aligned}
$$

Thus, it is proven that $D_{n+2} - D_n > D_n - D_{n-2}$ for even $n > 4$. Further, with $d_0 = \frac{1}{2}(D_{n_0} - D_{n_0-2})$, it follows for $n > n_0$ that

$$D_{n+2} - D_n > 2d_{n_0}.$$

Secondly, it is shown that $D_n > D_{n_0} + (n - n_0)d_{n_0}$ is valid. Since n and n_0 are even and $n > n_0 \geq 4$, it holds that

$$
\begin{aligned}
D_n &= D_{n_0} + \sum_{k=1}^{\frac{1}{2}(n-n_0)} \left(D_{n_0+2k} - D_{n_0+2(k-1)}\right) \\
&> D_{n_0} + \sum_{k=1}^{\frac{1}{2}(n-n_0)} 2d_{n_0} \\
&= D_{n_0} + (n - n_0)d_{n_0}.
\end{aligned}
$$

□

With the proof that the linear function $D_{n_0} + (n - n_0)d_{n_0}$ is a lower bound for the length of the largest diagonal in a regular polygon with n edges of length D, the approximation ratio of the CBN heuristic is investigated.

Theorem 5.4. *For the CBN heuristic, the approximation ratio is unbounded.*

Proof. For the proof, the following instance is analyzed: Let n be even. Assuming n jobs are arranged in a regular polygon with n edges of length $D \in \mathbb{N}$. The travel costs between the jobs are equal to the rounded-up Euclidean distances. Then, the travel costs to the nearest neighbor are always D and the travel costs to the diagonally opposite job amount to $\lceil D_n \rceil$. The travel time is the Euclidean distance

multiplied with factor f_t and rounded up. The jobs are numbered clockwise from 1 to n. The customer cost coefficients are set to

$$c_i := \begin{cases} (n-2i)B & \text{if } i \leq \frac{n}{2}, \\ 2(n-i)B & \text{if } i > \frac{n}{2}. \end{cases}$$

Consequently, always the two opposite jobs have the same customer cost coefficient and its maximum is $B(n-2)$. All jobs have to be served by one vehicle and its depots are located on the same position as job 1 which is one of the two most-expensive jobs. The start time of the depots is defined by $t_0 = (0, 0)$. The working duration of each job is a. The number of jobs per day is calculated as $n_{\text{day}} = \frac{u+\lceil f_t D \rceil}{a+\lceil f_t D \rceil}$. Consequently, it is independent from the number of jobs.

In this instance, the CBN heuristic proceeds as follows:

1. Starting in the depot, job 1 is the nearest job and also one of the two most-expensive jobs. Consequently, its balanced costs are minimal and job 1 is appended at first to the route.

2. With route $\Pi_{\text{CBN}} = (1)$, mainly two jobs have to be considered:

   - Job 2 is the nearest job.
     Its balanced costs are $c_b(1, 1) = \beta D - (1-\beta)\frac{n-1}{n_{\text{day}}} B(n-4)$.
   - Job $\frac{n}{2}+1$ is the most-expensive unplanned job and is located at the opposite of job 1.
     Its balanced costs are $c_b(\frac{n}{2}+1, 1) = \beta D_n - (1-\beta)\frac{n-1}{n_{\text{day}}} B(n-2)$.

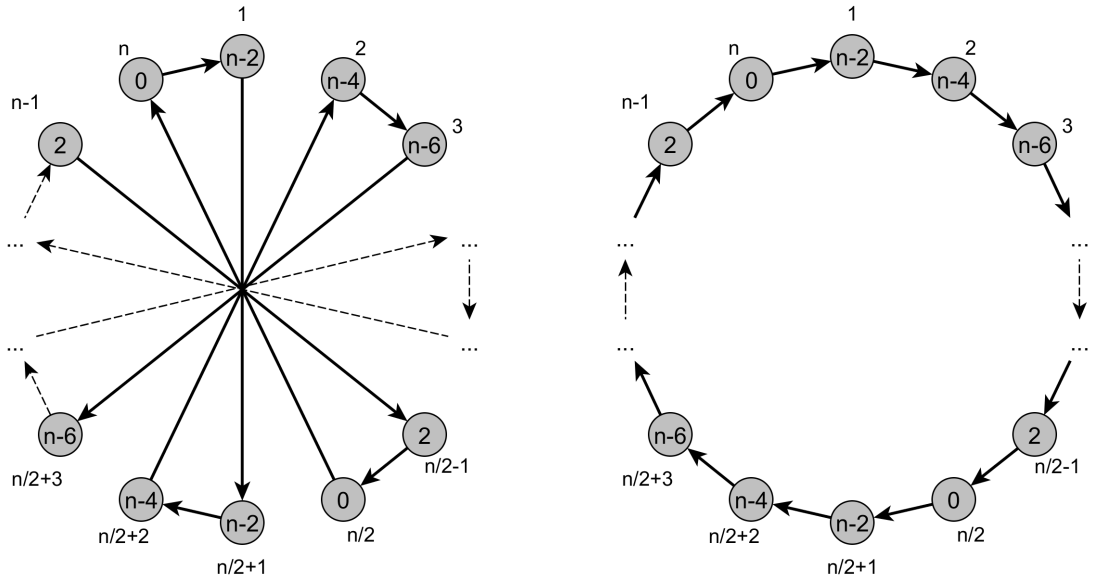
   If $B > \frac{\beta}{1-\beta}\frac{n_{\text{day}}}{2(n-1)}(D_n - D)$, the smallest balanced costs are $c_b(\frac{n}{2}+1, 1)$ and the opposite job $\frac{n}{2}+1$ is selected and appended to the route. Then, it is $\Pi_{\text{CBN}} = (1, \frac{n}{2}+1)$.

3. With $\frac{n}{2}+1$ as current last job of the route, $\frac{n}{2}+2$ is the nearest job and also one of the two most-expensive jobs. Because of that, its balanced costs are minimal and $\frac{n}{2}+2$ is appended to route 1.

4. With route $\Pi_{\text{CBN}} = (1, \frac{n}{2}+1, \frac{n}{2}+2)$, again two jobs have to be considered:

   - Job $\frac{n}{2}+3$ is the nearest job.
     Its balanced costs are $c_b(\frac{n}{2}+3, 1) = \beta D - (1-\beta)\frac{n-3}{n_{\text{day}}} B(n-6)$.
   - Job 2 is the most-expensive unplanned job and located at the opposite of job $\frac{n}{2}+1$.
     Its balanced costs are $c_b(2, 1) = \beta D_n - (1-\beta)\frac{n-3}{n_{\text{day}}} B(n-4)$.

   If $B > \frac{\beta}{1-\beta}\frac{n_{\text{day}}}{2(n-3)}(D_n - D)$, again the opposite job is selected and appended to the route. Then, it is $\Pi_{\text{CBN}} = (1, \frac{n}{2}+1, \frac{n}{2}+2, 2)$.



(a) solution of CBN heuristic

(b) solution of NN solution

Figure 5.5.: Example of an instance where the cost-balanced neighbor heuristic has an unbounded approximation ratio.

These steps are repeated until the set of unplanned jobs is empty. In case of

$$B > \frac{\beta}{1-\beta}\frac{n_{\text{day}}}{2}(D_n - D),$$

always the nearest job and the opposite job are appended alternately. Consequently, if n can be divided by 4, the CBN heuristic leads to the route

$$\Pi_{\text{CBN}} = (1, \tfrac{n}{2}+1, \tfrac{n}{2}+2, 2, 3, \ldots, \tfrac{n}{2}+3, \ldots, n, \tfrac{n}{2})$$

and otherwise to

$$\Pi_{\text{CBN}} = (1, \tfrac{n}{2}+1, \tfrac{n}{2}+2, 2, 3, \ldots, \tfrac{n}{2}+3, \ldots, \tfrac{n}{2}, n).$$

The CBN heuristic solution S_{CBN} of this instance is shown in Figure 5.5 on the left.

In the following, the costs of the solution S_{CBN} are estimated. A lower bound for the days needed to travel to the opposite job is $T_n := \left\lfloor \frac{f_t D_n}{h} \right\rfloor$. Then, job 1 is executed on the start day, which is day 0. The next two jobs, which are $\frac{n}{2}+1$ and $\frac{n}{2}+2$, are executed not before day T_n. The jobs 2 and 3, visited after $\frac{n}{2}+2$, are done not before day $2T_n$, and the next two jobs not before day $3T_n$. Continuing this observation leads to $T_n \left\lfloor \frac{i}{2} \right\rfloor$ is a lower bound for the start day of the job π_i. Note, job π_i of the route has the customer cost coefficient $c_{\pi_i} = B(n - 2\lceil \frac{i}{2} \rceil)$. Thus, for the customer costs of the obtained solution S_{CBN}, it holds that

$$
\begin{aligned}
g^c(S_{\mathrm{CBN}}) &\geq \sum_{i=1}^{n} T_n \left\lfloor \tfrac{i}{2} \right\rfloor B\left(n - 2\left\lceil \tfrac{i}{2} \right\rceil\right) \\
&= B\left[0(n-2) + T_n(n-2) + T_n(n-4) + 2T_n(n-4) + 2T_n(n-6) + \cdots\right] \\
&> B\left[0(n-2) + 0(n-2) + T_n(n-4) + T_n(n-4) + 2T_n(n-6) + \cdots\right] \\
&= \sum_{i=1}^{n/2} 2T_n(i-1)B(n-2i) = \sum_{i=1}^{n/2} 2\left\lfloor \frac{f_t D_n}{h} \right\rfloor (i-1)B(n-2i).
\end{aligned}
$$

As shown in Lemma 5.3, $D_n > D_{n_0} + (n - n_0)d_{n_0}$ for any $n > n_0 \geq 4$. Replacing D_n by this approximation and utilizing $\lfloor x \rfloor > x - 1$, leads to

$$
g^c(S_{\mathrm{CBN}}) > \sum_{i=1}^{n/2} 2\left(\frac{f_t}{h}(D_{n_0} + (n - n_0)d_{n_0}) - 1\right)(i-1)B(n-2i) \in \Theta(n^4).
$$

The travel costs can be estimated as

$$
g^d(S_{\mathrm{CBN}}) = \frac{n}{2}(D + \lceil D_n \rceil) > \frac{n}{2}(D + D_{n_0} + (n - n_0)d_{n_0}) \in \Theta(n^2)
$$

Consequently, $g(S_{\mathrm{CBN}}) = g^c(S_{\mathrm{CBN}}) + g^d(S_{\mathrm{CBN}}) \in \Theta(n^4)$.

In contrast, the NN heuristic leads to the solution $S_{\mathrm{NN}} = ((N, (1, 2, 3, \ldots, n-1, n)))$, shown in Figure 5.5 on the right. The travel costs are nD. To calculate the customer costs of S_{NN}, the start day of each job has to be determined: Starting with day 0, $\eta = \lfloor n_{\mathrm{day}} \rfloor = \left\lfloor \frac{u + \lceil f_t D \rceil}{a + \lceil f_t D \rceil} \right\rfloor$ jobs are executed everyday. Thus, the i-th job is done on day $\left\lceil \frac{i}{\eta} \right\rceil - 1$. Taking into account that the jobs on route position i and $i + \frac{n}{2}$, with $i \leq \frac{n}{2}$, have the same customer cost coefficient $B(n-2i)$, the costs of the NN solution can be expressed as follows:

$$
\begin{aligned}
g(S_{\mathrm{NN}}) &= \sum_{i=1}^{n/2} \left(\left\lceil \frac{i}{\eta} \right\rceil + \left\lceil \frac{i + \frac{n}{2}}{\eta} \right\rceil - 2\right) B(n-2i) + nD \\
&< \frac{B}{\eta} \sum_{i=1}^{n/2} (2i + \frac{n}{2})(n-2i) + nD \\
&\in \mathcal{O}(n^3)
\end{aligned}
$$

because $\lceil x \rceil < x + 1$.

To determine the approximation ration of the CBN heuristic, the costs of the NN heuristic solution are used. Since the optimal solution has smaller or equal costs as S_{NN}, it is

$$
\frac{g(S_{\mathrm{CBN}})}{g(S^*)} \geq \frac{g(S_{\mathrm{CBN}})}{g(S_{\mathrm{NN}})} \in \Theta(n),
$$

because the costs of the CBN heuristic are in order of $\Theta(n^4)$ and the costs of the NN heuristic are in order of $\Theta(n^3)$. Consequently, there exists no constant bound for the approximation ratio of the CBN heuristic. □

5.1.4. Best-of-Greedy Algorithm

In the Sections 5.1.1 to 5.1.3, three different greedy heuristics were presented that append one job after the other to a route regarding different selection criteria. Each of these criteria offers the possibility to adjust it by a parameter. Dependent on the customer cost coefficients of the jobs and the costs to travel from job to job, the solution quality resulting from different values for the parameters varies, which can be seen in the results of computational experiments provided in Section 5.4.1. To be able to solve a wide range of different instances, all greedy heuristics can be combined to one best-of-greedy heuristic by selecting the solution with minimal costs from a set of solutions obtained with the NN, MEN and CBN heuristic and different parameters. This approach is called best-of-greedy algorithm (BoG) and shown in Algorithm 5 in Appendix B. Note, an uncompleted solution has an infinite cost value.

In detail, eleven solutions are generated applying the NN heuristic with factors $f \in [0, 0.1, \ldots, 1]$; ten solutions are computed by the MEN heuristic with factors $f \in [0.1, 0.2, \ldots, 1]$; and eleven solution are determined by the CBN heuristic with weights $\beta \in [0, 0.1, \ldots, 1]$. From these 32 solutions, the one with smallest total costs is returned. Because of that, the computational costs of the BoG algorithm are 32 times larger than of a single greedy heuristic.

Remark For all three basic heuristics, the approximation ratio is unbounded. It can be expected that also for the BoG algorithms no constant upper bound for the approximation ratio exists. However, the proof of this conjecture is still pending.

5.2. Rollout Algorithm

The rollout algorithm was introduced in [16] for combinatorial optimization problems. Inspired by dynamic programming, the algorithm is designed for problems that can be solved step-by-step taking in each step one decision. In the greedy heuristics described in Section 5.1, each decision is taken by looking for the locally best choice. In dynamic programming, in each step the decision is selected that minimizes the optimal cost-to-go function which gives for each decision the minimal costs needed to complete the partial solution. In many applications, this approach is not practicable because the optimal cost-to-go function cannot be computed in acceptable time. The idea of the rollout algorithm is to replace the optimal cost-to-go function by an approximation based on a heuristic $\mathcal{H}$ that is able to find a solution based on a partial solution. Then, in each step the decision is taken that leads to minimal costs by completing the partial solution with $\mathcal{H}$.

To ensure that the solution of the rollout algorithm cannot be worse than the solution of the used heuristic $\mathcal{H}$, the heuristic has to be sequentially consistent. This means, that a heuristic solution is always reproduced by the heuristic completion of its partial solutions, see [16] for further information.

The rollout algorithm for the VRPCC works as follows: Again, the solution is build up iteratively by appending jobs at the end of routes. The algorithm starts with an empty solution $\widetilde{S} := (\emptyset, ())_{k \in M}$ and the set of unplanned job $\widetilde{N}$ contains all jobs. Then, in each step the best feasible job-route-pair is searched which consists of three substeps:

1. If no feasible job-route-pair exists, return the uncompleted schedule.
2. Analyze each feasible job-route-pair (j, k), with $j \in \widetilde{N}$ and $k \in M$. For this purpose,
   2.1. append j to route k.
   2.2. Complete the partial solution by one of the presented greedy heuristics.
   2.3. Compute and store the approximated cost-to-go value which equals to the total costs of the obtained solution.
   2.4. Remove job j and all before unplanned jobs from the schedule.
3. Select the job-route-pair with minimal approximated cost-to-go value and append the job to the corresponding route.

These steps are repeated until all jobs are added to a route. The computational complexity is $\mathcal{O}(n^4m^2)$: The algorithm takes n steps to append all jobs to a route. In each step, at most nm possible job-route-pairs have to be analyzed. And to analyze one job-route-pair, a greedy heuristic with an effort of $\mathcal{O}(n^2m)$ is used. Note, the obtained solution can contain empty routes. Thus, finally empty routes are removed by the FILL algorithm. The pseudocode of the rollout algorithm is given in Appendix B, Algorithm 6.

To ensure that the NN, MEN and CBN heuristic are sequentially consistent, the job and route with smallest indices are chosen if several job-route-pairs were best. As mentioned in [16], also a set of heuristics can be the base of the approximated cost-to-go function. Thus, also the BoG algorithm, which is the best solution from solving VRPCC with different greedy heuristics, can be used.

To reduce the calculation effort, it is possible to reduce the number of analyzed job-route-pairs or the number of decisions taken by the heuristic to evaluate a job-route-pair. The first-mentioned variant is restricting the width of the search tree by a value $w \in [1, n]$. Then, in every step a candidate list with at most w jobs is created and only these jobs are analyzed. To determine good candidates, similar selection criteria as in the greedy heuristics can be used:

- When the candidates are selected by the *nearest neighbor criteria*, the unplanned jobs are arranged in increasing order with regard to the smallest travel costs to the current last job of any route. The first w jobs of this order are the candidates.
- When the candidates are selected by the *most-expensive neighbor criteria*, the w jobs with highest customer cost coefficient are the candidates. If less than

w jobs are afflicted with a non-zero customer cost coefficient, the remaining jobs are selected by the nearest neighbor criteria.

- When the candidates are selected by the *cost-balanced neighbor criteria*, the unplanned jobs are arranged in increasing order with regard to the minimal cost evaluation for any route. The first w jobs of this order are the candidates.

With a restriction of the search width, the computational complexity is decreased to $\mathcal{O}(n^3m^2w)$. However, the finally obtained solution $\widetilde{S}$ can be worse than the heuristic solution because the choice of the selection criteria used by the heuristic can be missing in the candidate list. To compensate this, the algorithm should store the best solution of all solutions generated in substep 2.3. during the solution process and finally return either the computed solution, or, if better, the best of all produced solutions.

With the second-mentioned variant to reduce the calculation effort, the depth of the search tree is restricted. Then, for each analyzed job-route-pair at most d jobs are added with the selected greedy heuristic. The approximated cost-to-go function is the sum of the costs of the uncompleted solution obtained by adding d jobs with the greedy heuristic and an estimation of the costs for the unplanned jobs which is computed by

$$c_{est}(\widetilde{N}) = \sum_{i \in \widetilde{N}} t^d_{est} c_i \; + \min_{j \in \widetilde{N},\; i \neq j} r_{ij},$$

where $t^d_{est} = \frac{1}{m}(\frac{|\widetilde{N}|}{n_{\text{day}}} + \sum_{k \in M} t^d_{l_k})$ is an estimation for the average start day of the unplanned jobs with n_{day} as defined for the CBN heuristic in equation (5.1). This approach leads to a decrease in the computational complexity to $\mathcal{O}(n^3m^2d)$: n steps are necessary to append all jobs. For each step, at most nm job-route-pairs are analyzed. To evaluate one job-route-pair, the computational complexity is reduced to $\mathcal{O}(dnm)$. Not that for $d < n$, the rollout algorithm can produce a solution with higher costs than the solution of the base heuristic. This drawback cannot be compensated by returning the best solution determined during the solution process. From the step, where a completed solution was obtained via the limited base heuristic, the solution can only be improved in the further steps because all unplanned jobs are analyzed. Thus, the last obtained solution is not worse than a previously obtained completed solution.

Using both restrictions of the search tree leads to a decrease in the time complexity to $\mathcal{O}(n^2m^2dw)$: In each of the n steps, at most wm job-route-pairs are analyzed, and analyzing one job-route-pair has a complexity of $\mathcal{O}(dnm)$.

5.3. Local Search Algorithms

Computational experiments on benchmark `S` showed that there is a significant gap between heuristic solutions and optimal solutions, see Section 5.4. A common way to improve a heuristic solution is local search, see, e.g., [1], which means to iteratively

change an initial solution in order to find a better one. The set of all solutions that can be reached by one change is called neighborhood. The most common local search algorithms are first and best improvement [68], simulated annealing [85] and tabu search [61]. How to apply local search to VRPs can be found, e.g., in [1, 23].

For a variant of the VRPCC with a single machine, two different neighborhoods are compared in [70] applied to simulated annealing: the two-opt neighborhood known from solving TSPs as presented, e.g., in [128], where two edges are exchanged, and the m-move neighborhood where a part of the route is moved to another position. Thereby, it was found out that the two-opt neighborhood does not lead to good results because the part of the route between the exchanged edges is visited in reverse order. Consequently, the jobs that were previously visited first are visited last after the edge exchange which often leads to larger customer costs. The second neighborhood m-move led to better results. But the best solutions were obtained, when both neighborhoods were combined.

This section provides an approach for best and first improvement of a heuristic solution of the VRPCC. In contrast to simulated annealing, only neighbor solutions with smaller costs are accepted. The improvement algorithms work as follows:

1. Generate a start solution S.
2. Search for an improved solution in the neighborhood of solution S. Thereby, first improvement takes the first found neighbor with smaller objective value, and best improvement takes the neighbor that leads to the largest improvement.
3. If an improved neighbor was found, replace the current solution S by its neighbor and repeat from step 2.
4. If none of the neighbors has a smaller objective value, the algorithm terminates.

Note that the obtained solution is a local minimum with respect to the neighborhood. In Appendix B Algorithm 7, the pseudocode for first and best improvement is given.

To apply the improvement algorithms, a neighborhood of a feasible solution has to be defined. For this purpose, three basic operations to modify the current solution are used:

- Shift one job to another position in the same or another route,
- switch two jobs, and
- move a part of a route to another route similar to [70].

To analyze, whether a certain neighbor leads to an improvement, its costs have to be determined. This can be achieved by generating the neighbor solution and computing the start times as well as the total costs. But, to speed up the algorithm, the change in the costs can also be approximated without generating the neighbor. Then, the true costs of the neighbor are only computed, if the approximated cost

change has a negative value or a small positive value, which means an improvement is probable. Otherwise, the next neighbor is analyzed because an improvement is not expected.

To generate a start solution for the improvement algorithm, two variants are common: applying a heuristic and creating a random solution. For the first case, it was decided to compute the start solution by the BoG heuristic because with a good start solution, less improvement steps are necessary to reach a local minimum.

As noted before, the improvement algorithms cannot leave the reached local minimum. To be able to explore a larger area of the solution space, the improvement algorithm can be restarted several times. For each run, another start solution should be used. In case of a random start solution, in each run a new start solution can be generated to explore different areas of the solution space. And in case of a heuristic start solution, it was decided to randomly change the last obtained solution and restart the improvement algorithm with this changed solution.

5.4. Computational Results

This section provides a comparison of the presented heuristics in terms of solutions quality. The most computational experiments were done with benchmark `L` that consists of one hundred instances with one hundred jobs. More information can be found in Appendix A. To observe the behavior of the heuristics for different kinds of instances, benchmarks dominated by travel costs or customer costs are defined by multiplying the travel costs of the edges or the customer cost coefficients of the jobs with factor 10 or 100. Therewith, five kinds of benchmark `L` are on hand for the experiments. Since the instances of benchmark `L` cannot be optimally solved, the heuristics are compared in terms of the percentage gap to the best obtained solution. For this purpose, let $\{S_1, S_2, \ldots, S_p\}$ be a set of solutions of one instance obtained by the heuristics $\{H_1, H_2, \ldots, H_p\}$. Then,

$$\text{gap}_{\{H_1,H_2,\ldots,H_p\}}(H_q) := \left(\frac{g(S_q)}{\min\{g(S_1), g(S_2), \ldots, g(S_p)\}} - 1 \right) \cdot 100\% \tag{5.4}$$

is the percentage gap of the solution obtained by heuristic variant H_q compared to the heuristics $\{H_1, H_2, \ldots, H_p\}$. Furthermore, let S' be the solution with smallest costs of the set $\{S_1, S_2, \ldots, S_p\}$.

Firstly, an overview of the main results of this section will be provided: In Section 5.4.1, the greedy heuristics are compared with the observation that all three greedy heuristics should be combined to the BoG heuristic to obtain good heuristic solutions. After that, in Section 5.4.2, several variants of the rollout algorithm are analyzed. It turns out, that the BoG heuristic was the best variant to compute the approximated cost-to-go function. Further, it is noticed that restricting the rollout width to the half of the jobs led to better solutions than non-restricting the rollout algorithm. Finally, in Section 5.4.3, first and best improvement are compared with

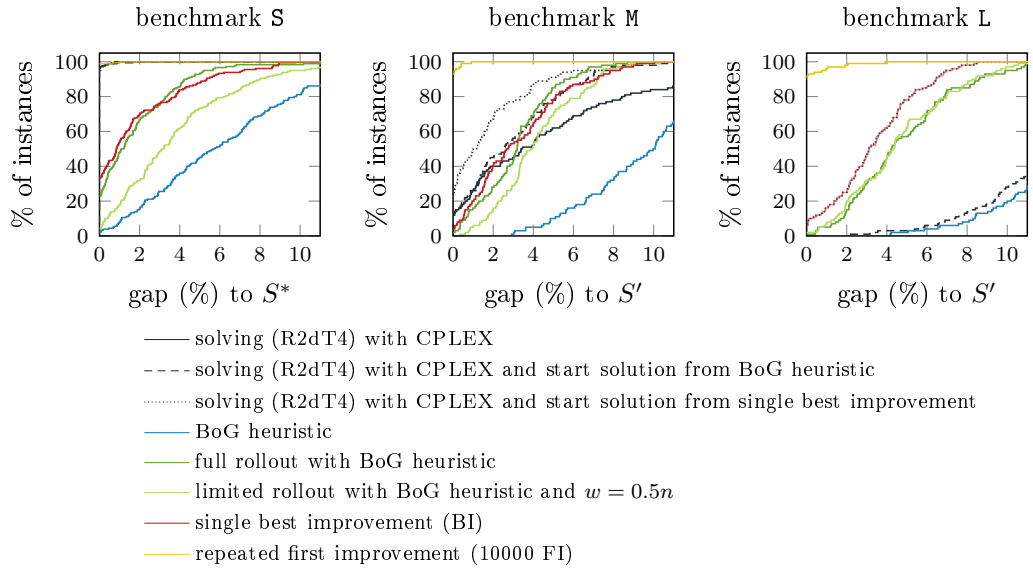


Figure 5.6.: Comparison of several heuristics and solving a MILP with CPLEX on three benchmarks.

the result that in case of a single run best improvement led to the best results but in case of several runs, first improvement was better.

To summary the results, Figure 5.6 shows a comparison of the solution quality of the BoG heuristic, the two best variants of the rollout algorithm, a single run of the best improvement algorithm and several runs of the first improvement algorithm. Furthermore, the results for applying CPLEX to (R2dT4) are analyzed initializing the solution process with different start solutions. For the comparison, not only benchmark `L` is used, but also the smaller benchmarks `S` and `M`. Note that for each instance of benchmark `S`, an optimal solution S^* is known. Consequently, for benchmark `S` the gap to the optimal value is given. For benchmark `M` and `L`, the heuristics are compared with each other: The percentage gap to S', which is the solution with smallest costs obtained by one of the analyzed heuristics, is computed by equation (5.4). For all solution methods and benchmarks, the computational time was restricted to ten minutes. The quartiles of the computational times are provided in Table 5.1.

In the left plot of Figure 5.6, the results on benchmark `S` are compared, which consists of 180 instances each with 15 jobs and two vehicles. There, the best solutions were obtained by the repeated first improvement algorithm, which is to apply at most 10000 times first improvement to the randomly changed last obtained solution. With this approach, for 178 of 180 instances an optimal solution was found. Note that with this local search algorithm optimality cannot be proven. Also solving (R2dT4) with CPLEX showed a good performance. In case of applying CPLEX without a start solution, for 174 instances, an optimal solution was found and for 134 instances, the optimality was proven. Initializing the solution process of CPLEX with a start

	benchmark S			benchmark M			benchmark L		
	Q1	Q2	Q3	Q1	Q2	Q3	Q1	Q2	Q3
CPLEX	11.0	70.4	600	600	600	600	600	600	600
BoG heuristic	<0.1	<0.1	<0.1	<0.1	<0.1	<0.1	<0.1	<0.1	<0.1
lim. rollout	<0.1	<0.1	<0.1	0.1	0.1	0.1	25.3	41.4	62.1
full rollout	<0.1	<0.1	<0.1	0.2	0.3	0.4	37.7	61.4	92.0
BI	<0.1	<0.1	<0.1	<0.1	<0.1	<0.1	10.6	12.7	15.6
10000 FI	2.3	2.8	3.4	30.4	35.8	42.4	600	600	600

Table 5.1.: Quartiles of computational times for several heuristics.

solution did not change the performance significantly. Comparing the computational times shows that 10000 times first improvement requires only a fraction of the time needed by CPLEX which can be gathered from the rows CPLEX and 10000 FI in Table 5.1. As expected, the worst solutions were obtained with the BoG heuristic. For half of the instances, the solutions obtained with the BoG heuristic were more than 5.6% more expensive than an optimal solution.

The results on benchmark M, which consists of 100 instances each with 30 jobs, are shown in the middle of Figure 5.6. Note that the gap is computed with respect to the best solution obtained with any of the presented solution approaches because optimal solutions are not known. For 93 instances, the best objective value was obtained by the repeated first improvement algorithm. Solving (R2dT4) with CPLEX did not lead to good solutions in benchmark M. In case of providing no start solution, for 12 instances a best solution was found, but for half of the instances the obtained solution was more than 3.4% more expensive as the best obtained solution and for ten percent of the instances, the gap was larger than 12.5%. Initializing the solution process with a start solution resulted in significant better results but nevertheless, solving (R2dT4) with CPLEX was clearly outperformed by the repeated first improvement algorithm because of the fact that with the latter, more cost-efficient schedules were computed and also less computational time was required. From the faster computed heuristics, which are the heuristics computed in less than a second as shown in Table 5.1, slightly better schedules were obtained with a single run of the best improvement algorithm and the full rollout algorithm than with the limited rollout algorithm: With these two heuristics, a best solution was found for 4 and 2 instances, respectively. For half of the instances, the gap exceeded 2.6% and 3.2%, respectively. And for each instance of benchmark M, the gap of the solution obtained by the best improvement algorithm and the full rollout algorithm to the best solution was smaller than 10%.

The right plot of Figure 5.6 shows the performance profiles for the gap to the best solution on benchmark L. Again, the solutions with smallest costs were mostly obtained with the repeated first improvement algorithm. Note, that in case of benchmark L not all 10000 improvement runs were made because of the time limit

of ten minutes. Applying one times best improvement to the BoG solution led to better solution than the rollout algorithms which can be seen on the performance profiles and was also faster computed as shown in Table 5.1. With the BoG heuristic, worse solutions were obtained: for 80 of 100 instances the gap to the best solution was larger than 10%. As expected, also CPLEX led not to good solutions: In case of not providing a start solution, the obtained solutions were even much worse than a heuristic solution and the smallest gap to a best solution was 58%. With providing the solution of the BoG heuristic as start solutions, the obtained solutions were slightly better than the start solution, which can be seen on the small distance between the both corresponding performance profiles in Figure 5.6. But in case of providing the solution of single best improvement as start solution, it was rarely improved by applying CPLEX for ten minutes which can be seen on almost identical performance profiles. The failing of applying CPLEX is not surprising because for such large instance, a computational time of ten minutes is short for an exact solver.

In summary, it is observed that the repeated first improvement algorithm outperforms the other presented heuristics. Solving a MILP formulation with the commercial solver CPLEX showed a good performance on benchmark `S`, where each instance contains only fifteen jobs. But for the two benchmarks `M` and `L`, where each instance contains 30 and 100 jobs, respectively, the solutions obtained in ten minutes were not better than an heuristic solution and also optimality was not reached.

5.4.1. Comparison of Greedy Heuristics

In this section, firstly, for each greedy heuristic, the influence of its parameter to the solution quality is analyzed. Secondly, the greedy heuristics are compared with each other based on instances dominated by travel costs and instances dominated by customer costs.

Nearest Neighbor Heuristic

For the NN heuristic, the influence of the factor f is analyzed, which gives the allowed deviation from the smallest travel cost when searching for a nearest neighbor with large customer cost coefficient. The NN heuristic was tested on the basic benchmark `L`, and on the benchmarks derived by increasing the travel costs by factor ten or hundred. An increase of the customer cost coefficients was not analyzed, because the NN heuristic is designed to minimize travel costs and with it, it is not suitable for instances with high customer costs. The factor f was varied between zero and one. Note that with $f = 0$ only feasible job-route-pairs with a nearest neighbor can be selected and with $f = 1$, it is allowed to append a job to a route with at most the double of the minimal travel costs of a feasible job-route-pair.

Figure 5.7 shows performance profiles of the percentage gap to the best obtained solution for the NN heuristic with different factors f on three different kinds of benchmark `L`. Note that only the solutions obtained with the NN heuristic applying any of the discussed factors is taken into account to compute the percentage gap.

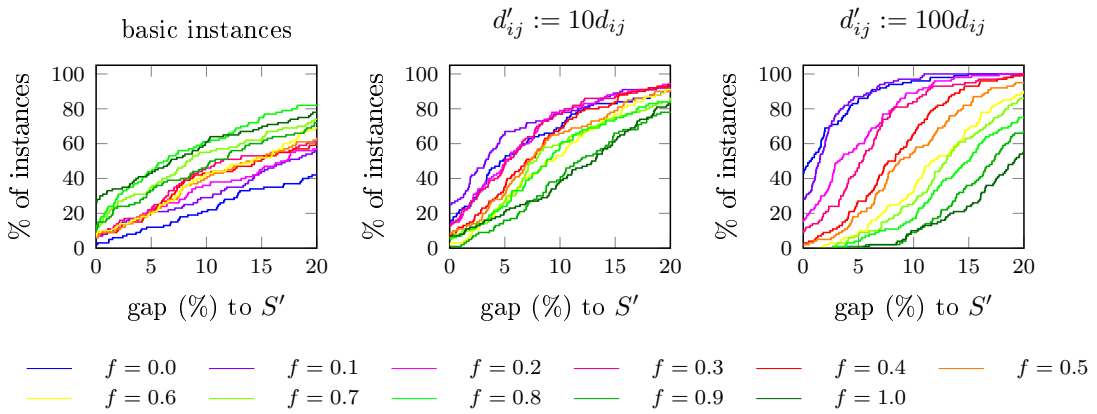


Figure 5.7.: Comparison of different factors f for the nearest neighbor heuristic on variants of benchmark `L`.

As it can be seen on the left plot, for the basic instances, larger factors show a better performance than smaller factors. For each factor, in some instances the gap is zero which means that with it the best NN heuristic solution was found. In contrast, for the instances with increased travel costs, smaller factors f resulted in significant better solutions. There, the best choice for factor f was 0.1, because with this value more often a small gap was achieved than with other factors.

To apply the NN heuristic as base heuristic in the rollout algorithm, a factor has to be selected. Because with $f = 0.1$, worse solutions were obtained in the basic instances, another factor is selected. The factor $f = 0.3$ seems to be a compromise and is used in further computational experiments with the NN heuristic.

Most-Expensive Neighbor Heuristic

The MEN heuristic was tested on benchmark `L` in its basic definition, and also with increased customer cost coefficients, once by factor ten and once by factor hundred. Different factors f were analyzed, from $f = 1$, which implies that always a job with highest customer cost coefficient is appended, to $f = 0.1$, which means that the next job is appended with a higher priority to travel costs and start day because all jobs, whose customer cost coefficient is not smaller than ten percent of the maximal customer cost coefficient of all unplanned jobs, are taken into account by choosing the best job-route-pair.

Figure 5.8 shows performance profiles of the percentage gap to the best obtained solution of the MEN heuristic with different factors f on three different kinds of benchmark `L`. In all three benchmark variants, f close to one led to worse solutions. This is because the jobs with high customer costs are rarely close together. Consequently, appending the jobs in the order of customer cost coefficient leads to much traveling which is associated with large travel times. But large travel times lead to a late execution of jobs which in turn results in high customer costs in the objective function. Thus, not only the travel costs of the solutions obtained with the MEN

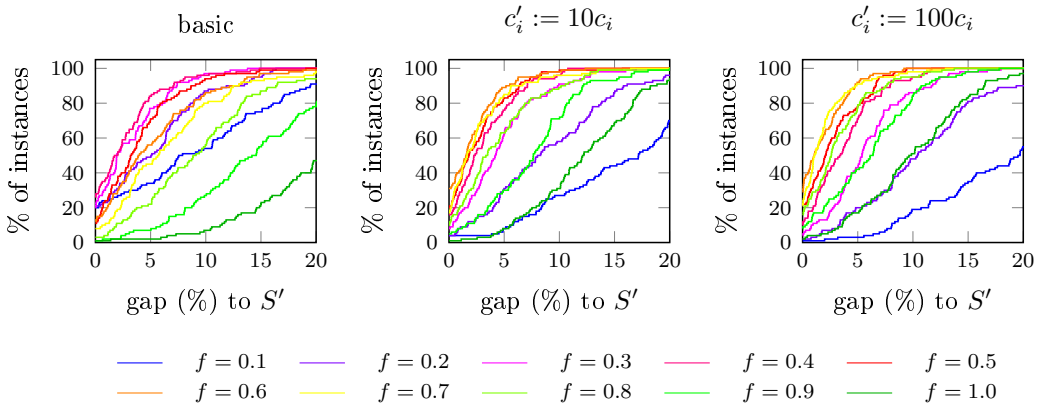


Figure 5.8.: Comparison of different factors f for the most-expensive neighbor heuristic on variants of benchmark L.

heuristic and factor $f = 1.0$ were large, but also the customer costs.

The left plot of Figure 5.8 shows that in the basic instances of benchmark L, smaller values of the factor f led to the best results. Good solutions were most often obtained applying factor $f = 0.4$ which can be seen on the fact that the corresponding profile is mostly to the left of the other profiles. Recap, in case of factor $f = 0.4$, all jobs with at least forty percent of the highest customer cost coefficient are considered by selecting the best job-route-pair. Consequently, the selection is more based on the cost evaluation than on the customer cost coefficients.

In the instances with increased customer cost coefficients, which are shown in the middle and in the right plot of Figure 5.8, better solutions were obtained with the values 0.6 and 0.7 for factor f. With these factors, more often a best solution was found, which can be seen on the high rate of instances with gap equal to zero. Furthermore, the performance profiles remain well above and to the left of that for the other factors. Due to factor $f = 0.6$ led also to good solutions in the basic benchmark variant, this heuristic is used with factor $f = 0.6$ in further computational experiments.

Cost-Balanced Neighbor Heuristic

The CBN heuristic was tested for different values of $\beta \in [0, 1]$ which is an algorithm parameter used to give more weight on travel costs or customer costs by selecting the best job-route-pair. Firstly, it is analyzed, how the travel costs and customer costs vary in dependence of β. Note that $\beta = 0$ leads to a focus on customer costs and that with $\beta = 1$ only travel costs are considered. After that, the gap to the best solution is analyzed for different values of the parameter β on benchmark L and its variants.

Figure 5.9 shows statistic values for the percentage gap of customer costs, travel costs and total costs with respect to the minimal value of a solution obtained with the CBN heuristic. Given are the minimum, 25-th percentile, median, 75th-percentile

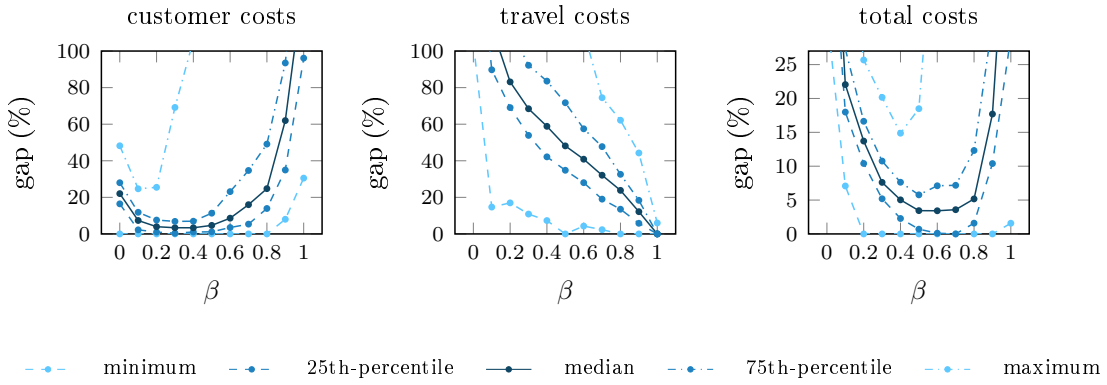


Figure 5.9.: Comparison of cost values for solutions obtained with the CBN heuristic applying different values for parameter β on benchmark L.

and maximum of the gap.

By comparing the customer cost value obtained with different values of the parameter β, which are shown in the left plot of Figure 5.9, it is observed that the smallest customer cost value was not obtained with $\beta = 0$ even with $\beta = 0$, only customer costs are taken into account by selecting the best job-route-pair and jobs with high customer cost coefficient are appended firstly to the routes. As already mentioned by analyzing the influence of factor f to the quality of a MEN heuristic, this leads not only to large travel costs but also to larger travel times which cause, due to the later execution of jobs, also high customer costs. Increasing the parameter β up to 0.4 resulted in smaller customer cost values. A further increasing led to higher customer costs because the focus is then more and more on minimizing travel costs than on minimizing customer costs.

Analyzing the obtained travel costs dependent on the chosen value for the parameter β, for which statistic values are provided in the middle plot of Figure 5.9, shows that, as expected, with $\beta = 1$ solutions with smallest travel costs were obtained. Interesting is the strong increase in travel costs with decreasing the parameter β. For $\beta = 0.5$ almost half of instances showed a gap larger than 50% to the solution with minimal travel costs as shown in the middle plot of Figure 5.9.

Finally, the total costs, which are the sum of travel costs and customer costs, are compared. As shown in the right plot of Figure 5.9, the smallest median of total costs were obtained when the parameter β was set to 0.5, 0.6 and 0.7. For larger values of the parameter β, the increase in customer costs was not compensated by the savings in travel costs. And for smaller values of the parameter β, the increase in travel costs was not compensated by the savings in customer costs or, for $\beta \leq 0.2$, travel and customer costs increased.

Figure 5.10 shows performance profiles for the percentage gap to the best obtained solution for the CBN heuristic with different values of the parameter β on three variants of benchmark L. In the basic variant of benchmark L, which is shown in the middle plot of Figure 5.10, several values of the parameter β led to good results,

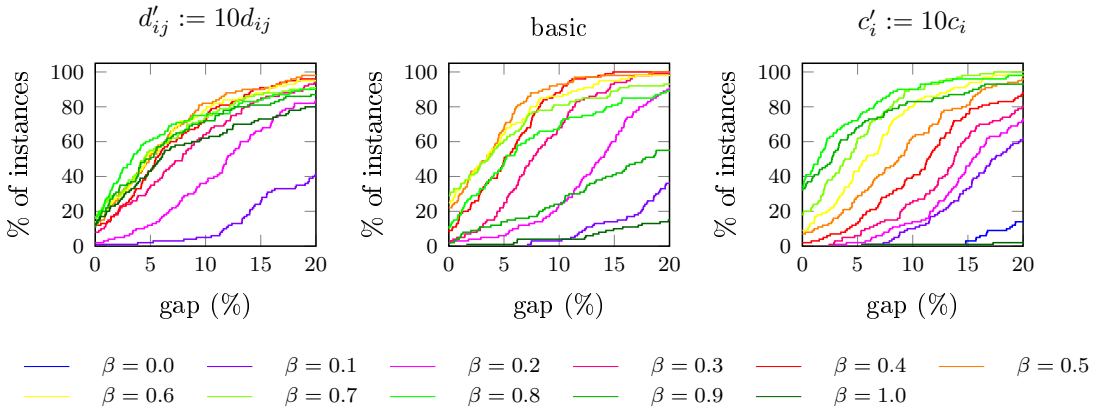


Figure 5.10.: Comparison of different parameters β for the cost-balanced neighbor heuristic on variants of benchmark `L`.

i.e., the values 0.5 and 0.6 showed the best performance in terms of solution quality. Applying $\beta = 0$ or $\beta = 1$ resulted in solutions with much higher costs. With it, the previously made observations are confirmed. For the instances with increased travel costs, shown in the left plot of Figure 5.10, applying small values for the parameter β, i.e., $\beta \leq 0.3$, resulted in solutions with higher costs because travel costs were not adequate considered by selecting the best job-route-pairs to build up the routes. However, also with $\beta = 1$, where only travel costs are taken into account by the selection of the best job-route-pair, the CBN heuristics produced solutions with higher costs. Finally, the right plot of Figure 5.10 shows the results for instances with increased customer cost coefficients. As it can be seen and as expected, $\beta = 0$ and $\beta = 1$ led not to good solutions. But interesting is that also in the benchmarks with increased customer costs, the best performance of the CBN heuristic in terms of solution quality was reached with the parameters $\beta = 0.7$ or $\beta = 0.8$ even these parameters lead to a higher focus on travel costs by the selection of the best job-route-pair.

In further computational experiments, the CBN heuristic was used with $\beta = 0.6$ because this parameter value showed a good performance in all three variants of benchmark `L`.

Best-of Greedy Heuristic

In this subsection, the NN, MEN and CBN heuristic are compared with each other and with the BoG algorithm. For the computational experiments, five variants of benchmark `L` were used: its basic variant, as defined in the Appendix A, and four variants obtained by multiplying the travel costs on the edges or the customer cost coefficients of the jobs with factor ten or hundred. The NN, MEN and CBN heuristic were applied with the parameter setting defined based on the previous computational experiments. In detail, the NN heuristic was processed with $f = 0.3$, the MEN heuristic with $f = 0.6$ and the CBN heuristic was applied with $\beta = 0.6$.

	factor travel costs			factor customer costs	
	100	10		10	100
NN heuristic	7.0%	11.7%	61.0%	121.7%	141.5%
MEN heuristic	55.1%	29.3%	6.5%	2.2%	1.8%
CBN heuristic	4.6%	7.6%	8.0%	12.7%	21.4%

Table 5.2.: Average gap to the BoG solution of solutions obtained by a greedy heuristic in different variants of benchmark `L`.

Due to the fact that the BoG algorithm returns the best solution obtained with various parameter setting of the NN, MEN and CBN heuristic inclusive the analyzed parameter settings, the costs of a solution obtained with the BoG heuristic S_{BoG} cannot be higher than the costs of a solution computed with the NN, MEN or CBN heuristic.

In Table 5.2, the greedy heuristics are compared in terms of the average percentage gap to the BoG solution S_{BoG}, which is computed by

$$\text{gap}_{BoG}(S) := \frac{g(S) - g(S_{BoG})}{g(S_{BoG})} 100\%. \tag{5.5}$$

As expected, with the NN heuristic applying factor $f = 0.3$, worse solutions were obtained for instances dominated by customer costs. But also for the basic variant of benchmark `L` and the two variants with increases travel costs, the solutions obtained with the NN heuristic had significantly higher costs than the solutions obtained with the other heuristics on average. Unsurprisingly, the MEN heuristic applying factor $f = 0.6$ showed a reverse trend: In the benchmark variants dominated by customer costs, the solutions computed with the MEN heuristic were very close to the solutions of the BoG algorithm. But on the benchmark variants dominated by travel costs, the MEN heuristic produced solutions with much higher costs than the BoG heuristics. This effect is due to the fact that also in case of small customer cost coefficients, jobs with non-zero customer cost coefficient are visited first which results in higher travel costs. The CBN heuristic applying parameter $\beta = 0.6$ showed the same trend as the NN heuristic: Even travel and customer costs are considered by selecting the best job-route-pair, in instances dominated by customer costs, the solutions obtained with the CBN heuristic were inferior. Consequently, applying one of the greedy heuristics is not suitable to obtain good heuristic solutions for a wide range of instances. Instead, the BoG algorithm should be preferred.

5.4.2. Rollout Algorithm

In this subsection, different variants of the rollout algorithm, which is presented in Section 5.2, are analyzed and compared. For a consistent analysis, again the percentage gap to the solution obtained with the BoG algorithm for the same instance

	factor travel costs			factor customer costs	
	100	10		10	100
R_{NN}	-5.0%	-6.2%	5.0%	17.3%	20.8%
R_{MEN}	18.7%	4.9%	-5.0%	-5.3%	-5.5%
R_{CBN}	-6.0%	-7.1%	-5.1%	0.1%	3.4%
R_{BoG}	-7.9%	-9.4%	-7.7%	-6.2%	-6.2%

Table 5.3.: Average gap to a BoG solution of solutions obtained with the rollout algorithm applying different greedy algorithms as base heuristic in different variants of benchmarks `L`.

is used as quality measurement. Note that a gap smaller zero is an improvement compared to the solution of the BoG heuristic.

At first, it is analyzed which of the presented greedy heuristics is suitable as base heuristic for the rollout algorithm. Subsequently, different variants to restrict the search tree by limiting its width or depth are analyzed.

For the rollout algorithm with applying different heuristics as base heuristic, Table 5.3 shows the average gap of the obtained solution to the solutions of the BoG heuristic. In detail, the analyzed rollout variants are

- R_{NN} that applies the NN heuristic with $f = 0.3$,
- R_{MEN} that applies the MEN heuristic with $f = 0.6$,
- R_{CBN} that applies the CBN heuristic with $\beta = 0.6$, and
- R_{BoG} that applies the BoG algorithm as base heuristic to evaluate the feasible job-route-pairs.

It can be seen that the rollout algorithm followed the trend of the applied base heuristics. The variants R_{NN} and R_{CBN} were not suitable to solve instances dominated by customer costs; and the rollout variant R_{MEN} produce worse solutions for instances dominated by travel costs. This can be seen on the fact that there the average gap to the BoG solution was greater than zero.

Applying the BoG algorithm as base heuristic, the rollout algorithm performed well in all kinds of benchmark. As expected due to the algorithm design, all solutions obtained by the rollout algorithm R_{BoG} were not worse than solution of the BoG heuristic. Comparing all entries in the row corresponding to R_{BoG} of Table 5.3 shows that in all benchmark variants, on average a significant improvement was reached. This is due to the fact that the BoG algorithm uses all three greedy heuristics to obtain a solution. Consequently, in instances dominated by customer costs a solution of the MEN heuristic can be returned as best and in instances dominated by travel costs a solution of the CBN or NN heuristic can be returned as best. Note that using

	factor travel costs			factor customer costs	
	100	10		10	100
nearest	-7.2%	-8.2%	-4.4%	-2.6%	-2.7%
most-expensive	-2.5%	-4.8%	-4.9%	-4.9%	-5.3%
cost-balanced	-7.2%	-8.6%	-6.4%	-5.2%	-5.2%
mixed	-6.9%	-8.2%	-5.6%	-4.2%	-4.5%

Table 5.4.: Average gap to a BoG solution of solutions obtained with the limited rollout algorithm applying different candidate selection variants.

the BoG algorithm as base heuristic results in 32 times the computational cost of using the NN, MEN or CBN heuristic.

The limited rollout algorithm

Next, it is analyzed how a limitation of the search tree of the rollout algorithm affects to the solution quality. At first, the rollout algorithm with a limited search width is investigated. In this variant of the rollout algorithm, not all possible job-route-pairs are taken into account by searching for the best job-route-pair, but only w job candidates are analyzed. In Table 5.4, different variants to get candidates are compared. The candidates are selected by

- nearest neighbors: the candidates are the w unplanned jobs with the smallest distance to the current last job of any route,
- most-expensive neighbors: the candidates are the w unplanned jobs with highest customer cost coefficient,
- cost-balanced neighbors: the candidates are the w unplanned jobs with minimal balanced costs considering all routes, and
- mixed neighbors: the candidates are the union of $\frac{w}{3}$ nearest neighbors, $\frac{w}{3}$ most-expensive neighbors and $\frac{w}{3}$ cost-balanced neighbors.

The rollout was used with $w = 10$, $d = 100$ and the BoG algorithm as base heuristic. As shown in Table 5.4, the nearest neighbor selection yielded a significant improvement in the instances dominated by travel costs compared to the solutions obtained with the BoG algorithm. In comparison to the results of the rollout algorithm R_{BoG} with $w = 100$, as shown in Table 5.3, the improvement is still high by a significant reduced computational effort. But for the basic variant and the variants with increased customer costs of benchmark `L`, the average improvement of the obtained solutions was considerably smaller than the average improvement of the full rollout algorithm. Using the most-expensive neighbors as candidates led to a moderate

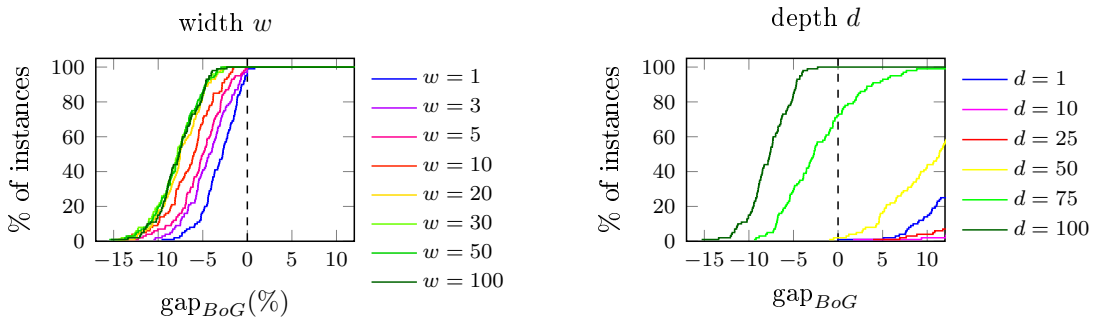


Figure 5.11.: Comparison of the rollout algorithm with varying width and depth on benchmark `L`.

average improvement in all benchmark variants beside the one with travel costs increased by factor hundred, which can be seen in Table 5.4. On average, the obtained solutions were 2.5% better than the BoG solution in the benchmark variant with hundredfold travel costs and about 5% better in the other four benchmark variants. Using cost-balanced or mixed neighbors as candidates led to a high average improvement in all kinds of benchmark `L` whereby slightly smaller improvements were observed with the mixed neighbors. Comparing the limited rollout with cost-balanced candidates with the full rollout algorithm (see Table 5.3, row R_{BoG}) shows that limiting the width to ten jobs resulted in a changed average gap to the solutions obtained with the BoG heuristic by about one percentage point to the worse by a drastically reduced computational effort. Consequently, next it is analyzed whether a limited search tree width can lead to a similar solution quality than the full rollout algorithm by reduced computational effort.

In Figure 5.11, the influence of width and depth on the solution quality is shown by means of performance profiles for the percentage gap to the solution of the BoG heuristic. In the left plot, the rollout algorithm was applied with a width $w \in \{1, 3, 5, 10, 20, 30, 50, 100\}$, depth $d = 100$ and the BoG algorithm as base heuristic. As candidates, the w jobs with smallest balanced costs are chosen. For width $w = 1$, where for the job with minimal balanced costs the best route to append this job is selected by the rollout approach, 95 instances were improved by up to 10%. In only one instance, the solution of the rollout algorithm was more worse than the solution of the BoG heuristic. Increasing the width up to $w = 20$ led to a significant improvement of the solution quality which can be seen in the left plot of Figure 5.11 where the corresponding performance profiles are clearly beside each other. A further increase of the width w did not lead to a significant better performance in terms of solution quality, which can be seen on the fact that the performance profiles overlap, but to a strong increase in the computational time, as shown in Figure 5.12. With width $w = 30$ and $w = 50$, a slightly better performance was observed than with width $w = 100$ and $w = 20$.

In the right plot of Figure 5.11, the results of varying the depth of the rollout algorithm are shown: The rollout algorithm was applied with with $w = 100$, a depth

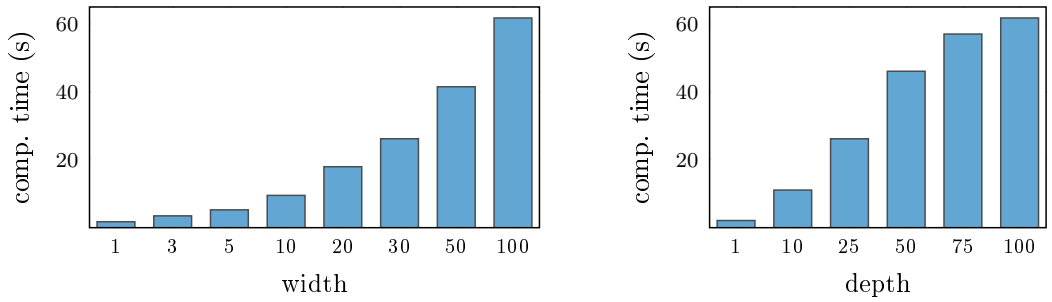


Figure 5.12.: Influence of the width and depth of the rollout algorithm on the computational time.

$d \in \{1, 10, 25, 50, 75, 100\}$ and the BoG algorithm as base heuristic. The evaluation of each decision was based on the costs of the (uncompleted) schedule and an estimation of the costs of the unplanned jobs. It turns out that limiting the depth led to worse solution. The costs of an uncompleted solution seems not to be suitable to evaluate a decision. For the depths 1, 10 and 25, the obtained solutions of the rollout algorithm were more expensive than the solution of the BoG heuristic, which can be seen on the fact that all gaps were larger than zero. Also with depth $d = 50$, for only two instances a better solution was found than with the base heuristic. Even with depth $d = 75$, in still only 72% of the instances a better solution was obtained with the limited rollout algorithm than with the BoG heuristic. Thus, limiting the search depth is not a suitable approach to reduce the computational time of the rollout algorithm because the obtained solutions were inferior.

In summary, for the VRPCC, the rollout algorithm led to better solutions than the greedy heuristics. Furthermore, the results showed that not all jobs have to be analyzed by selecting the best job-route-pair with the rollout approach, but that analyzing only a part of the jobs with small balanced costs resulted in solutions of similar quality by a significant reduced computational time.

5.4.3. Local Search Algorithms

This subsection shows, how much a solution of the BoG heuristic can be improved by the local search algorithms first improvement and best improvement as presented in Section 5.3. Again, the algorithms are compared in terms of the percentage gap to the solution obtained by the BoG heuristic, which is denoted by gap_{BoG}, as defined in equation (5.5).

Figure 5.13 gives performance profiles for the gap to the solution of the BoG heuristic of the algorithms first improvement (FI), best improvement with approximation (BIA) and a second variant of best improvement (BIE), where the cost of all neighbors are exactly computed. The local search procedures are initialized on the one hand by a heuristic solution (subscripted HEU), to be more precise by the solution of the BoG heuristic, and on the other hand by a random solution (subscripted RAND).

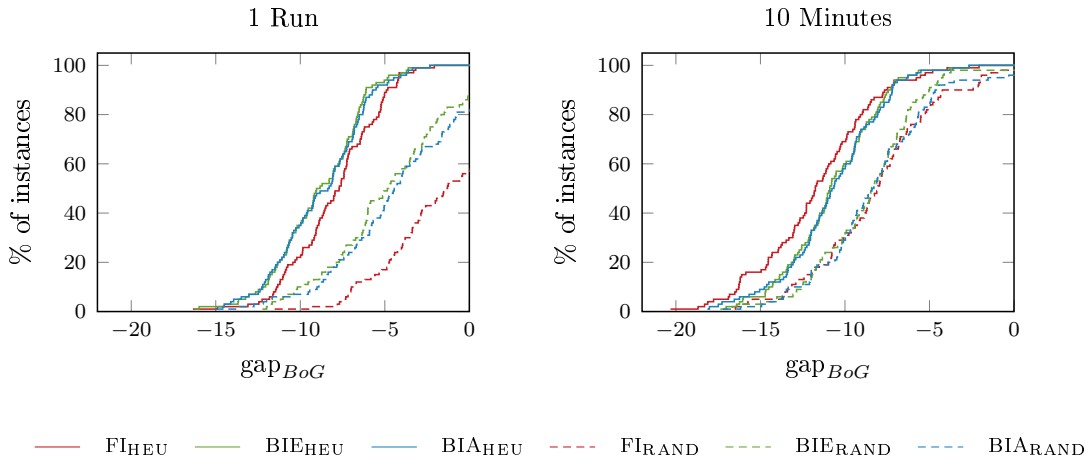


Figure 5.13.: Comparison of first and best improvement on benchmark `L`.

Start Solution	heuristic	random
FI: first improvement	1.25	7.77
BIA: best improvement with approximation	10.48	37.96
BIE: best improvement	12.94	41.94

Table 5.5.: Average computational time for first and best improvement algorithms (in seconds).

The left plot of Figure 5.13 shows the gap reached in a single run of the improvement algorithms. There, it is observed that both best improvement variants led to similar results but that the computational time can be reduced if the approximation is applied. This effect is due to the fact that cost changes are computed only for promising neighbors. With the first improvement algorithm, smaller improvements of the initial solution were found. It can also be seen, that initializing the local search procedures by a random solution resulted in worse solutions than starting with a heuristic solution. For some instances, in this case the obtained solution was not better than the solution computed with the BoG heuristic.

The right plot of Figure 5.13 shows the results of repeating the improvement algorithms for ten minutes. Due to first improvement and best improvement cannot leave a local minimum with respect to the neighborhood, the local search is restarted if no further improvement was found. In case of a heuristic start solution, the last obtained solution is randomly changed until a solution with at least 15% higher costs is found to explore another area of the solution space. As it can be seen, in this case, first improvement was better than best improvement. One reason is that first improvement is a faster algorithm than best improvement because only a part of the neighborhood has to be analyzed, which can also be seen on the average com-

putational times provided in Table 5.5. As a consequence, more improved solutions were generated during the time limit of ten minutes resulting in a higher probability of finding a better solution. Another reason could be that best improvement tends to obtain similar solutions in several runs because it undoes the random changes quickly. In case of choosing the start solution by random, each run of the improvement algorithms is started with a new random solution. As it can be seen in the right plot of Figure 5.13, in this case, all three improvement algorithms showed a similar performance. Further, it can be observed that the improvements were smaller if random solutions are used as start solutions. One reason for this behavior is that the computational time of a single improvement algorithm run was significantly larger if the start solution was chosen by random because more improvement steps were necessary to reach a local minimum with respect to the neighborhood. Consequently, less improved solutions were obtained within ten minutes. Furthermore, as observed by analyzing a single run of the improvement algorithms initialized with a random solution, mostly better solutions were found when the improvement algorithm is initialized by a heuristic solution.

In conclusion it can be said firstly, that the proposed local search procedures should be initialized by a heuristic solution, and secondly, that, if one run or only few runs of local search are possible, then the best improvement algorithm should be used, but if more time is available, it is better to run the first improvement algorithm several times.

5.5. Conclusion

In this chapter, different heuristics were presented to solve the VRPCC approximately. In Section 5.1, three newly developed greedy heuristics were investigated. It was proven that the approximation ratio of each of them is unbounded. Furthermore, the three greedy heuristics were combined to a best-of-greedy algorithm. In Section 5.2, an application of the rollout algorithm to the VRPCC was investigated which applies the previously presented heuristics to evaluate decisions by building up the schedule job by job. Finally, in Section 5.3, the local search algorithms first and best improvement were presented and a definition of a neighborhood for the VRPCC was given.

Computational experiments were carried out to compare the performance of the heuristics in terms of solution quality. The results were presented in Section 5.4. Based on large instances with one hundred jobs, it was analyzed, which settings of the heuristics yielded the best solutions. It was observed that, dependent on the instance, different greedy heuristics with different parameter settings produced the best solution. Consequently, the combination of them, which is the best-of greedy heuristic, was the best choice to obtain a good approximate solution for a large spectrum of instances. Further, it was found out that the rollout algorithm significantly improved the greedy heuristics. Also the presented first and best improvement algorithms led to significant improvements of the greedy solutions.

The BoG heuristic, the rollout algorithm and local search were compared with solving one of the presented MILPs by CPLEX. It turned out that in benchmark `S`, where the optimal solution is known, repeating first improvement several times often obtained an optimal solution, even more often than solving (R2dT4) with CPLEX. Also in benchmark `M` and `L`, repeated local search outperformed the other heuristics and applying the commercial solver. Because the exact solution method was outperformed by the heuristics, in the next chapter an alternative exact solution approach will be investigated.

6. Two Branch-and-Bound Algorithms for the Partition and Permutation Model

After applying a commercial solver to the VRPCC, it turned out that the customer costs lead to a problem harder to solve because the time variables and the route variables are hardly linked which leads to poor LP relaxation values. Because of that, other exact solution approaches were searched. As explained in Section 2.3, the branch-and-bound method is a suitable approach to solve combinatorial problems like VRPs. In this chapter, it is described how the general solution approach branch-and-bound is applied to the VRPCC. At first, the branch-and-bound method is introduced in detail. Then, Section 6.2 provides lower bounds for the objective value, which are a fundamental part of this solution method. After that, in Section 6.3, two branch-and-bound algorithms for the VRPCC are presented. In detail, with the proposed branching strategies the routes are build-up job by job. The algorithms are implemented in a depth-first fashion applying backtracking and allowing parallel computation. Finally, Section 6.4 compares the lower bounds and the two branching strategies using computational experiments. The comparison of the computational costs of the two branch-and-bound algorithms and of solving a MILP of the VRPCC with a commercial solver shows that the developed branch-and-bound algorithms in combination with the lower bounds efficiently solve small instances of the VRPCC.

6.1. General Principles of the Branch-and-Bound Method

Branch-and-bound is a common principle to solve complex combinatorial optimization problems. An introduction to this method can be found, e.g., in [10, 90]. The main idea is to successively break up the solution space $\mathcal{S}$ into certain subsets S_i and to discard subsets that cannot contain an optimal solution. In doing so, a search tree is produced as shown in Figure 6.1: Each node i corresponds to a solution space S_i which is a subset of $\mathcal{S}$. To break-up the solution space of a node i into subsets, several child nodes are created each with a smaller solution space such that the union of the solution spaces of the child nodes is equal to the solution space of node i. If a node corresponds to (at most) one feasible solution, a leaf of the search tree is reached. The root of the search tree is the node corresponding to the complete

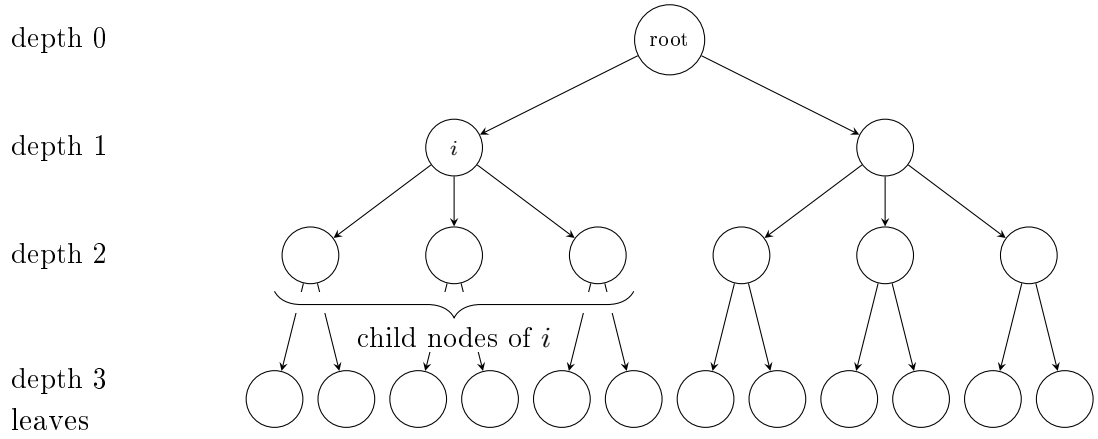


Figure 6.1.: Illustration of a search tree.

solution space $\mathcal{S}$. In the following, the distance of a node to the root will be called depth of the node in the search tree.

The branch-and-bound algorithm for an optimization problem

$$\min\{g(s) | \, s \in \mathcal{S}\} \tag{6.1}$$

essentially consists of the following steps:

Initialize: The search tree is initialized with the root node to which the whole solution space $\mathcal{S}$ belongs to. Further, an upper bound UB for $g(s)$ is determined, this could be for example the objective value of a feasible solution which is stored as best solution so far s^*.

Select: Take a node from the search tree that has not been analyzed yet.

- If the solution space of the node contains several feasible solutions, continue.
- Otherwise, if a single feasible solution s corresponds to the node, calculate its objective value $g(s)$. If $g(s) < \text{UB}$, store s as the best solution achieved so far s^* and set $\text{UB} = g(s)$. Repeat step **Select**.

Bound: Calculate a lower bound LB for the solutions that belongs to the selected node.

Branch or Prune:

- If $\text{LB} < \text{UB}$, **branch** on the node by dividing its solution space into (disjoint) subsets. Each of them creates a new node.
- Otherwise, **prune** the node because no better solution than s^* belongs to its solution space.

Go to step **Select**.

Return optimal solution: When all nodes of the tree are analyzed, the best solution so far s^* is optimal for the problem (6.1). This is true because the branch-and-bound algorithm ensures that the objective value of any feasible solution is not smaller than $g(s^*)$. If no feasible solution was found, the solution space of (6.1) is empty and the optimization problem is infeasible.

Note that if the upper bound is not based on a feasible solution, then branching is necessary in the branch-or-prune step even for nodes with LB = UB until a feasible solution is found. The first found feasible solution s with $g^c(s) \leq$ UB has to be stored.

To design a branch-and-bound algorithm, mainly three questions have to be answered:

- *How to select the next node?*
  Two selection strategies are common, see [42]:
  - "Last in, first out": This strategy leads to a depth-first search where each branch is explored as far as possible. One advantage of this strategy is that possibly a feasible solution can be obtained in shorter time, which is important if no upper bound was found. But often, the objective value of this first solution is far from the optimal value. A second advantage of "last in, first out" is that, in comparison to the strategy described below, fewer nodes have to be stored simultaneously.
  - "Best first": With this strategy, the best node, e.g., the one with smallest lower bound, is selected in each step in order to quickly find a good solution that improves the upper bound. This allows to prune more nodes which keeps the search tree small. However, several nodes and their lower bound must be stored, which can require a lot memory capacity or additional effort to store and retrieve the information.

- *How to determine a lower bound?*
  This question strongly depends on the problem and its formulation. In many applications, a relaxation of the problem is used, which can be quickly calculated via a suitable algorithm, e.g., the LP relaxation.

- *How to create new nodes?*
  The branching strategy strongly depends on the problem formulation and also on the chosen lower bound. If the lower bound results from the LP relaxation of a MILP formulation, two variants are common [116]:
  - Variable dichotomy: one integer variable is selected that has a fractional value in the solution of the LP relaxation. Two new nodes are generated: One where the variable is bounded from below by the rounded up value, and another where the variable is bounded from above by the round down value.

– Generalized upper bound dichotomy: In several MILPs, generalized upper bound constraints occur, which are $\sum_{j \in Q} x_j \leq U$ for some binary variables of the problem. For example, the constraints (4.35) and (4.36) of the route formulation (R2) presented in Section 4.3 are generalized upper bound constraints. To generate two new nodes, the set Q is split into two non-empty subsets $Q_1 \subset Q$ and $Q \setminus Q_1$. Then for the first node, it is stipulated that $\sum_{j \in Q_1} x_j = 0$. And for the second node, the constraint $\sum_{j \in Q \setminus Q_1} x_j = 0$ is added. With it, the solution space is split into two disjunctive sets.

The branch-and-bound method can be improved by some tests to strengthen the bounds, discard subsets or fix further variables.

For the VRPCC, in this thesis two branch-and-bound algorithms are developed that rely on the formulation (PP) presented in Section 3.2. Recap, in (PP), a schedule is defined by a partition of the jobs into m non-empty subsets N_k and a permutation for each partition $\Pi^k(N_k))$ with $k \in M$. In the branch-and-bound algorithms, each node of the search tree refers to a partial solution $\widetilde{S}$, which is defined by a set of unplanned jobs $\widetilde{N}$ and an uncompleted schedule $(\widetilde{N}_k, \Pi^k(\widetilde{N}_k))_{k \in M}$, where $\widetilde{N} \cup \left(\bigcup_{k \in M} \widetilde{N}_k \right) = N$. The solution space of the node is the set of all schedules that can be generated by completing the partial solution according to the branching strategy. To get an **upper bound**, one of the heuristics presented in Section 5.1.4 is used. To **select** the next branching node, the depth-first approach is chosen. In detail, the design of the branching rules allows to use the backtracking approach [116, 141] which leads to a single, not yet analyzed tree node. For the **bound** step, lower bounds for travel costs and customer costs are calculated separately and the sum of both is used as lower bound for the total costs, see Section 6.2. For the **branch**ing step, two strategies are developed. In the first branching strategy, presented in Section 6.3.1, new nodes are created by *append*ing one job at the end of one route of the current uncompleted schedule. Applying the second strategy, new nodes are generated by *includ*ing a certain job inside a route of the current uncompleted schedule, see Section 6.3.2. It will be shown that the first branching strategy produces an unbalanced search tree, but allows the application of tighter bounds compared to the second branching strategy, which however results in a more balanced search tree.

6.2. Lower Bounds

Tight and fast computed lower bounds are fundamental for the efficiency of branch-and-bound methods. With tight bounds, most branches that do not lead to an optimal solution can be discarded and fewer leaves of the search tree need to be generated. A first idea to obtain a lower bound for the VRPCC could be to use the LP relaxation of a MILP formulation, e.g., one of the formulations proposed in Chapter 4 where the binary and integer variables are replaced by continuous ones.

However, as it will be seen later, there exists lower bounds that are more appropriate. For the novel customer cost part of the objective function, new developed bounds will be presented in Section 6.2.1. These bounds are investigated both analytically and computationally. For the latter, the quality of the bounds and the computational time are analyzed using computational experiments, see Section 6.4.1.1. For the travel cost part of the objective function, lower bounds of the TSP can be applied by making small changes to the travel cost matrix. A collection of possible bounds is presented in Section 6.2.2. A computational analysis of them is given in Section 6.4.1.2.

6.2.1. Lower Bounds for Customer Costs

In this thesis, several lower bounds for the customer cost value $g^c(s)$ defined by equation (3.6) are developed, in particular:

- A simple lower bound that is based on an upper bound for the number of jobs that can be visited per day,
- some lower bounds that are derived from solving different special bin packing problems and
- one lower bound that is computed by a polynomial algorithm which provides a solution of the LP relaxation of one of the presented bin packing problems.

The last bound was considered because solving bin packing problems is NP-hard.

The computational effort of the lower bounds on customer costs depends on the number of analyzed jobs. To reduce the computational time, it would be beneficial to consider only jobs with non-zero customer cost coefficient which are given by the set $N_c = \{i \in N | \, c_i > 0\}$. The following lemma indicates that jobs with zero customer cost coefficient can be ignored when calculating lower bounds for customer costs.

Lemma 6.1. *Let $N' \subseteq N$ a set of considered jobs. Then, for each feasible solution S on N, a feasible solution S' on N' can be deduced by removing the jobs of the set $N \setminus N'$ from the routes. This solution is feasible without changing the start times. Furthermore, if for S' the start times are calculated based on* (3.8)*, the customer cost value cannot be higher than the customer cost value of S.*

Proof. For each job $i \in N'$, let p_i and p'_i be its predecessor in S and S', respectively. As assumed, the travel times satisfy the triangle inequality, thus it is $r_{ij} + a_j + r_{jk} \geq r_{ik}$ for each $i \neq j \neq k \in N$. Then, a feasible start time t_i of job i in S is also feasible in S':

- if $p_i = p'_i$, then t_i is obviously feasible; and

- if $p_i \neq p'_i$, then in S at least one job is visited between p'_i and i. Let π_i be the position of i in S. Then, $p_i = \pi_{i-1}$. Let further π_{i-j} be the position of p'_i. Because S is feasible and the travel times satisfy the triangle inequality, it holds that

$$\begin{aligned}\xi(t_i) = \xi(t_{\pi_i}) &\geq \xi(t_{\pi_{i-1}}) + a_{\pi_{i-1}} + r_{\pi_{i-1}\pi_i} \\ &\geq \xi(t_{\pi_{i-j}}) + \sum_{k=0}^{k=j-1} \left(a_{\pi_{i-j+k}} + r_{\pi_{i-j+k}\pi_{i-j+k+1}}\right) \\ &\geq \xi(t_{\pi_{i-j}}) + a_{\pi_{i-j}} + r_{\pi_{i-j}\pi_i} \\ &= \xi(t_{p'_i}) + a_{p'_i} + r_{p'_i i}.\end{aligned}$$

Thus, for the solution S' the start time t_i satisfies constraint (3.2). Consequently, with the start times calculated based on the schedule S, S' is a feasible solution.

Furthermore, a recalculation of the start times for S' based on equation (3.8) does not lead to increased start times as reasoned in Section 3.1. Because of that, the customer cost value of S' is not larger than the customer cost value of S. □

For the sake of readability, the following definitions are introduced:

- Only the jobs with non-zero customer cost coefficient have to be considered which are given by the set

$$N_c := \{i \in N |\, c_i > 0\}$$

  with cardinality n_c.

- The minimal travel time from job $i \in N_c$ to any other job in N_c is

$$r_i := \min\{r_{ij} |\, j \in N_c, j \neq i\}. \tag{6.2}$$

- For the trips outside the working shift, the time R_m has to be reserved which is given by

$$R_m := \max\left\{\textstyle\sum_{i\in I} r_i |\, I \subseteq N_c, |I| = m\right\}. \tag{6.3}$$

- The k-th largest travel time $r^k_{\max}$ of the set $\{r_i |\, i \in N_c\}$ is defined as

$$r^1_{\max} := r_{i^1_{\max}} \text{ with } i^1_{\max} = \min\{i |\ r_i \geq r_j,\ i, j \in N_c\} \tag{6.4}$$

  and

$$r^k_{\max} := r_{i^k_{\max}} \tag{6.5}$$

$$\text{with } i^k_{\max} = \min\{i |\ r_i \geq r_j,\ i, j \in N_c \setminus \{i^{k'}_{\max} |\, 1 \leq k' \leq k\}\}, \tag{6.6}$$

  respectively.

- The set of feasible start times is given by T. For the benchmarks used in the computational experiments, which are defined in Appendix A, the start time is $(0, 480)$ which means that the first job is executed on day one. Consequently, $T = \{1, 2, \ldots, d_{\max}\}$. If another start time is chosen, T has to be adapted.

6.2.1.1. A Lower Bound based on a Constant Number of Jobs per Day

The idea of the first presented lower bound for the customer costs is to calculate, how many jobs can be done at most per day; and then, to allocate the most expensive jobs to the first day, the next most expensive jobs to the second day, and so on.

At first, a simple calculation of $\tilde{\eta}$, which is the maximal number of jobs executable at one day, is given. Let $t_{\min} := \min\{a_i + r_i | \, i \in N_c\}$ be the minimal time needed per job, where a_i is the working duration and r_i is the minimal time needed to travel to the next job calculated by equation (6.2). For each day, u minutes are available for job execution and the time $r^1_{\max}$, as defined in equation (6.4), has to be reserved for the trip outside the working shift. Thus, not more than

$$\tilde{\eta} := m \left\lfloor \frac{u + r^1_{\max}}{t_{\min}} \right\rfloor \tag{6.7}$$

jobs can be done per day.

The estimation of (6.7) can be improved by replacing $t_{\min}$ with the smallest sum of execution and travel time of η jobs and by using different approaches to consider m vehicles. For the first approach, the number of jobs that can be done by one vehicle on one day is computed and then multiplied by the number of vehicles m. For the second approach, the time available for all vehicles is considered to calculate how many jobs can be executed per day. The minimum of both values is then η. Recap, $a_i + r_i$ is the minimal time needed for execution and travel for job $i \in N_c$. Then, the upper bound for the number of jobs executable by one vehicle at one day is the value η_1 that meets the conditions

$$\min_{\{I \subseteq N_c | \, |I| = \eta_1\}} \sum_{i \in I} (a_i + r_i) \leq u + r^1_{\max} \tag{6.8}$$

and

$$u + r^1_{\max} < \min_{\{I \subseteq N_c | \, |I| = \eta_1 + 1\}} \sum_{i \in I} (a_i + r_i). \tag{6.9}$$

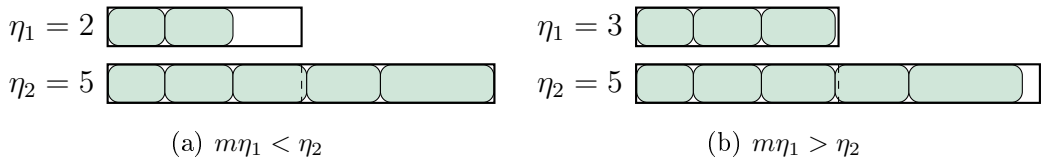
Inequality (6.8) ensures that a subset I with cardinality η_1 exists, such that the sum of the shortest possible time for execution and travel of the jobs of I does not exceed the maximum time available per vehicle per day. And inequality (6.9) guarantees that η_1 is an upper bound, because there exists no set with more than η_1 jobs, such that the sum of the shortest possible times of its job fits to the total available time.

A similar problem is solved to calculate the upper bound η_2 for the number of jobs that can be executed on one day by all m vehicles:

$$\min_{\{I \subseteq N_c | \, |I| = \eta_2\}} \sum_{i \in I} (a_i + r_i) \leq mu + R_m \tag{6.10}$$

and

$$mu + R_m < \min_{\{I \subseteq N_c | \, |I| = \eta_2 + 1\}} \sum_{i \in I} (a_i + r_i). \tag{6.11}$$



(a) $m\eta_1 < \eta_2$ (b) $m\eta_1 > \eta_2$

Figure 6.2.: Schematic illustration of the comparison of $m\eta_1$ and η_2.

Each of both approaches has an advantage: For η_1, the advantage is that the time available per day is better approximated. As shown in Figure 6.2(a) by an example with two vehicles, if the remaining space in the bin for one day per vehicle is large, it can happen that more than $m\eta_1$ items fit into the bin for one day and all vehicles. But, as illustrated in Figure 6.2(b), the advantage of η_2 is that more items are taken into account. Consequently, if the item have strongly different sizes, it is possible that fewer than $m\eta_1$ jobs fit into the bin for one day and all vehicles because the items are packed in order of increasing item size and with it also larger items can be included.

Lemma 6.2. *The value* $\eta := \min\{m\eta_1, \eta_2\}$ *computed by the inequalities* (6.8)–(6.11) *is an upper bound for the number of jobs processed on one day.*

Proof. Let S be a solution on N and S_c the solution derived from S by removing the jobs that does not belong to N_c. For each job $i \in N_c$, the successor in S_c is denoted by q_i. Assuming, it exists a day $t \in T$ where more than η jobs are executed, which means $|\{i \in N_c | \, t_i^d = t\}| > \min\{m\eta_1, \eta_2\}$.

If $|\{i \in N_c | \, t_i^d = t\}| > m\eta_1$, then there exists a route $k \in M$ where more than η_1 jobs are visited on day t. Let $I_{k,t} := \{i \in N_c \cap N_k | \, t_i^d = t\}$ be the jobs visited on day t by vehicle k with $|I_{k,t}| > \eta_1$. As shown in Lemma 6.1, S_c is a feasible solution, thus for this route k and day t it holds that

$$\sum_{i \in I_{k,t}} (a_i + r_{iq_i}) \leq u + r_{jq_j},$$

with j is the job that is executed at last on day t, thus $j \in I_{k,t}$ but $q_j \notin I_{k,t}$. Rearranging leads to

$$\begin{aligned} u &\geq \sum_{i \in I_{k,t}} (a_i + r_{iq_i}) - r_{jq_j} \\ &= \sum_{i \in I_{k,t} \setminus \{j\}} (a_i + r_{iq_i}) + a_j \\ &\geq \sum_{i \in I_{k,t} \setminus \{j\}} (a_i + r_i) + a_j, \end{aligned}$$

because $r_i \leq r_{iq_i}$. Adding r_j on both sides leads to

$$u + r_j \geq \sum_{i \in I_{k,t}} (a_i + r_i).$$

Due to $r_j \leq r^1_{\max}$, it follows that $\sum_{i \in I_{k,t}} (a_i + r_i) \leq u + r^1_{\max}$. This is a contradiction to (6.9), because if $|I_{k,t}| > \eta_1$, it holds that

$$u + r^1_{\max} < \min_{\{I \subseteq N_c \,|\, |I| = \eta_1 + 1\}} \sum_{i \in I} (a_i + r_i) \leq \sum_{i \in I_{k,t}} (a_i + r_i) \leq u + r^1_{\max}.$$

Secondly, it is shown that also $|\{i \in N_c \,|\, t^d_i = t\}| > \eta_2$ leads to a contradiction. Let $I_t := \{i \in N_c \,|\, t^d_i = t\}$ be the set of jobs visited on day t with $|I_t| > \eta_2$. Then, it is

$$mu \geq \sum_{i \in I_{k,t}} a_i + \sum_{\substack{i \in I_{k,t} \\ t^d_{q_i} = t}} r_{iq_i} \geq \sum_{i \in I_{k,t}} a_i + \sum_{\substack{i \in I_{k,t} \\ t^d_{q_i} = t}} r_i.$$

Adding the travel times for the jobs executed after day t on both sides leads to

$$mu + \sum_{\substack{i \in I_{k,t} \\ t^d_{q_i} > t}} r_i \geq \sum_{i \in I_{k,t}} (a_i + r_i).$$

Since $|\{r_i \,|\, i \in I_{k,t},\ t^d_{q_i} > t\}| \leq m$ it holds that the sum of these travel times does not exceeds R_m. Consequently, it holds that $\sum_{i \in I_{k,t}} (a_i + r_i) \leq mu + R_m$, which is a contradiction to inequality (6.11) of the definition of η_2:

$$mu + R_m < \min_{\{I \subseteq N_c \,|\, |I| = \eta_2 + 1\}} \sum_{i \in I} (a_i + r_i) \leq \sum_{i \in I_{k,t}} (a_i + r_i) \leq mu + R_m.$$

Thus, in a feasible solution neither more than $m\eta_1$ nor more than η_2 jobs can be visited per day and $\eta = \min\{m\eta_1, \eta_2\}$ is an upper bound for the number of jobs executable per day. □

To calculate η_1 and η_2 efficiently, the set N_c is ordered by increasing values of $a_i + r_i$. The complexity of the sorting is $\mathcal{O}(n_c \log(n_c))$. The values are summed up in increasing order until the time bounds, $u + r^1_{\max}$ and $mu + R_m$, respectively, will be exceeded with the next summand.

To obtain the lower bound $\mathrm{LB}^c_{\mathrm{s}}$, an allocation of jobs to days with minimal customer costs is sought. The problem is described by (6.12)-(6.14).

$$\text{(S)} \qquad \mathrm{LB}^c_{\mathrm{s}} := \min \quad \sum_{i \in N_c} c_i \tau_i \qquad\qquad\qquad\qquad (6.12)$$

$$\text{s.t.} \quad \tau_i \in T \qquad\qquad i \in N_c \qquad (6.13)$$

$$|\{i \in N_c \,|\, \tau_i = t\}| \leq \eta \qquad\qquad t \in T \qquad (6.14)$$

The decision variable τ_i is the day to which job $i \in N_c$ is assigned. The objective function (6.12) is the sum of the time-dependent customer costs and has to be minimized. Constraints (6.13) ensure that a day is allocated to each job. Finally, constraints (6.14) bound the number of jobs allocated to day $t \in T$ by the upper

bound η. The solution can be obtained in polynomial time by ordering the jobs by decreasing values of c_i. The first η jobs of the ordered list are allocated to the first day of T, the next η jobs are allocated to the second day and so on.

Theorem 6.3. *LB_s^c is a lower bound for the customer costs of VRPCC.*

Proof. As shown in Lemma 6.2, if S_c is a feasible schedule of the jobs N_c, than at most η jobs are visited per day. Thus, each feasible solution of the VRPCC can be transformed into a feasible solution of (S) by setting $\tau_i = t_i^d$ for the jobs $i \in N_c$, where the objective value equals to the customer cost value of S_c. Consequently, LB_s^c cannot be larger than the customer cost value of a solution of the VRPCC. □

6.2.1.2. Lower Bounds from Minimum Weighted Sum Bin Packing Problems

The first lower bound LB_s^c is a straightforward approach. Instead of using a fixed number of jobs per day, one could also be inspired by bin packing problems to allocate jobs to days. The resulting problems are similar to the minimum weighted sum bin packing problem (MWSBP) as introduced in [48]. In the MWSBP, each item is afflicted with a weight and the objective is to minimize the costs c which are the sum of the item weights multiplied with the index of the bin in which the item is packed. Two polynomial algorithms are presented in [48], one to calculate a lower bound for the optimal value c^* and one to get an approximated solution with $c \leq c^*(1+\delta)$ where $1/\delta$ is an integer. Both algorithms are based on the search for a shortest path in a graph, where the size of the graph increases with the desired accuracy of the result.

The in this thesis presented bin packing problems to obtain a lower bound for customer costs are not solved with these polynomial algorithms because their computational effort increases strongly with the accuracy. Furthermore, applying the bin packing problems to the branching strategies leads to additional constraints and non-identical bin sizes. Adapting the algorithms to these restrictions is hardly possible. Because of that, CPLEX is used to solve the bin packing problems.

There are two possibilities to consider traveling outside the working shift:

- Increase the time available per day for working and traveling to pack working time and travel time together as one item in the bins.
- Define separate items for the working time and travel time and pack them in bins of the same size as the working shift.

The bins can be designed either one per day or one per day and vehicle. In the latter case, the time available per day is more precisely modeled and better bounds can be obtained. However, for each day, there are m bins of similar size which can lead to several solutions of same costs with interchanged bin allocation. This can lead to an increased computational effort, as more solutions have to be analyzed.

Bin Packing Problems with Bin Size Increased by Time Reserved for Traveling Outside the Working Shift

At first, the bin packing problems (BP1) and (BP2) are presented where for each job one item has to be packed into a bin. The size of the items represents the sum of the working duration and the minimal travel time to another job. The bin size corresponds to the total time available for work plus the time reserved to travel outside the work shift. This leads to the following problem definition:

$$
\begin{aligned}
\text{(BP1)} \quad \mathrm{LB}^c_{\mathrm{BP1}} := \min \quad & \sum_{i \in N_c} \sum_{t \in T} c_i t x_{it} & & \\
\text{s.t.} \quad & \sum_{t \in T} x_{it} = 1 & & i \in N_c, \\
& \sum_{i \in N_c} x_{it}(a_i + r_i) \leq mu + R_m & & t \in T, \\
& x_{it} \in \{0,1\} & & i \in N_c,\ t \in T.
\end{aligned}
$$

In (BP1), each bin corresponds to one day. The bin size is set to the time available for maintenance and travel per day $mu + R_m$, where R_m is the time reserved for traveling outside the working shift as defined in equation (6.3). The allocation of items, or rather jobs, to bins is done by binary variables x_{it}, $i \in N_c,\ t \in T$, where $x_{it} = 1$ if and only if job i is allocated to the bin of day t. The size of each job $i \in N_c$ is $a_i + r_i$, which is the time needed for the execution of the job and the minimal time needed to travel to the next job as defined in equation (6.2). The weight of each job is the customer cost coefficient c_i. The objective is to minimize the sum of the customer costs resulting from the job allocation.

As mentioned above, a better bound can be obtained if for each vehicle a separate bin per day is defined. This leads to the following problem formulation:

$$
\begin{aligned}
\text{(BP2)} \quad \mathrm{LB}^c_{\mathrm{BP2}} := \min \quad & \sum_{i \in N_c} \sum_{t \in T} \sum_{k \in M} c_i t x_{itk} & & \\
\text{s.t.} \quad & \sum_{t \in T} \sum_{k \in M} x_{itk} = 1 & & i \in N_c, \\
& \sum_{i \in N_c} x_{itk}(a_i + r_i) \leq u + r^k_{\max} & & t \in T,\ k \in M, \\
& x_{itk} \in \{0,1\} & & i \in N_c,\ t \in T,\ k \in M.
\end{aligned}
$$

The problem (BP2) is a bin packing problem with $m|T|$ bins, one per day and vehicle. Each bin is associated to a route $k \in M$ and has the size $u + r^k_{\max}$, where $r^k_{\max}$ is the k-th largest value of the set $\{r_i | \, i \in N_c\}$ as defined in equations (6.4) and (6.6). Then, the bin size equals the sum of the working shift length and the time reserved for the trip to the first job executed on the next day. Again, each job $i \in N$ has the size $a_i + r_i$ and the weight c_i. The allocation of jobs to bins is done by three-index variable x_{itk}, $i \in N_c,\ t \in T,\ k \in M$, where $x_{itk} = 1$ if and only if job i is allocated to the bin associated to day t and route k. The objective is to minimize the sum of customer costs resulting from the job allocation.

The next theorem shows that $\mathrm{LB}^c_{\mathrm{BP1}}$ and $\mathrm{LB}^c_{\mathrm{BP2}}$ are lower bounds for the customer cost value.

Theorem 6.4. *LB^c_{BP1} and LB^c_{BP2} are lower bounds for the customer cost part of the VRPCC.*

Proof. Let S be a solution on N and S_c the solution derived from S by removing the jobs $N \setminus N_c$. The successor of a job $i \in N_c$ in schedule S_c is denoted by q_i. Then, for each day $t \in T$, the working shift constraint leads to

$$\sum_{\substack{i \in N \\ t_i^d = t}} a_i + \sum_{\substack{i \in N \\ t_i^d = t_{q_i}^d = t}} r_{iq_i} \leq mu.$$

Because $r_i \leq r_{iq_i}$, it is also true that

$$\sum_{\substack{i \in N_c \\ t_i^d = t}} a_i + \sum_{\substack{i \in N_c \\ t_i^d = t_{q_i}^d = t}} r_i \leq mu. \tag{6.15}$$

To match the capacity constraint of (BP1), the sum of the minimal travel times for the trips to the jobs that are executed on the next day has to be added. This sum contains at most m summands, thus

$$\sum_{\substack{i \in N_c \\ t_i^d = t < t_{q_i}^d}} r_i \leq \max\{\sum_{i \in I} r_i |\, I \subseteq N_c, |I| = m\} = R_m.$$

Consequently, adding these costs on both sides of inequality (6.15) implies that for each day $t \in T$,

$$\sum_{\substack{i \in N_c \\ t_i^d = t}} (a_i + r_i) \leq mu + \sum_{\substack{i \in N_c \\ t_i^d = t < t_{q_i}^d}} r_i \leq mu + R_m.$$

Thus, each feasible solution S of the VRPCC can be transformed into a feasible solution of the bin packing problem (BP1) by allocating each job to the bin that corresponds to the start day of the job. The costs of this solution are equal to $g^c(S)$. Consequently, also the schedule with minimal customer cost value can be transformed into a feasible solution of $\mathrm{LB}^c_{\mathrm{BP1}}$. Consequently, $\mathrm{LB}^c_{\mathrm{BP1}}$ is a lower bound for the customer costs of the VRPCC.

The proof for $\mathrm{LB}^c_{\mathrm{BP2}}$ is similar: Let $N_{c,k}$ be the set of jobs allocated to route k in S_c. For each $t \in T$ and $k \in M$, it is valid that

$$\sum_{\substack{i \in N_{c,k} \\ t_i^d = t}} a_i + \sum_{\substack{i \in N_{c,k} \\ t_i^d = t_{q_i}^d = t}} r_i \leq \sum_{\substack{i \in N_{c,k} \\ t_i^d = t}} a_i + \sum_{\substack{i \in N_{c,k} \\ t_i^d = t_{q_i}^d = t}} r_{iq_i} \leq u. \tag{6.16}$$

Furthermore, for each day $t \in T$, at most one trip to a job executed on the next day has to be considered per vehicle. Thus, there is at most one job $i \in N_{c,k}$ with $t_i^d = t < t_{q_i}^d$. Because $\{r_{\max}^j\}_{j=1,\dots,m}$ are the m largest values of $\{r_i | \, i \in N_c\}$, there exists a mapping from $k \in M$ to $j_{k,t} \in M$ such that

$$\sum_{\substack{i \in N_{c,k} \\ t_i^d = t < t_{q_i}^d}} r_i \leq r_{\max}^{j_{k,t}},$$

for each $k \in M$, and

$$\bigcup_{k \in M} \{j_{k,t}\} = M.$$

Then, adding the costs reserved for the job that is executed on the next day on both sides of inequality (6.16) leads to

$$\sum_{\substack{i \in N_{c,k} \\ t_i^d = t}} (a_i + r_i) \leq u + \sum_{\substack{i \in N_{c,k} \\ t_i^d = t < t_{q_i}^d}} r_i \leq u + r_{\max}^{j_{k,t}}.$$

Consequently, each feasible solution S can be transformed into a feasible solution of (BP2) with equal customer costs. Thus, the minimal customer costs of a solution of the VRPCC cannot be smaller than $\mathrm{LB}_{\mathrm{BP2}}^c$. □

By means of the following theorem, the bounds are compared analytically.

Theorem 6.5. *It holds that LB_{BP2}^c is at least as good as LB_s^c and LB_{BP1}^c.*

Proof. Firstly, it is shown that $\mathrm{LB}_{\mathrm{BP2}}^c \geq \mathrm{LB}_{\mathrm{s}}^c$. Since

$$\begin{aligned} \sum_{k \in M} \sum_{i \in N_c} (a_i + r_i) x_{itk} &\leq \sum_{k \in M} \left(u + r_{\max}^k\right) \\ &\leq m(u + r_{\max}^1) \\ &\leq m \min_{\{I \subset N_c | \, |I| = \eta_1 + 1\}} \sum_{i \in I} (a_i + r_i) \end{aligned}$$

and

$$\begin{aligned} \sum_{k \in M} \sum_{i \in N_c} (a_i + r_i) x_{itk} &\leq \sum_{k \in M} \left(u + r_{\max}^k\right) \\ &= mu + R_m \\ &\leq \min_{\{I \subset N_c | \, |I| = \eta_2 + 1\}} \sum_{i \in I} (a_i + r_i), \end{aligned}$$

it follows that $\sum_{k \in M} \sum_{i \in N_c} x_{itk} \leq \eta$ for each $t \in T$. Thus, each feasible solution of (BP2) can be transformed into a feasible solution of (S) with the same objective value and it is valid that $\mathrm{LB}_{\mathrm{s}}^c$ is not larger than $\mathrm{LB}_{\mathrm{BP2}}^c$.

Note that $\mathrm{LB}_{\mathrm{s}}^{c}$ and $\mathrm{LB}_{\mathrm{BP2}}^{c}$ are not equal because not each feasible solution of (S) can be transformed into a feasible solution of (BP2) with same cost value. This is not difficult to see: Let $I \subset N_c$ a set with $|I| \leq \eta$ but $\sum_{i \in I}(a_i + r_i) > mu + R_m$. Such a set can exist, because η is an upper bound of the jobs executable per day and it is not ensured that each day η jobs are executable. Then, all jobs of I can be allocated to the same day in (S), but it is not possible to pack all jobs into the m bins of $\mathrm{LB}_{\mathrm{BP2}}^{c}$ associated to one day.

Secondly, it is shown that $\mathrm{LB}_{\mathrm{BP2}}^{c} \geq \mathrm{LB}_{\mathrm{BP1}}^{c}$. Because of

$$\sum_{k \in M} \sum_{i \in N_c} (a_i + r_i) x_{itk} \leq \sum_{k \in M} \left(u + r_{\max}^{k}\right) = mu + R_m \quad t \in T,$$

each feasible solution of (BP2) can be transformed into a feasible solution of (BP1) by defining $x_{it} = \sum_{k \in M} x_{itk}$ and both solutions have the same cost value. Thus, $\mathrm{LB}_{\mathrm{BP1}}^{c}$ is not larger than $\mathrm{LB}_{\mathrm{BP2}}^{c}$.

Finally, it is shown that $\mathrm{LB}_{\mathrm{BP2}}^{c}$ and $\mathrm{LB}_{\mathrm{BP1}}^{c}$ are not equal. For a feasible solution of $\mathrm{LB}_{\mathrm{LPBP1}}^{c}$, let $I_t := \{i \in N_c | \, x_{it} = 1, k \in M\}$ be the set of jobs allocated to day $t \in T$. To obtain a feasible solution of (BP2), a partition of I_t into m blocks I_{tk} with $k \in M$ has to be found, such that each block satisfies the constraint $\sum_{i \in I_{tk}}(a_i + r_i) \leq u + r_{\max}^{k}$. Although in both problems the time available per day is equal, it can be impossible to find such a partition. Consequently, it is $\mathrm{LB}_{\mathrm{BP2}}^{c} \geq \mathrm{LB}_{\mathrm{BP1}}^{c}$. □

Note that if the bin sizes are similar, there exists several solutions of (BP2) with the same cost value. This could lead to a loss of performance in solving (BP2) compared to (BP1).

Remark For the bounds $\mathrm{LB}_{\mathrm{s}}^{c}$ and $\mathrm{LB}_{\mathrm{BP1}}^{c}$, neither $\mathrm{LB}_{\mathrm{s}}^{c} \geq \mathrm{LB}_{\mathrm{BP1}}^{c}$ nor $\mathrm{LB}_{\mathrm{BP1}}^{c} \geq \mathrm{LB}_{\mathrm{s}}^{c}$ is true. The reasons are that on the one hand, a feasible solution of (S) is infeasible for (BP1) when a set $I \subset N_c$ with $|I| \leq \eta$ exists, for which the sum $\sum_{i \in I}(a_i + r_i)$ is larger than $mu + R_m$. And on the other hand, if $m\eta_1 < \eta_2$, a feasible solution of (BP1) can be infeasible for (S), because a set $I \subset N_c$ with $\sum_{i \in I}(a_i + r_i) \leq mu + R_m$ can contain more than $m\eta_1$ jobs.

However, if $\eta_2 \leq m\eta_1$, it is true that $\mathrm{LB}_{\mathrm{BP1}}^{c} \geq \mathrm{LB}_{\mathrm{s}}^{c}$ because a bin of a solution of (BP1) cannot contain more than η_2 jobs due to its definition given by inequality (6.11). Then, each feasible solution of (BP1) can be transformed into a feasible solution of (S) with same cost value.

Remark The lower bounds $\mathrm{LB}_{\mathrm{BP1}}^{c}$ and $\mathrm{LB}_{\mathrm{BP2}}^{c}$ can be improved by adding constraints $\sum_{i,j \in I} x_{ij} \leq |I|$ and $\sum_{i,j \in I} \sum_{k \in M} x_{ijk} \leq |I|$, respectively, to exclude subsets $I \subseteq N_c$, where the sum of the start times and the minimal travel times the between the jobs of the subset I exceeds the bin size. For (BP1), a subset I can be excluded if

$$\sum_{i \in s}(a_i + \min_{j \in I \setminus \{i\}} r_{ij}) - \max_{\substack{\bar{I} \subseteq I \\ |\bar{I}| = m}} \sum_{i \in \bar{I}} \min_{j \in I \setminus \{i\}} r_{ij} > mu, \tag{6.17}$$

whereby the m redundant travel time r_{ij} were subtracted. And for (BP2), a subset I cannot be processed by one vehicle within one day if it satisfies

$$\sum_{i \in I} (a_i + \min_{j \in I \setminus \{i\}} r_{ij}) - \max_{i \in I} \min_{j \in I \setminus \{i\}} r_{ij} > u. \tag{6.18}$$

Both kinds of subsets can be generated iteratively from a previous solution of the bin packing problem based on the items packed into one bin. Hence, the iterative algorithm starts by solving the bin packing problem without any excluded subsets. Then, each set of jobs allocated to one bin is analyzed whether the inequality (6.17) and (6.18), respectively, is true. If so, this subset is excluded in all further iterations. In order to generate a sufficiently large pool of excluded subsets quickly, not only the subsets obtained by the bin packing problem should be analyzed, but also sets with one job fewer and sets with one job exchanged should be tested and excluded if necessary.

Bin Packing Problems with Working Items and Travel Items

In (BP1) and (BP2), the fact that it is allowed to travel to the next job outside the working shift is modeled by increasing the bin size by the maximal possible trip duration. In the following, two bin packing problems, denoted as (BP3) and (BP4), are formulated, where the bin size is not increased. To allow trips outside the working shift, separate items for working duration a_i and minimal possible travel time r_i are packed. Thereby, the number of travel items per bin can be smaller than the number of packed working items.

At first, (BP3) is presented where one bin per day is defined.

$$\text{(BP3)} \quad \text{LB}^{\text{c}}_{\text{BP3}} := \min \quad \sum_{i \in N_c} \sum_{t \in T} c_i t x_{it} \tag{6.19}$$

$$\text{s.t.} \quad \sum_{i \in N_c} (x_{it} a_i + y_{it} r_i) \leq mu \qquad t \in T \tag{6.20}$$

$$\sum_{i \in N_c} (x_{it} - y_{it}) \leq m \qquad t \in T \tag{6.21}$$

$$y_{it} - x_{it} \leq 0 \qquad i \in N_c,\, t \in T \tag{6.22}$$

$$\sum_{t \in T} x_{it} = 1 \qquad i \in N_c \tag{6.23}$$

$$\sum_{t \in T} y_{it} \leq 1 \qquad i \in N_c \tag{6.24}$$

$$x_{it} \in \{0,1\},\, y_{it} \in \{0,1\} \qquad i \in N_c, t \in T \tag{6.25}$$

In (BP3), a binary variable x_{it} is one if and only if the working duration of job i is allocated to day $t \in T$. Further, y_{it} takes value one if and only if the travel time of job i is assigned to the working shift of day t. The objective function (6.19), which has to be minimized, is the sum of the customer cost coefficient c_i of job $i \in N_c$

multiplied by the day t to which the job is assigned to. Thus, for each job, there are two items that could be packed into a bin associated to a day: the working item a_i and the travel item r_i. With constraint (6.20), it is ensured that the sum of the allocated items is not larger than the available time for working per day, i.e., mu. The inequalities (6.21) realize that at most m working items more than travel items are packed into each bin. By constraints (6.22) is stipulated that the travel items have to be packed into the same bin as the corresponding working item. Furthermore, equations (6.23) stipulate that each working item has to be allocated to exactly one bin, and inequalities (6.24) ensure that each travel item is allocated to at most one bin.

Remark Relaxing (6.22) leads to a problem that can be solved faster. Then, it is not further stipulated that the travel item of job i is allocated to the same bin as the working item of job i. Note that according to (6.21) nevertheless each bin contains several travel items.

Better lower bounds can be obtained, if m bins are used per day, i.e., one for each vehicle. The corresponding bin packing problem is denoted as (BP4).

$$\text{(BP4)} \quad \mathrm{LB}^c_{\mathrm{BP4}} := \min \sum_{i \in N_c} \sum_{t \in T} \sum_{k \in M} c_i t x_{itk} \tag{6.26}$$

$$\text{s.t.} \quad \sum_{i \in N_c} (x_{itk} a_i + y_{itk} r_i) \leq u \qquad t \in T,\ k \in M \tag{6.27}$$

$$\sum_{i \in N_c} (x_{itk} - y_{itk}) \leq 1 \qquad t \in T,\ k \in M \tag{6.28}$$

$$y_{itk} - x_{itk} \leq 0 \qquad i \in N_c,\ t \in T,\ k \in M \tag{6.29}$$

$$\sum_{t \in T} \sum_{k \in M} x_{itk} = 1 \qquad i \in N_c \tag{6.30}$$

$$\sum_{t \in T} \sum_{k \in M} y_{itk} \leq 1 \qquad i \in N_c \tag{6.31}$$

$$x_{itk} \in \{0,1\},\ y_{itk} \in \{0,1\} \qquad i \in N_c,\ t \in T,\ k \in M$$

The problem (BP4) is similar to (BP3). A binary variable x_{itk} or y_{itk} is equal to one if and only if the working item or travel item of job i, respectively, is allocated to the bin associated to day $t \in T$ and vehicle $k \in M$. There are mainly three differences between (BP3) and (BP4):

- In (BP4), totally $m|T|$ bins are available, instead of $|T|$ bins as in (BP3).
- The size of each bin of (BP4) equals to the length of the working shift u, instead of the time available for working per day mu.
- In (BP4), only one more working item than travel items can be allocated to a bin, instead of at most m more working items than travel items as in (BP3).

The following theorem shows that $\mathrm{LB}^c_{\mathrm{BP3}}$ and $\mathrm{LB}^c_{\mathrm{BP4}}$ are lower bounds for the customer costs value of the VRPCC.

Theorem 6.6. *LB^c_{BP3} and LB^c_{BP4} are lower bounds for the customer cost part of the VRPCC and it is $LB^c_{BP4} \geq LB^c_{BP3}$.*

Proof. The first goal is to show that each solution S can be transformed into a feasible solution of (BP4) and that then $g^c(S)$ is equal to the costs of the corresponding solution of (BP4).

Let S be a feasible solution on N and S_c be the feasible solution deduced from S taking into account only the jobs of N_c. Further, let q_i be the successor of job i in S_c. For each $k \in M$, let $N_{c,k}$ be the jobs visited by vehicle k in S_c. For each day $t \in T$, the working shift constraint leads to

$$u \geq \sum_{\substack{i \in N_{c,k} \\ t_i^d = t}} a_i + \sum_{\substack{i \in N_{c,k} \\ t_i^d = t_{q_i}^d = t}} r_{iq_i} \geq \sum_{\substack{i \in N_{c,k} \\ t_i^d = t}} a_i + \sum_{\substack{i \in N_{c,k} \\ t_i^d = t_{q_i}^d = t}} r_i.$$

The binary variables are set to

$$x_{itk} = \begin{cases} 1 & \text{if } i \in N_{c,k} \text{ and } t_i^d = t \\ 0 & \text{otherwise} \end{cases} \quad \text{and} \quad y_{itk} = \begin{cases} 1 & \text{if } i \in N_{c,k} \text{ and } t_i^d = t_{q_i}^d = t \\ 0 & \text{otherwise} \end{cases}.$$

Obviously, $\sum_{i \in N_c} \sum_{t \in T} \sum_{k \in M} c_i t x_{itk} = g^c(S)$.
It remains to show that this definition of the binary variables leads to a feasible solution of (BP4). Firstly, the bin size constraints (6.27) are proven. Applying the binary variables to the working shift constraints leads to

$$u \geq \sum_{\substack{i \in N_{c,k} \\ t_i^d = t}} a_i + \sum_{\substack{i \in N_{c,k} \\ t_i^d = t_{q_i}^d = t}} r_i = \sum_{i \in N_c} (x_{itk} a_i + y_{itk} r_i) \qquad \forall t \in T,\ k \in M.$$

Thus, the bin size constraints (6.27) are valid.

Secondly, the other constraints have to be proven: Because for each day and route there is at most one trip outside the working shift, constraints (6.28) are true. Furthermore, since each job is allocated to exactly one bin, which is the bin associated to day t and vehicle k where the job is visited, also constraints (6.30) are true. Moreover, for each job $i \in N_{c,k}$ with $t_i^d = t$, the travel time r_i is allocated also to the bin of day t and vehicle k if the job and its successor q_i are both visited on day t by vehicle k. Otherwise, $y_{itk} = 0$. Thus, also constraints (6.31) and (6.29) are true. In summary, this implies that each feasible solution of VRPCC can be transformed to a feasible solution of (BP4). With it, $\mathrm{LB}^c_{\mathrm{BP4}}$ is a lower bound of the customer cost part of VRPCC.

Next, it is shown that $\mathrm{LB}^c_{\mathrm{BP4}} \geq \mathrm{LB}^c_{\mathrm{BP3}}$, which implies that also $\mathrm{LB}^c_{\mathrm{BP3}}$ is a lower bound for the customer costs of VRPCC. To transform a feasible solution of (BP4) to (BP3), let $x_{it} = \sum_{k \in m} x_{itk}$ and $y_{it} = \sum_{k \in m} y_{itk}$. Hence, the objective value does not change and the constraints (6.23) and (6.24) are valid. Further, the binarity of x_{it} and y_{it} is ensured because of (6.30) and (6.31), respectively. For each $t \in T$, from constraints (6.27) of (BP4) follows that

$$\sum_{k \in M} \sum_{i \in N_c} (x_{itk} a_i + y_{itk} r_i) = \sum_{i \in N_c} (x_{it} a_i + y_{it} r_i) \leq \sum_{k \in M} u = mu.$$

Thus, constraints (6.20) of (BP3) are valid. And from (6.28), it results that

$$\sum_{k \in M} \sum_{i \in N_c} (x_{itk} - y_{itk}) = \sum_{i \in N_c} (x_{it} - y_{it}) \leq \sum_{k \in M} 1 = m \quad \forall t \in T$$

which means that (6.21) is also true. Finally, for each $i \in N_c$ and $t \in T$, constraints (6.29) lead to

$$\sum_{k \in M} (y_{itk} - x_{itk}) = y_{it} - x_{it} \leq 0.$$

Thus, also (6.22) is satisfied. In summary follows that each feasible solution of (BP4) can be transformed into a feasible solution of (BP3) and $\mathrm{LB}^c_{\mathrm{BP3}} \not> \mathrm{LB}^c_{\mathrm{BP4}}$. But $\mathrm{LB}^c_{\mathrm{BP3}}$ and $\mathrm{LB}^c_{\mathrm{BP4}}$ are not equal because it is not always possible to split up the jobs of a bin of (BP3) into m bin of (BP4) associated to the same day. Thus, not each feasible solution of $\mathrm{LB}^c_{\mathrm{BP3}}$ can be transformed into a feasible solution of (BP4). With it, it is shown that $\mathrm{LB}^c_{\mathrm{BP4}} \geq \mathrm{LB}^c_{\mathrm{BP3}}$. □

Theorem 6.7. *It holds that LB^c_{BP3} is not less than LB^c_{BP1}; and LB^c_{BP4} is not less than LB^c_s, LB^c_{BP1}, LB^c_{BP2} and LB^c_{BP3}.*

Proof. Firstly, it is shown that $\mathrm{LB}^c_{\mathrm{BP3}} \geq \mathrm{LB}^c_{\mathrm{BP1}}$. For this purpose, it has to be proven that each feasible solution of (BP3) is also a feasible solution of (BP1). Let $I_t \subset N_c$ be the set of jobs allocated to day $t \in T$ in a solution of (BP3). Thus, $x_{it} = 1$ for each $i \in I_t$. Since constraint (6.20) is satisfied, it holds that

$$\sum_{i \in I_t} (a_i + r_i) \leq mu + \max_{\{\tilde{I}_t \subset I_t | |\tilde{I}_t| = m\}} \sum_{i \in \tilde{I}_t} r_i \leq mu + R_m.$$

Thus, the binary variables x_{it} lead to a feasible solution of (BP1) with same costs. The reversal is not necessarily true because of the increased bin size of $\mathrm{LB}^c_{\mathrm{BP1}}$. Consequently, it follows that $\mathrm{LB}^c_{\mathrm{BP3}} \geq \mathrm{LB}^c_{\mathrm{BP1}}$.

Secondly, it remains to show that $\mathrm{LB}^c_{\mathrm{BP4}}$ is not worse than the other bounds: From Theorem 6.5, it is obtained that $\mathrm{LB}^c_{\mathrm{BP2}} \geq \mathrm{LB}^c_{\mathrm{BP1}}$ and $\mathrm{LB}^c_{\mathrm{BP2}} \geq \mathrm{LB}^c_{\mathrm{s}}$ and from Theorem 6.6 follows that $\mathrm{LB}^c_{\mathrm{BP4}} \geq \mathrm{LB}^c_{\mathrm{BP3}}$. Thus, it remains to show that $\mathrm{LB}^c_{\mathrm{BP4}} \geq \mathrm{LB}^c_{\mathrm{BP2}}$. Let $\{x_{itk}\}$ and $\{y_{itk}\}$, with $i \in N_c$, $t \in T$ and $k \in M$, be a feasible

solution of (BP4). The goal is to show that this solution can be transformed into a feasible solution of (BP2). At first, for each vehicle exists at most one trip to a job executed on the next day. Recap, $\{r^j_{\max}\}_{j=1,\dots,m}$ are the m largest values of $\{r_i | \, i \in N_c\}$. Then, for each $t \in T$, there exists a mapping from $k \in M$ to $j_{kt} \in M$ with $\bigcup_{k \in M}\{j_{kt}\} = M$ such that

$$\sum_{\substack{i \in N_c \\ x_{itk}=1, y_{itk}=0}} r_i \leq r^{j_{kt}}_{\max} \tag{6.32}$$

for each $k \in M$. This means that for each travel item, which is not packed into the bin of the corresponding working item, a bin of (BP2) can be found, such that this travel item does not exceed the time reserved for traveling outside the working shift. With it, for each bin associated to day $t \in T$ and vehicle $k \in M$, it holds that

$$\begin{aligned} \sum_{i \in N_c} (x_{itk} a_i + y_{itk} r_i) &\leq \sum_{i \in N_c} x_{itk}(a_i + r_i) && (6.33) \\ &\leq \sum_{i \in N_c} (x_{itk} a_i + y_{itk} r_i) + r^{j_{kt}}_{\max} && (6.34) \\ &\leq u + r^{j_{kt}}_{\max}. && (6.35) \end{aligned}$$

Inequality (6.33) shows the transition from (BP4) to (BP2). The formula on the left provides the total size of the items of the bin in (BP4) which is smaller than the total size of the items packed into the bin of (BP2) because there is one additional summand r_i. As shown by (6.32), this additional summand is less than $r^{j_{kt}}_{\max}$ and thus, inequality (6.34) is satisfied. The fact that in (BP4) the bin size is limited to u, leads to inequality (6.35). Consequently, the variables x_{itk} of a feasible solution of (BP4) lead also to a feasible solution of (BP2).

The reversal is not necessarily true. Let I_{tk} be the set of jobs allocated to day $t \in T$ and vehicle $k \in M$ in a feasible solution of $\mathrm{LB}^c_{\mathrm{BP2}}$. Transforming this solution to (BP4) leads to

$$\sum_{i \in I_{tk}} (a_i + r_i) - \max_{i \in I_{tk}} r_i \leq u + r^k_{\max} - \max_{i \in I_{tk}} r_i \lesseqgtr u$$

if the largest travel item is not allocated to the bin. In the most cases, $r^k_{\max}$ is larger than the removed travel item, thus, it is not possible to make a statement whether the sum of the working items and selected travel items is less than, equal or larger than u. Thus, not each feasible solution of (BP2) can be transformed into a feasible solution of (BP4). Consequently, $\mathrm{LB}^c_{\mathrm{BP4}} \geq \mathrm{LB}^c_{\mathrm{BP2}}$. □

Remark For instances with $m\eta_1 < \eta_2$, $\mathrm{LB}^c_{\mathrm{s}}$ can be a better bound than $\mathrm{LB}^c_{\mathrm{BP3}}$. This is caused by the fact that by computing $\mathrm{LB}^c_{\mathrm{BP3}}$, not more than η_2 jobs were allocated to one day, but it is possible to allocate more than $m\eta_1$ jobs to one day because the bin size for day t equals to mu. But if $\eta_2 \leq m\eta_1$, it holds that $\mathrm{LB}^c_{\mathrm{BP3}} \leq \mathrm{LB}^c_{\mathrm{s}}$.

6.2.1.3. Linear Programming Relaxation of Bin Packing Bounds

The computational experiments showed that solving the bin packing problems is to time-consuming in relation to the bound quality, compare the results provided in Section 6.4.1.1. For this reason, applying the corresponding lower bounds to one of the presented branch-and-bound algorithm is not useful. But a fast calculable lower bound for a bin packing problem is its LP relaxation. To obtain an LP relaxation, the integer constraints on the binary variables are removed and the binary variables become positive numbers. With it, items can be packed fractional in different bins and the bins can be completely filled. As it will be shown in Section 6.4.1.1, the LP relaxations are not only faster solvable with CPLEX, but lead also to lower bounds on customer costs with a similar quality as the original bin packing problems. Furthermore, as shown later, the LP relaxation of $\mathrm{LB}^c_{\mathrm{LPBP1}}$ can be calculated with an algorithm that has a complexity of $\mathcal{O}(n_c \log(n_c) + n_c)$.

Theorem 6.8. *The bin packing problems (BP1) and (BP2) have the same LP relaxation value. Also the LP relaxation values of (BP3) and (BP4) are equal.*

Proof. Firstly, it is shown that (BP1) and (BP2) have the same LP relaxation values. For this purpose, let (LPBP1) and (LPBP2) be the LP relaxation of (BP1) and (BP2), respectively. In (LPBP1), one bin of size $mu + R_m$ is associated to day t. And in (LPBP2), m bins are associated to day t and the total size of these m bins is $\sum_{k \in M} \left(u + r^k_{\max}\right) = mu + R_m$. Because of that, each packing of (LPBP2) can be transformed to a feasible packing of (LPBP1) with same costs by defining $x_{it} = \sum_{k \in M} x_{itk}$. And also from each feasible packing of (LPBP1), a feasible packing of (LPBP2) with same costs can be transformed, e.g., by defining $x_{itk} = x_{it} \frac{u + r^k_{\max}}{mu + R_m}$. Consequently, both problems have the same optimal value.

For the LP relaxations of (BP3) and (BP4), named (LPBP3) and (LPBP4), an analog observation is made: In both problems, the total size of the bins associated to day t is mu. In detail, to show that the LP relaxation values are equal, firstly it is observed that $x_{it} = \sum_{k \in M} x_{itk}$ and $y_{it} = \sum_{k \in M} y_{itk}$ transform a feasible solution of (LPBP4) to a feasible solution of (LPBP3) with same costs, compare the proof of Theorem 6.6. Secondly, it is proven that a solution of (LPBP4) can be obtained from a solution of (LPBP3) by defining $x_{itk} = \frac{x_{it}}{m}$ and $y_{itk} = \frac{y_{it}}{m}$. Clearly, the bin size constraints (6.27) of (LPBP4) are valid. For constraints (6.28), it follows from (6.21) of (LPBP3) that

$$\begin{aligned} 1 \overset{!}{\geq} \sum_{i \in N_c} (x_{itk} - y_{itk}) &= \sum_{i \in N_c} (x_{it} - y_{it}) \frac{1}{m} \\ &\leq m \frac{1}{m} = 1. \end{aligned}$$

And from constraints (6.22) of (LPBP3), it follows that (6.29) of (LPBP4) is satisfied, because

$$y_{itk} - x_{itk} = \frac{y_{it}}{m} - \frac{x_{it}}{m} \leq 0.$$

Finally, the constraints (6.30) and (6.31) of (LPBP4) apply because the constraints (6.23) and (6.24) of (LBPP3) are satisfied. Thus, also each feasible solution of (BP3) can be transformed into a feasible solution of (BP4) with same costs. Consequently, both problems have the same optimal value. □

For the branch-and-bound method, lower bounds have to be tight, but it is also important that the bound can be calculated fast. The following theorem shows that the LP relaxation of (BP1) can be calculated by an algorithm with polynomial runtime.

Theorem 6.9. *LB^c_{LPBP1} can be calculated in $\mathcal{O}(n_c \log(n_c))$.*

Proof. For the proof of this theorem, several fractional knapsack problems are solved. A fractional knapsack problem is given as $\max\{x^\top c | \, \mathbf{0} \leq x \leq \mathbf{1}, x^\top w \leq W\}$, see [90], where $c \in \mathbb{N}^n$ is the cost vector, $w \in \mathbb{N}^n$ the weight vector and $W \in \mathbb{N}$ the maximal weight. A solution of the fractional knapsack problem is computed by sorting the items such that $\frac{c_i}{w_i} \geq \frac{c_{i+1}}{w_{i+1}}$ for $i \in \{1, 2, \ldots, n-1\}$ and selecting the first items k of this order until $\sum_{i=1}^{k+1} w_i > W$. Then, a solution is given by the vector $x \in \mathbb{R}^n$ with

$$x_i = \begin{cases} 1 & \text{if } i \leq k, \\ \frac{1}{w_{k+1}} \left(W - \sum_{i=1}^{k} w_i \right) & \text{if } i = k+1, \text{and} \\ 0 & \text{otherwise.} \end{cases}$$

To compute $\mathrm{LB}^c_{\mathrm{LPBP1}}$, for each job $i \in N_c$ an item is defined with costs c_i and weight $w_i = a_i + r_i$. The total weight is $W = mu + R_m$. Following the approach to solve the fractional knapsack problem, the jobs are sorted by $\frac{c_i}{w_i} = \frac{c_i}{a_i + r_i}$ in decreasing order. Then, for each day $t \in T$ in increasing order, the fractional knapsack problem is solved in order to allocate maximal costs to the day. After that, the allocated items are (fractional) removed from the item set. Consequently, for each day $t \in T$, the fractional knapsack problem can be formulated as follows:

$$C_t := \max\{x_t^\top c | \, \mathbf{0} \leq x_t - \sum_{\tau \in T, \tau < t} x_\tau \leq \mathbf{1}, x^\top w \leq W\}. \tag{6.36}$$

Consequently, it holds that $C_\tau \geq C_t$ for $\tau < t$ and further that $\sum_{t \in T} x_t = 1$.

Next, it is shown that with $\{x_t\}_{t \in T}$ an optimal solution of $\mathrm{LB}^c_{\mathrm{LPBP1}}$ is obtained. The corresponding customer cost value is equal to $\sum_{t \in T} t C_t$. Clearly,

$$\sum_{t \in T} C_t = \sum_{i \in N_c} c_i. \tag{6.37}$$

Assuming another solution $\{x'_t\}_{t\in T}$ with $C'_\tau < C_\tau$ for any $\tau \in T$. Then, due to equation (6.37), there must be at most one day $\theta \in T$ with $C'_\theta > C_\theta$. W.l.o.g., let there be only two days with differing costs. Thus, $C'_t = C_t$ for each $t \in T \setminus \{\tau, \theta\}$ and further $C'_\tau = C_\tau - \delta$ and $C'_\theta = C_\theta + \delta$ with $\delta > 0$. If $\tau < \theta$, it holds that

$$\sum_{t\in T} tC'_t = \sum_{t\in T} tC_t + (\theta - \tau)\delta > \sum_{t\in T} tC_t.$$

Thus, solution $\{x'_t\}_{t\in T}$ has a higher objective value than solution $\{x_t\}_{t\in T}$. Otherwise, if $\theta < \tau$, C_θ was not maximal which is a contradiction to its definition given by equation (6.36) for $t = \theta$. Consequently, the costs C_t, $t \in T$ lead to a minimal customer cost value.

Finally, the complexity of solving the fractional knapsack problems is analyzed. The algorithm consists of the subprocesses sorting the jobs by the customer costs per time unit and allocating the items to the bins. Using, e.g., heapsort [33], the complexity of the first subprocess is $\mathcal{O}(n_c \log(n_c))$. Further, using an appropriate implementation, the complexity of the second subprocess is $\mathcal{O}(n_c)$. Consequently, the algorithm to obtain the lower bound $\mathrm{LB}^c_{\mathrm{LPBP1}}$ has a complexity of $\mathcal{O}(n_c \log(n_c))$. □

An pseudo-code of the algorithms to compute $\mathrm{LB}^c_{\mathrm{LPBP1}}$ is given in Algorithm 8 in Appendix B.

From classical bin packing problems, it is known that the LP relaxation has a worst-case performance ratio of $\frac{1}{2}$, see [111]. The following theorem shows, that this observation is also true for the here presented (BP1) and its LP relaxation. For the proof, which is inspired by the proof provided in [111], an instance is constructed where $\mathrm{LB}^c_{\mathrm{LPBP1}}$ is only about a half of $\mathrm{LB}^c_{\mathrm{BP1}}$.

Theorem 6.10. *For $T = \{1, 2, \ldots\}$, the LP relaxation of (BP1) has a worst-case performance ratio of $\frac{1}{2}$.*

Proof. In an optimal solution of $\mathrm{LB}^c_{\mathrm{BP1}}$, at most one bin can be filled by less than a half. Otherwise, if there are two or more bins that are packed by less than half, the items of two such bins can be packed into one bin, which cannot lead to higher costs.

Assuming an instance, where all bins are filled slightly over a half which can only be optimal if all items are slightly larger than the half of the bin size. Then, in the solution of (BP1), to each bin a single item is allocated to. But in the solution of (LPBP1), the items of two bins are packed nearly completely in one bin. To compare the cost values of both solutions, let C_t be the costs of item $i \in N$ allocated to bin $t \in T$ in an optimal solution of (BP1). Then, the optimal value $\mathrm{LB}^c_{\mathrm{BP1}}$ equals to $\sum_{t\in T} tC_t$. In the solution of the LP relaxation, the items of bin one and two are nearly packed together into bin one, the items of bins three and four into bin two,

and so on. Thus, an item, which is in an optimal solution of (BP1) allocated to bin t, is in an optimal solution of (LPBP1) mainly allocated to bin $\lceil \frac{t}{2} \rceil$. Consequently, the cost value can be approximated as $\mathrm{LB}^c_{\mathrm{LPBP1}} = \sum_{t \in T} \lceil \frac{t}{2} \rceil C_t + \varepsilon_t$ where $\varepsilon_t > 0$ is a positive number caused by the fact that the sum of the size of two items is slightly larger than the bin size and a small part of one item has to be shifted to the next bin. Consequently, for such an instance it is

$$\frac{\mathrm{LB}^c_{\mathrm{LPBP1}}}{\mathrm{LB}^c_{\mathrm{BP1}}} = \frac{\sum_{t \in T} \lceil \frac{t}{2} \rceil C_t + \varepsilon_t}{\sum_{t \in T} t C_t} > \frac{\frac{1}{2} \sum_{t \in T} t C_t}{\sum_{t \in T} t C_t} = \frac{1}{2}.$$

□

Even this worst-case performance ratio is small, the computational results provided in Section 6.4.1.1 showed that $\mathrm{LB}^c_{\mathrm{LPBP1}}$ is close to $\mathrm{LB}^c_{\mathrm{BP1}}$.

Remark In general, neither $\mathrm{LB}^c_{\mathrm{LPBP1}} \leq \mathrm{LB}^c_{\mathrm{s}}$ nor $\mathrm{LB}^c_{\mathrm{s}} \leq \mathrm{LB}^c_{\mathrm{LPBP1}}$ hold for all instances.

6.2.2. Lower Bounds for Travel Costs

Lower bounds for travel costs are well known from research concerning the TSP. A collection of several lower bounds for the TSP can be found, e.g., in [128]. To apply TSP bounds to the developed branch-and-bound algorithms, the m routes of a feasible VRPCC solution are joined to one big route, as shown in the two-index models provided in Section 4.3. To this end, the travel costs between the depots z_k and s_{k+1}, $k = 1, \ldots, m-1$, as well as z_m and s_1 are set to zero, and all the other travel costs between depots are set to infinity.

In this section, a small selection of lower bounds for the TSP is presented which were applied in the designed branch-and-bound algorithms: At first, the fast calculated two-neighbor bound is given. In Section 6.2.2.2, two lower bounds are presented which are based on the assignment problem. And in Section 6.2.2.3, two lower bounds are provided that are obtained from minimum spanning trees.

6.2.2.1. The Two-Neighbor Bound

A straightforward lower bound for the travel costs is the two-neighbor bound, see [128]. For the VRPCC, this bound is computed by

$$\mathrm{LB}^d_{\mathrm{2N}} := \left\lceil \frac{1}{2} \left(\sum_{i \in N} \min\{d_{ji} + d_{ik} | \, j \in N \cup N_s, k \in N \cup N_z, j \neq i, k \neq i, k \neq j\} \right. \right.$$
$$\left. \left. + \sum_{i \in N_s} \min\{d_{ij} | \, j \in N\} + \sum_{i \in N_z} \min\{d_{ji} | \, j \in N\} \right) \right\rceil .$$

For each job $i \in N$, the travel costs to its two nearest neighbors are summed up taking into account, that the predecessor and the successor must be different jobs,

that the predecessor must not be an end depot and that the successor must not be a start depot. Also the minimal travel costs to leave the start depots and to reach the end depots are added. In doing so, the costs of $2n + 2m$ edges are summed up. Because a tour in a graph has as many edges as the graph has nodes, see Proposition 2.1, the route through all jobs and all depots must have $n + 2m$ edges. Due to the m edges that connect the end depot z_k with the start depot $s_{[k+1 \pmod m]}$, $k \in M$ have zero travel costs, only $n + m$ edges must be added to compute a lower bound. Consequently, the sum of the travel costs is divided by two and rounded up.

6.2.2.2. Two Bounds from the Assignment Problem

A lower bound for the travel costs of a TSP can be derived from the assignment problem which results from the TSP by relaxing subtour elimination constraints:

$$
\begin{aligned}
\text{(AP)} \quad \mathrm{LB}^d_{\mathrm{AP}} := \min \quad & \sum_{i \in N_a} \sum_{j \in N_a} d_{ij} x_{ij} & \\
\text{s.t.} \quad & \sum_{j \in N_a \setminus \{i\}} x_{ij} = 1 & \forall i \in N_a \\
& \sum_{j \in N_a \setminus \{i\}} x_{ji} = 1 & \forall i \in N_a \\
& x_{ij} \in \{0, 1\} & \forall i, j \in N_a
\end{aligned}
$$

The result of (AP) is a set of cycles. It can be solved in $\mathcal{O}(n^3)$ using the Hungarian method, as shown in [115]. The assignment problem equals to the problem of finding n independent elements in an $n \times n$-matrix with minimal sum where independent means that not two of the selected items are in the same column or in the same row. The idea of the algorithm is to reduce the matrix iteratively until n independent zeros can be selected. Then, the positions of the selected zeros represent an optimal assignment and the optimal value is the sum of the corresponding elements in the original cost matrix.

The assignment bound is known to be weak because it contains several small cycles. An improvement of this lower bound is provided in [27], where an approach is proposed that iteratively adds the costs necessary to connect the cycles. The algorithm starts with the cost matrix. For this matrix, a solution of (AP) is calculated using the Hungarian method. The resulting costs initialize the lower bound $\mathrm{LB}^d_{\mathrm{APC}}$ and the reduced matrix is stored. Then, the cycles of the solution are contracted, which means a single node replaces the nodes of one cycle. For these new nodes, a new cost matrix is computed from the reduced matrix by the minimal costs between the cycles, taking into account the triangle inequality. Based on this new cost matrix, again the assignment problem is solved. The calculated optimal value is added to the current lower bound. If the new solution consists of more than one cycle, the iterative approach continuous with contracting the cycles. Otherwise, the calculated lower bound $\mathrm{LB}^d_{\mathrm{APC}}$ is returned. To control the computational effort, the process can be stopped after a predefined number of iterations. The solution process is shown schematically in Figure 6.3.

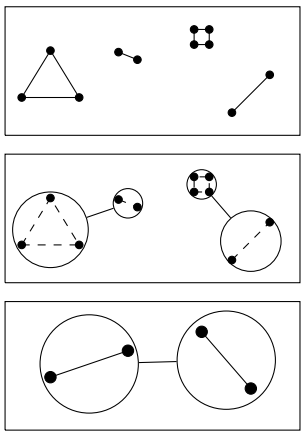


Figure 6.3.: Schematic representation of the solution process of $\mathrm{LB}^d_{\mathrm{APC}}$.

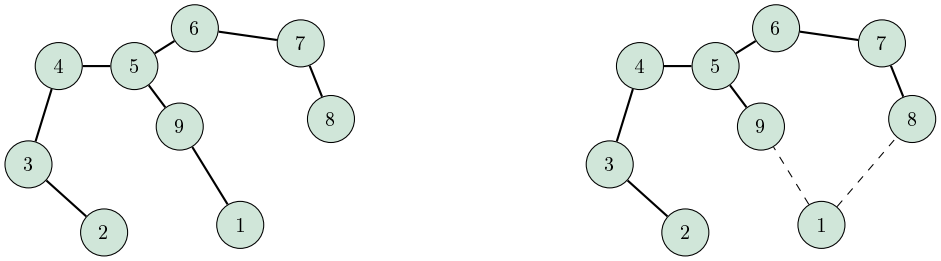
6.2.2.3. Minimum Spanning Tree Based Lower Bounds

Another lower bound for the travel costs of a TSP can be obtained by relaxing the constraints that ensure that each job is connected to exactly two other jobs. Then, a minimum spanning tree problem has to be solved. As shown in Proposition 2.1, a tree has n-1 edges, but a tour has n edges. Because of that, an 1-tree is used, see, e.g., [90, 116], which is a tree-like subgraph with n edges. To get a minimum-weight 1-tree, a minimum spanning tree on all but one vertices is computed and the remaining vertex is connected to this tree by two edges of minimum weight. It can be shown, that a tour is an 1-tree where the tree equals to a path. Because of that, a minimum-weight 1-tree is a lower bound for the weight of a minimal tour which is the solution of the TSP. To improve the 1-tree bound, for each vertex an 1-tree is computed and the maximum of the obtained bound values is returned. The 1-tree bound is named as $\mathrm{LB}^d_{\mathrm{MST}}$.

To calculate a minimum-weight spanning, the algorithm of Jarník/Prim/Dijkstra [126] or the one of Kruskal [93] can be applied, see also [33]. By a suitable implementation and data structure, the complexity of the algorithms for a complete graph with n vertexes can be stated as $\mathcal{O}(n^2)$ and $\mathcal{O}(n^2 \log n)$, respectively. Note, the cost matrix has to be symmetric or an approach for the asymmetric TSP has to be used, e.g., the algorithm to solve the min-sum arborescence problem presented in [53]. To compute $\mathrm{LB}^d_{\mathrm{MST}}$, a symmetric cost matrix $\tilde{D}$ is derived from the cost matrix D by $\tilde{d}_{ij} = \min\{d_{ij}, d_{ji}\}$.

In Figure 6.4, a minimal spanning tree and a minimum-weight 1-tree computed on vertex 1 are shown. For the latter, the two edges to connect vertex 1 are drawn by dashed lines.

To improve the 1-tree bound, an iterative approach was presented in [74], where vertices with degree larger than two are penalized. This is done by setting the edge weight to $w_{ij} = \tilde{d}_{ij} + \pi_i + \pi_j$ where π_i is a penalty for vertex i. Let $W^*(\pi)$ be the



(a) A minimal spanning tree of G.

(b) A 1-tree of G computed on vertex 1.

Figure 6.4.: Example for an 1-tree.

minimal weight of an 1-tree with penalty vector π, then $W^*(\pi) - 2\sum_{i \in N_a} \pi_i$ is a lower bound of the travel costs for the TSP. Thus, the best lower bound is given by $\max_\pi W^*(\pi) - 2\sum_{i \in N_a} \pi_i$, as shown in [74]. To approximate the optimal bound value, the penalty vector π is changed iteratively. Let d_i be the degree of vertex i in the 1-tree computed in step $k \in \mathbb{N}$. Then, the penalty for the next iteration is set to $\pi_i^{k+1} = \pi_i^k + t(d_i - 2)$, where t is an appropriately chosen step size. This reduces the weight of edges incident to a vertex with degree 1 and increases the weight of edges incident to a vertex with degree greater 2. The improved 1-tree bound is referred to by $\mathrm{LB}^d_{\mathrm{iMST}}$.

6.3. Two Branching Strategies

To design an efficient branch-and-bound algorithm, a suitable branching strategy has to be found. For the VRPCC, two branching strategies are developed in Section 6.3.1 and 6.3.2. Both are based on the partition and permutation model (PP) provided in Section 3.2. A new node is generated based on a partial solution $\widetilde{S}$ which is defined by the set of unplanned jobs $\widetilde{N}$ and the uncompleted schedule $(\widetilde{N}_k, \Pi^k(\widetilde{N}_k))_{k \in M}$, where $\widetilde{N} \cup \left(\bigcup_{k \in M} \widetilde{N}_k\right) = N$. To reduce the storage effort, both branching strategies are implemented as depth-search with backtracking.

6.3.1. Branching Strategy Append

The first developed branching strategy is to build-up the routes successively by appending one job to the end of a route. Then, it can be expected that tight lower bounds can be found for the partial solutions, since travel costs and customer costs of the so far appended jobs are fixed. Furthermore, the costs of a partial solution can be determined very effectively because only the start time of the appended job has to be calculated. To avoid multiple solutions (which will otherwise occur very often), the routes are filled with jobs one after the other.

Let $(\widetilde{N}_k, \Pi^k(\widetilde{N}_k))_{k \in M}$ be the partial solution of the current node and let k' be the non-empty route with largest index. Then, the routes $1, 2, \ldots, k'-1$ are called closed

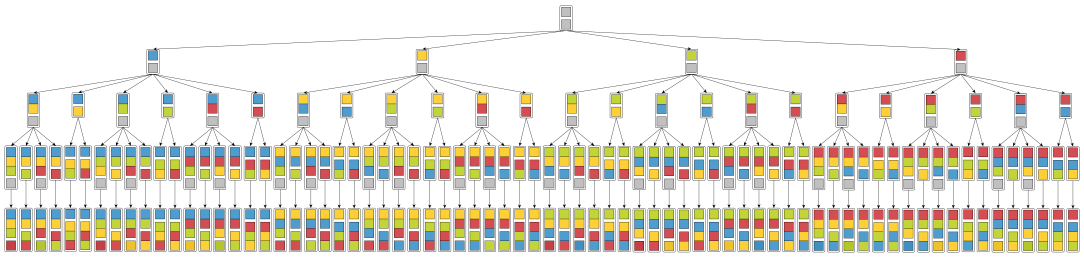


Figure 6.5.: Example of the search tree for branching strategy Append.

and the routes $k', k'+1, \ldots, m$ are called open. Note, that the routes $k'+1, \ldots, m$ are still empty. For this node, child nodes are generated by appending each unplanned job $i \in \widetilde{N}$ on the one hand at the end of route k' and on the other hand, if exists, as first job at route $k'+1$. This approach is similar to the branching strategy presented in [89]. In the following, the branch-and-bound algorithm with branching strategy Append is called BB-Append algorithm.

The search tree resulting from this branching strategy is illustrated in Figure 6.5 on a small example with four jobs and two routes. Each node of the search tree is related to a (partial) solution. In Figure 6.5, on each node the partial solution is drawn by boxes of different colors that represent the jobs. The order of the boxes from top to bottom equals to the job order of route one and route two. An empty route is represented by a gray box.

As it can be seen in Figure 6.5, the structure of the search tree is unfavorable: In the branching steps on nodes with small depth, many more child nodes are generated than on nodes with a large depth: From the root node, which has depth zero, n child nodes are created: one for each job. And for a node with depth k, for each so far unplanned job one or two child nodes are generated. One where the job is appended to the current route and, if possible, a second where the job is appended to the next route as first job. Consequently, for each node with depth k, $n-k$ or $2(n-k)$ new nodes are produced. And for nodes with depth n-1, where a single job is unplanned, only one child node has to be created. This leads to a high number of partial solutions compared to the number of feasible solutions. For example for an instance with four jobs and two routes, the search tree has 101 internal nodes but only 72 leaves, see Figure 6.5. Furthermore, the tree is unbalanced. On nodes, where the routes $1, 2, \ldots, m-1$ are closed, only half of child nodes are generated, as it is not possible to append a job to route $m+1$. This can also be seen in the small example of Figure 6.5 by comparing two nodes with depth two: One with both jobs appended to route one, which has four child nodes and corresponds to four feasible solutions. And another with one job appended to route one and one job appended to route two, which has only two child nodes and two feasible solutions. Due to pruning is mostly possible on deeper nodes, this structure of the search tree leads to the effect that by eliminating a partial solution often only a small subspace of the solution space is removed. With it, perhaps more nodes have to be analyzed compared to a better structured search tree.

6.3.1.1. Implementation

In the following, the branch-and-bound steps as introduced in Section 6.1 are explained for the BB-Append algorithm.

Initialize: The BB-Append algorithm gets as input the jobs in a predefined order N_{sort}, i.e., the jobs are sorted by the customer cost coefficient in decreased order. At first, an upper bound is determined by help of a greedy heuristic. In detail, a schedule is calculated with the BoG heuristic and improved by the best improvement heuristic. Then, the upper bound UB is set to the costs of this schedule.

After initializing, the first node is generated with the set of unplanned jobs $\widetilde{N} = N \setminus \{j_1\}$ and the partial solution $\widetilde{S} = (\{j_1\}, (j_1), \emptyset, (), \ldots, \emptyset, ())$ which is a uncompleted schedule where job j_1 is the only planned job, scheduled as first job of route one.

Select: The branching strategy Append is implemented as backtracking [141]. Consequently, in each step a single node is present and the selection step is not required.

Bound: In this step, the lower bound for the current node is computed which is the base to decide whether it should be further analyzed or pruned. Computing the lower bound of a node consists of at most three substeps:

1. Check, whether the corresponding partial solution can lead to a feasible solution. If not, the lower bound is set to infinity.

2. Otherwise and if all jobs are planned, a leaf of the tree is reached. The cost value is calculated and compared with the upper bound UB. If the costs are smaller, the best solution so far and the upper bound are updated. Due to the node is fully analyzed, the lower bound is set to infinity.

3. If not all jobs are planned, the total lower bound is the sum of the costs for the jobs planned so far $g^c(\widetilde{S})$ and the lower bounds computed on the unplanned job LB^c and LB^d.

   To compute $g^c(\widetilde{S})$, the start times of all planned jobs are determined to obtain the customer cost value and the travel costs between planned jobs and depots are summed up whereby the end depots of the open routes are ignored. Note that if the current node is created as child node of the last analyzed node, only the costs for the new appended job have to be added to the cost value of the last node.

   Only if the costs of the partial solution $g^c(\widetilde{S})$ are smaller than the upper bound, a lower bounds LB^c and LB^d are computed for the costs that appear if all unplanned jobs are appended.

Branch or Prune: As mentioned, the branching step is designed as backtracking approach. Thus, if the computed total lower bound of the node is below the upper bound, the current branch has to be further investigated and one child node is generated as next node. For this purpose, the first unplanned job j of the sorted jobs

list N_{sort} is searched and appended to the open route with smallest index because the routes are filled one after the other.

If the lower bound is not smaller than the upper bound, the current branch has not to be further investigated and is pruned. Then, a backtracking step is executed which means to follow back the path in the search tree until a node is reached which has a not yet analyzed child node. From this node, one not yet analyzed child node is generated as next node. In detail, let j be the last appended job and k the corresponding route. Then, there are the following four cases:

1. If $k < m$ and more than one job is assigned to route k, job j is shifted to route $k+1$. Consequently, route k is closed and route $k+1$ is the one open route that contains jobs.
   To illustrate backtracking embedded in the BB-Append algorithm, some examples are given and illustrated by Figure 6.6, which shows a part of the search tree based on an instance with job list $N_{\text{sort}} = (j_1, j_2, j_3, j_4)$ and two vehicles. As highlighted by number 1, if the node with the partial solution $\Pi_1 = (j_1, j_2)$, $\Pi_2 = ()$ is pruned, the backtracking algorithm leads to the node with the partial solution $\Pi_1 = (j_1)$, $\Pi_2 = (j_2)$.

2. If job j is the first in route k, shifting job j to the next route would lead to an empty route k because it closes route k. Then, the schedule is infeasible. Thus, job j is not shifted to route $k+1$. Instead, another job j' is searched and appended to route $k-1$ to obtain the next child node of the parent node of the current node. Again, only jobs that are in N_{sort} after job j are allowed. In Example 3 in Figure 6.6, the node with the partial solution $\Pi_1 = (j_1)$, $\Pi_2 = (j_3)$ is pruned. Then, $j = j_3$ and $k = 2$. The next unplanned job from N_{sort} behind job j is the job j_4 which is appended to route $k - 1 = 1$. Thus, the backtracking algorithm produces $\Pi_1 = (j_1, j_4)$, $\Pi_2 = ()$.

3. If $k = m$, no further route can be opened and it is verified whether another unplanned job can be visited instead of job j. To ensure that solutions are not generated twice, only jobs which are in the sorted job list N_{sort} after job j are allowed.
   As illustrated by Example 2 in Figure 6.6, backtracking on the node with solution $\Pi_1 = (j_1)$, $\Pi_2 = (j_2, j_3)$ leads to $\Pi_1 = (j_1)$, $\Pi_2 = (j_2, j_4)$ because job j_3 is replaced by job j_4.

4. However, if $k = m$ or j is the first job and also all jobs that are in N_{sort} after j are allocated to a route of the partial solution, neither job j can be shifted nor another unplanned job can be appended instead of j. In this case, the current node is the lastly analyzed child node of its parent node. Consequently, job j is removed from route k to obtain the parent node and backtracking is repeated. For example, backtracking on the leaf with the partial solution $\Pi_1 = (j_1, j_3)$, $\Pi_2 = (j_4, j_2)$ leads in the first iteration to $\Pi_1 = (j_1, j_3)$, $\Pi_2 = (j_4)$, because neither j_2 can be shifted to a new route nor an unplanned job exists. Also in

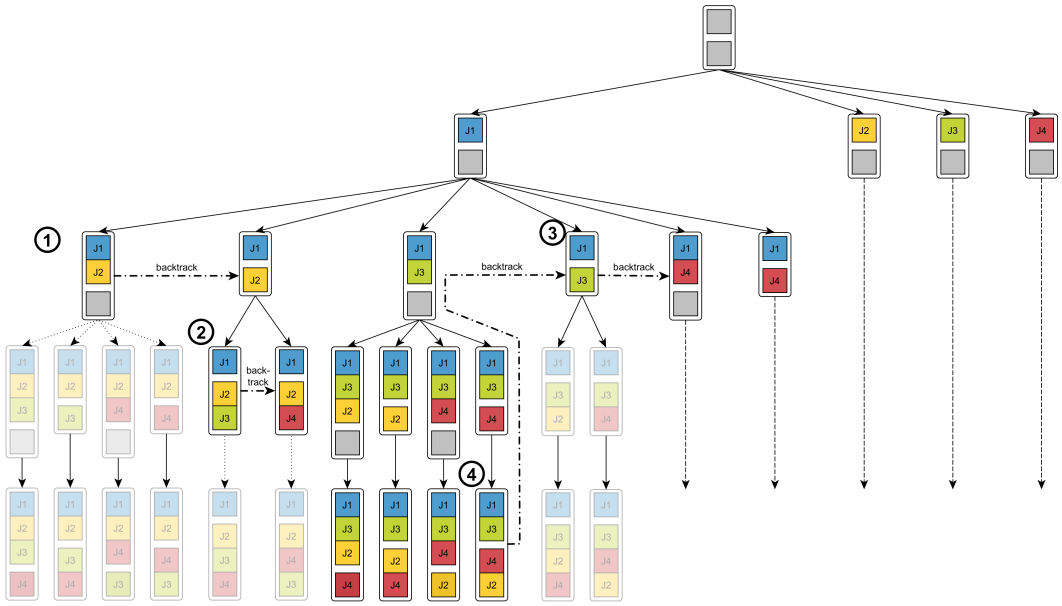


Figure 6.6.: Details of a search tree resulting from branch-and-bound with branching strategy append.

the second iteration the last appended job, which is now j_4, is removed from the schedule because $k = m$ and j_4 is the last job of the list N_{sort}. For the next iteration step, the partial solution is $\Pi_1 = (j_1, j_3)$, $\Pi_2 = ()$. Now, $k = 1 < m$ and the last appended job j_3 is shifted to route $k + 1 = 2$. The uncompleted schedule $\Pi_1 = (j_1)$, $\Pi_2 = (j_3)$ is returned. Following the path back in the search tree until a new node is created is shown as Example 4 in Figure 6.6.

Return optimal solution If backtracking reaches the root node and all child nodes of the root node are analyzed, the branch-and-bound algorithm is finished

In Appendix B, the pseudocode of the BB-Append is shown in Algorithm 9 and the backtracking algorithm applied in the branching step is given in Algorithm 10.

Parallelization

To speed up the solution process, the algorithm is parallelized. For this purpose, the whole search tree is split into several subtrees and for each subtree a task is defined. These tasks can be computed in parallel whereby the number of processors of the used computer defines how many tasks can be computed simultaneously. To use the available computing power optimal, it is aimed that all processors finish their tasks at the same time. Since it is not known in advance how many nodes of a subtree will be analyzed, more tasks then processors are defined. If any processor completes a task, the next from the tasks list is chosen. The tasks with the smallest lower bound of the partial solution are handled first in order to quickly improve the upper bound.

For the parallelized BB-Append, each task is initialized with a partial solutions visiting exactly three jobs. Consequently, for instances with four or more jobs and at least three machines, which means $n \geq 4$ and $m \geq 3$, the number of tasks is $4\frac{n!}{(n-3)!}$, since there are $\frac{n!}{(n-3)!}$ possibilities to select an order of three jobs and four possibilities to allocate the jobs to at most the first three routes, which are

- all jobs to route one,
- the first two jobs to route one and the third job to route two,
- the first job to route one and the second and third job to route two and
- the first job to route one, the second job to route two and the third job to route three.

For example with $n = 15$ and $m = 3$, 10920 tasks are created.

6.3.1.2. Adaptations to Calculate Lower Bounds

To calculate the lower bounds for the BB-Append algorithm, some adaptations are required to benefit from the knowledge of the decisions made so far. Since the already planned jobs are fixed, the lower bound of the node is the sum of the costs of the partial solution $g(\widetilde{S})$, the applied customer cost bound LB^c and the applied travel cost bound LB^d.

By computing a lower bound on customer costs, the time available per day is reduced by considering that some routes are closed and that one open route contains at least one job. Further, only unplanned jobs of N_c have to be considered. To calculate lower bounds on travel costs, depots of closed routes and planned jobs are not taken into account because their travel costs are fixed. However, the last appended job is considered because it serves as start depot of the corresponding route.

The computational experiments showed the customer costs bounds are tighter than the travel cost bounds, see Section 6.4.1. Because of that the lower bound of the customer cost part is calculated first. If the sum of current costs and the lower bound for customer costs is smaller than the upper bound, also the lower bound for the travel cost part is computed and added to the lower bound.

Adaptation for $\mathrm{LB}^c_{\mathrm{s}}$

There are two approaches to adapt the customer cost bound $\mathrm{LB}^c_{\mathrm{s}}$ to the branching scheme Append which differ in the way to calculate the upper bound η_t for the number of jobs executable on day $t \in T$. The resulting lower bounds are the faster computed bound $\mathrm{LB}^c_{\mathrm{s,\,fast}}$ and the more accurate bound $\mathrm{LB}^c_{\mathrm{s,\,accurate}}$.

Recap, the set of unplanned jobs, which are the jobs not allocated to a route in the partial solution, is denoted by $\widetilde{N}$. Let j_l be the lastly appended job and k the corresponding route. Then, the routes $1, 2, \ldots, k-1$ are closed, thus no other job

can be appended to these routes. To route k, further jobs can be appended in the next branching steps which cannot start before day $t_l = t^d_{j_l}$. The routes $k+1, \ldots, m$ are still empty.

For $\mathrm{LB}^c_{\mathrm{s,\,fast}}$, the value η_1 is calculated at the beginning of the process with all jobs with non-zero customer cost coefficient based on the inequalities (6.8) and (6.9). Recap, η_1 is an upper bound for the number of jobs executable per day by one vehicle. Then, η_t for $t \in T$ is calculated as follows:

$$\eta_t = \begin{cases} (m-k)\eta_1 & \text{if } t < t_l, \\ (m-k)\eta_1 + \left\lfloor \left(1 - \frac{t^m_{j_l} + a_{j_l} + r_{j_l}}{u + r^1_{\max}}\right)\eta_1 \right\rfloor & \text{if } t = t_l, \\ (m-k+1)\eta_1 & \text{if } t > t_l. \end{cases}$$

Thus, for each day earlier than the start day of the last appended job j_l, jobs can be appended to only $m-k$ routes. On day t_l, the next job appended to route k cannot start before minute $t^m_{j_l} + a_{j_l} + r_{j_l}$. Because of that, for vehicle k the remaining time has to be added to $(m-k)\eta_1$. And for each day later than t_l, to each of the $m-k+1$ open routes η_1 jobs can be allocated to. The advantage of this approach is that η_1 is calculated only once which reduces the calculation effort. But this affects the accuracy because the minimal travel time to the next job r_i is not restricted to the jobs $N_c \cap \widetilde{N}$ which could lead to larger values for r_i and a reduction of η_t.

In the accurate approach $\mathrm{LB}^c_{\mathrm{s,\,accurate}}$, the values r_i are calculated based on the partial solution by $r_i = \min\{r_{ij} \,|\, j \in N_c \cap \widetilde{N}, j \neq i\}$. Thus, the already planned jobs are not considered. Further, $R_{m-k} = \max\{\sum_{i \in I} r_i \,|\, I \subseteq N_c \cap \widetilde{N}, |I| = m-k\}$ is the time reserved for the trips outside the working shift. Then, following the definition of η_1 via inequalities (6.8) and (6.9), and of η_2 via inequalities (6.10) and (6.11), the values η_t are calculated as follows:

- if $t < t_l$ then $\eta_t = \min\{(m-k)\eta_1, \eta_{2,m-k}\}$, where $\eta_{2,m-k}$ is calculated with $(m-k)u + R_{m-k}$ as time available per day;
- if $t = t_l$ then $\eta_t = \min\{(m-k)\eta_1 + \tilde{\eta}_1, \tilde{\eta}_{2,m-k}\}$, where $\tilde{\eta}_1$ is calculated with $u + \max\{r_i \,|\, i \in N_c \cap \widetilde{N}\} - (t^m_{j_l} + a_{j_l} + r_{j_l})$ as time available, and $\tilde{\eta}_{2,m-k}$ is calculated with $(m-k+1)u - (t^m_{j_l} + a_{j_l} + r_{j_l}) + R_{m-k+1}$ as time available and
- if $t > t_l$ then $\eta_t = \min\{(m-k+1)\eta_1, \eta_{2,m-k+1}\}$, where $\eta_{2,m-k+1}$ is calculated with $(m-k+1)u + R_{m-k+1}$ as time available per day.

This approach leads to a higher calculation effort, because each time the values r_i have to be calculated and the k smallest values have to be determined. But the values η_t are a tighter upper bound for the number of jobs executable on day t since only so far unplanned jobs are considered in the calculations.

Adaptation for $\mathrm{LB}^c_{\mathrm{LPBP1}}$

To calculate $\mathrm{LB}^c_{\mathrm{LPBP1}}$, only unplanned jobs are taken into account. Thus, for job $i \in \widetilde{N} \cap N_c$, the minimal travel time to another job is $r_i = \min\{r_{ij} \,|\, j \in N_c \cap \widetilde{N}, j \neq i\}$

and the time reserved for traveling to the jobs executed on the next day for $m-k$ vehicles equals to R_{m-k}. Based on these definitions, the size B_t of the bin $t \in T$ is calculated as

$$B_t = \begin{cases} (m-k)u + R_{m-k} & \text{if } t < t_l, \\ (m-k+1)u + R_{m-k+1} - \left(t^m_{j_l} + a_{j_l} + \min_{i \in N_c \cap \widetilde{N}} r_{j_l i}\right) & \text{if } t = t_l, \\ (m-k+1)u + R_{m-k+1} & \text{if } t > t_l. \end{cases}$$

Thus, for each day before the start day of the last planned job t_l, the bin size equals to the time for $(m-k)$ vehicles. On each day after day t_l, the bin size equals to the time for $m-k+1$ vehicles. And on day t_l, from the time available for $m-k+1$ vehicles the already planned time of vehicle k is subtracted, which is the start minute of the last job j_l plus its working duration and the travel time to the next job of the set $N_c \cap \widetilde{N}$.

Adaptation for Travel Cost Bounds

By calculating lower bounds for the travel costs of the remaining jobs, only unplanned jobs and open routes have to be considered. Further, the last appended job j_l acts as start depot for route k. Thus, considered are

- the unplanned jobs,
- the last appended job j_l that works as start depot for route k,
- the start depots of the routes $k+1, \ldots, m$, and
- the end depots of the routes $k, k+1, \ldots, m$.

With it, the size of the travel cost matrix decreases with increasing depth in the branch-and-bound tree. As mentioned in Section 6.2.2, the travel costs between the depots z_l and s_{l+1}, $l = k, k+1, \ldots, m-1$, as well as z_m and j_l are set to zero. The other travel costs between depots including j_l are set to infinity. This leads to some asymmetries in the distance matrix, which are not considered in the travel cost bounds based on minimum spanning trees.

The iteratively improved travel cost bounds $\mathrm{LB}^d_{\mathrm{APC}}$ and $\mathrm{LB}^d_{\mathrm{iMST}}$ can be aborted if the bound exceeds $\mathrm{UB} - g(\widetilde{S}) - \mathrm{LB}^c$ with UB is the current upper bound of the branch-and-bound solution process, $g(\widetilde{S})$ is the total cost value of the analyzed partial solution $\widetilde{S}$ and LB^c is the value of the applied customer cost bound calculated for $\widetilde{S}$. Because in this case, the lower bound exceeds the upper bound and the branch will be discarded.

6.3.2. Branching Strategy Include

As shown in Section 6.3.1 based on Figure 6.5, the search tree generated during the BB-Append algorithm is unfavorable and it is expected that only small parts

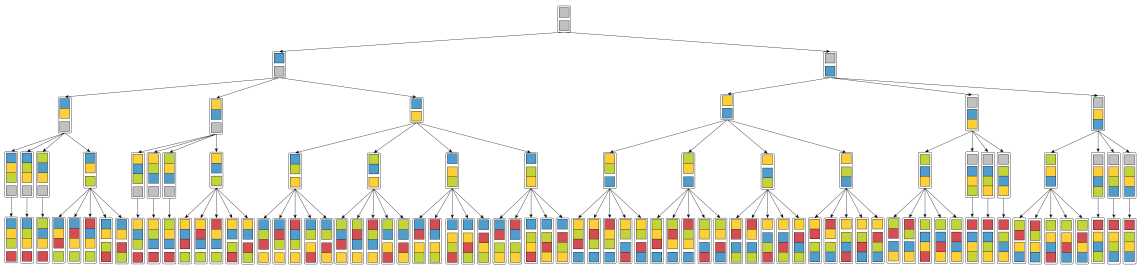


Figure 6.7.: Example of the search tree for branching strategy include.

of the solution space are discarded when a branch is pruned. Because of that, a second branching strategy is developed in this dissertation: With branching strategy include, the child nodes are created by including a certain job on different positions into the routes of the current partial solution.

To be more precise, the first unplanned job of the sorted list is included at the first, the second, till to the last position of each route. Consequently, $|N| - |\widetilde{N}| + m$ branches are generated, because the job can be included after each already planned job, which leads to $|N| - |\widetilde{N}|$ child nodes; and as first job of each route which leads to m further child nodes.

In the following, the branch-and-bound algorithm with branching strategy Include is called *BB-Include* algorithm.

The resulting search tree is illustrated in Figure 6.7 based on an instance with four jobs sorted as $N_{\text{sort}} = \{j_1, j_2, j_3, j_4\}$ and two vehicles. As it can be seen on this small example, the search tree generated by the BB-Include algorithm is better structured compared to the search tree of the BB-Append algorithm: For the instance of Figure 6.7, only 33 branching nodes are generated to get all 72 leaves, instead of 101 branching nodes when branching strategy Append is used. The reason for this better structure is that on nodes deeper in the search tree more branches are generated than on nodes close to the root. For the example shown in Figure 6.7, from the root node with the empty schedule two branches are generated: Including job j_1 to route one and to route two, respectively. For each of both tree nodes with depth one, three new branches are generated including job j_2 on each possible position which is before job j_1, after job j_1 and in the other route. And for each node with depth two, four new branches are generated independent from the structure of the corresponding partial solution. This can be seen in Figure 6.7 comparing, e.g., branching on the node on the left where job j_1 and j_2 are assigned to route one, and on the node which is the third from the left where job j_1 and j_2 are allocated to different routes. Finally, branching on nodes with depth three leads either to five leaves or, if one route is still empty, to only one leave. Then, for the instance of Figure 6.7, in the search tree resulting from branching strategy include, 16 or 20 feasible solutions belong to a node with depth two, instead of two or four feasible solutions when the BB-Append algorithm is applied. Consequently, pruning a node on a certain level of the search tree discards more solutions than pruning a node of the same depth in the search tree corresponding to branching strategy append.

However, since jobs are inserted within the routes, the costs for the already planned jobs are not fixed. For a planned job, the start day can be shifted backward, if jobs are inserted before its position in the route. And also the travel costs of a planned job can change, if a new predecessor or successor is included. Because of that, all jobs have to be taken into account by computing lower bounds and the computational effort does not decrease with increasing depth of the search tree. Furthermore, it is expected that the lower bounds are less tight than the lower bounds computed within the BB-Append algorithm.

In contrast, in the BB-Append algorithm, the costs of the planned jobs are fixed and the lower bounds are only computed for the unplanned jobs. Consequently, in the BB-Append algorithm, the computational effort for lower bounds decreases with increasing depth in the search tree and it is expected that on deeper nodes better lower bounds are obtained because fewer feasible solutions regards to the branch of a node.

6.3.2.1. Implementation

In this section, implementation details are given for the branch-and-bound steps of the BB-Include algorithm.

Initialize: The input of the BB-Include algorithm consists of a sorted job list $N_{\text{sort}} = (j_1, j_2, \ldots, j_n)$ and the depot sets N_s and N_z. In N_{sort}, the jobs are sorted by the customer cost coefficient in decreased order. This has two reasons: Firstly, if all jobs of N_c are planned, the lower bound on customer costs does not need to be calculated. And secondly, the lower bound on the customer cost value of an unplanned job can be very small because, theoretically, each job can be inserted as first job of a route. But for a planned job, its customer cost value cannot become smaller than the current value. Consequently, the more jobs with non-zero customer costs are planned, the better is the lower bound on customer costs.

To obtain an upper bound, a schedule is determined by a heuristic and the corresponding cost value is the upper bound UB. The first node is generated with the uncompleted schedule where job j_1 is the only job visited by vehicle one.

Select: The BB-Include algorithm is implemented as depth-search with a backtracking algorithm. Then, always a single not yet analyzed node exists and the selection step is needless.

Bound: In this step, the lower bounds are computed which are needed to decide whether the current branch should be further analyzed or pruned. Similar to the BB-Append algorithm, this step consists of three substeps:

1. Firstly, it is checked whether the corresponding partial solution can lead to a feasible solution. If this is not possible, i.e., because there are fewer jobs unplanned than routes are empty or $d_{\max}$ is exceeded, the lower bound is set to infinity.

2. Otherwise and if all jobs are planned, the cost value can be calculated and compared with the upper bound UB. If the current solution $\widetilde{S}$ is an improvement,

i.e., $g(\widetilde{S}) < \text{UB}$, the best solution so far and the upper bound are updated. Due to the node is fully analyzed and further branching is not possible, the lower bound is set to infinity.

3. Also if not all jobs are planned, the costs of the partial solution are determined. If these costs are not smaller than the upper bound, the lower bound is set to infinity because completing the current partial solution cannot lead to less costs. Otherwise, the lower bound of the node is the sum of a lower bound for travel costs and a lower bound for customer costs, which are calculated taking into account all jobs and considering that the unplanned jobs can be included inside the current routes.

Branch or Prune: If the lower bound is less than the upper bound UB, the branch can contain a solution better than the best so far and has to be further analyzed. Then, the first unplanned job of the list N_{sort} is included as last job of route one to create a child node.

Otherwise, the branch is not further investigated and a new node is generated via backtracking: Let j be the last included job, k the corresponding route and p its position in the route. Note, that the last included job always equals to the $(|N|-|\widetilde{N}|)$-th job of N_{sort} because the jobs are included strictly in the order provided by N_{sort}. In Figure 6.8, examples for backtracking are shown for an instance with four jobs and two machines. There are three cases:

1. If $p > 1$, the job j is brought forward to position $p-1$.
   As shown by Example 1 in Figure 6.8, backtracking on the node with partial solution with $\Pi_1 = (j_1, j_2, j_3)$ and $\Pi_2 = ()$ leads to $\Pi_1 = (j_1, j_3, j_2)$ and $\Pi_2 = ()$ because j_3 is brought forward by one position.

2. If job j is the first one of route k and if $k < m$, it can be shifted to the next route and included as last job of route $k+1$.
   The example for this case is highlighted by number two in Figure 6.6. As it can be seen, backtracking on the leaf with solution $\Pi_1 = (j_4, j_1, j_2)$ and $\Pi_2 = (j_3)$ leads to the leaf with solution $\Pi_1 = (j_1, j_2)$ and $\Pi_2 = (j_3, j_4)$.

3. Otherwise, all child nodes of the current parent node are analyzed. In this case, job j is removed to generate the current parent node and from this backtracking is repeated until a new partial solution was generated or all jobs are removed.
   For this case, an example is shown in Figure 6.8 highlighted by number three. Starting from the node with $\Pi_1 = (j_1, j_2)$ and $\Pi_2 = (j_4, j_3)$, at first, job j_4 is removed because it is the first job of route m which means that all possible positions of job j_4 are already analyzed. The obtained parent node has the partial solution $\Pi_1 = (j_1, j_2)$ and $\Pi_2 = (j_3)$. As it can be seen, all possibilities to include job j_3 are analyzed. Consequently, again the parent node is generated by removing job j_3. Now, the current node is the one with partial solution $\Pi_1 = (j_1, j_2)$ and $\Pi_2 = ()$. Its parent root has child nodes

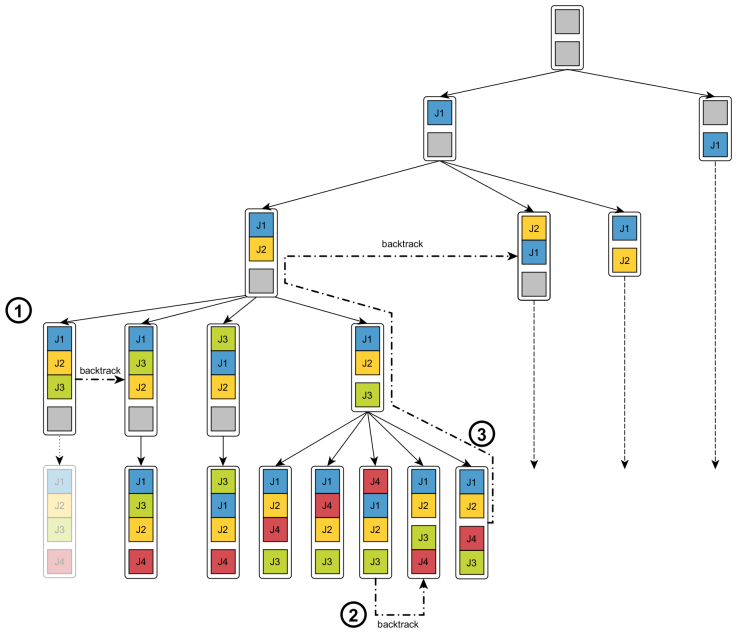


Figure 6.8.: Details of a search tree resulting from branch-and-bound with branching strategy include.

which are not proceeded so far. According to the backtracking algorithm, job j_2 can be brought forward in the corresponding route and the next node is the one with partial solution $\Pi_1 = (j_2, j_1)$ and $\Pi_2 = ()$.

Return optimal solution: If backtracking reaches the root node and all child nodes of it are analyzed, the BB-Include algorithm is finished.

In Appendix B, the pseudocode of the BB-Include algorithm is shown in Algorithm 11 and the corresponding backtracking method is given in Algorithm 12. With this procedure, the solutions of an instance with four jobs and two routes are generated in the order shown in Figure 6.7 from left to right.

Parallelization

Also the BB-Include algorithm can be parallelized by splitting the search tree into disjunctive subtrees. To optimally use the available computing power, it is aimed that all processors finish their tasks at the same time. Because the computational effort to analyze a subtree depends on many factors, significantly more tasks then processors are defined. Then, a new task can be taken from the task list if any processor completes a task. To define the tasks, all nodes of the search tree of a certain level d are generated which are all possible partial solutions where the first d jobs of N_{sort} are planned. The depth d is chosen dependent on the number of vehicles such that at least 500 tasks are produced which is a sufficient large number of treads to appropriately utilize the processors. Therefore, the number of nodes on level d is calculated which is $\prod_{j=1}^{d}(m+j-1)$. Given this, a suitable value for d can

be chosen. For example, for an instance with three vehicles, d is set to five to obtain 2520 tasks. In hope to improve the upper bound faster, the tasks are sorted by the lower bound of the corresponding partial solution in increasing order and computed in this order.

6.3.2.2. Adaptations to Calculate Lower Bounds

To calculate lower bounds for the BB-Include algorithm, all jobs have to be taken into account because jobs can be included between already planned jobs. The lower bound of the branch is then the sum of the customer cost bound and the travel cost bound both calculated based on all jobs. Let $N_p \subset N_c$ be the set of jobs allocated to a route in the partial solution. These jobs have a day for their execution in the partial solution, which is $t_i^d,\ i \in N_p$. In the further solution process, these jobs can only be shifted behind the current start time. Thus, the job i cannot be executed earlier than t_i^d. Further, for these jobs only the travel costs to the predecessor and successor in the route of the partial solution and to all unplanned jobs have to be considered.

In the BB-Include algorithm, at first the jobs with non-zero customer cost coefficient are included into the routes to know the minimal customer costs as soon as possible. If all these jobs are planned, the lower bound of the customer cost value equals to the customer cost value of the uncompleted schedule. Consequently, the lower bounds for customer costs are only calculated if any job $i \in N_c$ is unplanned. If the lower bound of the customer cost plus the travel costs of the partial solution are below the upper bound UB, a bound for the travel cost value is determined.

Adaptation for $\mathbf{LB_s^c}$

To calculate the lower bound of the customer costs of all solutions corresponding to the analyzed node, additional constraints are added which ensure that already allocated jobs are only shifted behind the current start day. The maximal number of jobs per day η is calculated as described in Section 6.2.1.1, Lemma 6.2. The optimization problem is then

$$
\begin{aligned}
\mathrm{LB}^c_{\text{s, include}} := \min & \sum_{i \in N_c} c_i \tau_i & & \\
\text{s.t. } & \tau_i \in T & & i \in N_c \\
& |\{i \in N_c |\, \tau_i = t\}| \leq \eta & & t \in T \\
& t_i^d \leq \tau_i & & i \in N_c \cap N_p
\end{aligned}
$$

To solve this optimization problem, the jobs are sorted by its customer cost coefficient in decreasing order. The start time of already unplanned jobs is set to $t_i^d = 0$, $i \in N_c \setminus N_p$. Then, the first η jobs with $t_i^d \leq 1$ are allocated to day 1 and removed from the list. From the remaining jobs, again the first η jobs with $t_i^d \leq 2$ are allocated to day 2 and removed. This is repeated, until all jobs are allocated. The

computation effort is $\mathcal{O}(|N_c| \log(|N_c|))$ to sort the jobs and $\mathcal{O}(|T||N_c|)$ to allocate the jobs to days.

Adaptation for $\mathbf{LB^c_{LPBP1}}$

To capitalize from the knowledge of the partial solution, also for the lower bound obtained from the LP relaxation of (BP1), additional constraints are included to ensure that jobs which are planned in the current partial solution are only shifted behind. These additional constraints lead to the following bin packing problem:

$$
\begin{aligned}
\mathrm{LB}^c_{\text{LPBP1, include}} := \min \quad & \sum_{i \in N_c} \sum_{t \in T} c_i t x_{it} & & \\
\text{s.t.} \quad & \sum_{t \in T} x_{it} = 1 & & i \in N_c \\
& \sum_{i \in N_c} x_{it}(a_i + r_i) \leq mu + R_m & & t \in T \\
& x_{it} \geq 0 & & i \in N_c,\ t \in T \\
& x_{it} = 0 & & i \in N_c \cap N_p,\ t \leq t^d_i \in T
\end{aligned}
$$

To obtain a solution in polynomial time, the solution procedure presented in Section 6.2.1.3 is applied with a small change: To fill a bin t, only the items of jobs with $t^d_i \leq t$ are taken into account. For this purpose, for currently unplanned jobs $i \in N_c \setminus N_p$, the start day is set to $t^d_i = 0$.

Adaptation to Calculate Travel Cost Bounds

By defining the travel cost matrix for a partial solution generated insight the BB-Include algorithm, all jobs are taken into account. Thus, the size of the travel cost matrix is always $n + 2m$. Let $N_p := \bigcup_{k \in M} \widetilde{N}_k$ be the set of the already planned jobs. For these jobs, only travel costs between consecutive jobs have to be considered. Because currently not planned jobs could be allocated at each position of the current routes, for these jobs the travel costs to all other jobs have to be considered. For the depots, the rules to transform the m routes to one large route are applied. In summary, the following travel costs are used:

$$
\tilde{d}_{ij} = \begin{cases}
\infty, & \text{if } i = j \in N_a, & \text{(6.38a)} \\
d_{ij}, & \text{if } i \in N_p \cup N_s,\ j \in N_p \cup N_z \text{ and } i \text{ is the predecessor of } j, & \text{(6.38b)} \\
\infty, & \text{if } i \in N_p \cup N_s,\ j \in N_p \cup N_z \text{ and } i \text{ is not the predecessor of } j, & \text{(6.38c)} \\
d_{ij}, & \text{if } i \in N_p \cup N_s, \text{ and } j \in N \setminus N_p, & \text{(6.38d)} \\
d_{ij}, & \text{if } i \in N \setminus N_p \text{ and } j \in N_p \cup N_z, & \text{(6.38e)} \\
d_{ij}, & \text{if } i, j \in N \setminus N_p,\ i \neq j, & \text{(6.38f)} \\
\infty, & \text{if } i \in N_s,\ j \in N_z, & \text{(6.38g)} \\
0, & \text{if } i = z_k,\ j = s_{[k+1 \pmod m]} \text{ and } k \in M, & \text{(6.38h)} \\
\infty & \text{if } i = z_k,\ j = s_l,\ k, l \in M \text{ and } l \neq k + 1 \pmod m. & \text{(6.38i)}
\end{cases}
$$

The travel costs $\tilde{d}_{ii}$ are set to infinity by equation (6.38a) because these edges are irrelevant in calculating travel cost bounds. If the jobs or depots i, j are visited consecutively in the current solution with i is the predecessor of j, the travel costs d_{ij} have to be considered, as defined by equation (6.38b). But for planned jobs or depots which are not visited consecutively, the travel costs between them are not taken into account and set to infinity, see (6.38c). By means of equation (6.38d) and (6.38e), the travel costs between an unplanned job and a planned job or depot are taken into account. And of course also the costs between two unplanned jobs $i, j \in N \setminus N_p$ are used as defined in (6.38f). Finally, travel costs between depots are defined. The travel costs from each start depot to an end depot are set to infinity by equation (6.38g) because empty routes are forbidden. To connect the m routes to one large route, the costs between the end depot of route $k \in M$ and the start depot of the subsequent route $k+1 \pmod m$ are set to zero via (6.38h) and all other travel costs between an end depot and a start depot are set to infinity by equation (6.38i). Note that the resulting travel cost matrix contains more asymmetries than a cost matrix obtained by computing a travel cost bound for a node in the BB-Append algorithm. Because of that, probably the travel cost bounds based on the assignment problem could be better than the bounds based on the minimum-weight 1-trees since for the computation of the minimal spanning tree, not symmetric entries are replaced by its minimum.

6.4. Computational Results

In this section, it is analyzed, how efficiently the VRPCC is solved by means of the presented branch-and-bound algorithms. For this purpose, at first the lower bounds are compared in terms of bound quality and computational time because for the branch-and-bound method, tight and fast computable lower bounds are needed. The results are presented in Section 6.4.1.1, where the customer cost bounds are analyzed, and Section 6.4.1.2 that compares the presented travel cost bounds. After that, the performance of the branch-and-bound algorithms for selected lower bounds is compared in order to find the best variant. Thereby, the two branching strategies are analyzed separately in the Sections 6.4.2.1 and 6.4.2.2. Finally, in Section 6.4.2.3, the best variant of both developed branch-and-bound algorithms are compared with the best variant of solving the VRPCC with the commercial solver CPLEX on several benchmarks.

To summary the results, initially the BB-Append and BB-Include algorithm, both applying the lower bounds $\mathrm{LB}^c_{\mathrm{LPBP1}}$ and $\mathrm{LB}^d_{\mathrm{APC}}$, are compared with solving (R2dT4) with CPLEX. For this purpose, Table 6.1 provides the statistic values minimum, 25-th, 50-th and 75-th percentile and maximum (denoted by min, Q1, Q2, Q3 and max) for the computational time. Furthermore, the number of not optimally solved instances and the average gap to an optimal value for these instances are given. Note that the branch-and-bound algorithms are initialized with a start solution which is obtained by the BoG algorithm. In contrast, CPLEX is not initialized

	computational time					not	avg
	min	Q1	Q2	Q3	max	opt.	gap
CPLEX	0.6	11.0	70.4	600	600	46	0.03%
BB-Append	0.4	1.3	2.4	5.8	128.7	0	–
BB-Include	0.1	0.7	2.0	6.6	141.3	0	–

Table 6.1.: Comparison of the BB-Append and BB-Include algorithm as well as solving (R2dT4) with CPLEX on benchmark `S`.

with a start solution because this did not improve the performance. The results show that the smallest computational times were obtained with the branch-and-bound algorithms. With the BB-Append and the BB-Include algorithm, 75% of the instances were solved in less than 5.8 and 6.6 seconds, respectively, compare Table 6.1, Column Q3. And all instances were optimally solved within the time limit of ten minutes. Instead, solving the MILP (R2dT4) with CPLEX led to significant larger computational times and for more than a fourth of the instances, the time limit was exceeded. To be more precise, 46 instances were not solved within ten minutes. On average, the gap to the optimal solution in these 46 instances was 0.03%, where in 41 instances the gap was zero and the maximal gap of the remaining instances was 0.7%. Consequently, in benchmark `S`, the two branch-and-bound algorithms clearly outperformed solving (R2dT4) with CPLEX.

6.4.1. Comparison of Lower Bounds

In this subsection, the lower bounds for customer costs and for travel costs are compared with each other in computational experiments. For this purpose, the bounds were computed for the instances of benchmark `L`, each with 100 jobs, because in these large instances, differences in bound quality should be more obviously. Furthermore, the bounds were calculated for the 180 instances of benchmark `S` and compared with the minimal customer cost or travel cost value to evaluate the quality of the bounds. The time to compute a lower bound was limited to 60 seconds.

Firstly, the main result will be summarized. For this purpose, the LP relaxation bound of the MILP (R2dT4) is compared with different sums of a customer cost bound and a travel cost bound to evaluate the quality of the total bound. In Figure 6.9, performance profiles for the gap of the total bound value to the costs of an optimal solution or the best bound are shown based on benchmark `L` and benchmark `S`. Table 6.2 provides statistic values for the computational times. Compared are

- the LP relaxation value,
- the sum of the bounds $\mathrm{LB}^{c}_{\mathrm{LPBP1}}$ and $\mathrm{LB}^{d}_{\mathrm{APC}}$, which showed the best performance applied to one of the branch-and-bound algorithms,

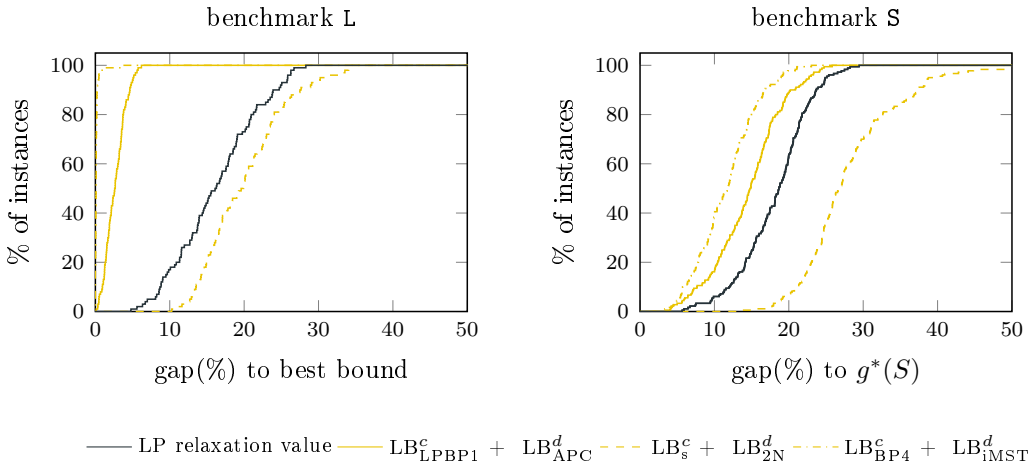


Figure 6.9.: Comparison of lower bounds for total costs.

	benchmark L				
bound	min	Q1	Q2	Q3	max
LP relaxation	158	178	188	208	289
$LB^c_{LPBP1} + LB^d_{APC}$	7	10	12	16	101
$LB^c_s + LB^d_{2N}$	< 1	< 1	< 1	1	2
$LB^c_{BP4} + LB^d_{iMST}$	669	60194	60376	60722	62398

	benchmark S				
bound	min	Q1	Q2	Q3	max
LP relaxation	< 1	2	10	10	39
$LB^c_{LPBP1} + LB^d_{APC}$	< 1	< 1	< 1	< 1	4
$LB^c_s + LB^d_{2N}$	< 1	< 1	< 1	< 1	1
$LB^c_{BP4} + LB^d_{iMST}$	11	60	149	303	1408

Table 6.2.: Computational times for lower bounds for total costs (in milliseconds).

- the sum of the bounds $\mathrm{LB}^c_{\mathrm{s}}$ and $\mathrm{LB}^d_{\mathrm{2N}}$, which were the two fasted computed lower bounds, and
- the sum of the bounds $\mathrm{LB}^c_{\mathrm{BP4}}$ and $\mathrm{LB}^d_{\mathrm{iMST}}$, which were the two bounds with highest bound values.

As it can be seen in Figure 6.9 and Table 6.2, the lower bound derived from $\mathrm{LB}^c_{\mathrm{LPBP1}}$ and $\mathrm{LB}^d_{\mathrm{APC}}$ is tighter than the LP relaxation bound and faster computed. A better total bound was achieved summing $\mathrm{LB}^c_{\mathrm{BP4}}$ and $\mathrm{LB}^d_{\mathrm{iMST}}$. However, the computational time was drastically greater. With the two fastest computed lower bounds $\mathrm{LB}^c_{\mathrm{s}}$ and $\mathrm{LB}^d_{\mathrm{2N}}$, the LP relaxation value was not reached.

6.4.1.1. Comparison of Lower Bounds for Customer Costs

In this subsection, the developed lower bounds for the customer cost values are compared. At first, the obtained values for η_1 and η_2, which are two different variants to obtain an upper bound for the number of jobs that can be visited per day, are compared in Table 6.3. Furthermore, η_{H} denotes the maximal number of jobs started at one day in the solution obtained with the BoG heuristic. The first column shows the percentage of instances in which η_1 and η_2 was smaller than the other of both. And the further columns give the statistic values minimum, 25-th, 50-th and 75-th percentile and maximum (denoted by min, Q1, Q2, Q3 and max). In benchmark `S`, for half of the instances, η_2 was a better bound than η_1. For the other half, both bound values were equal. Recap, the advantage of η_2 is that more jobs are considered by computing the upper bound. Comparing the statistic values of η_2 with the number of jobs executed at most per day in a heuristic solution η_{H} shows that the upper bound is about one or two larger. Indeed, η_{H} is not an upper bound. In benchmark `L`, the differences between η_1 and η_2 were more obviously. In 94% of the instances, η_2 was the tighter upper bound. But the gap between η_2 and the maximal number of jobs solved per day in a heuristic solution η_{H} was large.

	better	min	Q1	Q2	Q3	max
			benchmark `S`			
$m\eta_1$	0%	6	8	8	8	12
η_2	50.0%	6	7	8	8	10
η_{H}	–	4	5	6	7	8
			benchmark `L`			
$m\eta_1$	0.02%	18	21	28	35	35
η_2	94.0%	18	19	24	28	32
η_{H}	–	10	11	14	16	20

Table 6.3.: Comparison of η_1 and η_2 based on benchmark `S` and benchmark `L`.

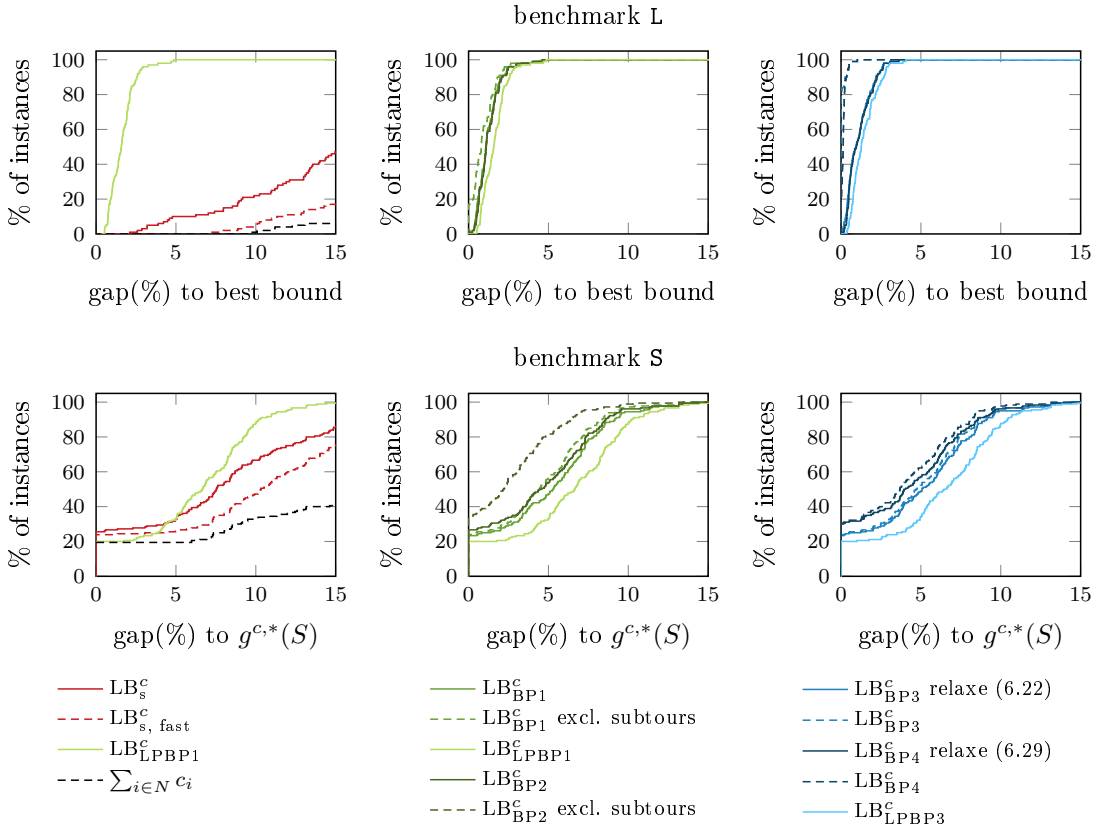


Figure 6.10.: Comparison of the lower bounds on customer costs.

Figure 6.10 shows performance profiles for the gap of the lower bounds for customer costs. The results are presented in six diagrams arranged in two rows and three columns. In the first row, the results obtained on benchmark `L` are shown. There, the bound value is compared with the best bound value because an optimal value is not known. The second row of Figure 6.10 provides the results for the instances of benchmark `S`, where the gap between the bound value and the minimal possible customer cost value $g^{c,*}(S)$ is used. Note, $g^{c,*}(S)$ is obtained by solving the instances minimizing only customer costs. Table 6.4 provides statistic values for the computational time in milliseconds.

The bounds are presented in three groups The first group, shown in the first column of Figure 6.10, contains the very fast computed bounds. In detail, the bounds LB^c_s and $LB^c_{s,\,fast}$, which are presented in Section 6.2.1.1 and Section 6.3.1.2, respectively, are shown together with LB^c_{LPBP1} as introduced in Section 6.2.1.3. Recap, LB^c_s and $LB^c_{s,\,fast}$ are computed based on η, which is a constant upper bound for the number of jobs that can be visited per day, and LB^c_{LPBP1} is calculated by solving the LP relaxation of the bin packing problem (BP1). For these three lower bounds, for the most instances less than a millisecond was needed to obtain the bound value, compare Table 6.4. The performance profiles of $LB^c_{s,\,fast}$ and LB^c_s in Figure 6.10 on the left show that $LB^c_{s,\,fast}$ provided smaller bounds. This is caused by the fact that

	benchmark L				
bound	min	Q1	Q2	Q3	max
LB^c_s	< 1	< 1	< 1	< 1	2
$LB^c_{s,\,fast}$	< 1	< 1	< 1	< 1	2
LB^c_{LPBP1}	< 1	< 1	< 1	1	2
LB^c_{BP1}	14	136	301	661	15593
LB^c_{BP1} excl. subtours	135	59522	60000	60000	60000
LB^c_{BP2}	135	6425	60000	60000	60000
LB^c_{BP2} excl. subtours	44844	60160	60000	60000	60000
LB^c_{BP3}	98	887	2977	9708	60000
LB^c_{BP3} relax constraint (6.22)	96	484	1498	3259	31403
LB^c_{BP4}	784	60000	60000	60000	60000
LB^c_{BP4} relax constraint (6.29)	293	15746	60000	60000	60000
LB^c_{LPBP3}	3	36	100	233	2284

	benchmark S				
bound	min	Q1	Q2	Q3	max
LB^c_s	< 1	< 1	< 1	< 1	1
$LB^c_{s,\,fast}$	< 1	< 1	< 1	< 1	1
LB^c_{LPBP1}	< 1	< 1	< 1	< 1	1
LB^c_{BP1}	3	5	54	70	682
LB^c_{BP1} excl. subtours	2	4	58	129	4972
LB^c_{BP2}	7	11	79	119	2134
LB^c_{BP2} excl. subtours	7	35	232	882	6921
LB^c_{BP3}	6	10	71	101	518
LB^c_{BP3} relax constraint (6.22)	7	10	77	136	403
LB^c_{BP4}	9	57	120	220	983
LB^c_{BP4} relax constraint (6.29)	11	27	148	303	1407
LB^c_{LPBP3}	< 1	1	2	2	4

Table 6.4.: Computational times for lower bounds on customer costs (in milliseconds).

for $\mathrm{LB}^c_{\mathrm{s,\,fast}}$ the number of jobs per day is $m\eta_1$. But in $\mathrm{LB}^c_{\mathrm{s}}$, the number of jobs per day equals to the minimum from $m\eta_1$ and η_2 which leads often to a better approximation because, as shown in Table 6.3, η_2 was often tighter than η_1. Significantly better bounds were obtained with the LP relaxation of (BP1). For all instances of benchmark `L`, the gap of $\mathrm{LB}^c_{\mathrm{LPBP1}}$ was below 5%. And also in benchmark `S`, often (BP1) was more close to the minimal possible customer cost value than $\mathrm{LB}^c_{\mathrm{s}}$ and $\mathrm{LB}^c_{\mathrm{s,\,fast}}$.

The second group consists of the bounds based on the bin packing problems (BP1) and (BP2), where the bin size is increased to reserve time for traveling outside the working shift. In detail, the bounds $\mathrm{LB}^c_{\mathrm{BP1}}$, $\mathrm{LB}^c_{\mathrm{BP1}}$ excluding iteratively unfeasible subsets and the LP relaxation $\mathrm{LB}^c_{\mathrm{LPBP1}}$, as well as $\mathrm{LB}^c_{\mathrm{BP2}}$ and $\mathrm{LB}^c_{\mathrm{BP2}}$ excluding unfeasible subsets are shown. Recap, $\mathrm{LB}^c_{\mathrm{BP2}}$ cannot be smaller than $\mathrm{LB}^c_{\mathrm{BP1}}$ as shown in Theorem 6.5. In benchmark `L`, all variants led to almost the same bound values, but the computational times differ strongly. As it can be seen in Table 6.4, for the most instances solving (BP2) exceeded the time limit of 60 seconds, but the LP relaxation value was obtained in less than one millisecond. In benchmark `S`, $\mathrm{LB}^c_{\mathrm{BP2}}$ was slightly better than $\mathrm{LB}^c_{\mathrm{BP1}}$. Excluding subsets that cannot be packed into one bin resulted in much better bounds. All variants, besides the LP relaxation of (BP1), required much computational time, compare Table 6.4. For the half of the instances of benchmark `S`, the computational time for $\mathrm{LB}^c_{\mathrm{BP1}}$ and $\mathrm{LB}^c_{\mathrm{BP2}}$ exceeds 54 and 79 milliseconds, respectively. The LP relaxation led to the smallest lower bounds but in almost all instances the computational time was less than one millisecond.

And the third group are the bin packing problems $\mathrm{LB}^c_{\mathrm{BP3}}$ and $\mathrm{LB}^c_{\mathrm{BP4}}$, where working duration and travel time are separately packed into the bins. Again, using m bins per day led to a small improvement of the bound value, see Figure 6.10, but to a strong increase of the computational effort as shown in Table 6.4. Relaxing the constraint that working duration and travel time of a certain job have to be packed into the same bin led to similar bound values, but did not lead to a sufficient reduction of the computational time to allow its application to one of the presented branch-and-bound algorithms. Also the LP relaxation of these bin packing problems is not an usable approach. Even the obtained bound values of $\mathrm{LB}^c_{\mathrm{LPBP3}}$ were slightly better than the values of $\mathrm{LB}^c_{\mathrm{LPBP1}}$, the computational effort was too high compared to the improvement of the bound value because it could not be solved by a polynomial algorithm but was solved by CPLEX. During the branch-and-bound algorithms, millions of lower bounds are computed such that 100 milliseconds is to slow to apply the bound.

Consequently, only the bounds of the first group should be applied to branch-and-bound because the calculation of the other bounds was too time-consuming compared to the bound quality. Thereby, $\mathrm{LB}^c_{\mathrm{LPBP1}}$ showed the best bound quality.

6.4.1.2. Comparison of Lower Bounds for Travel Costs

In this subsection, the presented travel costs bounds are compared in computational experiments. At first, different parameter settings to calculate $\mathrm{LB}^d_{\mathrm{iMST}}$ are compared

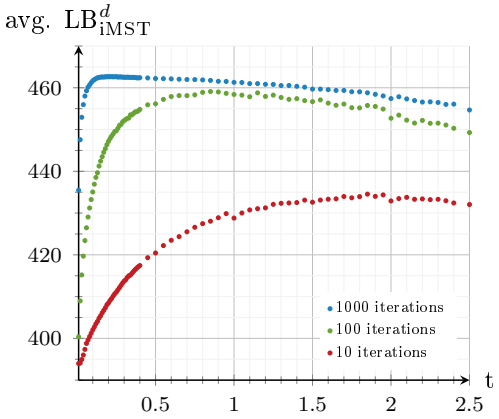


Figure 6.11.: Comparison of different constant step sizes t to compute the travel cost bound LB^d_{iMST}.

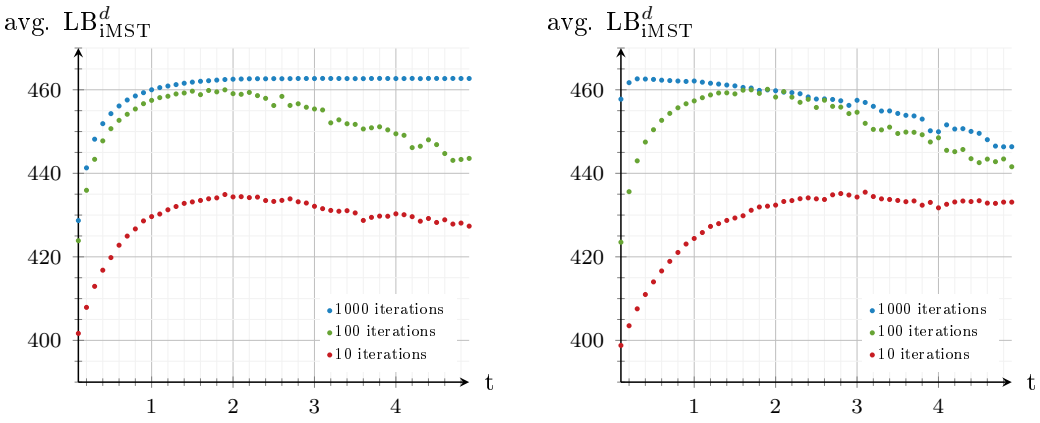
and a setting suitable for the used benchmark is selected. After that, the bounds are compared.

Parameter for Minimum-Weight 1-Tree Bound LB^d_{iMST}

In this paragraph, different settings for the step size t to calculate the improved minimum-weight 1-tree bound LB^d_{iMST} are compared. The computational experiment was done on 60 instances with 15, 20 or 25 jobs.

For the first test, the step size was chosen constant. For the comparison, the bounds obtained with at most 10, 100 and 1000 iterations are analyzed computed with $t \in [0.01, 2.5]$. Figure 6.11 shows for different step sizes the average bound value LB^d_{iMST}. As expected, more iterations led to better bounds especially for small values of t. In case of 1000 iterations, in average the best bounds were obtained with $t \in [0.12, 0.65]$, where the average of LB^d_{iMST} was above 462, which can be seen in Figure 6.11. Reducing the number of iterations to 100 led to slightly smaller bounds. The best average bound was about 459 obtained with $t = 0.85$. With $t \in [0.65, 1.15]$, the average bound exceeded 458. Larger step sizes led to a smaller average bound value. And with only 10 iterations, the obtained bounds were even smaller. Then, the step size had to be chosen higher in order to improve the bound faster. The best results were obtained with $t \in [1.85, 1.95]$, where the average of LB^d_{iMST} exceeded 434, compare Figure 6.11.

In the two plots of Figure 6.12, two variants for a nonconstant step size of t are presented. On the left, a parameter schema with exponentially decreasing step size is shown. There, the step size of the k-th iteration was set to $t_k = 0.985^k t$. The coefficient 0.985 was found by some computational tests. With at most 1000 iterations, $t \in [2, 10]$ led to the best bounds. There, LB^d_{iMST} was in average above 462.5 as shown in Figure 6.12(a). The best average results with 100 iterations were obtained with $t \in [1.3, 2.2]$, where the average of LB^d_{iMST} was about 459, compare Figure 6.12(a). And allowing only 10 iterations performed best with $t \in [1.8, 2.3]$



(a) $t_k = 0.985^k t$ (b) $t_k = t - k\Delta(t)$

Figure 6.12.: Comparison of exponential and linear parameter schemes for the step size to compute the travel cost bound LB^d_{iMST}.

resulting to an average bound of 434. Thus, in contrast to the constant step size, one parameter setting fitted for all numbers of iterations which is $t \approx 2$. Furthermore, for each iteration number, this exponential parameter scheme resulted to similar average bound values as the best respective solution with a constant step size $t_k = t$.

Also a linear decreased step size $t_k = t - k\Delta_t$ was tested with Δ_t is chosen such that after n_I iterations t_k is zero. To be more precise, $\Delta_t = \frac{t}{n_I}$. The obtained results are shown in Figure 6.12(b) for n_I equal to 1000, 100 and 10. As observed by constant parameters shown in Figure 6.11, also linear decreased parameters require an adaptation of the parameter setting to the number of iterations in order to obtain good solutions. As shown in 6.12(b), the more iterations are allowed, the smaller the step size should be chosen. For each iteration number, the best average values were similar to the best values obtained with the other parameter schemes.

After obtaining a parameter setting that obtains good results for various iteration numbers, the iteration number itself is analyzed. As expected, in the previous experiments it was observed that more iterations led to better bounds. However, more iterations also require more computational time. Because of that, in the following a trade-off between computational time and bound quality is searched. For this purpose, Table 6.5 contains the average bound value and the average computational time for LB^d_{MST} and LB^d_{iMST} calculated with different numbers of iterations. In the first group, for each of the n_a jobs an 1-tree is calculated and the best is chosen for the iteration step. And in the second group, only one 1-tree is determined with job 1 as the vertex connected finally to the spanning tree. As it can be seen, more iterations led to better bounds at the expense of an increased computational time. Further, comparing the best of n_a 1-trees with only one 1-tree shows that with the latter similar results can be obtained by decreased computational effort because more iterations can be executed when in each iteration only one minimum spanning

Bound	Bound value	Comp. Time (ms)
Use best of n_a 1-trees		
Initial solution $\mathrm{LB}^d_{\mathrm{MST}}$	393,1	0,3
$\mathrm{LB}^d_{\mathrm{iMST}}$ with 10 iterations	434,4	1,5
$\mathrm{LB}^d_{\mathrm{iMST}}$ with 25 iterations	445,0	2,9
$\mathrm{LB}^d_{\mathrm{iMST}}$ with 50 iterations	453,6	5,6
$\mathrm{LB}^d_{\mathrm{iMST}}$ with 100 iterations	459,1	9,5
$\mathrm{LB}^d_{\mathrm{iMST}}$ with 1000 iterations	462,5	33,3
Use single 1-trees on job 1		
Initial solution $\mathrm{LB}^d_{\mathrm{MST}}$	356,6	0,0
$\mathrm{LB}^d_{\mathrm{iMST}}$ with 10 iterations	409,9	0,1
$\mathrm{LB}^d_{\mathrm{iMST}}$ with 100 iterations	452,4	0,4
$\mathrm{LB}^d_{\mathrm{iMST}}$ with 250 iterations	456,2	0,6
$\mathrm{LB}^d_{\mathrm{iMST}}$ with 500 iterations	456,7	0,8
$\mathrm{LB}^d_{\mathrm{iMST}}$ with 1000 iterations	457,0	1,0

Table 6.5.: Comparison of number of iterations and number of calculated 1-trees.

tree has to be determined. Using one 1-tree and 250 iterations showed the best trade-off between bound quality and computational time.

Consequently, the bound $\mathrm{LB}^d_{\mathrm{iMST}}$ is applied with 250 iterations. The step size in iteration k is set to $t_k = 0.985^k\, 2$. And in each iteration, a single 1-tree is calculated whereby job 1 represents the vertex that is finally connected to the minimum weighted spanning tree by means of two edges of minimal weight. Also the bound $\mathrm{LB}^d_{\mathrm{MST}}$ is obtained from a single 1-tree computed on job 1 because computing n_a trees is too time consuming compared to the achieved bound quality.

Comparison of the Travel Cost Bounds

In the following, the travel costs bounds

- $\mathrm{LB}^d_{\mathrm{2N}}$, which is the half of the sum of the travel costs to the two nearest neighbors of each job,
- $\mathrm{LB}^d_{\mathrm{AP}}$ and $\mathrm{LB}^d_{\mathrm{APC}}$, which are the bounds calculated from the assignment problem, as well as
- $\mathrm{LB}^d_{\mathrm{MST}}$ and $\mathrm{LB}^d_{\mathrm{iMST}}$, which are the bounds calculated from minimum spanning trees,

are compared in terms of initial gap and computational time because for an efficient branch-and-bound algorithm, bounds are needed that are calculated fast and have a high bound value. Again, the comparison is done on benchmark L, where each

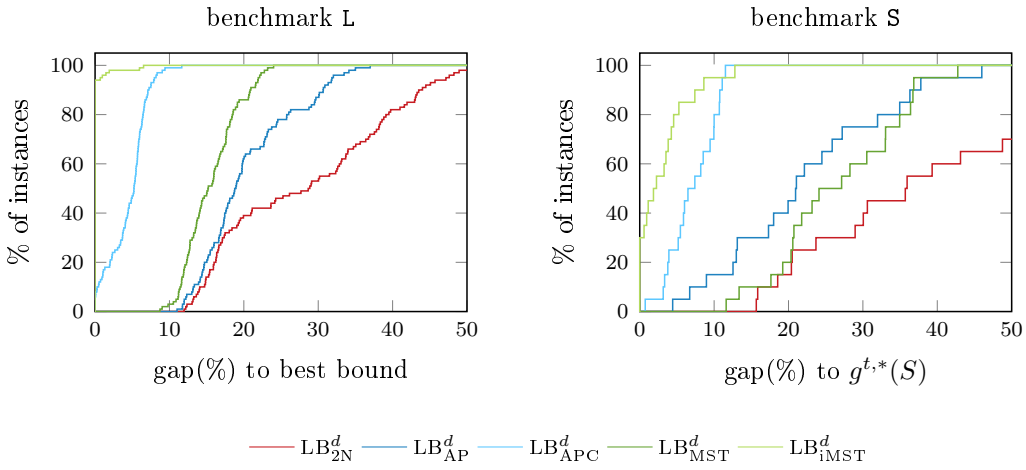


Figure 6.13.: Comparison of the lower bounds on travel costs.

	benchmark L					benchmark S				
bound	min	Q1	Q2	Q3	max	min	Q1	Q2	Q3	max
LB^d_{2N}	< 1	< 1	< 1	< 1	1	< 1	< 1	< 1	< 1	1
LB^d_{AP}	5	8	11	14	155	< 1	< 1	< 1	< 1	2
LB^d_{APC}	7	10	12	15	99	< 1	< 1	< 1	< 1	3
LB^d_{MST}	< 1	< 1	< 1	1	7	< 1	< 1	< 1	< 1	1
LB^d_{iMST}	14	24	29	32	287	< 1	< 1	1	1	53

Table 6.6.: Computational times for lower bounds on travel costs (in milliseconds).

instance contains 100 jobs, because in these large instances, differences in bound quality should be more apparent; and on the 180 instances of benchmark S, because there the obtained bound value can be compared with the minimal travel costs.

Figure 6.13 shows performance profiles for the gap of the lower bounds for travel costs. In the left plot, for instances of benchmark L, the gap to the best lower bound is provided. In the right plot, for instances of benchmark S, the gap to the minimal feasible travel cost value is given which is computed by solving an instance minimizing only travel costs and ignoring the customer cost value during the optimization. Furthermore, Table 6.4 provides percentiles of the computational time in milliseconds. In detail, the minimum, 25th percentile, median, 75th percentile and the maximum are given denoted by min, Q1, Q2, Q3 and max.

As it can be seen in Figure 6.13, the trivial bound LB^d_{2N} led to the smallest bound values. Comparing the assignment bound LB^d_{AP} with the 1-tree bound LB^d_{MST}, which is obtained from one minimum spanning tree calculated on the jobs $N_a \setminus \{1\}$, shows that in benchmark L, LB^d_{MST} outperformed LB^d_{AP} because better bound values were obtained and less time was required for the computation. Also for the iteratively

improvements of these bounds, the bound value computed from minimal spanning trees LB^d_{iMST} was tighter than the improved assignment bound LB^d_{APC}, but at the expense of higher computational times. Note, that for LB^d_{APC} at most 6 iterations were needed to found an assignment with a single cycle, but LB^d_{iMST} was executed with up to 250 iterations.

Concluding, it can be seen that all travel cost bounds have their pros and cons: LB^d_{AP} and LB^d_{iMST} were fast calculated and showed a similar approximation quality. LB^d_{APC} led to significantly improved bounds by a small increase of the computational effort. And LB^d_{iMST} showed the best performance but also the highest computational times (which are still acceptable). Only the simple two-neighbor bound LB^d_{2N} was unconvincingly. Therefore, only LB^d_{2N} will not be tested applied to the developed branch-and-bound algorithms.

6.4.2. Comparison of the Branch-and-Bound Algorithms

In the following, the performance of both developed branch-and-bound algorithms in combination with the presented lower bounds is analyzed. The algorithms were tested based on the 180 instances of benchmark `S` and the computational time was limited to ten minutes. With it, the results can be compared with those of Section 4.4 where for several MILP formulations of the VRPCC, the performance of a commercial solver is analyzed.

6.4.2.1. Applying Branching Strategy Append

In this section, it is analyzed which lower bounds lead to the best performance of the branch-and-bound algorithm with branching strategy append. The analyzed customer cost bounds are the two variants $LB^c_{s,\,fast}$ and $LB^c_{s,\,accurate}$, where a fixed number of jobs executable per day is computed, and LB^c_{LPBP1} that is the solution of the LP relaxation of a special bin packing problem. A detailed description can be found in Section 6.2.1. The investigated lower bounds for travel costs are the two assignment bounds LB^d_{AP} and LB^d_{APC} and the two bounds LB^d_{MST} and LB^d_{iMST} obtained from minimum spanning trees, which are introduced in Section 6.2.2.

The BB-Append algorithm is implemented as described in Section 6.3.1.1 as parallelized depth-first search with backtracking. Needed adaptations of the lower bounds to the branching strategy are explained in Section 6.3.1.2.

Figure 6.14 shows performance profiles for the computational times of the BB-Append algorithm applying different lower bounds. Due to the fact, that the most instances of benchmark `S` were solved in less than one minute, a logarithmic scale is used for the horizontal axis. In Table 6.7, for each combination of customer cost bound and travel cost bound, the number of instances, where the solution process exceeded the time limit of ten minutes, is given together with the average gap to the optimal value for these instances. Furthermore, Table 6.8 provides the number of nodes analyzed during the solution process, which is a good indicator of the quality of the used lower bounds. Tighter lower bounds lead to fewer analyzed nodes because

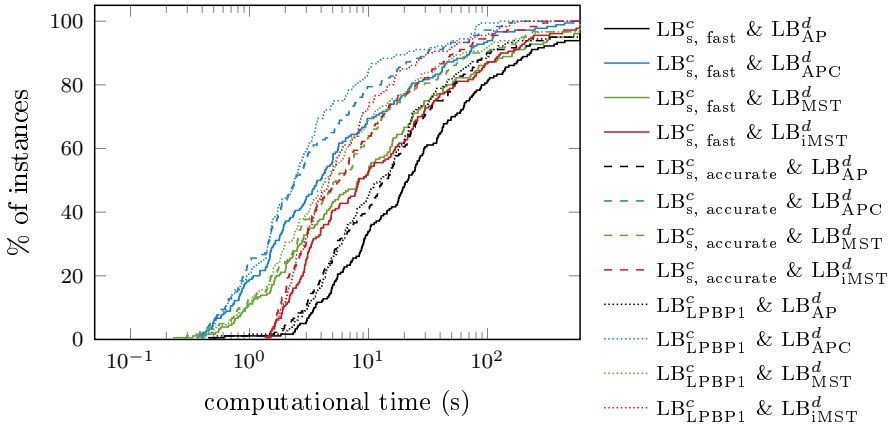


Figure 6.14.: Performance profiles for computational time of the BB-Append algorithm applying different lower bounds on benchmark `S`.

	LB^d_{AP}	LB^d_{APC}	LB^d_{MST}	LB^d_{iMST}
$LB^c_{s,\ accurate}$	9 (0.28%)	0	6 (0.28%)	0
$LB^c_{s,\ fast}$	11 (0.38%)	0	7 (0.24%)	4 (0.00%)
LB^c_{LPBP1}	9 (0.28%)	0	6 (0.21%)	0

Table 6.7.: Number of instances where the time limit was exceeded and the average gap to the optimal value for these instances for the BB-Append algorithm applying different lower bounds.

	LB^d_{AP}	LB^d_{APC}	LB^d_{MST}	LB^d_{iMST}
$LB^c_{s,\ accurate}$	94.6	12.2	60.0	5.4
$LB^c_{s,\ fast}$	177.9	34.7	110.4	15.5
LB^c_{LPBP1}	96.7	10.0	58.3	3.9

Table 6.8.: Number of analyzed nodes (in millions) during the solution process of the BB-Append algorithm applying different lower bounds.

branches can be pruned closer to the root. Each field of Table 6.8 shows how many nodes of the search tree were analyzed when the branch-and-bound algorithm was applied with the customer cost bound of the corresponding row and the travel cost bound of the corresponding column.

Firstly, the travel cost bounds are compared. The performance profiles of Figure 6.14 show that applying LB^d_{APC}, which is the improved assignment bound, led to the smallest computational times. With the faster computed travel cost bounds LB^d_{AP} and LB^d_{MST}, significantly more nodes were analyzed than with LB^d_{APC}, which can be seen in Table 6.8. This is caused by the fact that the bound values LB^d_{AP} and LB^d_{MST} were less tight than LB^d_{APC} as shown in Section 6.4.1.2. This was not compensated by the smaller computational time required to compute LB^d_{AP} and LB^d_{MST} such that the branch-and-bound algorithm took more time if these two travel cost bounds were applied. Furthermore, as it can be seen in Table 6.7, some instances were not solved within ten minutes and for these instances the average gap to the optimal value was larger than zero. In contrast, with LB^d_{iMST}, which is a tighter bound than LB^d_{APC}, more nodes were pruned which can be seen on the smaller number of analyzed nodes, compare Table 6.8. However the computational times were higher due to the larger computational effort to calculate LB^d_{iMST} which can be seen on the corresponding performance profiles in Figure 6.14.

Secondly, the customer cost bounds are analyzed. On the performance profiles provided in Figure 6.14, it can be seen that the customer cost bound $LB^c_{s,\,fast}$ resulted in higher computational times than $LB^c_{s,\,accurate}$ and LB^c_{LPBP1}. Recap, the customer cost bound $LB^c_{s,\,fast}$ was designed to be fast computed at the expense of bound quality. As it can be seen on the comparable high number of nodes analyzed during branch-and-bound with customer cost bound $LB^c_{s,\,fast}$, provided in Table 6.8, the loss of bound quality was too high to be compensated by the reduced effort to compute $LB^c_{s,\,fast}$. The other two bounds led to a similar performance of the BB-Append algorithm: the computational times were close together, as shown in Figure 6.14, and also the number of analyzed nodes was similar, compare Table 6.8. However, the algorithm performed slightly better when LB^c_{LPBP1} was applied because the corresponding performance profiles are mostly to the left of the profiles of $LB^c_{s,\,accurate}$.

Concluding, due to the relatively small computational times and the small number of analyzed nodes, the best performance of the BB-Append algorithm was achieved with the customer cost bound LB^c_{LPBP1} in combination with travel cost bound LB^d_{APC}.

6.4.2.2. Applying Branching Strategy Include

In this section, it is analyzed which lower bounds led to the best computational performance of the BB-Include algorithm. The analyzed customer cost bounds are $LB^c_{s,\,include}$, which is based on a fixed number of jobs executable per day , and $LB^c_{LPBP1,\,include}$, which is the solution of the LP relaxation of a bin packing problem. A detailed description is given in Section 6.2.1. The investigated travel cost bounds

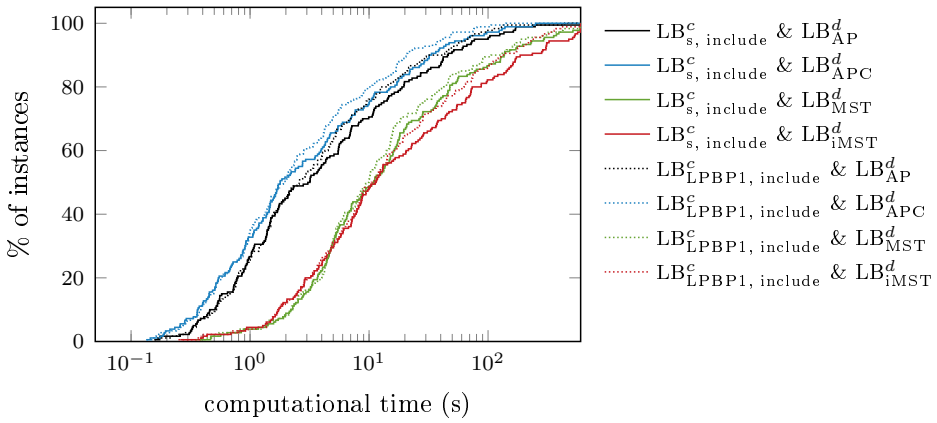


Figure 6.15.: Performance profiles for computational time of the BB-Include algorithm applying different lower bounds on benchmark `S`.

	LB^d_{AP}	LB^d_{APC}	LB^d_{MST}	LB^d_{iMST}
$LB^c_{s,\,include}$	1 (0.00%)	0	3 (0.00%)	5 (2.36%)
$LB^c_{LPBP1,\,include}$	0	0	3 (0.00%)	1 (0.00%)

Table 6.9.: Number of instances where the time limit was exceeded and the average gap to the optimal value for these instances for the BB-Include algorithm applying different lower bounds.

	LB^d_{AP}	LB^d_{APC}	LB^d_{MST}	LB^d_{iMST}
$LB^c_{s,\,include}$	32.8	16.8	147.5	10.2
$LB^c_{LPBP1,\,include}$	22.7	9.7	122.9	6.6

Table 6.10.: Number of analyzed nodes (in millions) during the solution process of the BB-Include algorithm applying different lower bounds.

are the two assignment bounds LB^d_{AP} and LB^d_{APC}, and the two bounds LB^d_{MST} and LB^d_{iMST} obtained from minimum spanning trees, as introduced in Section 6.2.2.

The BB-Include algorithm is implemented as parallelized depth-first search with backtracking. A detailed description is given in Section 6.3.2.1. Due to the fact that branching strategy Include leads to changes inside the routes, also already planned jobs have to be considered by computing lower bounds. This leads to several adaptations: On the one hand, by computing lower bounds on customer costs, it has to be ensured that the start time of already planned jobs can only be later than the current start time. And on the other hand, to compute lower bounds for travel costs, an (asymmetric) distance matrix is defined that considers, among others, that an already planned job can only be connected to its predecessor and successor or a job unplanned so far. More details are given in Section 6.3.2.2.

Figure 6.15 shows performance profiles for computational times of BB-Include algorithm applying different lower bounds. For the horizontal axis, a logarithmic scale is used because the most instances were solved in less than one minute. Table 6.9 gives the number of instances that were not optimally solved within the time limit of ten minutes and in braces the average gap to the optimal value for these instances. And Table 6.10 provides the average number of analyzed search tree nodes. In both tables, each field refers to the BB-Include algorithm applied with the customer cost bound of the corresponding row and the travel cost bound of the corresponding column.

Firstly, the travel cost bounds are compared. As it can be seen in Figure 6.15, the instances were solved faster if an assignment bound for the travel costs is applied to BB-Include algorithm. The corresponding performance profiles remain well above and to the left of the profiles of the lower bounds obtained from 1-trees. Furthermore, the time limit was exceed in a single case and there, the obtained solution value was equal to the optimal value. A reason for the higher computational effort of the lower bound on travel costs LB^d_{MST} and LB^d_{iMST} could be that the distance matrix for the travel cost bounds has a lot of asymmetric entries, which is not taken into account by these bounds. Nevertheless, LB^d_{iMST} yielded tighter bound values than LB^d_{AP} and LB^d_{APC}, which can be seen on the fact that less nodes were analyzed during the solution process if LB^d_{iMST} was applied to branch-and-bound, see Table 6.10. But, as shown in Section 6.4.1.2, the effort to compute LB^d_{iMST} is higher than for LB^d_{APC}. From the both assignment bounds, LB^d_{AP} was clearly outperformed by the improved variant LB^d_{APC}.

Secondly, the two lower bound on customer costs are compared. Figure 6.15 shows that $LB^c_{LPBP1,\ include}$ led to a better performance of the BB-Include algorithm than $LB^c_{s,\ include}$. Independent from the combined travel cost bound, the computational times with $LB^c_{LPBP1,\ include}$ were smaller and less nodes were analyzed than with $LB^c_{s,\ include}$ which results from the better bound quality of $LB^c_{LPBP1,\ include}$ obtained with similar computational effort.

Concluding, the best combination of lower bounds for BB-Include algorithm was $LB^c_{LPBP1,\ include}$ and LB^d_{APC}.

		BB-Append	BB-Include	Solving (R2dT4)	
all instances		9.3	8.5	224.5	(46)
r_{cc}	0.33	9.8	2.8	20.3	
	0.66	6.4	2.9	201.5	(9)
	1.00	11.7	19.8	451.6	(37)
job location	uniformly	3.0	7.1	180.8	(16)
	clustered	15.6	10.0	268.2	(30)

Table 6.11.: Average computational time (in seconds) for the two branch-and-bound algorithms BB-Append and BB-Include and solving the MILP (R2dT4) via CPLEX.

6.4.2.3. Comparison with Applying a Commercial Solver

In the following, the two developed branch-and-bound algorithms are compared with solving a MILP of the VRPCC via CPLEX. The two branch-and-bound algorithm BB-Append and BB-Include are used applying the customer cost bound LB^c_{LPBP1} and the travel cost bound LB^d_{APC}. The solved MILP was (R2dT4), which was solved fastest with CPLEX as shown in Section 4.4. Recap, (R2dT4) is a formulation of the VRPCC, where the routes are defined by two-index binary variables to define for each job the predecessor and successor in the route and integer variables to allocate each job to a route, see Section 4.3.4. The start time of a job is given by a continuous variable that is the start time in minutes counted from day zero. To define the start days of the jobs, binary variables are used. With it, the number of jobs per day is limited which leads to a tighter LP relaxation, compare Section 4.2.3. The used upper bound to limit the jobs executed per day is η as defined in Section 6.2.1.1.

Influence of Instance Characteristics to the Computational Effort

As defined in Appendix A, the instances of benchmark `S` are generated in groups, e.g., each 60 of 180 instances have the same number of jobs with non-zero customer cost coefficient. In this paragraph, it is analyses whether the computational effort to solve an instance differs between these groups. For this purpose, Table 6.11 provides the average computation time of the solution methods for different groupings of the instances of benchmark `S`.

Comparing the computational times as average over all instances, which is provided in the first row of Table 6.11, shows that BB-Include had the best performance. On average, only 8.5 seconds were necessary to solve one instance of benchmark `S`. With the BB-Append algorithm, one instance of benchmark `S` was solved on average in 9.3 seconds. Solving the MILP (R2dT4) with the commercial solver CPLEX led to the highest average computational time which is 224.5 seconds. Note, that with CPLEX, not all instances were optimally solved within the time limit of ten

minutes. In 46 instances, the solution process was not completed which is given in Table 6.11 in parentheses.

At first, three groups are defined that categorize the 180 instances according to the ratio of jobs with non-zero customer cost coefficients r_{cc}. Instances generated with $r_{cc} = 0.33$, where a third of the jobs is afflicted with non-zero customer costs, were easy to solve for each of the three algorithms which can be seen on the small average computational times provided in Table 6.11. As obtained for all instances, with the BB-Include algorithm, the instances were solved faster than with the other solution methods. The instances generated with $r_{cc} = 0.66$, which means that ten of fifteen jobs have a non-zero customer cost coefficient, were significantly harder to solve applying CPLEX. The average computational times was increased tenfold. Indeed, the computational times of the two branch-and-bound algorithms were not drastically changed. The instances, where all jobs are afflicted with non-zero customer cost coefficients, were significantly harder to solve by the commercial solver. This can be seen on the higher average computational times of the instances with $r_{cc} = 1$ shown in Table 6.11. For these instances, solving the MILP with CPLEX was not a suitable solution method because several instances were not solved optimally within ten minutes. Also the BB-Include algorithm showed a significantly increased computational effort: Compared to the instances generated with $r_{cc} = 0.66$, the average computational time was more than sextuplet. For the BB-Append algorithm, the increase in computational time was small.

To investigate, why the computational time of the BB-Include algorithm was drastically increased between instances generated with $r_{cc} = 0.66$ and $r_{cc} = 1$, the search strategy has to be analyzed in more detail. Recap, if all jobs with non-zero customer cost coefficient are planned, the customer cost value of the solution is known and a calculation of the customer cost bound is not required. Furthermore, the known customer cost value of the partial solution is better than each provided lower bound. With branching strategy include, the jobs with non-zero customer costs are planned first. Consequently, if several jobs are not afflicted with customer costs, then for the deeper levels of the search tree, which contains the most nodes, the customer cost value of the partial solution is a tight lower bound and only the travel cost bound has to be computed. Because of that, the computational effort to compute lower bounds decreases and furthermore, more nodes can be discarded because the tight lower bound shows that an optimal solution does not belongs to its solution space. In contrast, with branching strategy Append the jobs are not appended in this order and with it, also on nodes far away from the root of the search tree, it could be necessary to compute both bound values. However, if all jobs are afflicted with a non-zero customer cost coefficient, the customer cost bound is computed in each node of the search tree. Due to the structure of the search trees, in the BB-Include algorithm the number of customer cost bound computations increases much more then in the BB-Append algorithm. Furthermore, as mentioned in Section 6.3.2.1, all jobs have to be considered by computing lower bounds for the BB-Include algorithm but only the unplanned jobs have to be taken into account by computing lower bounds for the BB-Append algorithm. Both together led to a

smaller increase of the computational time for the BB-Append algorithm than for the BB-Include algorithm.

Comparing the instances with uniformly distributed jobs and clustered instances shows that the latter are harder to solve because on average the computational times were higher as shown in Table 6.11. For the BB-Include algorithm, the difference in the average computational times is much smaller than for the BB-Append algorithm or solving the MILP (R2dT4) with CPLEX.

Influence of Cost Structure to Computational Effort

The previous results showed that instances, where more jobs are afflicted with a non-zero customer cost coefficient, were harder to solve. Because of that, in the following it is analyzed how the three compared methods solve instances dominated by travel costs or by customer costs. For this purpose, four further kinds of benchmark `S` are defined by increasing the travel costs or the customer costs each by factor ten and hundred: There are two variants of benchmark `S`, where the jobs $i \in N_a$ have travel costs $d'_{ij} := 10d_{ij}$ or $d'_{ij} := 100d_{ij}$ to each job $j \in N_a$. Consequently, these instances are dominated by travel costs in which 85% and 98%, respectively, of the total costs are travel costs on average. And there are two variants of benchmark `S`, where the customer cost coefficient of each job $i \in N$ is set to $c'_i := 10c_i$ or $c'_i := 100c_i$. Note, that the number of jobs with non-zero customer cost coefficient is not changed: in each kind of benchmark `S`, one third of the instances have 5, 10 and 15 jobs with non-zero customer cost coefficient, respectively. These two variants of benchmark `S` are dominated by customer costs, 91% and 99%, respectively, of the total costs are customer costs. In the basic variant of benchmark `S`, 45% of the total costs are travel costs and 55% are customer costs.

Compared are the BB-Append and BB-Include algorithm, both applying the lower bounds $\mathrm{LB}^c_{\mathrm{LPBP1}}$ and $\mathrm{LB}^d_{\mathrm{APC}}$, as well as solving (R2dT4) with CPLEX.

Firstly, the results on the instances dominated by travel costs are investigated. For these instances, the two plots in the first row of Figure 6.16 show performance profiles for the computational time in seconds and for the gap to the optimal value, respectively. As it can be seen, all three algorithms solved the most instances fast. To be more precise, 62.5%, 49.2% and 47.8% of instances were solved in less than one second with the BB-Append algorithm, the BB-Include algorithm and CPLEX, respectively. The two branch-and-bound algorithms solved all instances within ten minutes, and with CPLEX, in seven of 360 instances, the time limit was exceeded and for two of them, the optimal value was not found.

Secondly, the results on the instances dominated by customer costs are analyzed based on the two plots given in the second row of Figure 6.16. The performance profiles for the computational times show that in these instances, the BB-Include algorithm clearly outperformed the other two solution methods. The computational times were significantly smaller and all instances were solved within ten minutes. With the BB-Append algorithm, in 23 of 360 instances the solution process was terminated after ten minutes and for 14 of these instances, the optimal value was

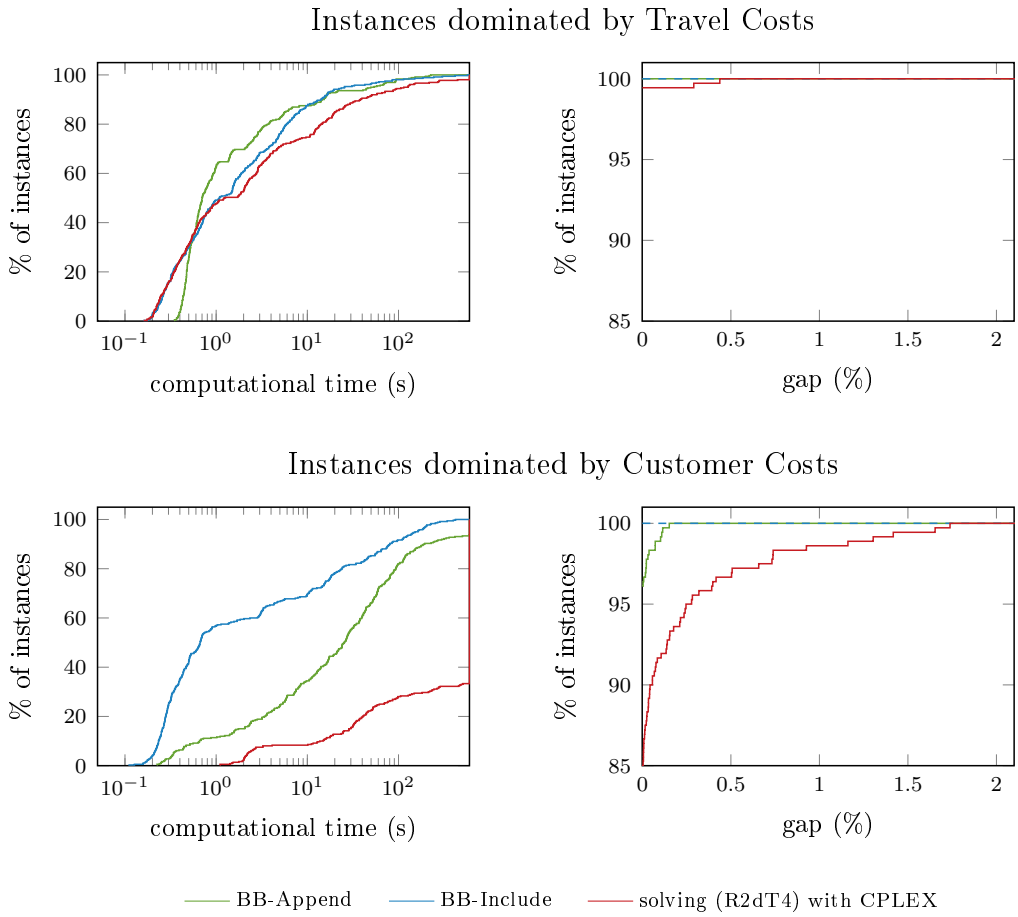


Figure 6.16.: Comparison of the BB-Append and BB-Include algorithm with solving (R2dT4) with CPLEX on variants of benchmark `S` with increased travel costs and increased customer cost coefficients.

not found. The highest computational times were achieved applying CPLEX to the MILP (R2dT4). Only a third of the instances were solved in less than ten minutes and for 15.6% of the instances, the optimal value was not found.

Influence of $d_{\max}$ to the Computational Performance

As defined in Definition 3.4, in the time-constrained VRPCC, all jobs have to be served within a given time horizon $d_{\max}$. In the following, it is analyzed how a short time horizon affects to the computational effort. For this purpose, two variants of benchmark `S` with a small value for $d_{\max}$ are defined: In the first one, $d_{\max}$ is set to the smallest value for which a feasible solution can be found. This variant is called "tight". In the second variant of benchmark `S`, every instance is infeasible, which is enforced by setting $d_{\max}$ one day smaller than in the tight benchmark variant. Therefore, this variant is called "infeasible".

To solve these instances with CPLEX, the corresponding value of $d_{\max}$ is used

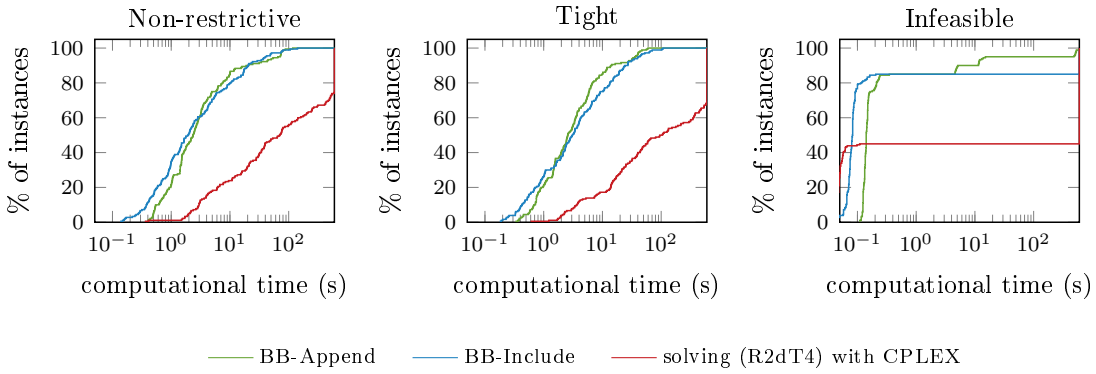


Figure 6.17.: Comparison of the BB-Append and BB-Include algorithm with solving (R2dT4) with CPLEX on variants of benchmark `S` where $d_{\max}$ is small.

as upper bound for the start day. Also the two branch-and-bound algorithm can be easily adapted to this variant of the VRPCC. The customer cost bounds can be applied to check whether all unplanned jobs can be scheduled without exceeding the time horizon $d_{\max}$ because computing lower bounds on customer costs is done by allocating jobs to days. Consequently, not only the jobs with a non-zero customer cost coefficient are taken into account but all jobs which means $N_c = N$. Then, if for the partial solution of any node, the LP relaxation of (BP1) with $T = \{t_0^d, t_0^d + 1, \ldots, d_{\max}\}$ is infeasible, $\mathrm{LB}^c_{\mathrm{LPBP1}}$ is set to infinity because the node can be pruned.

In Figure 6.17, performance profiles for the computational time of the BB-Append and BB-Include algorithm, both applying the lower bounds $\mathrm{LB}^c_{\mathrm{LPBP1}}$ and $\mathrm{LB}^d_{\mathrm{APC}}$, as well as solving (R2dT4) with CPLEX are given. The results are shown on the left for the benchmark `S` with $d_{\max} = n$ which does not restrict the solution space, in the middle for the tight variant of benchmark `S` which means there exists a feasible solution for each instance, and on the right for the infeasible variant where $d_{\max}$ was chosen just small enough that no feasible solution exists. For all algorithms, a time limit of ten minutes was defined and the BoG heuristic was applied to find a feasible start solution. If no feasible start solution was found, the upper bound for the branch-and-bound algorithms was set sufficiently large to $2^{31} - 1$, which is the largest integer value in the used programming language Java.

For each instance with tight $d_{\max}$, all algorithms found a feasible solution within the time limit of ten minutes. The two branch-and-bound algorithms BB-Append and BB-Include solved all instances within ten minutes which can be seen in the middle plot of Figure 6.17. In case of solving the instances with CPLEX, for 56 of 180 instances, the solution process took more than ten minutes. In comparison to benchmark `S` with a non-restrictive value of $d_{\max}$, shown in the left diagram of Figure 6.17, it turned out that the tight value for $d_{\max}$ does not lead to a significant performance change. In the tight benchmark variant, the computational times for the BB-Include algorithms and also solving (R2dT4) with CPLEX were slightly larger than in the normal variant of benchmark `S`. The reason for the small increase

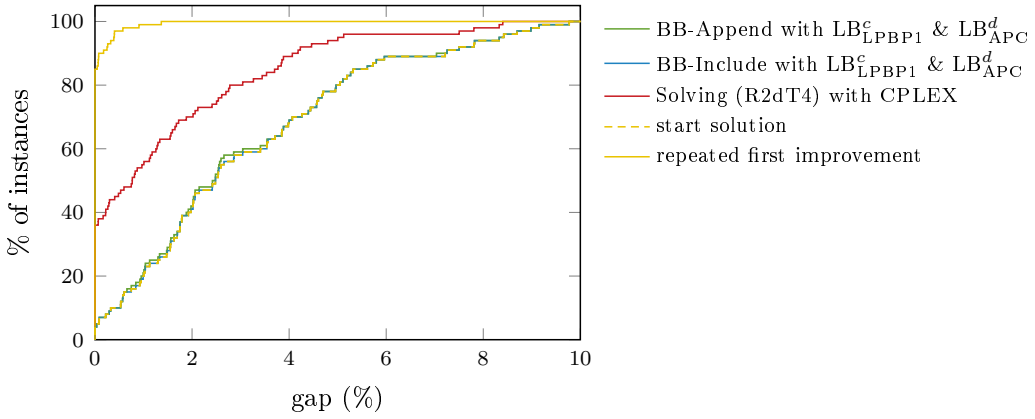


Figure 6.18.: Comparison of the BB-Append and the BB-Include algorithm as well as solving (R2dT4) with CPLEX in benchmark `M`.

of the computational time is that for some instances no feasible start solution was found which leads to additional effort to find a start solution and a suitable upper bound. The BB-Append algorithms had shown almost the same computational performance in both variants.

For the benchmark variant with infeasible instances, the computational times varied strongly as shown in the left plot of Figure 6.17. There are some instances, where it can be observed fast that no feasible solution exists, but for some instance it seems to be hard. With CPLEX, for 81 of these 180 instances, it was observed in less than 0.11 seconds that no feasible solution exists. But for the rest of the instances, infeasibility could not be shown. With the BB-Include algorithm, for 153 of 180 instances it was obtained in less than 0.21 seconds that no feasible solution exists. But for the remaining 27 instances, infeasibility could not be observed. And with the BB-Append algorithm, for 153 instances, less than 0.5 seconds were needed to show that they are infeasible. For further 18 instances, infeasibility was shown in less than 15.2 seconds. And for further 8 instances, infeasibility was shown during the time limit of ten minutes. For only one instance, the time limit was exceeded.

Increase of the Computational Time with Increasing Number of Jobs

Finally, it will be analyzed how the computational time increases with increasing number of jobs. So far, mainly the computational performance on benchmark `S` was investigated. Thereby, it was observed that the two branch-and-bound algorithms outperformed solving (R2dT4) with CPLEX.

Firstly, based on benchmark `M` the exact solution methods BB-Append, BB-Include and solving (R2dT4) by CPLEX are compared with the best heuristic, which is repeated first improvement as described in Section 5.3. All three exact solution methods were initialized with the solution obtained by the BoG algorithm improved by a single run of the best improvement algorithm. It turned out that with the branch-and-bound algorithms none of the 100 instances with 30 jobs was

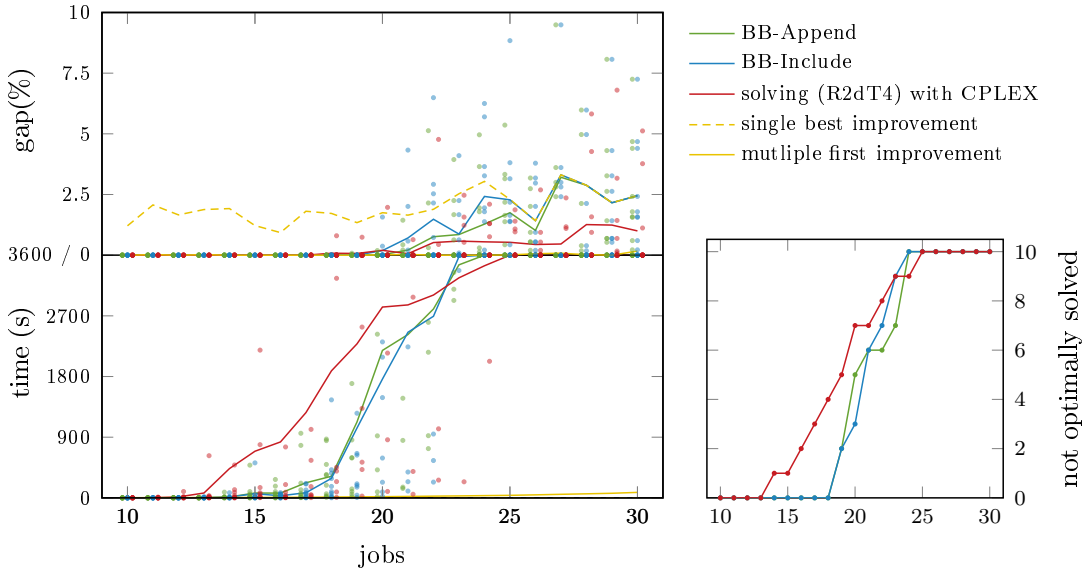


Figure 6.19.: Average computational time and gap of the BB-Append and BB-Include algorithm as well as solving (R2dT4) with CPLEX dependent on the number of jobs.

solved within the time limit of one hour. And only a single instance of benchmark `M` was solved optimal in one hour by CPLEX. Because of that, Figure 6.18 provides performance profiles for the gap to the best obtained solution. As it can be seen, both branch-and-bound algorithm hardly found solutions better than the provided start solution. Significantly better solutions were obtained with CPLEX. The best solutions were obtained with repeated first improvement.

This is in contrast to the results obtained for benchmark `S` where both branch-and-bound algorithm outperformed CPLEX. Because of that, secondly the algorithms are compared for instances with different job numbers. For this purpose, initially ten instances with ten jobs were defined. To obtain from them ten instances with eleven jobs, one job is added to each instance. In this manner, also the further instances are generated such that an instance with n jobs contains all jobs of the according instance with $n-1$ jobs. In the instances, on average 78% of the jobs are afflicted with a non-zero customer cost coefficient. The ratio of customer costs on the total costs varies between 26% and 82% and its median is 48%. The computational time was limited to one hour.

Figure 6.19 shows the computational results for the branch-and-bound algorithms BB-Append and BB-Include as well as solving the MILP (R2dT4) with CPLEX for instances with an increasing number of jobs from 10 to 30. In detail, the main plot shows the average computational times in seconds with the average gap to the best obtained solution on the top. Beside this, a second plot shows the number of not optimally solved instances. As it can be seen, solving the MILP (R2dT4) showed the highest average computational time for the most job numbers. Between $n = 13$ and $n = 25$, the average computational time increases until for $n = 25$ none of the

instance was solved within one hour. The two branch-and-bound algorithms BB-Append and BB-Include outperformed the commercial solver up to $n = 22$. But, between $n = 18$ and $n = 22$, the average computational times increase strongly so that for larger instances, CPLEX led to better solutions, which can be seen on the gap provided on the top of Figure 6.19. It can also be observed, that for instances with more than 25 and 27 jobs, respectively, the branch-and-bound algorithms did not found solutions better than the start solution. The best solutions were obtained by the local search algorithm with 10000 first improvement iterations.

6.5. Conclusion

In this chapter, two branch-and-bound algorithms for the VRPCC were presented. Both are based on the partition and permutation model which was presented in Chapter 3. After giving a detailed description of the branch-and-bound method, some lower bounds for the VRPCC were presented. In detail, in Section 6.2.1, several lower bounds for the customer cost part were developed and analytically compared. The computational experiments showed that the lower bound $\mathrm{LB}^{c}_{\mathrm{LPBP1}}$, which is the LP relaxation value of a special bin packing problem, had the best trade-off between small computational time and high bound values. In Section 6.2.2, several lower bounds for the travel cost part were investigated which are known from the classical TSP.

After providing lower bounds, in Section 6.3, two branching strategies were designed that build-up the routes successively. With it, the start times of the jobs can be computed from the partial solution and are not decision variables as in a MILP formulation of the VRPCC. The computational experiments provided in Section 6.4 showed that both developed branch-and-bound algorithms outperformed solving one of the presented MILPs of the VRPCC by CPLEX. However, in medium-sized problems, the branch-and-bound algorithms were not suitable to solve the VRPCC. As applying CPLEX to solve a MILP formulation of the VRPCC, both branch-and-bound algorithms were not able to obtain an optimal solution in reasonable time. To be more precise, a solution better than the start solution was rarely obtained. This is caused by the fact that the search tree is scanned in a unfavorable order because of the used backtracking approach.

7. Summary and Outlook

In this thesis, a new variant of the vehicle routing problem was investigated which results from a railway maintenance planning problem. The aim of this optimization problem is the scheduling of maintenance jobs that correct failures which occurred unexpectedly. In contrast to common vehicle routing problems, the objective function contains not only travel costs, but also customer costs. These customer costs are time-dependent and represent penalties that have to be paid for each day from the time of failure detection to the time of maintenance completion. Consequently, for each maintenance job, which is represented by a customer, the customer costs are computed by multiplying its customer cost coefficient with its start day which is the day on which maintenance is planned.

This new vehicle routing problem with customer costs (VRPCC) was introduced in this thesis by a non-linear partition and permutation model. In this model, a feasible solution is defined by a partition of the job set into subsets that represents the allocation of jobs to vehicles and a permutation for each subset that represents the order of processing the jobs. Then, the start times of the jobs were calculated based on the orders given by the permutations. It was taken into account that work can only be done in eight hour shifts during the night. Based on the start times, the customer cost value of each job is computed, which are the penalty costs paid for this job. Then, the costs of a schedule are calculated via the sum of travel costs and customer costs.

To solve the VRPCC by commercial optimization software, the VRPCC was formulated as mixed-integer linear program. In doing so, the start times became decision variables. Furthermore, it turned out that including customer costs led to problems harder to solve than vehicle routing problems where only travel costs are minimized. This was caused by the fact that in the LP relaxation, the start days were set close to the earliest feasible start day. Consequently, the customer cost value of the LP relaxation was small. Since the lower bound for the branch-and-cut algorithm of the solver is the LP relaxation, this lower bound is not tight unless the customer cost value of an optimal solution is insignificantly small. To improve the LP relaxation, several alternative formulations were developed. Computational experiments were conducted to compare the formulations in terms of LP relaxation value, computational time and, in case of not reaching the optimum, the gap to an optimal solution, if available. The best performance was achieved with a formulation that uses binary variables for the start days. With it, the number of jobs allocated to a single day was limited by an upper bound. Nevertheless, small instances, where the customer costs have a significant impact on the obtained solution, were still hard to solve. Medium-sized and large instances could not be solved efficiently by the

applied commercial solver.

For practical applications, an optimal solution is often not necessary. Instead, heuristics are used to obtain a good and feasible solution in short time. For this purpose, several construction heuristics for the VRPCC were designed and investigated. Furthermore, two local search algorithms, first and best improvement, were applied. The computational experiments showed that the repeated first improvement algorithm, which is applying the first improvement algorithm several times to different start solutions, was the best heuristic with respect to solution quality.

Further, in order to compare the classical approach, i.e., solving a mixed-integer linear program of the VRPCC by a commercial solver, with the heuristics, the computational time was limited to ten minutes. In this setting, the numerical experiments indicated that the repeated first improvement algorithm outperformed the classical approach. To be more precise, for instances with fifteen jobs the repeated first improvement algorithm found more often an optimal solution than the commercial solver. However, the drawback of this heuristic is that the approximation quality of the obtained solution cannot be determined.

Although the improvement algorithms obtained good heuristic solutions, one aim of this thesis was to find an exact solution method to optimally solve small instances of the VRPCC in reasonable time. As mentioned above, the classical approach, where the problem is formulated as mixed-integer linear program and then solved by a commercial solver, was not suitable mainly due to two reasons: Firstly, the start times of the jobs become decision variables, when the VRPCC is formulated as mixed-integer linear program, which increases the solution space. And secondly, the LP relaxations of the developed formulations were not tight because the values for the start days can be chosen small when the integer constraints are removed. Because the commercial solver uses a branch-and-cut method to solve mixed-integer linear programs where the lower bound equals to the LP relaxation improved by some additional cuts, the lower bounds were small and less branches were discarded.

Since the branch-and-bound method is a suitable approach to solve complex combinatorial optimization problems, this approach was chosen and, in this thesis, two special branch-and-bound algorithms for the VRPCC were developed. To design a branch-and-bound algorithm that solves a combinatorial problem efficiently, tight and fast computable lower bounds are needed to discard parts of the solution space. In this thesis, new lower bounds for the customer cost part of the objective function were formulated, where most of the bounds are based on solving bin packing problems. All lower bounds for customer costs were compared analytically and experimentally, whereby the computational experiments showed that the lower bound computed from the LP relaxation of a specific bin packing problem had the best trade-off between computational effort and bound quality. In this thesis, a suitable algorithm was developed for this bound, which computes the bound value in $\mathcal{O}(n \log(n) + n)$. For the travel cost part of the objective function, several known lower bounds from the TSP were compared. The improved assignment bound, firstly provided in [27], showed the best performance applied to the designed branch-and-bound algorithms.

To design a branch-and-bound algorithm, beside efficient lower bound, also suitable branching strategies are necessary to split the problem space into smaller subspaces. In this thesis two branching strategies were developed which are based on the non-linear partition and permutation model to take advantage from the problem structure. To be more precise, new branches are generated by appending or including a job in an uncompleted schedule. Consequently, the start times can be computed directly from the so far planned jobs and more tight lower bounds can be computed for the so far unplanned jobs.

In this thesis, the first approach for such a branching strategy was to generate a new branch by appending an unplanned job at the end of an uncompleted route. It was reasoned that this strategy leads to tight lower bounds because the costs of the planned jobs are fixed. However, an analysis of the resulting search tree indicated that the number of generated nodes is large. Because of that, in this thesis a second branching strategy was developed where new branches are generated by including an unplanned job inside an uncompleted route. This implies that costs of the so far planned jobs are no longer fixed because including a job can shift planned jobs behind, but the corresponding search tree contains significantly less nodes and has a more suitable structure. The two corresponding branch-and-bound algorithms were implemented as depth-first search with backtracking to reduce the storage effort.

By means of computational experiments, the developed branch-and-bound algorithms were compared with the classical approach, which means solving a mixed-integer linear program of the VRPCC by a commercial solver. The results showed that both branch-and-bound algorithms solved the small instances faster than the classical approach.

Finally, some possible directions for future research on the VRPCC are pointed out and discussed.

For practical applications, some additional requirements can be imposed on the solution. In Section 3.3, some possible extensions of the VRPCC are mentioned. To apply the presented heuristics and solution approaches, some adaptations would be necessary. For example, for the branch-and-bound algorithms, the limitation of the planning horizon or time window restrictions can be integrated in computing lower bounds on customer costs. Also the requirement that maintenance crews have certain skills and can only process corresponding tasks can be integrated. For the application of a commercial solver, according constraints can be formulated as shown, e.g., in [151]. It would be interesting to investigate how such additional constraints affect to the solution performance.

As stated in Section 2.3.3, in this thesis the branch-and-price method was not applied to the VRPCC because of the large amount on feasible routes and the time-dependent costs in the objective function. However, in further research it might be worth to explore how branch-and-price can be efficiently applied to the VRPCC. Recent papers like, e.g., [56, 122] show that also vehicle routing problems with less restrictions to the routes can be solved by the branch-and-price method. And also problems with time-dependent subproblems were addressed, e.g., in [100, 138].

As shown in Section 4.4, the customer costs lead to problems harder to solve by a commercial solver: Even some small instances with fifteen jobs were not solved within ten minutes. To obtain near-optimal solutions in medium-sized instances, MIP based local search, compare, e.g., [77, 81, 124], might be a topic of further research. For example, after obtaining a start solution by a heuristic, iteratively some variables can be fixed such that a small subset of variables remains to be optimized applying a commercial solver.

Also the two developed branch-and-bound algorithms were not suitable to solve medium-sized instances, which was observed in the computational experiments presented in Section 6.4.2.3. This is caused by the backtracking approach: To avoid storing millions of branching nodes, the search tree is investigated in a specific order. Because of that, the branch-and-bound algorithms analyze at first unfavorable solutions where the most jobs are allocated to a single route. Further research could improve the branch-and-bound algorithms, e.g., by developing a more suitable best-first search strategy in order to find faster an improved solution or by tightening the lower bounds. It might also be worth to develop additional valid constraints like an upper bound for the number of jobs per route or for the latest start day.

Bibliography

[1] E. Aarts and J. K. Lenstra. *Local search in combinatorial optimization*. Princeton University Press, 2003.

[2] S. Almoustafa, S. Hanafi, and N. Mladenović. New exact method for large asymmetric distance-constrained vehicle routing problem. *European Journal of Operational Research*, 226:386–394, 2013.

[3] J. Andrews, D. Prescott, and F. De Rozières. A stochastic model for railway track asset management. *Reliability Engineering & System Safety*, 130:76–84, 2014.

[4] C. Archetti, N. Bianchessi, and M. G. Speranza. Branch-and-cut algorithms for the split delivery vehicle routing problem. *European Journal of Operational Research*, 238(3):685–698, 2014.

[5] G. B. Arfken. *Mathematical Methods for Physicists*. Number Bd. 1 in Mathematical Methods for Physicists. Academic Press, 1966.

[6] N. Ascheuer. *Hamiltonian path problems in the on-line optimization of flexible manufacturing systems*. PhD thesis, Technische Universität Berlin, Germany, 1996.

[7] N. Ascheuer, M. Fischetti, and M. Grötschel. Solving the asymmetric travelling salesman problem with time windows by branch-and-cut. *Mathematical Programming*, 90(3):475–506, 2001.

[8] P. Augerat, D. Naddef, J. M. Belenguer, E. Benavent, A. Corberán, and G. Rinaldi. Computational results with a branch and cut code for the capacitated vehicle routing problem. Technical report, Institute for Systems Analysis and Computer Science, 1995.

[9] N. Balakrishnan. Simple heuristics for the vehicle routeing problem with soft time windows. *Journal of the Operational Research Society*, 44(3):279–287, 1993.

[10] E. Balas and M. Guignard. Report of the session on branch and bound/implicit enumeration. In *Annals of Discrete Mathematics*, volume 5, pages 185–191. Elsevier, 1979.

[11] R. Baldacci, N. Christofides, and A. Mingozzi. An exact algorithm for the vehicle routing problem based on the set partitioning formulation with additional cuts. *Mathematical Programming*, 115(2):351–385, 2008.

[12] R. Baldacci, E. Hadjiconstantinou, and A. Mingozzi. An exact algorithm for the capacitated vehicle routing problem based on a two-commodity network flow formulation. *Operations Research*, 52(5):723–738, 2004.

[13] R. Baldacci, A. Mingozzi, and R. Roberti. Recent exact algorithms for solving the vehicle routing problem under capacity and time window constraints. *European Journal of Operational Research*, 218(1):1–6, 2012.

[14] M. M. Baldi, F. Heinicke, A. Simroth, and R. Tadei. New heuristics for the stochastic tactical railway maintenance problem. *Omega*, 63:94–102, 2016.

[15] M. Bellmore and G. Nemhauser. The traveling salesman problem: A survey. *Operations Reseach*, 16:538–558, 1968.

[16] D. P. Bertsekas, J. N. Tsitsiklis, and C. Wu. Rollout algorithms for combinatorial optimization. *Journal of Heuristics*, 3(3):245–262, 1997.

[17] S. Boğ, A. K. Nemani, and R. K. Ahuja. Iterative algorithms for the curfew planning problem. *Journal of the Operational Research Society*, 62(4):593–607, 2011.

[18] N. Boland, T. Kalinowski, and S. Kaur. Scheduling network maintenance jobs with release dates and deadlines to maximize total flow over time: Bounds and solution strategies. *Computers & Operations Research*, 64:113–129, 2015.

[19] N. Boland, T. Kalinowski, H. Waterer, and L. Zheng. Mixed integer programming based maintenance scheduling for the hunter valley coal chain. *Journal of Scheduling*, 16(6):649–659, 2013.

[20] C. Borraz-Sánchez and D. Klabjan. Strategic gang scheduling for railroad maintenance. Technical report, Department of Industrial Engineering and Management Sciences, Northwestern University, 2012.

[21] O. Bräysy and M. Gendreau. Vehicle routing problem with time windows. Part I: Route construction and local search algorithms. *Transportation science*, 39(1):104–118, 2005.

[22] O. Bräysy, P. P. Porkka, W. Dullaert, P. P. Repoussis, and C. D. Tarantilis. A well-scalable metaheuristic for the fleet size and mix vehicle routing problem with time windows. *Expert Systems with Applications*, 36(4):8460–8475, 2009.

[23] A. V. Breedam. *An analysis of the behavior of heuristics for the vehicle routing problem for a selection of problems with vehicle-related, customer-related, and time-related constraints*. PhD thesis, Faculty of Applied Economics, University of Antwerp - RUCA, 1994.

[24] G. Budai, D. Huisman, and R. Dekker. Scheduling preventive railway maintenance activities. Technical report, Econometric Institute, Erasmus University Rotterdam, 2004.

[25] F. Camci. The travelling maintainer problem: Integration of condition-based maintenance with the travelling salesman problem. *Journal of the Operational Research Society*, 65(9):1423–1436, 2014.

[26] Z. Cao, H. Guo, J. Zhang, D. Niyato, and U. Fastenrath. Improving the efficiency of stochastic vehicle routing: A partial lagrange multiplier method. *IEEE Transactions on Vehicular Technology*, 65(6):3993–4005, 2015.

[27] N. Christofides. Technical note – bounds for the travelling-salesman problem. *Operations Research*, 20(5):1044–1056, 1972.

[28] N. Christofides, A. Mingozzi, and P. Toth. Exact algorithms for the vehicle routing problem, based on spanning tree and shortest path relaxations. *Mathematical Programming*, 20(1):255–282, 1981.

[29] G. Clarke and J. W. Wright. Scheduling of vehicles from a central depot to a number of delivery points. *Operations Research*, 12(4):568–581, 1964.

[30] G. Codato and M. Fischetti. Combinatorial benders' cuts for mixed-integer linear programming. *Operations Research*, 54(4):756–766, 2006.

[31] A. C. Consilvio, C. C. Crovetto, B. G. Guyot, A. K. Kirwan, N. M. Mazzino, and F. P. Papa. Towards an intelligent and automated platform for railway asset management. *Proceedings of 7th Transport Research Arena TRA 2018*, 2018.

[32] C. Contardo and R. Martinelli. A new exact algorithm for the multi-depot vehicle routing problem under capacity and route length constraints. *Discrete Optimization*, 12:129–146, 2014.

[33] T. H. Cormen, C. E. Leiersin, R. L. Rivest, and C. Stein. *Introduction to algorithms*. MIT Press [u.a.], 2. edition, 2001.

[34] S. Dabia, S. Ropke, T. Van Woensel, and T. De Kok. Branch and price for the time-dependent vehicle routing problem with time windows. In *Proceedings of the VII ALIO–EURO – Workshop on Applied Combinatorial Optimization*, pages 141–144, 2011.

[35] L. Dai, M. Stålhane, and I. B. Utne. Routing and scheduling of maintenance fleet for offshore wind farms. *Wind Engineering*, 39(1):15–30, 2015.

[36] K. Dalmeijer and R. Spliet. A branch-and-cut algorithm for the time window assignment vehicle routing problem. *Computers & Operations Research*, 89:140–152, 2018.

[37] G. B. Dantzig and J. H. Ramser. The truck dispatching problem. *Management science*, 6(1):80–91, 1959.

[38] N. De Jaegere, M. Defraeye, and I. Van Nieuwenhuyse. The vehicle routing problem: State of the art classification and review. Technical report, Faculty of Economics and Business Ku Leuven, 2014.

[39] M. Desrochers and G. Laporte. Improvements and extensions to the Miller-Tucker-Zemlin subtour elimination constraints. *Operations Research Letters*, 10(1):27–36, 1991.

[40] M. Desrochers, J. K. Lenstra, M. W. P. Savelsbergh, and F. Soumis. Vehicle routing with time windows: Optimization and approximation. *Vehicle Routing: Methods and Studies*, 1988.

[41] E. D. Dolan and J. J. Moré. Benchmarking optimization software with performance profiles. *Mathematical Programming*, 91(2):201–213, 2002.

[42] A. Domschke, Wolfgang Drexl, R. Klein, and A. Scholl. *Einführung in Operations Research*. Springer Gabler, 9. edition, 2015.

[43] M. Dror, D. Fortin, and C. Roucairol. Redistribution of self-service electric cars: A case of pickup and delivery. Technical report, INRIA National Institute for Research in Digital Science and Technology, 1998.

[44] M. Dror, G. Laporte, and P. Trudeau. Vehicle routing with split deliveries. *Discrete Applied Mathematics*, 50(3):239–254, 1994.

[45] Y. Dumas, J. Desrosiers, and F. Soumis. The pickup and delivery problem with time windows. *European journal of operational research*, 54(1):7–22, 1991.

[46] B. Eksioglu, A. V. Vural, and A. Reisman. The vehicle routing problem: A taxonomic review. *Computers & Industrial Engineering*, 57(4):1472–1483, 2009.

[47] N. A. El-Sherbeny. Vehicle routing with time windows: An overview of exact, heuristic and metaheuristic methods. *Journal of King Saud University-Science*, 22(3):123–131, 2010.

[48] L. Epstein and A. Levin. Minimum weighted sum bin packing. In *International Workshop on Approximation and Online Algorithms,* 218–231. Springer, 2007.

[49] D. Feillet. A tutorial on column generation and branch-and-price for vehicle routing problems. *Operations Research*, 8:407–424, 2010.

[50] D. Feillet, P. Dejax, M. Gendreau, and C. Gueguen. An exact algorithm for the elementary shortest path problem with resource constraints: Application to some vehicle routing problems. *Networks*, 44(3):216–229, 2004.

[51] J. Feng, X. Jia, F. Zhu, Q. Yang, Y. Pan, and J. Lee. An intelligent system for offshore wind farm maintenance scheduling optimization considering turbine production loss. *Journal of Intelligent & Fuzzy Systems*, 37(5):6911–6923, 2019.

[52] M. A. Figliozzi. An iterative route construction and improvement algorithm for the vehicle routing problem with soft time windows. *Transportation Research Part C: Emerging Technologies*, 18(5):668–679, 2010.

[53] M. Fischetti and P. Toth. An efficient algorithm for the min-sum arborescence problem on complete digraphs. *ORSA Journal on Computing*, 5(4):426–434, 1993.

[54] M. L. Fisher and R. Jaikumar. A generalized assignment heuristic for vehicle routing. *Networks*, 11(2):109–124, 1981.

[55] M. L. Fisher, K. O. Jörnsten, and O. B. G. Madsen. Vehicle routing with time windows: Two optimization algorithms. *Operations Research*, 45(3):488–492, 1997.

[56] R. Fukasawa, Q. He, and Y. Song. A branch-cut-and-price algorithm for the energy minimization vehicle routing problem. *Transportation Science*, 50(1):23–34, 2016.

[57] B. Gavish and S. C. Graves. The travelling salesman problem and related problems. Technical report, Massachusetts Institute of Technology, Operations Research Center, 1978.

[58] B. Gavish and K. Srikanth. An optimal solution method for large-scale multiple traveling salesmen problems. *Operations Reseach*, 34(5):698–717, 1986.

[59] M. Gendreau, G. Laporte, and J.-Y. Potvin. Vehicle routing: Modern heuristics. In *Local search in combinatorial optimization*, pages 311–336. Princeton University Press, 2003.

[60] B. E. Gillett and L. R. Miller. A heuristic algorithm for the vehicle-dispatch problem. *Operations Research*, 22(2):340–349, 1974.

[61] F. Glover. Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research*, 13(5):533–549, 1986.

[62] B. Golden, S. Raghavan, E. A. Wasil, and B. L. Golden. *The vehicle routing problem: Latest advances and new challenges*. Springer, 2008.

[63] M. F. Gorman and J. J. Kanet. Formulation and solution approaches to the rail maintenance production gang scheduling problem. *Journal of Transportation Engineering*, 136:701–708, 2010.

[64] C. A. Grimes. Application of genetic techniques to the planning of railway track maintenance work. In *First International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications*, pages 467–472. IET, 1995.

[65] Z. G. Guo and K. L. Mak. A heuristic algorithm for the stochastic vehicle routing problems with soft time windows. In *Proceedings of the 2004 congress on evolutionary computation*, volume 2, pages 1449–1456. IEEE, 2004.

[66] E. Gustavsson. Scheduling tamping operations on railway tracks using mixed integer linear programming. *EURO Journal on Transportation and Logistics*, 4(1):97–112, 2015.

[67] T. Hanne and R. Dornberger. *Computational intelligence in logistics and supply chain management*, volume 244 of *International Series in Operation Research & Management Science*. Springer, 2017.

[68] P. Hansen and N. Mladenović. First vs. best improvement: An empirical study. *Discrete Applied Mathematics*, 154(5):802–817, 2006. IV ALIO/EURO Workshop on Applied Combinatorial Optimization.

[69] Q. He, H. Li, D. Bhattacharjya, D. P. Parikh, and A. Hampapur. Track geometry defect rectification based on track deterioration modelling and derailment risk assessment. *Journal of the Operational Research Society*, 66(3):392–404, 2015.

[70] F. Heinicke and A. Simroth. Application of simulated annealing to railway routine maintenance scheduling. In *Proceedings of the 14th International Conference on Civil, Structure and Environmental Engineering Computers*. Civil-Comp Press, 2013.

[71] F. Heinicke, A. Simroth, G. Scheithauer, and A. Fischer. A railway maintenance scheduling problem with customer costs. *EURO Journal on Transportation and Logistics*, 4:113–137, 2015.

[72] F. Heinicke, A. Simroth, and R. Tadei. On a novel optimisation model and solution method for tactical railway maintenance planning. In *2nd International Conference on Road and Rail Infrastructure*, 2012.

[73] F. Heinicke, A. Simroth, R. Tadei, and M. Baldi. Job order assignment at optimal costs in railway maintenance. In *ICORES 2013 - Proceedings of the 2nd International Conference on Operations Research and Enterprise Systems*, pages 304–309, 2013.

[74] M. Held and R. M. Karp. The traveling-salesman problem and minimum spanning trees: Part II. *Mathematical Programming*, 1(1):6–25, 1971.

[75] J. Hooker. *Integrated Methods for Optimization*. Springer, 2007.

[76] IBM Corporation. IBM ILOG CPLEX Optimization Studio 12.8 – User's Manual, 2017.

[77] R. J. W. James and B. Almada-Lobo. Single and parallel machine capacitated lotsizing and scheduling: New iterative MIP-based neighborhood search heuristics. *Computers & Operations Research*, 38(12):1816–1825, 2011.

[78] N. Jiménez-Redondo, N. Bosso, L. Zeni, A. Minardo, F. Schubert, F. Heinicke, and A. Simroth. Automated and cost effective maintenance for railway (ACEM–Rail). *Procedia-Social and Behavioral Sciences*, 48:1058–1067, 2012.

[79] N. Jiménez-Redondo, Á. C. Cordón, U. Kandler, A. Simroth, A. Reyes, F. J. Morales, J. Odelius, S. M. Famurewa, J. Morgado, E. Duarte, et al. INFRALERT: Improving linear transport infrastructure efficiency by automated learning and optimised predictive maintenance techniques. *Proceedings of 7th Transport Research Arena TRA 2018*, 2018.

[80] D. Jungnickel. *Graphs, Networks and Algorithms*. Springer, 1999.

[81] T. Kalinowski, J. Matthews, and H. Waterer. Scheduling of maintenance windows in a mining supply chain rail network. *Computers & Operations Research*, 115:104670, 2020.

[82] B. Kallehauge, J. Larsen, and O. B. G. Madsen. Lagrangian duality applied to the vehicle routing problem with time windows. *Computers & Operations Research*, 33(5):1464–1487, 2006.

[83] B. Kallehauge, J. Larsen, O. B. G. Madsen, and M. M. Solomon. Vehicle routing problem with time windows. In *Column generation*, pages 67–98. Springer, 2005.

[84] U. Kandler, A. Simroth, J. Morgado, and E. Duarte. Decision support for tactical planning – a use case of the INFRALERT project. *Proceedings of 7th Transport Research Arena TRA 2018*, 2018.

[85] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[86] T. Koch, T. Berthold, J. Pedersen, and C. Vanaret. Progress in mathematical programming solvers from 2001 to 2020. Technical report, Zuse Institute Berlin, 2021.

[87] N. Kohl. *Exact methods for time constrained routing and related scheduling problems*. PhD thesis, Technical University of Denmark, 1995.

[88] A. L. Kok, E. W. Hans, and J. M. J. Schutten. Vehicle routing under time-dependent travel times: The impact of congestion avoidance. *Computers & Operations Research*, 39(5):910–918, 2012.

[89] A. W. J. Kolen, A. H. G. Rinnooy Kan, and H. W. J. M. Trienekens. Vehicle routing with time windows. *Operations Research*, 35(2):266–273, 1987.

[90] B. Korte and J. Vygen. *Combinatorial optimization: Theory and algorithms*. Springer, 2000.

[91] A. Kovács, G. Erdös, L. Monostori, and Z. J. Viharos. Scheduling the maintenance of wind farms for minimizing production loss. *IFAC Proceedings Volumes*, 44(1):14802–14807, 2011.

[92] V. Kowalenko. Applications of the cosecant and related numbers. *Acta applicandae mathematicae*, 114(1):15–134, 2011.

[93] J. B. Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical Society*, 7(1):45–50, 1956.

[94] S. N. Kumar and R. Panneerselvam. A survey on the vehicle routing problem and its variants. *Intelligent Information Management*, 4(3):66–74, 2012.

[95] G. Laporte. The vehicle routing problem: An overview of exact and approximate algorithms. *European Journal of Operational Research*, 59(3):345–358, 1992.

[96] G. Laporte. What you should know about the vehicle routing problem. *Naval Research Logistics*, 54(8):811–819, 2007.

[97] G. Laporte, M. Desrochers, and Y. Nobert. Two exact algorithms for the distance-constrained vehicle routing problem. *Networks*, 14(1):161–172, 1984.

[98] G. Laporte, Y. Nobert, and S. Taillefer. Solving a family of multi-depot vehicle routing and location-routing problems. *Transportation Science*, 22(3):161–172, 1988.

[99] H. Lei, G. Laporte, and B. Guo. The capacitated vehicle routing problem with stochastic demands and time windows. *Computers & Operations Research*, 38(12):1775–1783, 2011.

[100] G. Lera-Romero, J. J. M. Bront, and F. J. Soulignac. An enhanced branch and price algorithm for the time-dependent vehicle routing problem with time window. Technical report, Optimization Online, 2018, 2018.

[101] G. Li. *Optimization-based decision support for inspection and maintenance of infrastructure networks*. PhD thesis, University of Texas at Austin, 2011.

[102] T. Lidén. Railway infrastructure maintenance – a survey of planning problems and conducted research. *Transportation Research Procedia*, 10:574–583, 2015.

[103] T. Lidén and M. Joborn. An optimization model for integrated planning of railway traffic and network maintenance. *Transportation Research Part C: Emerging Technologies*, 74:327–347, 2017.

[104] T. Lidén, T. Kalinowski, and H. Waterer. Resource considerations for integrated planning of railway traffic and maintenance windows. *Journal of Rail Transport Planning & Management*, 8(1):1–15, 2018.

[105] R. Lima and E. W. O. Seminar. IBM ILOG CPLEX – what is inside of the box. In *Proc. 2010 EWO Seminar*, pages 1–72, 2010.

[106] A. Löbel. *Optimal Vehicle Scheduling in Public Transit.* PhD thesis, TU Berlin, 1997.

[107] A. Lucena. Time-dependent traveling salesman problem – the deliveryman case. *Networks*, 20(6):753–763, 1990.

[108] R. Macedo, C. Alves, J. M. Valério de Carvalho, F. Clautiaux, and S. Hanafi. Solving the vehicle routing problem with time windows and multiple routes exactly using a pseudo-polynomial model. *European Journal of Operational Research*, 214(3):536–545, 2011.

[109] F. Maffioli and A. Sciomachen. A mixed-integer model for solving ordering problems with side constraints. *Annals of Operations Research*, 69:277–297, 1997.

[110] C. Malandraki and M. S. Daskin. Time dependent vehicle routing problems: Formulations, properties and heuristic algorithms. *Transportation science*, 26(3):185–200, 1992.

[111] S. Martello and P. Toth. *Knapsack Problems: Algorithms and Computer Implementations.* John Wiley & Sons Ltd., 1990.

[112] C. E. Miller, A. W. Tucker, and R. A. Zemlin. Integer programming formulation of traveling salesman problems. *Journal of the Association for Computing Machinery*, 7:326–329, 1960.

[113] M. Miwa. Mathematical programming model analysis for the optimal track maintenance schedule. *Quarterly Report of RTRI*, 43(3):131–136, 2002.

[114] A. Montero, I. Méndez-Díaz, and J. J. Miranda-Bront. An integer programming approach for the time-dependent traveling salesman problem with time windows. *Computers & Operations Research*, 88:280–289, 2017.

[115] J. Munkres. Algorithms for the assignment and transportation problems. *Journal of the Society for Industrial and Applied Mathematics*, 5(1):32–38, 1957.

[116] G. L. Nemhauser and L. A. Wolsey. *Integer and combinatorial optimization*, volume 18. John Wiley & Sons Ltd., 1988.

[117] J. Neuhold. Tamping within sustainable track asset management. *Monographic Series TU Graz. Railway research*, 6, 2020.

[118] J.-E. Nilsson and J. Nyström. Mapping railways maintenance contracts: The case of Netherlands, Finland and UK. Technical report, The Swedish National Road and Transport Research Institute (VTI), 2014.

[119] S. M. Oh, J. H. Lee, B. H. Park, H. U. Lee, and S. H. Hong. A study on a mathematical model of the track maintenance scheduling problem. *Computers in Railways X*, 88:85–96, 2006.

[120] M. W. Padberg. *Linear optimization and extensions.* Springer, 2., rev. and expanded edition, 1999.

[121] S. N. Parragh, K. F. Doerner, and R. F. Hartl. A survey on pickup and delivery problems. Part II: Transportation between pickup and delivery locations. *Journal für Betriebswirtschaft*, 58(2):81–117, 2008.

[122] D. Pecin, A. Pessoa, M. Poggi, and E. Uchoa. Improved branch-cut-and-price for capacitated vehicle routing. *Mathematical Programming Computation*, 9(1):61–100, 2017.

[123] F. Peng. *Scheduling of track inspection and maintenance activities in railroad networks.* PhD thesis, University of Illinois at Urbana-Champaign, 2011.

[124] F. Peng, S. Kand, X. Li, and Y. Ouyang. A heuristic approach to the railroad track maintenance scheduling problem. *Computer-Aided Civil and Infrastructure Engineering*, 26:129–145, 2011.

[125] S. M. Pour, K. Marjani Rasmussen, J. H. Drake, and E. K. Burke. A constructive framework for the preventive signalling maintenance crew scheduling problem in the danish railway system. *Journal of the Operational Research Society*, 70(11):1965–1982, 2019.

[126] R. C. Prim. Shortest connection networks and some generalizations. *The Bell System Technical Journal*, 36(6):1389–1401, 1957.

[127] A. G. Qureshi, E. Taniguchi, and T. Yamada. Exact solution for the vehicle routing problem with semi soft time windows and its application. *Procedia-Social and Behavioral Sciences*, 2(3):5931–5943, 2010.

[128] G. Reinelt. *The Traveling Salesman: Computational Solutions for TSP Applications.* Lecture Notes in Computer Science. Springer, 1994.

[129] L.-M. Rousseau, M. Gendreau, G. Pesant, and F. Focacci. Solving VRPTWs with constraint programming based column generation. *Annals of Operations Research*, 130(1-4):199–216, 2004.

[130] M. Savelsbergh. A branch-and-price algorithm for the generalized assignment problem. *Operations Research*, 45(6):831–841, 1997.

[131] M. M. Solomon. On the worst-case performance of some heuristics for the vehicle routing and scheduling problem with time window constraints. *Networks*, 16(2):161–174, 1986.

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

[133] Z. Su and B. De Schutter. Optimal scheduling of track maintenance activities for railway networks. *IFAC-PapersOnLine*, 51(9):386–391, 2018.

[134] M. Tagmouti, M. Gendreau, and J.-Y. Potvin. Arc routing problems with time-dependent service costs. *European Journal of Operational Research*, 181(1):30–39, 2007.

[135] É. Taillard, P. Badeau, M. Gendreau, F. Guertin, and J.-Y. Potvin. A tabu search heuristic for the vehicle routing problem with soft time windows. *Transportation science*, 31(2):170–186, 1997.

[136] K. C. Tan, L. H. Lee, Q. L. Zhu, and K. Ou. Heuristic methods for vehicle routing problem with time windows. *Artificial intelligence in Engineering*, 15(3):281–295, 2001.

[137] S.-Y. Tan and W.-C. Yeh. The vehicle routing problem: State-of-the-art classification and review. *Applied Sciences*, 11(21):10295, 2021.

[138] D. Taş, M. Gendreau, N. Dellaert, T. Van Woensel, and A. G. De Kok. Vehicle routing with soft time windows and stochastic travel times: A column generation and branch-and-price solution approach. *European Journal of Operational Research*, 236(3):789–799, 2014.

[139] R. Tavakkoli-Moghaddam, A. R. Saremi, and M. S. Ziaee. A memetic algorithm for a vehicle routing problem with backhauls. *Applied Mathematics and Computation*, 181(2):1049–1060, 2006.

[140] F. Theurich, A. Fischer, and G. Scheithauer. A branch-and-bound approach for a vehicle routing problem with customer costs. *EURO Journal on Computational Optimization*, 9:100003, 2021.

[141] P. Tittmann. *Einführung in die Kombinatorik*. Springer Spektrum, 2. edition, 2014.

[142] P. Toth and D. Vigo. An exact algorithm for the vehicle routing problem with backhauls. *Transportation science*, 31(4):372–385, 1997.

[143] P. Toth and D. Vigo. *The Vehicle Routing Problem*. Monographs on Discrete Mathematics and Applications. Society for Industrial and Applied Mathematics, 2002.

[144] P. Toth and D. Vigo. *Vehicle Routing Problems, Methods, and Applications*. Society for Industrial and Applied Mathematics, 2. edition, 2014.

[145] C. Vale, I. M. Ribeiro, and R. Calçada. Integer programming to optimize tamping in railway track as preventive maintenance. *Journal of Transportation Engineering*, 138:123–131, 2012.

[146] C. A. van Eijl. A polyhedral approach to the delivery man problem. Technical report, Department of Mathematics and Computing Science, Eindhoven University of Technology, 1995.

[147] J. I. van Zante-de Fokkert, D. den Hertog, F. J. van den Berg, and J. H. M. Verhoeven. The Netherlands schedules track maintenance to improve track workers' safety. *Interfaces*, 37(2):133–142, 2007.

[148] R. J. Vanderbei. *Linear Programming: Foundations and Extensions*. Springer, 5. edition, 2020.

[149] L. A. Wolsey. *Integer programming*. Wiley, 1998.

[150] R. T. Wong. Integer programming formulations of the traveling salesman problem. In *Proceedings of the IEEE international conference of circuits and computers*, pages 149–152. IEEE Press Piscataway NJ, 1980.

[151] S. Xie, C. Lei, and Y. Ouyang. A customized hybrid approach to infrastructure maintenance scheduling in railroad networks under variable productivities. *Computer-Aided Civil and Infrastructure Engineering*, 33(10):815–832, 2018.

[152] T. Zhang, J. Andrews, and R. Wang. Optimal scheduling of track maintenance on a railway network. *Quality and Reliability Engineering International*, 29(2):285–297, 2013.

[153] E. Zio, M. Marella, and L. Podofillini. Importance measures-based prioritization for improving the performance of multi-state systems: Application to the railway industry. *Reliability Engineering & System Safety*, 92(10):1303–1314, 2007.

List of Figures

List of Tables

.

Appendices

A. Benchmarks

In the following, benchmarks are defined that were used to compare the solution methods and heuristics experimentally. For heuristics, which do not guarantee to solve a problem optimally, the approximation quality is analyzed. Solution methods to optimally solve the VRPCC are compared in terms of computational time. For this purpose, computational experiments were made on 24 Intel Xeon CPUs with 2.93 GHz of a NUMA architecture. All developed algorithms and heuristics were written in the programming language Java. Mixed-integer linear programs were solved with CPLEX 12.8.

To evaluate and compare the solution approaches in terms of solution quality or computational time, performance profiles are used as presented in [41] which show cumulative distribution functions of different approaches for a selected performance measure.

Definition A.1. A particular VRPCC with defined input data is called an *instance*. A certain set of instances used for different computational experiments is called *benchmark*.

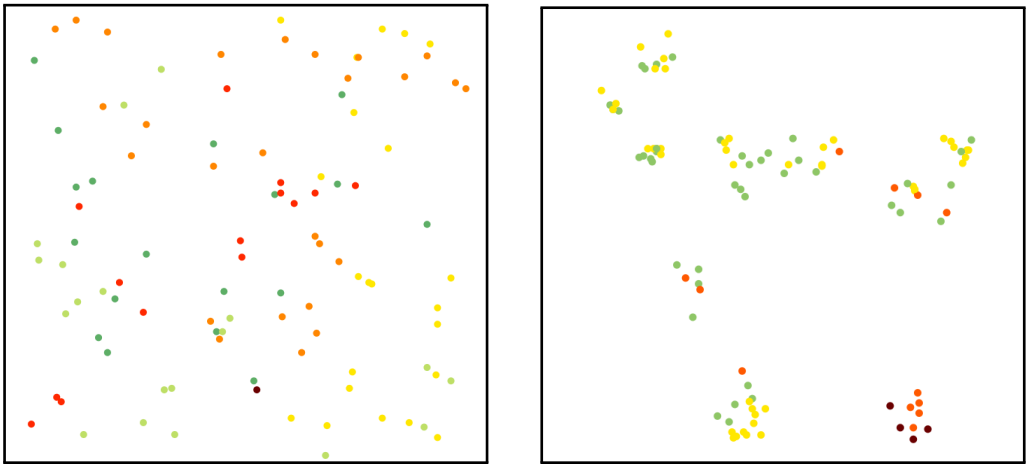
The computational tests were carried out with artificial instances, because real data of maintenance activities were rarely available. Furthermore, different companies have different maintenance strategies and approaches. Because of that, the maintenance data of several companies can be strongly different. To be able to test the solution methods and heuristics in general, the instances were created by random varying several design parameters which are explained in the next sections.

Working Shift and Start Time

The working shift has a length of eight hours which are 480 minutes. In all instances, the start times of the depots are set to $t_0 = (0, 480)$ because a rolling planning horizon is assumed which means that each maintenance machine ends them maintenance job at the end of the working shift and can travel over day to the first job of the new maintenance plan. Consequently, the first job can be visited on day one at the beginning of the working shift.

Definition of Job and Depot Locations

The location of a job or a depot is defined by an x- and a y-coordinate in a square with length l. Following the ideas of Marius M. Solomon[1], two variants to define job locations are used: uniformly distributed and clustered. In the uniformly distributed instances, the x- and y-coordinates of the job locations are integers, generated uniformly randomly in the interval $[1, l]$. In Figure A.1(a), an instance with 100 uniformly distributed jobs is shown.



(a) Uniformly distributed (b) Clustered

Figure A.1.: Example for the locations of 100 jobs.

The job locations of the clustered instances are defined as follows:

1. Select the number of clusters randomly from $[\frac{2}{3}\sqrt{n}, 2\sqrt{n}]$.
2. Generate the center $C \in \mathbb{R}^2 = (C_x, C_y)$ and the radius $R \in \mathbb{R}$ of each cluster whereby C_x and C_y are uniformly distributed in $[0.1l, 0.9l]$, and R is uniformly distributed in $[0.05l, 0.1l]$.
3. Create the locations of the jobs. For each job,
   3.1. choose a cluster randomly,
   3.2. generate a radius $r \in [0, R]$ and an angle $\phi \in [0, 2\Pi]$ by random, and
   3.3. calculate the coordinates as $x := \lfloor cos(\phi)r \rceil + C_x$ and $y := \lfloor sin(\phi)r \rceil + C_y$.

Figure A.1(b) shows an instance where the jobs location are defined clustered.

[1]The benchmark problems presented 1987 by Marius M. Solomon for the VRPTW [132] are common to analyze the performance of new developed solution approaches for the VRPTW. The geographical data of these benchmarks are generated randomly, clustered or randomly and clustered.

In both variants, the coordinates of the depots are either chosen randomly from the inner square $[0.25l, 0.75l] \times [0.25l, 0.75l]$ or all depots are defined in the midpoint $(\lfloor \frac{l}{2} \rceil, \lfloor \frac{l}{2} \rceil)$.

The travel costs and travel times are calculated based on the Euclidean distance between two jobs. Let l_i and l_j be the location of two jobs $i, j \in N_a$. Then, the travel costs and times are

$$d_{ij} := \lceil f_d \, \| l_i - l_j \|_2 \rceil \quad \text{and} \quad r_{ij} := \max\{5, \lceil f_t \, \| l_i - l_j \|_2 \rceil\},$$

With it, the triangle inequality is satisfied for the travel times and costs.

Definition of the Customer Cost Coefficients

The randomized definition of customer cost coefficients is based on three values: the ratio of jobs with non-zero customer cost coefficient $r_{\text{cc}} \in [0, 1]$, the type of customer cost distribution from the set $\{1, 2, 3\}$ and a factor $f_{cc} \in \mathbb{R}$. The first $\lfloor r_{\text{cc}} n \rceil$ jobs are afflicted with a non-zero customer cost coefficient. The customer cost coefficient of a job is computed by multiplying a random generated basic value with the customer cost factor f_{cc}. The product is rounded up to get an integer. Thereby, the basic value is chosen with a certain probability uniformly from a certain interval. For the three customer cost types, the probabilities and intervals are specified as shown in Table A.1. For example, the distribution of type three implies that a large base value is chosen with probability 0.1 and a small base value with probability 0.9. For each of the three customer cost types, the expected basic value is 40.

type	interval	probability
1	$[10, 70]$	1
2	$[10, 20]$	$1/3$
	$[30, 50]$	$1/3$
	$[60, 70]$	$1/3$
3	$[150, 200]$	$1/10$
	$[10, 40]$	$9/10$

Table A.1.: Base distributions to define customer cost coefficients.

Definition of Working Time

The working time of a job is the time needed for processing the job. It is defined by the parameter f_{wd} multiplied with a random value from $[0.5, 1.5]$.

	min	Q1	Q2	Q3	max
non-zero customer cost coefficient	8	21	33	49	256
travel time to nearest job	5	5	5	11	43
travel costs to nearest job	1	4	7	16	64
travel time to any job	5	18	46	68	127
travel costs to any job	1	26	69	102	190
working duration	57	106	133	177	240
percentage of $g^c(S^*)$	15.9%	37.5%	52.7%	65.3%	81.7%

Table A.2.: Characteristics of benchmark `S`.

Some Benchmark Classes

In the following, the three benchmark types `S`, `M` and `L` are described detailed.

Benchmark `S`

Benchmark `S` contains 180 instances. It is constructed to analyze the influence of the customer cost structure to the performance of a solution method.

Each instance consists of 15 jobs located in a square with a side length of 100 units which have to be served by two machines. For half of the instances, the job locations are uniformly distributed and for the other half, the locations are defined by clusters. For the latter, the number of clusters is between three and seven. The factor to calculate travel times from the Euclidean distance is one, and the travel costs factor is 1.5.

The customer costs are defined as follows: The customer cost factor f_{cc} is chosen randomly from $[0.5, 2]$. From the 90 instances with uniformly distributed job locations, 30 instances are generated with $r_{\text{cc}} = 0.33$, $r_{\text{cc}} = 0.66$ and $r_{\text{cc}} = 1$, respectively. From 30 instances with same ratio of jobs with non-zero customer cost coefficient, in ten instances the customer costs are defined as type 1, 2 and 3, respectively. The same holds for the 90 instances with clustered located jobs.

Table A.2 provides statistic values for customer cost coefficients, travel times and costs as well as working duration. Finally, the percentage of customer costs to the total costs in an optimal solution is provided. As it can be seen, there are instances more dominated by travel costs and instances more dominated by customer costs.

Benchmark `M`

Benchmark `M` collects 100 medium-sized instances. Each instance consists of 30 jobs which have to be served by two or three vehicles. The locations of the jobs are chosen randomly in a square of length 300, either uniformly distributed or clustered. The factors for travel costs and time are $f_d = f_t = 1$ for all instances. At least half

	min	Q1	Q2	Q3	max
jobs with non-zero customer cost coefficient	15	19	22	26	29
non-zero customer cost coefficient	6	26	47	77	471
travel time/costs to nearest job	5/1	7	14	25	117
travel time/costs to any job	5/1	72	136	196	389
working duration	33	73	97	133	216
percentage of $g^c(S)$	32.4%	53.5%	60.9%	68.8%	80.8%

Table A.3.: Characteristics of benchmark `M`.

	min	Q1	Q2	Q3	max
jobs with non-zero customer cost coefficient	25	43	62	75	98
non-zero customer cost coefficient	1	15	27	52	369
travel time/costs to nearest job	5/1	5	8	15	65
travel time/costs to any job	5/1	84	141	196	406
working duration	56	85	111	139	166
percentage of $g^c(S)$	16.7%	48.1%	62.1%	74.1%	88.1%

Table A.4.: Characteristics of benchmark `L`.

of jobs are afflicted with a non-zero customer cost coefficient. The customer cost factor f_{cc} is chosen uniformly distributed from $[0.5, 2.5]$ and the customer cost type is selected randomly between 1, 2 or 3. To determine the working duration, for each instance the factor f_{wd} is chosen randomly between four and nine.

Table A.3 gives statistic values for some characteristics of the instances of benchmark `M`: the number of jobs with non-zero customer cost coefficient, the customer cost coefficient, travel time/costs and working duration. Furthermore, the percentage of customer costs to the total costs in the best-known solution are shown.

Benchmark `L`

Benchmark `L` contains 100 large instances which includes 100 jobs that have to be served by three to five machines. The job locations are generated uniformly distributed or clustered in a square of length 300. The factors for travel costs and time are $f_d = f_t = 1$ for all instances. Between 25 and 100 jobs are afflicted with a non-zero customer cost coefficient. The customer cost type is selected randomly between 1, 2 or 3, and f_{cc} is chosen randomly from $[0, 2]$. The factor for the working time f_{wd} is equal to 23.

Table A.4 provides statistic values for the number of jobs with non-zero customer cost coefficient and the customer cost coefficients itself, travel times and costs as well as working duration. Finally, the percentage of customer costs to the total costs in

a best-known solution is given.

Because benchmark L is used to test heuristics and them sensitivity to the ratio of customer costs and travel costs, four additional variants of benchmark L are created by multiplying either the travel costs by factor ten or hundred or the customer cost coefficients by factor ten or hundred.

B. Algorithm Pseudocodes

Algorithm 1: Algorithm to fill empty routes (FILL)

Input: S
for $k = 1 \rightarrow m$ **do**
  if $N_k = \emptyset$ **then**
    $c_{\min} = \infty$
    for $l = 1 \rightarrow m$ **do**
      if $|N_l| > 1$ **then**
        for $j \in N_l$ **do**
          Create S' by shifting job j to route N_k.
          if $g(S') < c_{\min}$ **and** *all non-empty routes of* S' *are feasible* **then**
            $c_{\min} = g(S')$
            $j_B = j$
    Shift job j_B to route N_k in S.
return S

Algorithm 2: Nearest neighbor heuristic (NN)

Input: $f \geq 0$ *// flexibility: allowed variation from the minimal travel cost value*

$\widetilde{S} := (\emptyset, ())_{k \in M}$

$\widetilde{N} := N$

while $\widetilde{N} \neq \emptyset$ **do**

  $I = \{(j,k) |\, j \in \widetilde{N},\ k \in M,\ (j,k) \text{ is feasible}\}$

  if $|I| = 0$ **then**

    return *uncompleted schedule* $\widetilde{S}$

  Let $l_k \in N \cup N_s$ be the current last job of route $k \in M$.

  $d_{\min} := \min_{(j,k) \in I} d_{l_k j}$

  $c_{max} := \max_{(j,k) \in I} \{c_j |\, d_{l_k j} \leq d_{\min}(1+f)\}$

  $(j_B, k_B) := \arg\min_{(j,k) \in I} \{d_{l_k j} |\, c_j = c_{max}\}$

  Append job j_B at the end of route Π^{k_B} and set $N_{k_B} = N_{k_B} \cup \{j_B\}$ in $\widetilde{S}$

  $\widetilde{N} = \widetilde{N} \setminus \{j_B\}$

If necessary, fill empty routes.

return *completed schedule* S

Algorithm 3: Most-expensive neighbor heuristic (MEN)

Input: $f \in (0, 1]$ *// flexibility: allowed variation from the most-expensive job*

$\widetilde{S} := (\emptyset, ())_{k \in M}$

$\widetilde{N} := N$

while $\widetilde{N} \neq \emptyset$ **do**

  $c_{\max} = \max\{c_j | \, j \in \widetilde{N}\}$

  $e_{\min} = \infty$

  forall $j \in \widetilde{N}$ **do**

    if $c_j \geq f c_{\max}$ **then**

      for $k = 1 \to m$ **do**

        if (j, k) *is feasible* **then**

          Let l_k be the current last job of route k.

          $(t^d, t^m) := \zeta(\xi(t_{l_k}) + a_{l_k} + r_{l_k j}, u_j)$

          $e(j, k) := d_{l_k j} + t^d c_{\max}$

          if $e_{min} > e(j, k)$ **then**

            $e_{\min} = e(j, k)$

            $(j_B, k_B) = (j, k)$

  if (j_B, k_B) **is** *None* **then**

    return *uncompleted schedule* $\widetilde{S}$

  Append job j_B at the end of route Π^{k_B} and set $N_{k_B} = N_{k_B} \cup \{j_B\}$ in $\widetilde{S}$.

  $\widetilde{N} = \widetilde{N} \setminus \{j_B\}$

If necessary, fill empty routes.

return *completed schedule* $\widetilde{S}$

Algorithm 4: Cost-balanced neighbor heuristic (CBN)

Input: $\beta \in [0,1]$ *// weight to balance travel costs and customer costs*
n_{day} *// estimated number of jobs visited per day and vehicle, // see equation (5.1) on page 71*

$\widetilde{S} := (\emptyset, ())_{k \in M}$
$\widetilde{N} := N$
while $\widetilde{N} \neq \emptyset$ **do**
  $c_{\min} = \infty$
  forall $j \in \widetilde{N}$ **do**
    for $k = 1 \to m$ **do**
      if (j,k) *is feasible* **then**
        Let l_k be the current last job of route k.
        $c_b(j,k) = \beta d_{l_k j} - (1-\beta)\frac{\frac{n}{m} - |N_k|}{n_{\text{day}}} c_j$
        if $c_{min} > c_b(j,k)$ **then**
          $c_{\min} = c_b(j,k)$
          $(j_B, k_B) = (j,k)$
  if (j_B, k_B) **is** *None* **then**
    return *uncompleted schedule* $\widetilde{S}$
  Append job j_B at the end of route Π^{k_B} and set $N_{k_B} = N_{k_B} \cup \{j_B\}$ in $\widetilde{S}$.
  $\widetilde{N} = \widetilde{N} \setminus \{j_B\}$
If necessary, fill empty routes.
return *completed schedule* $\widetilde{S}$

Algorithm 5: Best-of-Greedy algorithm (BoG)

$S_{\text{best}} := \{(\emptyset, ())_{k \in M}\}$ with $g(\{(\emptyset, ())_{k \in M}\}) = \infty$
for $f \in \{0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1\}$ **do**
    $S :=$ schedule from NN heuristic with flexibility f
    if $g(S) < g(S_{best})$ **then**
        $S_{\text{best}} = S$
for $f \in \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1\}$ **do**
    $S :=$ schedule from MEN heuristic with flexibility f
    if $g(S) < g(S_{best})$ **then**
        $S_{\text{best}} = S$
for $\beta \in \{0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1\}$ **do**
    $S :=$ schedule from CBN heuristic with weight β
    if $g(S) < g(S_{best})$ **then**
        $S_{\text{best}} = S$
return S_{best}

Algorithm 6: Rollout algorithm

$\widetilde{S} := (\emptyset, ())_{k \in M}$
$\widetilde{N} := N$
while $\widetilde{N} \neq \emptyset$ **do**
  $g_{\min} = \infty$
  forall $j \in \widetilde{N}$ **do**
    $\widetilde{N} := \widetilde{N} \setminus \{j\}$
    for $k = 1 \rightarrow m$ **do**
      if (j, k) *is feasible* **then**
        Append job j at the end of route Π^k and set $N_k = N_k \cup \{j\}$.
        Let S_{H} be the schedule obtained by completing $\widetilde{S}$ by means of a greedy heuristic.
        if S_H *is feasible* **and** $g(S_H) < g_{min}$ **then**
          $g_{\min} = g(S_{\mathrm{H}})$
          $(j_B, k_B) = (j, k)$
        Remove job j from Π^k and set $N_k = N_k \setminus \{j\}$.
    $\widetilde{N} := \widetilde{N} \cup \{j\}$
  if (j_B, k_B) **is** *None* **then**
    return *uncompleted schedule* $\widetilde{S}$
  Append job j_B at the end of route Π^{k_B} and set $N_{k_B} = N_{k_B} \cup \{j_B\}$ in $\widetilde{S}$.
  $\widetilde{N} = \widetilde{N} \setminus \{j_B\}$
If necessary, fill empty routes.
return *completed schedule* $\widetilde{S}$

Algorithm 7: Base algorithm for first and best improvement

Input: K *// number of improvement runs*

Compute S by the BoG heuristic.

$S_{\text{best}} = S$

$g_{\text{min}} = g(S_{\text{best}})$

for K *runs* **do**

  do

    $S =$ first/best improved neighbor of S

  while *an improved neighbor of* S *exists*

  if $g(S) < g_{min}$ **then**

    $g_{\text{min}} = g(S)$

    $S_{\text{best}} = S$

  / Compute start schedule for the next iteration which should be at least 15% more expensive than last obtained schedule S. */*

  $g = 1.15\, g(S)$

  while $g(S) < g$ **do**

    $S =$ random neighbor of S

return S_{best}

Algorithm 8: Compute $\mathrm{LB}^c_{\mathrm{LPBP1}}$ with polynomial effort.

Input: N_c *// set of jobs with non-zero customer cost coefficient*

$T = \{t_1, t_2, \ldots, t_{|T|}\}$ *// start days with* $t_j < t_{j+1}\ \forall j \in \{1, 2, \ldots, |T|-1\}$

$\mathrm{LB}^c_{\mathrm{LPBP1}} = 0$

Sort the jobs decreased by $\frac{c_i}{a_i+r_i}$ to get $\frac{c_1}{a_1+r_1} \geq \frac{c_2}{a_2+r_2} \geq \cdots \geq \frac{c_{n_c}}{a_{n_c}+r_{n_c}}$.

$b = 0$

$j = 1$

for $i = 1$ **to** n_c **do**

  if $b + a_i + r_i \leq mu + R_m$ **then**

    $\mathrm{LB}^c_{\mathrm{LPBP1}} = \mathrm{LB}^c_{\mathrm{LPBP1}} + t_j c_i$

    $b = b + a_i + r_i$ *// Pack job i completely into bin* t_j.

  else

    $\mathrm{LB}^c_{\mathrm{LPBP1}} = \mathrm{LB}^c_{\mathrm{LPBP1}} + \left(t_j \frac{mu+R_m-b}{a_i+r_i} + t_{j+1}\left(1 - \frac{mu+R_m-b}{a_i+r_i}\right)\right) c_i$

    $j = j + 1$

    $b = a_i + r_i - (mu + R_m - b)$ *// Pack job i fractional into bin* t_j *and* t_{j+1}.

return LB^c_{LPBP1}

Algorithm 9: Branch-and-bound with branching strategy append (BB-Append).

Input: $N_{\text{sort}} = (j_1, j_2, \ldots, j_n)$
Calculate an heuristics solution S.
Initialize the best solution so far $S_{\text{best}} = S$ and the upper bound $\text{UB} = g(S)$.
Start with the first partial solution $\widetilde{S} = (\{j_1\}, (j_1), \emptyset, (), \ldots, \emptyset, ())$ and $\widetilde{N} = N \setminus \{j_1\}$.
while $\widetilde{S}$ *is not empty* **do**
  if *the current partial solution* $\widetilde{S}$ *can lead to a feasible solution* **then**
    if *all jobs are planned* **then**
      if $g(\widetilde{S}) < UB$ **then**
        $\text{UB} = g(\widetilde{S})$
        $S_{\text{best}} = \widetilde{S}$
      $\widetilde{S} =$ **backtrack**$(\widetilde{S})$
    else
      if $g(\widetilde{S}) < UB$ **then**
        Calculate lower bound LB.
      else
        Set $\text{LB} = 0$.
      if $g(\widetilde{S}) + LB < UB$ **then**
        Further investigate the branch:
        Search first job j in N_{sort} which is unplanned in $\widetilde{S}$.
        Append j to the route where the last job was appended to.
      else
        $\widetilde{S} =$ **backtrack**$(\widetilde{S})$
  else
    $\widetilde{S} =$ **backtrack**$(\widetilde{S})$
return S_{best}

Algorithm 10: Backtracking for branching strategy append.

Input: $\widetilde{S}$
while $\widetilde{S}$ *is not empty* **do**
  Let j be the last appended job and
  k the corresponding route.
  if $k < m$ **then**
    Shift j to route $k+1$.
    return $\widetilde{S}$
  Search for first j' behind j in the sorted list with j' is unplanned.
  if j' *exists* **then**
    Remove j from route k.
    Let k' be the non-empty route with largest index or, if all routes are empty, $k' = 1$. Append j' to route k'.
    return $\widetilde{S}$
  Remove job j from route k.
return $\widetilde{S}$

Algorithm 11: Branch-and-bound with branching strategy include (BB-Include).

Input: $N_{\text{sort}} = (j_1, j_2, \ldots, j_n)$
Calculate an heuristics solution S.
Initialize the best solution so far $S_{\text{best}} = S$ and the upper bound $\text{UB} = g(S)$.
Start with the first partial solution $\widetilde{S} = (\{j_1\}, (j_1), \emptyset, (), \ldots, \emptyset, ())$ and $\widetilde{N} = N \setminus \{j_1\}$.
while $\widetilde{S}$ *is not empty* **do**
  if *the current partial solution* $\widetilde{S}$ *can lead to a feasible solution* **then**
    if *all jobs are planned* **then**
      if $g(\widetilde{S}) < UB$ **then**
        $\text{UB} = g(\widetilde{S})$
        $S_{\text{best}} = \widetilde{S}$
      $\widetilde{S} =$ **backtrack**$(\widetilde{S})$
    else
      if $g(\widetilde{S}) < UB$ **then**
        Calculate lower bound LB.
        if $LB < UB$ **then**
          Further investigate the branch:
          Include the first unplanned job of N_{sort} as last of route 1.
        else
          $\widetilde{S} =$ **backtrack**$(\widetilde{S})$
      else
        $\widetilde{S} =$ **backtrack**$(\widetilde{S})$
  else
    $\widetilde{S} =$ **backtrack**$(\widetilde{S})$
return S_{best}

Algorithm 12: Backtracking for branching strategy include.

Input: partial solution $\widetilde{S}$
while $\widetilde{S}$ *is not empty* **do**
    Let j be the last included job,
    k the corresponding route and
    p its position in the route.
    if $p > 1$ **then**
        Switch j and its predecessor. Then, j is on position $p-1$ in route k.
        return $\widetilde{S}$
    if $k < m$ **then**
        Shift j to route $k+1$ and append it to the end of the route.
        return $\widetilde{S}$
    Remove job j from route k.
return $\widetilde{S}$

Erklärung des Promovierenden
zum Antrag auf Eröffnung des Promotionsverfahrens

1. Die folgende Promotionsordnung in ihrer gültigen Fassung erkenne ich an:

   Bereich Mathematik und Naturwissenschaften
   Promotionsordnung vom 23.02.2011

2. Die Promotion wurde an folgendem Institut/an folgender Professur durchgeführt:

   Institut für Numerische Mathematik der Technischen Universität Dresden
   in einem kooperativen Verfahren mit der Hochschule Mittweida

3. Folgende Personen haben die Promotion wissenschaftlich betreut und/oder mich bei der Auswahl und Auswertung des Materials sowie bei der Herstellung des Manuskripts unterstützt:

   Prof. Dr. Andreas Fischer
   Dr. Guntram Scheithauer
   Dr. Ute Gläser
   Axel Simroth
   Prof. Dr. Peter Tittmann
   Prof. Dr. Thomas Kalinowski

4. Ich bestätige, dass für meine Person bisher keine früheren, erfolglosen Promotionsverfahren stattgefunden haben.

5. Ich versichere weiterhin, dass

   (a) ich die vorliegende Arbeit mit dem Titel „On a Vehicle Routing Problem with Customer Costs and Multi Depots" ohne unzulässige Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel selbst angefertigt habe. Hilfe Dritter wurde nur in wissenschaftlich vertretbarem und prüfungsrechtlich zulässigem Ausmaß in Anspruch genommen. Es sind keine unzulässigen geldwerten Leistungen, weder unmittelbar noch mittelbar, im Zusammenhang mit dem Inhalt der vorliegenden Dissertation an Dritte erfolgt.

   (b) die aus fremden Quellen direkt oder indirekt übernommenen Gedanken als solche kenntlich gemacht sind.

   (c) ich die vorliegende Arbeit bisher weder im Inland noch im Ausland in gleicher oder ähnlicher Form einer anderen Prüfungsbehörde zum Zwecke einer Promotion oder eines anderen Prüfungsverfahrens vorgelegt habe.

6. Mir ist bekannt, dass die Nichteinhaltung dieser Erklärung oder unrichtige Angaben zum Verfahrensabbruch oder zum nachträglichen Entzug des Doktortitels führen können.

Dresden, 07.07.2022
Ort, Datum

Unterschrift: Antragsteller:in