EURO Journal on Computational Optimization 9 (2021) 100003

Contents lists available at ScienceDirect

EURO Journal on Computational Optimization

journal homepage: www.elsevier.com/locate/ejco

A branch-and-bound approach for a Vehicle Routing Problem with Customer Costs

Franziska Theurich[a,1,*], Andreas Fischer[b], Guntram Scheithauer[b]

[a] Fraunhofer IVI, Zeunerstraße 38, Dresden 01069, Germany
[b] Faculty of Mathematics, Technische Universität Dresden, Dresden 01062, Germany

ARTICLE INFO

Keywords:
Vehicle Routing Problem
Railway maintenance scheduling
Customer costs
Branch-and-bound

ABSTRACT

An important aspect in railway maintenance management is the scheduling of tamping actions in which two aspects need to be considered: first, the reduction of travel costs for crews and machinery; and second, the reduction of time-dependent costs caused by bad track condition. We model the corresponding planning problem as a Vehicle Routing Problem with additional customer costs. Due to the particular objective function, this kind of Vehicle Routing Problem is harder to solve with conventional methods. Therefore, we develop a branch-and-bound approach based on a partition and permutation model. We present two branching strategies, the first appends one job at the end of a route in each branching step and the second includes one job inside a route in each branching step; and analyze their pros and cons. Furthermore, different lower bounds for the customer costs and the travel costs are defined and compared. The performance of the branch-and-bound method is analyzed and compared with a commercial solver.

1. Introduction

An important issue of railway infrastructure maintenance is ballast tamping to stabilize the whole track line. If the ballast is too loose, the track becomes unstable and derailments may occur. To prevent damage and accidents, train speed or axle load has to be reduced. These limitations of the railway service lead to penalty costs for the maintenance operator. In the past, track sections with limitations were maintained preferentially, while the increase in travel costs of tamping machines caused by this strategy were not taken into account. To reduce the overall maintenance costs, the aim of our ballast tamping scheduling problem is to compute a maintenance plan that minimizes the sum of travel and penalty costs.

To this end, in Heinicke et al. (2013) a new variant of the Vehicle Routing Problem (VRP) is developed which is called *Vehicle Routing Problem with Customer Costs* (short VRP-CC). The additional time-dependent customer costs are introduced to consider penalties for the limitation of railway service. This novel variant of VRP does not include capacity constraints, but has a new kind of objective function. Not only the travel costs are minimized, rather a sum of travel costs and costs that depend on the time of job execution are considered.

Based on this, Heinicke et al. (2015) provides several formulations of the VRP-CC as mixed integer linear program (MILP). We have seen that the new objective function leads to problems whose computational solution is significantly harder. Hence, more efficient solution approaches have to be developed. To this end, exploiting the cost structure, we will show how the branch-and-bound approach can be tailored to solve the VRP-CC faster. In particular, we provide two different branching strategies based on a non-linear formulation in order to calculate the execution times of the jobs directly from a feasible schedule; we present several lower bounds for the customer cost part of the objective function; and we give suitable lower bounds for the travel costs. By means of computational tests, we analyze which lower bounds lead to the best performance of the branch-and-bound methods. The needed upper bounds are calculated based on heuristics proposed in Heinicke et al. (2013). Finally, we compare the performance of the developed branch-and-bound approaches against solving a corresponding MILP by means of a commercial solver (CPLEX). These computational results show significant improvements for our approach.

Branch-and-bound is a common principle to solve complex combinatorial problems. An introduction to the method can be found in Balas (1977). The main idea is to successively break up the solution space into certain subsets (branches). In order to discard some of these subsets and to reduce the solution space, lower bounds for the objective function (that shall be minimized) over the subsets are calculated. If the lower bound of a subset is larger than an already known objective value of a feasible point (upper bound), then this subset is removed. In general, the tighter the bounds are, the more subsets can be discarded.

* Corresponding author
E-mail address: franziska.theurich@ivi.fraunhofer.de (F. Theurich).
[1] Formerly F. Heinicke.

https://doi.org/10.1016/j.ejco.2020.100003
Received 6 October 2020; Accepted 6 October 2020

Let us now comment on some branch-and-bound methods developed for problems related to the VRP-CC. A branch-and-bound method for a traveling salesman problem (TSP) with m salesmen was presented by Gavish and Srikanth (1986). They used a depth first search branching on the binary variables that describe whether city j is visited directly after city i. To obtain a lower bound, a Lagrangian technique is applied to the constraints which enforce that each node has degree 2 in the solution. The resulting problem is to find a minimal spanning tree for the modified cost matrix. Lucena (1990) presented a time-dependent TSP and a branch-and-bound method for its solution. There, the travel time depends on the position of the city in the tour. In a branching step, a city not visited so far is selected and for each open position a new branch is generated in which this city is allocated to that position. To obtain a lower bound for the subproblem, a Lagrangian relaxation is used that neglects the constraint that each city has to be visited. The resulting problem is solved with dynamic programming. In Fisher et al. (1997) a VRP with time windows is solved by a branch-and-bound algorithm with Lagrangian relaxation and compared with a K-tree approach. The Lagrangian relaxation leads to a shortest path problem with time windows and capacity constraints. In this problem, it is not ensured that each city is visited exactly once by exactly one vehicle. Baker (1983) described a branch-and-bound method for the time-constrained TSP. A notable fact is that the problem is modeled without binary variables; the only decision variable is the time to visit each city. Lower bounds are obtained based on a relaxation that leads in the dual problem to a longest path problem. The dissertation of Kohl (1995) shows different decomposition methods for the VRP with time windows, which lead to shortest path problems with time windows and capacity constraints as lower bound embedded in a branch-and-bound approach. The shortest path problems are solved by means of a dynamic programming algorithm. In Laporte et al. (1988), a branch-and-bound approach for a variant of the TSP resulting from the Location Routing Problem is shown. The objective function of the subproblems is estimated using a relaxation that leads to a constrained assignment problem. To generate new subproblems, the solution of the relaxation is checked for violations of the constraints that are relaxed.

Among the lower bounds for the travel costs of the TSP, two main relaxations are useful for our branch-and-bound approach. Firstly, removing the subtour elimination constraints yields an assignment problem. Secondly, relaxing the node degree constraints leads to a particular minimum spanning tree problem, the 1-tree problem. Both relaxations can be solved in polynomial time. An improvement of the assignment bound can be found in Christofides (1972) and an enhancement of the bound obtained by 1-trees is presented in Held and Karp (1971). In Section 4.2, we discuss both variants in more detail.

In this paper, we present a branch-and-bound approach to solve the more specific VRP-CC. The paper is structured as follows: in the next section, we develop a discrete non-linear model of VRP-CC. Then, in Section 3, we propose two branching strategies and discuss their pros and cons. Different lower bounds for customer costs and travel costs are provided and compared in Section 4. The computational performance of the branch-and-bound algorithms for the non-linear model are investigated in Section 5. This includes a comparison with the application of CPLEX to a MILP model. The conclusions in Section 6 complete the presentation.

2. The Vehicle Routing Problem with Customer Costs

Let $N = \{1, 2, \dots, n\}$ be a set of jobs that have to be scheduled to a set of machines $M = \{1, 2, \dots, m\}$. Each machine $k \in M$ has a start depot s_k and an end depot z_k modeled as artificial jobs. The depot sets are given as $N_s := \{s_k : k \in M\}$ 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} from job $i \in N_a$ to job $j \in N_a, j \neq i$, and
- the travel time r_{ij} from job $i \in N_a$ to job $j \in N_a, j \neq i$.

All values are supposed to be non-negative integers and 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.

A feasible solution or *schedule* S of the VRP-CC consists of a partition $(N_k)_{k \in M}$ of N into m non-empty subsets and, for each $k \in M$, a permutation $\Pi^k(N_k) = (\pi_1^k, \pi_2^k, \dots, \pi_{|N_k|}^k)$ defining the *route* of machine k which is $s_k \mapsto \pi_1^k \mapsto \pi_2^k \mapsto \cdots \mapsto \pi_{|N_k|}^k \mapsto z_k$ with $\pi_i^k \in N_k$. For short, a schedule is represented by $S := (N_k, \Pi^k(N_k))_{k \in M}$.

In difference to many VRPs, which only (or mainly) consider the travel costs $g^d(S)$, in the VRP-CC an additional cost components $g^c(S)$, called customer costs, has to be regarded for each schedule S. Hence, the VRP-CC can be formulated as the following non-linear partition and permutation problem:

$$\begin{aligned} \text{(PPP)}\ \min\ & g(S) = g^d(S) + g^c(S) \\ \text{s.t.}\ & S := \left(N_k, \Pi^k\left(N_k\right)\right)_{k \in M}, \\ & \left(N_k\right)_{k \in M} \text{is a partition of } N \text{ into exactly } m \text{ non-empty subsets}, \\ & \Pi^k\left(N_k\right) \text{is a permutation of the elements of } N_k, k \in M. \end{aligned} \tag{1}$$

Whereas the travel costs of S depend on the particular order of the jobs, namely

$$g^d(S) := \sum_{k \in M} \left(d_{s_k \pi_1^k} + \sum_{i=1}^{|N_k|-1} d_{\pi_i^k \pi_{i+1}^k} + d_{\pi_{|N_k|}^k z_k}\right), \tag{2}$$

and the customer costs of S are determined by the day $t_i^d \in \mathbb{N}$ on which job $i \in N$ is executed resulting from the particular route $\Pi^k(N_k)$ to which job i belongs:

$$g^c(S) := \sum_{i \in N} c_i t_i^d. \tag{3}$$

To calculate the day of execution for all jobs, we have to analyze a special feature of the above scheduling problem. Here, tamping is only possible during an eight-hour working shift in the night, but traveling of the tamping machine is possible all day long. To model the working shift, each day any tamping may start at minute 0 and has to end at minute $u := 480$ at the latest. Thus, a job i cannot start if the machine reaches the location after minute $u_i := u - a_i$. In this case, the job has to be postponed to the next day. For a correct scheduling, we calculate the *execution time* t_i of a job i, which is the time when the execution of the job starts, in a special way. We split it into two components: $t_i = (t_i^d, t_i^m)$. The first component $t_i^d \in \mathbb{N}$ represents the execution day and is the base for calculating the customer costs. The second component $t_i^m \geq 0$ is the execution minute on day t_i^d that is restricted by the working shift, see condition (6) below. To avoid infeasibility of an instance, the duration of any job $i \in N$ must be bounded by $a_i \leq u$. Since travel time and working duration are in minutes, the execution time $t_i = (t_i^d, t_i^m)$ of job i is converted into minutes by the function $\xi : \mathbb{N} \times \mathbb{R} \to \mathbb{R}$ with

$$\xi(t_i) := h\, t_i^d + t_i^m, \tag{4}$$

where $h := 1440$ represents the minutes per day. The execution 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:

$$\xi(t_i) \geq \xi(t_{p_i}) + a_{p_i} + r_{p_i i} \tag{5}$$

and

$$0 \leq t_i^m \leq u_i. \tag{6}$$

Condition (5) ensures that the job i does not start before its predecessor job p_i is completed and the machine has traveled to job i. By restriction (6), the compliance with the working shift is guaranteed. For each job $i \in N$, the arrival time t_i^a is the time when the machine reaches the location of job i and is calculated as

$$t_i^a := \xi(t_{p_i}) + a_{p_i} + r_{p_i i}, \tag{7}$$

where t_{p_i} is the execution time of predecessor job p_i which needs to be already computed. Moreover we define the function $\zeta : \mathbb{R} \times \mathbb{N} \to \mathbb{N} \times \mathbb{R}$ to convert a time t, given in minutes, into a pair (t^d, t^m) that satisfies the working shift condition (6):

$$\zeta(t, u_i) := \begin{cases} \left(\left\lfloor \frac{t}{h} \right\rfloor, t - \left\lfloor \frac{t}{h} \right\rfloor \cdot h \right) & \text{if } t - \left\lfloor \frac{t}{h} \right\rfloor \cdot h \le u_i, \\ \left(\left\lfloor \frac{t}{h} \right\rfloor + 1, 0 \right) & \text{otherwise.} \end{cases} \tag{8}$$

Then, the execution times of the jobs are calculated based on the schedule $S = (N_k, \Pi^k(N_k))_{k \in M}$. For the start depots, we define the start time $t_0 = (0, u)$ and set

$$t_{\pi_0^k} := t_{s_k} = t_0, \ k \in M. \tag{9}$$

Thus, each route starts at the end of day 0 in the start depot. The execution time of the jobs are calculated in the order defined by the permutations based on the arrival time as

$$t_{\pi_i^k} := \zeta\left(t^a_{\pi_i^k}, u_{\pi_i^k}\right) = \zeta\left(\xi\left(t_{\pi_{i-1}^k}\right) + a_{\pi_{i-1}^k} + r_{\pi_{i-1}^k \pi_i^k}, u_{\pi_i^k}\right), \quad i = 1, 2, \dots, |N_k|, \ k \in M. \tag{10}$$

This discrete non-linear model, described by S, the objective function $g(S)$ (1) with travel costs (2) and time-dependent customer costs (3), and the calculation of the execution times via the Eqs. (9) and (10), is very intuitive. The advantages of this model are the low memory requirements, a solution can be stored in an array of length $n + 2m$, and that the execution times are calculated directly from the so defined schedule. In our branch-and-bound approach, we want to profit from the possibility to calculate the execution time directly based on the schedule. Therefore, we use the non-linear model and develop a branch-and-bound method to solve it. Later on, this approach shall be compared with the commercial solver CPLEX applied to a linearized model. In Heinicke et al. (2015), different linearizations of VRP-CC (i.e., MILP formulations) were developed with the consequence that the execution times become decision variables. It turned out that CPLEX reached its best performance for a two-index formulation of the routing variables and a big-$\mathcal{M}$ linearization of time constraints. Therefore, we use this model for comparison and call it here (MILP). In this model, the routes are arranged one after another so that one tour visiting all jobs and depots represents a schedule.

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

$$\text{s.t.} \sum_{j \in N_a \setminus \{i\}} x_{ij} = 1, \qquad i \in N_a \tag{11}$$

$$\sum_{j \in N_a \setminus \{i\}} x_{ji} = 1, \qquad i \in N_a \tag{12}$$

$$x_{z_m s_1} = 1 \tag{13}$$

$$x_{z_k s_{k+1}} = 1, \qquad k \in M \setminus \{m\} \tag{14}$$

$$x_{s_k z_l} = 0, \qquad k, l \in M \tag{15}$$

$$x_{ij} \in \{0, 1\}, \qquad i, j \in N_a \tag{16}$$

$$y_{s_k} = k, \qquad k \in M \tag{17}$$

$$y_{z_k} = k, \qquad k \in M \tag{18}$$

$$y_i - y_j \le (1 - x_{ij})\, m, \qquad i \in N_s \cup N, \ j \in N_z \cup N \tag{19}$$

$$y_i \in \{1, 2, \dots, m\}, \qquad i \in N_a \tag{20}$$

$$\left(t^d_{s_k}, t^m_{s_k}\right) = t_0, \qquad k \in M \tag{21}$$

$$h\, t_i^d + t_i^m - h\, t_j^d - t_j^m + \left(\mathcal{M} + a_i + r_{ij}\right) \cdot x_{ij} \le \mathcal{M}, \quad i \in N_s \cup N, j \in N_z \cup N, \ i \ne j \tag{22}$$

$$t_0^d \le t_i^d \le d_{\max}, \quad t_i^d \in (\mathbb{N}), \qquad i \in N_a \tag{23}$$

$$0 \le t_i^m \le u_i, \quad t_i^m \in (\mathbb{R}), \qquad i \in N_a \tag{24}$$

In (MILP), the binary variables x_{ij} are 1, if and only if job i is the predecessor of j in the solution. The constraints (11) and (12) ensure that each job has one predecessor and one successor. The trips between the depots are regulated by the Eqs. (13)–(15). The integer variable y_i represents the machine that executes job i. This is necessary to guarantee the correct order of the depots. Eqs. (17) and (18) allocate the depots to the correct machine. Inequality (19) ensures that the jobs between a start depot and an end depot are allocated to the same machine as the depots itself. The execution time of a job $i \in N_a$ is given by the decision variables t_i^d and t_i^m, with the same meaning as before. By inequality (22), with a sufficiently large $\mathcal{M}$, it is guaranteed that $h\, t_j^d + t_j^m \ge h\, t_i^d + t_i^m + a_i + r_{ij}$ holds if i is predecessor of j, which means that job j does not start before the predecessor is finished and the machine has reached the location of job i.

3. The branching strategies

The definition of the branching strategies relies on the formulation (PPP). Thus, we do not branch in respect to binary or integer variables from a MILP formulation, but we create new subproblems by integrating a further job to a partial solution. A partial solution $\widetilde{S}$ is defined by the set of not yet planned jobs $\widetilde{N}$ that will also be called unplanned jobs, and the uncompleted schedule $(\widetilde{N}_k, \Pi^k(\widetilde{N}_k))_{k \in M}$, whereby $\widetilde{N} \cup \left(\bigcup_{k \in M} \widetilde{N}_k\right) = N$. Let $\mathcal{S}$ denote the set of all open partial solutions. At the beginning, we set $\mathcal{S} := \{(N, (\emptyset, ())_{k \in M})\}$. In a branching step, we select one open partial solution $\widetilde{S} \in \mathcal{S}$ and create a set of new partial solutions by integrating one job $i \in \widetilde{N}$ into the uncompleted schedule. We consider two strategies how a job is integrated. The first branching strategy produces an unbalanced search tree, but permits the application of tighter bounds in comparison to the second branching strategy, which, however, leads to a more balanced search tree. Both strategies will be described in detail in the next subsections and compared via computational experiments in Section 5.

3.1. Append a job on a partial solution

With the first branching strategy, we create a set of new partial solutions by appending jobs at the end of routes. We call this strategy shortly *Append*. The motivation of this approach is that travel costs and customer costs of the already appended jobs are fixed, which permits lower bounds that are more precise. Furthermore, the costs of a partial solution can be determined very efficiently since only the execution time of the appended job has to be calculated to add the resulting travel costs and customer costs of this job.

In detail, the branching strategy Append works as follows: Let $\widetilde{S} \in \mathcal{S}$ be a partial solution. We create a set of new partial solutions by appending each unplanned job at the end of certain routes. To avoid repeated solutions (which would be generated very often otherwise), we fill the routes one after the other. To this end, index $l := l(\widetilde{S}) \in M$ denotes the route where the last job was appended to this partial solution. We do not permit to append a job to a route $j < l$ in the following branches. Because of that, we call the routes $1, 2, \dots, l-1$ closed and the routes $l, l+1, \dots, m$ open. Note, that the routes $l+1, \dots, m$ of the partial solution are empty. Further, it is not allowed to append a job to a route $j > l+1$ because otherwise route $l+1$ will remain empty which is not feasible. The index of the first partial solution is set to $l((N, (\emptyset, ())_{k \in M})) = 0$.

If $l = l(\widetilde{S}) > 0$, we create for each unplanned job $i \in \widetilde{N}$ one new partial solution by appending this job to the route l which leads to the new

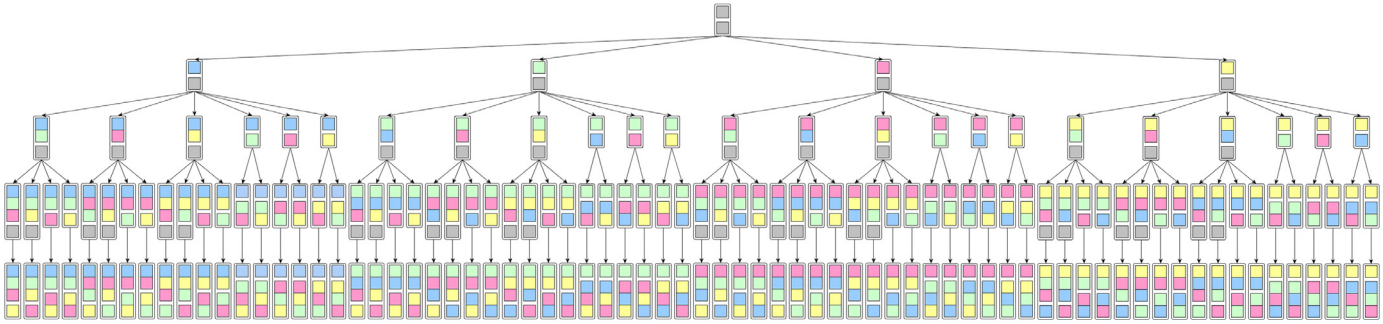

Fig. 1. Example for a search tree for branching strategy Append.

partial solutions

$$S_{i,l} = \left(\tilde{N} \setminus \{i\}, \left(\tilde{N}_l \cup \{i\}, \left(\pi_1^l, \ldots, \pi_{|\tilde{N}_l|}^l, i\right)\right), \left(\tilde{N}_k, \Pi^k(\tilde{N}_k)\right)_{k \in M \setminus \{l\}}\right).$$

And, if $l = l(\widetilde{S}) < m$, we create for each unplanned job $i \in \widetilde{N}$ one new partial solution by initializing route $l+1$ with this job, which leads to

$$S_{i,l+1} = \left(\tilde{N} \setminus \{i\}, (\{i\}, (i)), \left(\tilde{N}_k, \Pi^k(\tilde{N}_k)\right)_{k \in M \setminus \{l+1\}}\right)$$

With it, we create $|\widetilde{N}|$ or $2|\widetilde{N}|$ new partial solutions.

We illustrate the branching strategy Append by means of a small example with four jobs and two routes in Fig. 1. Each node of the search tree defines a partial solution, which is represented by the order of the jobs of route one and route two from top to bottom by means of colored boxes for the jobs. An empty route is symbolized by a gray square. Starting with two empty routes, we create four branches each starting with one job in route one. Based on one of these partial solutions, either one job is appended to route one or to route two. In branches where the second job was appended to route two, only two new branches are generated on the next branching level. But if the second job was appended to route one, four new branches are created next. This effect leads to unbalances in the search tree. Also the number of branches on each level is interesting. There are four branches with one planned job, 20 branches with two planned jobs and 72 branches with three and four planned jobs, respectively. Noticeable is that there is only one possibility to append the fourth job – always to route two since all routes must be used. Thus, there are many branching nodes created in the first branching levels and a high number of branching nodes is generated to create all 72 leaves of the search tree.

An advantage of the strategy Append consists in the fact that the execution times of the already planned jobs of a partial solution are fixed. Therefore, the costs can easily be updated if a new job is appended to a route without recalculating all execution times. Additionally, the effort of computing lower bounds decreases with the branching depth.

A difficulty of this branching strategy is the structure of the resulting search tree. Let the level of a branching step be the distance in the search tree from the root, which is branching on the empty partial solution. This distance is equal to the number of planned jobs, since branching on a partial solution leads to partial solutions with one more job scheduled. In the first branching levels, much more branches are created than later: in level 0, we have to create n branches, one for each job. In level κ, at most $2(n-\kappa)$ branches are produced, because $n-\kappa$ jobs are unplanned. And in the last level before reaching a leaf of the search tree, only one branch is created. Furthermore, the search tree is very unbalanced because of filling the routes one after the other, which could be seen in the small example shown in Fig. 1. Both together leads to the effect that by eliminating a partial solution only a small part of the set of feasible solution might be removed and, perhaps, more partial solutions have to be analyzed than in a better structured search tree.

3.2. Include a job into a partial solution

With the second branching strategy, we generate new partial solutions by including a certain job inside the routes. This strategy is called *Include*. More precisely, based on a partial solution, we select one not yet planned job and create new partial solutions by including this job at the first, the second, till to the last position of each route. The reason for that approach is that this branching strategy leads to a balanced search tree.

In detail, the branching strategy Include works as follows: let $\widetilde{S} \in S$ be a partial solution. To create a set of new partial solutions, we select one unplanned job that will be included into each position of the current routes. Let $i \in \widetilde{N}$ be this job. Then, we generate the following new partial solutions:

$$S_{j,l} := (\widetilde{N} \setminus \{i\}, (\widetilde{N}_l \cup \{i\}, (\pi_1^l, \ldots, \pi_{j-1}^l, i, \pi_j^l, \ldots, \pi_{|\widetilde{N}_l|}^l)), (\widetilde{N}_k, \Pi^k(\widetilde{N}_k))_{k \in M \setminus \{l\}})$$
$$j \in \{1, 2, ..., |\widetilde{N}_l|, |\widetilde{N}_l| + 1\}, \; l \in M.$$

With it, we create $|N| - |\widetilde{N}| + m$ new partial solutions, because the job i can be included after each already planned job, which leads to $|N| - |\widetilde{N}|$ new partial solutions; and as first job of each route which leads to m further partial solutions.

We illustrate the branching strategy Include in Fig. 2 with four jobs and two routes. In branching level 0, two branches are generated: Including the first job to route one and to route two. For each of both partial solutions, three new branches are generated including the second job on each possible position: before the first job, after the first job and in the other route. Independent of the structure of the partial solution, four new branches are generated in the next branching step, which can be seen comparing branching on a partial solution with two jobs in route one with branching on a partial solution with one job in route one and one in route two. Only on the last branching level, where the fourth job is included, partial solutions with an empty route lead not to the expected five schedules, but to one schedule, because the last job has to be included to the empty route. Compared to the branching strategy Append, we need only 33 branching nodes to get all 72 leafs (instead of 97 branching nodes with strategy Append) because the number of partial solutions on each branching level is smaller: two partial solutions with one jobs, six with two jobs, 24 partial solutions with three jobs and finally 72 feasible solutions. The search tree is very balanced and on deeper branching levels, more branches are generated. Thus, eliminating a partial solution eliminates a larger part of the set of feasible solutions.

However, there are also some disadvantages caused by the fact that the partial solution is changed by including jobs in later branching steps inside a partial route. On the one hand, this leads to an increased calculation effort because some execution times have to be recalculated to obtain the costs of a partial solution. On the other hand, the calculation of lower bounds is not as simple as in the strategy Append, because the execution times and the travel costs of already included jobs can

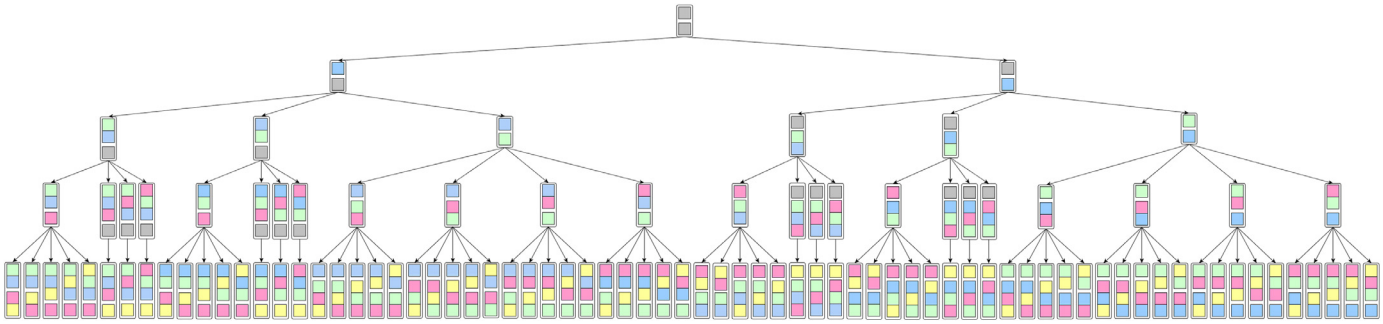

Fig. 2. Example for the search tree for branching strategy Include .

change by including another job into the route. Hence, the costs so far are not fixed. Because of that, all jobs have to be taken into account when calculating lower bounds.

In our comparative tests of Section 5, we analyze whether better lower bounds or a more balanced search tree leads to faster computation times.

4. Lower bounds

In this section, we present a collection of lower bounds for the VRP-CC. A first idea to obtain a lower bound for VRP-CC could be to use the linear programming relaxation of the linearized model (MILP) presented in Section 2. This means the binary variables are replaced by continuous ones in [0,1] and the integer time variables are represented by real numbers. The customer cost part of an optimal solution of this relaxation is called LB^c_{LP}, the travel cost part is called LB^d_{LP}. However, as we will see, lower bounds are needed that are more appropriate.

In the first subsection, we propose some lower bounds for the customer cost part and compare their quality and computation time. For the most suitable lower bound on customer costs, we present needed modifications to apply it in both branching strategies. In Section 4.2, we outline some lower bounds for the travel cost part based on lower bounds for the TSP. These bounds are compared and modifications needed for their application in the branching process are presented.

4.1. Lower bounds for customer costs

To calculate lower bounds of the customer costs, only the subset of jobs with non-zero customer costs has to be considered which is given by $N_c := \{i \in N | \, c_i > 0\}$ with $n_c = |N_c|$. We set $T = \{1, 2, \dots\}$ to be the set of days for job execution taking into account that $t_0 = (0, 480)$.

4.1.1. Trivial lower bound for customer costs

The idea of this bound is very simple: we calculate how many jobs can be done at most per day and select the most expensive jobs for the first day, the next most expensive jobs for the second day, and so on. At first, we search for the value η that gives an upper bound for the number of jobs executable per day. Therefore, we determine the minimal time needed per job for execution and to travel to the next one, which is $a_i + r_i$, with $r_i := \min\{r_{ij} | j \in N_c, j \neq i\}$ is the minimal time needed to travel to the next job. Each day, we have altogether mu minutes for job execution. For traveling to the jobs executed on the next day, we have to reserve the time

$$R_m := \max\{\sum_{i \in I} r_i | \, I \subseteq N_c, |I| = m\}. \tag{25}$$

All in all, we search for the value of η that meets the conditions

$$\min_{\{I \subseteq N_c | \, |I| = \eta\}} \sum_{i \in I} (a_i + r_i) \leq mu + R_m \tag{26}$$

and

$$mu + R_m < \min_{\{I \subseteq N_c | \, |I| = \eta + 1\}} \sum_{i \in I} (a_i + r_i) \tag{27}$$

The solution can be found easily by sorting the jobs appropriately. Then, to obtain a lower bound, we solve the problem described by (28)–(30).

$$LB^c_s := \min \sum_{i \in N_c} c_i \tau_i \tag{28}$$

$$\text{s.t. } \tau_i \in T, \qquad i \in N_c \tag{29}$$

$$\left|\left\{i \in N_c | \, \tau_i = t\right\}\right| \leq \eta \qquad t \in T \tag{30}$$

To solve this problem fast, we sort the jobs by decreasing customer cost coefficients. Then, the first η jobs of the order are allocated to day 1, the next η jobs are allocated to day 2, and so on. Because of the needed sorting, the computation time is $\mathcal{O}(n_c \log(n_c))$.

4.1.2. Lower bound from minimum weighted sum bin packing problems

The first lower bound LB^c_s is very simple. Instead of using a fixed number of jobs per day, one could also be inspired by bin packing problems to allocate jobs to days. To obtain a lower bound for the customer costs, we present some problems similar to the minimum weighted sum bin packing problem (Ebstein and Levin, 2007). In contrast to the common bin packing problem, where a given set of items with different sizes has to be packed into a minimal number of bins, see Scheithauer (2018) and the references therein, the items are afflicted with bin-dependent costs and we search for a packing of minimal total costs. A similar problem is the generalized bin packing problem with bin-dependent item profits introduced by Baldi et al. (2019), where the items are afflicted with a bin-dependent profit and the usage of bins leads to costs.

The first problem is an allocation of jobs to days, see (BP1).

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

where

- $r_i := \min\{r_{ij} | j \in N_c, j \neq i\}$ is again the minimal travel time from job $i \in N_c$ to any other job in N_c,
- R_m is the time reserved for the trips outside the working shift, see (25), and
- $x_{it} = 1$ if and only if job i is allocated to day $t \in T$.

In (BP1), each bin corresponds to one day. The bin size is set to the time available for maintenance and travel per day $mu + R_m$. The time needed for a job i to execute it and travel to the next one is at least $a_i + r_i$ which is the size of the job in the bin packing problem. In the objective function, the total customer cost of job i is calculated based on the day t with $x_{it} = 1$.

To improve the bound, we define a second bin packing problem (BP2) where for each machine an own bin per day is defined.

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

Problem (BP2) is a bin packing problem with $m\cdot|T|$ bins, one per day and machine. Each bin associated to route $k\in M$ has a capacity of $u + r^k_{\max}$ where $r^k_{\max}$ is the kth largest value of the set $\{r_i | i\in N_c\}$. Thus, the bin size is the sum of the working shift length and the time reserved for the trip to the job executed on the next day. Again, the size of the jobs in the bin packing problem is $a_i + r_i$ which is the time for execution and the minimal time to travel to the next job. The objective function is the sum of the customer costs of all jobs considering the corresponding day of the allocation.

The fact that traveling from job to job is allowed outside the working shift, is modeled in (BP1) and (BP2) by increasing the bin capacity by the maximal trip length. The following bin packing problem (BP3) models this by considering working duration a_i and minimal possible travel time r_i as separated items.

$$\begin{aligned}(\text{BP3})\quad \mathrm{LB}^c_{\mathrm{BP3}} := \min\ & \textstyle\sum_{i\in N_c}\sum_{b\in B} c_i\, t_b\, x_{ib} \\ \text{s.t.}\ & \textstyle\sum_{i\in N_c} x_{ib}\, a_i + \sum_{i\in N_c} y_{ib}\, r_i \le u, && b\in B \\ & \textstyle\sum_{i\in N_c} x_{ib} \le \sum_{i\in N_c} y_{ib} + 1, && b\in B \\ & \textstyle\sum_{b\in B} x_{ib} = 1, && i\in N_c \\ & \textstyle\sum_{b\in B} y_{ib} \le 1, && i\in N_c \\ & y_{ib} - x_{ib} \le 0, && i\in N_c, b\in B \\ & x_{ib}\in\{0,1\},\ y_{ib}\in\{0,1\}, && i\in N_c, b\in B\end{aligned}$$

where

- B denotes the set of bins (one per day and machine),
- t_b is the day of bin b with $t_b\in T$ and $|\{b\in B | t_b = t\}| = m,\ t\in T$,
- $r_i = \min\{r_{ij} | j\in N_c, j\neq i\}$ is the minimal travel time from job $i\in N_c$ to any other job in N_c,
- $x_{ib} = 1$ if and only if the execution of job i is allocated to bin b, and
- $y_{ib} = 1$ if and only if a travel from job i to another job is allocated to bin b.

In (BP3), for each job we have two items that could be packed into a bin: the working item a_i and the travel item r_i. We have to ensure, that the sum of the allocated items is not larger than the working shift. In each bin, we can allocate one working item more than travel items. Furthermore, each working item has to be allocated, but not each travel item. Finally, we claim that the travel items have to be packed in the same bin as the corresponding working item ($y_{ib}\le x_{ib}$).

Finally, note that $\mathrm{LB}^c_{\mathrm{s}}$, $\mathrm{LB}^c_{\mathrm{BP1}}$, $\mathrm{LB}^c_{\mathrm{BP2}}$ and $\mathrm{LB}^c_{\mathrm{BP3}}$ are indeed lower bounds. Moreover the following result is satisfied.

Theorem 1. *For any instance of VRP-CC, we have* $\mathrm{LB}^c_{\mathrm{BP3}} \ge \mathrm{LB}^c_{\mathrm{BP2}} \ge \mathrm{LB}^c_{\mathrm{BP1}} \ge \mathrm{LB}^c_{\mathrm{s}}$.

Proof. Theorem 1 First, we show that $\mathrm{LB}^c_{\mathrm{BP3}}$ is not worse than $\mathrm{LB}^c_{\mathrm{BP2}}$. Let $\{x_{ib}\}$ and $\{y_{ib}\}$, $i\in N_c, b\in B$, be a feasible solution of (BP3). We have to show that this solution can be transformed into a solution of same costs of (BP2). At first, for each machine we have at most one trip to a job executed on the next day. For all $t\in T$, there exists a mapping from bin b to one route j_b in (BP2) so that

$$\sum_{\substack{i\in N_c \\ x_{ib}=1,\, y_{ib}=0}} r_i \le r^{j_b}_{\max} \quad \forall b\in B \quad \text{with} \quad t_b = t$$

$$\text{and} \bigcup_{\{b\in B | t_b = t\}} \{j_b\} = M. \tag{31}$$

Then, we have

$$\sum_{i\in N_c} x_{ib}\, a_i + \sum_{i\in N_c} y_{ib}\, r_i \le \sum_{i\in N_c} x_{ib}\,(a_i + r_i) \tag{32}$$

$$\le \sum_{i\in N_c} x_{ib}\, a_i + \sum_{i\in N_c} y_{ib}\, r_i + r^{j_b}_{\max} \tag{33}$$

$$\le u + r^{j_b}_{\max} \quad \forall b\in B \tag{34}$$

Inequality (32) shows the transition from (BP3) to (BP2). The formula on the left side equals to the size of the items of a bin in (BP3), which is smaller than the size of the items of the bin in (BP2) because this has an additional summand r_i. This additional summand is not larger than $r^{j_b}_{\max}$ and with it, inequality (33) is true. The fact that in (BP3) the bin size is limited to u, leads to inequality (34). Thus, the variables x_{ib} of a feasible solution of (BP3) lead to a feasible solution of (BP2). The contrary is not necessarily true, because in (BP2) we have

$$\sum_{i\in N_c} x_{ib}\,(a_i + r_i) \le u + r^k_{\max}.$$

To transform a feasible solution of (BP2) into a feasible solution of (BP3), we have to ensure that $\sum_{i\in N_c} x_{ib}\,(a_i + r_i) - \max_{i\in N_c} x_{ib} r_i \le u$, because in (BP3) one travel item is removed and the bin size is decreased by $r^k_{\max}$. This is not guaranteed for each feasible solution, because normally $\max_{i\in N_c} x_{ib} r_i < r^k_{\max}$. Thus we get $\mathrm{LB}^c_{\mathrm{BP3}} \ge \mathrm{LB}^c_{\mathrm{BP2}}$.

Second, we proof that $\mathrm{LB}^c_{\mathrm{BP2}} \ge \mathrm{LB}^c_{\mathrm{BP1}}$. Because of

$$\sum_{k\in M}\sum_{i\in N_c} (a_i + r_i) x_{itk} \le \sum_{k\in M} \left(u + r^k_{\max}\right) = mu + R_m \quad \forall t\in T,$$

each feasible solution of (BP2) is also feasible for (BP1). However, not each feasible solution of (BP1) is feasible for (BP2). Let $\hat{N}(t) := \{i\in N_c | x_{it} = 1\}$ denote the set of jobs executed at day $t\in T$. To obtain a feasible solution of (BP2), we have to find a partition of $\hat{N}(t)$ into m subsets $\hat{N}_k(t)$, $k\in M$, with $\sum_{i\in\hat{N}_k(t)}(a_i + r_i) \le u + r^k_{\max}$ which is not necessarily possible. Thus, we have $\mathrm{LB}^c_{\mathrm{BP2}} \ge \mathrm{LB}^c_{\mathrm{BP1}}$.

Finally, it remains to show, that $\mathrm{LB}^c_{\mathrm{BP1}} \ge \mathrm{LB}^c_{\mathrm{s}}$. Because of $\sum_{i\in N_c} x_{it}(a_i + r_i) \le u + R_m$ and the definition of η we obtain $\sum_{i\in N_c} x_{it} \le \eta$ for each $t\in T$. Thus each feasible solution of (BP1) satisfies the constraints used to calculate $\mathrm{LB}^c_{\mathrm{s}}$ and we get $\mathrm{LB}^c_{\mathrm{BP1}} \ge \mathrm{LB}^c_{\mathrm{s}}$. □

4.1.3. Numerical comparison of lower bounds for the customer cost part

For our numerical comparisons, we first define a set of 100 small random benchmark instances with 10 up to 19 jobs, 6% to 90% of the jobs are afflicted with a non-zero customer cost coefficient. In these instances, the percentage of customer costs in the optimal total costs is between 2% and 87%. The increase in travel costs caused by customer costs, which is calculated by comparing the travel cost part of an optimal solution with the optimal costs minimizing only the travel costs g^d, is in average 23%. Based on this set, we generate a second set of instances dominated by customer costs by afflicting all jobs with a high customer cost coefficient. In detail, we increase the customer cost coefficient of each job by 10 and multiply the resulting value with 100. With it, about 99% of the optimal total costs are customer costs. For most of the instances dominated by customer costs, the optimal solution value is known but there are some instances only with a best-known value. Further, we define a set of 25 large instances with 75 up to 100 jobs, where the optimal solution value could not be determined.

The bounds are compared in quality measured by $q := \frac{\text{bound value}}{g^c(S^*)}$ in percent, where $g^c(S^*)$ is the customer cost part belonging to an optimal or a best-known solution S^* obtained by minimizing the total costs $g(S)$. In the large instances, where the optimal value is not known and the best-known solution can be worse, we compare the average of the distances to the best bound $\Delta := 1 - \frac{\text{bound value}}{\max \text{obtained bound value}}$ in percent instead of the quality. Further, we compare the computational effort since for the branch-and-bound method, lower bounds are needed that can be calculated fast. We measure the computational effort as the average computation time. The lower bounds based on bin packing problems are computed with CPLEX. If no optimal solution was found within the time limit of 60 s, we use the minimal possible objective function value of the remaining unexplored nodes.

In Table 1, we compare different customer cost bounds: the bound of the LP-relaxation $\mathrm{LB}^c_{\mathrm{LP}}$, the trivial bound $\mathrm{LB}^c_{\mathrm{s}}$, and the bounds based

Table 1
Comparison of different customer cost bounds.

	LB^c_{LP}	LB^c_s	LB^c_{BP1}	LB^c_{BP2}	LB^c_{BP3}
100 instances with customer costs for a part of jobs					
Avg q	86%	94%	97%	97%	97%
$q \geq 95\%$	48%	70%	86%	87%	88%
$q \geq 90\%$	59%	78%	94%	94%	94%
Avg time (ms)	28	< 1	31	56	125
100 instances dominated by customer costs					
Avg q	75%	92%	99%	99%	99%
$q \geq 95\%$	25%	56%	96%	98%	99%
$q \geq 90\%$	30%	69%	100%	100%	100%
Avg time (ms)	35	< 1	78	178	633
25 large instances					
Avg Δ	32%	9%	0.7%	0.1%	0.4%
Avg time (ms)	1823	< 1	1072	26051	45773
Instances with unsolved bounds	0	0	0	9	17

on bin packing problems LB^c_{BP1}, LB^c_{BP2}, and LB^c_{BP3}. For the two sets of small instances, we provide the average bound quality q, how often the obtained bound quality q exceeds 95% and 90%, and the average computation time in milliseconds. For the 25 large instances, we show the average of the distances Δ in percent, the average computation time in milliseconds, and how many instances were not solved within 60 s.

The LP-relaxation leads to the worst bounds. The analysis of the results has shown that in the LP-relaxation all jobs are allocated to day 1. Because of that, this bound is not suitable to evaluate the customer costs in our branch-and-bound approaches. We believe that this worse estimation of the customer costs is a key problem by solving the VRP-CC with a commercial solver. The trivial bound LB^c_s performs well. In both kinds of small instances, the quality is in average about 94% and 92%, respectively. Analyzing the bounds obtained for the large instances, we see that LB^c_s is in average 9% worse than the best-obtained bound. The computation time is very small, less than one millisecond is necessary to obtain the bound. The bounds based on bin packing problems have all a high quality, but there are major differences in the computation time. The average quality of all three lower bounds is about 97% in the small instances and about 99% in the small instances with high customer costs. In only some instances, LB^c_{BP2} or LB^c_{BP3} are better than LB^c_{BP1}, which can be seen comparing how often the quality exceeds 95%. The computation time to calculate the bounds differs significantly. LB^c_{BP1} is calculated with the smallest computation effort, much more time is needed for LB^c_{BP2} and the most time is required for LB^c_{BP3}. Especially for the large instances, the increase of computation time is substantial. The bound LB^c_{BP1} could be calculated within the time limit of 60 s for each of the 25 large instances. In 9 of these instances, it was not possible to optimally solve (BP2) within the time limit, and the calculation of bound LB^c_{BP3} requires in 17 instances more than 60 s. Because of these unsolved instances, the average Δ-value of LB^c_{BP3} is not zero, in difference to the theoretical statement.

Because the bin packing problems are NP-hard, we also tested their LP relaxations. It turned out that the LP relaxation bounds are similar to the trivial bounds LB^c_s. Moreover, since the LP relaxation of (BP1) and (BP2) can be calculated fast with a suitable ordering of the jobs, we tested their application in the branch-and-bound methods. The performance is comparable to that of LB^c_s. Altogether, we decide to use only the bound LB^c_s in our branch-and-bound methods because its good quality and very short computation time.

4.1.4. Application to branch-and-bound strategy Append

By applying bound LB^c_s to the branch-and-bound strategy Append, we consider that no more jobs will be appended to the routes already closed and that the route l, where the last job was appended, has as start time the finish time of the last appended job. The routes $l+1, \dots, m$ are empty. Because of that, we adapt the number of jobs executable per day η. For each day $t \in T$, we calculate the number of executable jobs considering the restrictions from the partial solution based on an upper bound of the number of jobs executable by one machine at one day.

4.1.5. Application to branch-and-bound strategy Include

To calculate LB^c_s for a partial solution $\widetilde{S} = (\widetilde{N}, (\widetilde{N}_k, \Pi^k(\widetilde{N}_k))_{k\in M})$, we have to take into account all jobs with customer costs. The jobs, that are allocated to a route in the partial solution, have a day for their execution calculated based on the partial solution, which is t^d_i, $i \in N \backslash \widetilde{N}$. Using the branching strategy Include, we calculate the lower bound $LB^{c_{ep,include}}$ as follows:

$$LB^{c_{ep,include}} := \min \sum_{i\in N_c} c_i \tau_i \tag{35}$$

$$\text{s.t. } \tau_i \in T, \qquad i \in N_c \tag{36}$$

$$\left|\left\{i \in N_c \,|\, \tau_i = t\right\}\right| \leq \eta, \qquad t \in T \tag{37}$$

$$t^d_i \leq \tau_i, \qquad i \in N_c \setminus \widetilde{N} \tag{38}$$

The constraint (38) ensures that each already allocated job with non-zero customer costs gets an execution day τ_i in the relaxation (35)–(38), which is not earlier than the execution day of the current partial solution. This optimization problem can be solved by suitably ordering the jobs.

4.2. Lower bounds for travel costs

Lower bounds for travel costs are well known from research concerning the TSP, see Reinelt (1994). To apply TSP bounds in our branch-and-bound approaches, theVRP-CC is transformed into a TSP by appending the m routes to one big route and defining an appropriate distance matrix $D = (d_{ij})_{i,j\in N_a}$, see Heinicke et al. (2015). To this end, the travel costs between the depots z_k and s_{k+1}, $k = 1, \dots, m-1$, as well as z_m and s_1 are set to zero, and all the other travel costs between depots are set to infinity.

4.2.1. The two-neighbor bound

A simple lower bound for the travel costs is the so-called 2-neighbor bound (Reinelt, 1994):

$$LB^{d_{2N}} := \frac{1}{2}\left(\sum_{i\in N} \min\{d_{ji} + d_{ik} \,|\, j \in N \cup N_s, k \in N \cup N_z, j \neq i, k \neq i, k \neq j\} + \sum_{i\in N_s} \min\{d_{ij} \,|\, j \in N\} + \sum_{i\in N_z} \min\{d_{ji} \,|\, j \in N\}\right)$$

For each job $i \in N$, the costs to its two nearest neighbors are summed up. We regard that predecessor and successor have to be different jobs, that the predecessor cannot be an end depot and that the successor cannot be a start depot. Additionally, the minimal travel costs to leave the start depots and to reach the end depots are added. This sum is divided by two, because it contains the twice of the needed number of summands.

4.2.2. Lower bound from assignment problem

Another lower bound can be derived from the assignment problem:

$$\begin{aligned} (\text{AP})LB^{d_{AP}} := \quad & \min \textstyle\sum_{i\in N_a} \sum_{j\in N_a} d_{ij}\, x_{ij} & \\ & \text{s.t. } \textstyle\sum_{j\in N_a\backslash\{i\}} x_{ij} = 1 & i \in N_a \\ & \textstyle\sum_{j\in N_a\backslash\{i\}} x_{ji} = 1 & i \in N_a \\ & x_{ij} \in \{0,1\} & i,j \in N_a \end{aligned}$$

The result of (AP) is in general a set of cycles. To exclude cycles of length 1, we set $d_{ii} = \infty$ for each $i \in N_a$. The assignment problem leads to a lower bound for the TSP, because it is derived from the TSP by removing subtour elimination constraints. It can be solved in $\mathcal{O}(n^3)$ by means of a good implementation of the Hungarian method, as shown in Munkres (1957).

Christofides (1972) provides an improvement of the assignment bound. Normally, the solution of (AP) contains small cycles. He developed an iterative method to add the costs necessary to connect the cycles. The algorithm starts with the distance matrix. For this matrix, the

Table 2
Comparison of different travel cost bounds.

	LB^{d}_{LP}	$LB^{d_{2N}}$	$LB^{d_{AP}}$	$LB^{d_{APC}}$	$LB^{d_{MST}}$			
Iterations					1	10	100	1000
100 instances with customer costs for a part of jobs								
Avg q	63%	47%	62%	76%	62%	66%	76%	83%
$q \geq 95\%$	1%	0%	1%	8%	0%	0%	7%	26%
$q \geq 90\%$	3%	0%	3%	22%	0%	1%	22%	39%
Avg time (ms)	32	< 1	< 1	< 1	< 1	< 1	< 1	1
100 instances without customer costs								
Avg q	74%	56%	74%	90%	74%	79%	90%	99%
$q \geq 95\%$	3%	0%	3%	24%	0%	1%	45%	91%
$q \geq 90\%$	14%	0%	14%	64%	2%	11%	65%	98%
Avg time (ms)	30	< 1	< 1	< 1	< 1	< 1	< 1	1
25 large instances								
Avg Δ	21%	29%	22%	6%	17%	12%	4%	0%
Avg time (ms)	1898	< 1	10	11	< 1	< 1	11	65

solution of (AP) is calculated using the Hungarian method. The resulting reduced distance 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 distance matrix is computed from the reduced matrix by the minimal distance between the cycles considering the triangle inequality. Based on this new 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 $LB^{d_{APC}}$ is returned.

4.2.3. Minimal spanning tree based lower bound

In the paper of Held and Karp (1971), a lower bound for the travel costs of TSP is presented that uses minimal spanning trees, which is the base for the bound $LB^{d_{MST}}$. Therefore, we need a symmetric distance matrix. To ensure that, we set $\tilde{d}_{ij} = \min\{d_{ij}, d_{ji}\}$.

Let C^* be the minimal costs of a TSP and W^* be the weight of a minimum-weight 1-tree, which is a minimal spanning tree with vertex set $\{2, 3, \dots, n_a\}$ and two distinct edges of minimal weight incident to vertex 1. Then, W^* is a lower bound for C^*, because a tour of the TSP is a 1-tree in which each vertex has degree 2 and if the minimum-weight 1-tree is a tour, then it is a tour of minimum weight. To calculate a minimum-weight 1-tree, the algorithm of Prim/Dijkstra or the one of Kruskal can be applied, see Prim (1957) or Kruskal (1956). Then, using a suitable implementation and data structure, the complexity of the algorithms can be stated as $\mathcal{O}(n^2)$ and $\mathcal{O}(n^2 \log n)$, respectively, because we have $\mathcal{O}(n^2)$ edges.

To improve the simple bound of the minimum-weight 1-tree, Held and Karp proposed an iterative approach called subgradient ascent method where vertices with degree not equal to two are penalized. This is done by setting the edge weight to $w_{ij} = \tilde{d}_{ij} + \pi_i + \pi_j$ where π_i and π_j are the penalties of vertices i and j. Let $W^*(\pi)$ be the minimal weight of an 1-tree with penalty vector π, then we have $C^* \geq W^*(\pi) - 2\sum_{i \in N_a} \pi_i$. Thus, the best lower bound is given by $\max_\pi W^*(\pi) - 2\sum_{i \in N_a} \pi_i$. The penalty vector π is changed iteratively. Let d_i be the degree of vertex i in the last calculated 1-tree, then the penalty for the next iteration is set to $\pi_i^{m+1} = \pi_i^m + t d_i$ 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.

In our implementation of this lower bound, we use a constant value $t = 0.1$. In the next subsection, we test the bound quality for different numbers of iterations. Within the test, we prematurely stop the iterative process if no improvement was obtained in the last ten percent of the maximal allowed number of iterations.

4.2.4. Numerical comparison of lower bounds

To compare the bounds, we use again three groups of benchmark instances: firstly, the 100 small instances with 10 up to 19 jobs with customer costs for a part of the jobs from Section 4.1.3; secondly, 100 small instances without customer costs, which are generated from the previous set of instances by set $c_i = 0$ for each job $i \in N$; and thirdly, the 25 large instances with 75 up to 100 jobs from Section 4.1.3.

As before, we compare the quality $q := \frac{\text{bound value}}{g^d(S^*)}$ in percent, where $g^d(S^*)$ is the travel cost value of an optimal solution, or the distance to the best bound $\Delta := 1 - \frac{\text{bound value}}{\text{max obtained bound value}}$; and the computational effort as the average computation time. The maximal time to calculate a lower bound is restricted to 60 s, but all bounds were calculated without exceeding this time limit.

In Table 2, the different travel cost bounds are compared: the travel costs of the LP-relaxation of (MILP) LB^{d}_{LP}, the 2-neighbor bound $LB^{d_{2N}}$, the bound $LB^{d_{AP}}$ of the assignment problem and its improvement by Christofides $LB^{d_{APC}}$, and the bounds based on minimal spanning trees $LB^{d_{MST}}$ with 1, 10, 100, and 1000 iterations.

The LP-relaxation leads to an acceptable lower bound for the travel cost part of the solution. In the small instances with non-dominating cost parts, where detours for jobs with high customer costs are common, the bound quality is in average 63%. In the small instances dominated by travel costs, the average bound quality is 74%. A quality of more than 90% is rarely reached. The computation effort is about 30 ms, which is comparably high. In the large instances, the bounds of the LP-relaxation are in average 22% worse than the best obtained bound and the computation time for LB^{d}_{LP} is much higher than for the other bounds. The 2-neighbor bound $LB^{d_{2N}}$ is fast to calculate, but the quality is very low compared to the other bounds. In small instances, the quality is in average by 47% and 56%, respectively. The average distance to the best obtained bound in the large instances is 29%. The assignment bound $LB^{d_{AP}}$ leads to better bounds than $LB^{d_{2N}}$ and similar bounds as LB^{d}_{LP}. The computation time is very small, in average less than one millisecond in the small instances and in average 10 ms in the large instances. The approach of Christofides improves the assignment bound by more than 20% with a similar computation time. In the instances dominated by travel costs, this bound leads in nearly two-third of the instances to results with a quality over 90% and in nearly a quarter of the instances to results with a quality over 95%. The bounds calculated with minimum spanning trees are comparable to the assignment bounds. The bound quality is high and the computation time is low. As expected, more iterations lead to significant better lower bounds. If only one 1-tree is calculated, the bound quality is as good as $LB^{d_{AP}}$ but better than the 2-neighbor bound. With 100 iterations, the bound quality is similar to the quality of $LB^{d_{APC}}$; and with 1000 iterations, the quality is further improved. In the instances dominated by the travel costs, the bound $LB^{d_{MST}}$ with 1000 iterations is very close to the optimal costs. In the large instances, the bounds based on minimum spanning trees are in average a bit better than the assignment bounds. In 24 of the 25 large instances, $LB^{d_{MST}}$ with 1000 iterations leads to the best lower bound.

Table 3
Comparison of the most suitable travel cost bounds for both branch-and-bound methods.

	$LB^{d_{2N}}$	$LB^{d_{AP}}$	$LB^{d_{APC}}$	$LB^{d_{MST}}$		
iterations				10	100	1000
Append						
Avg time (min)	4.2	4.5	1.8	2.1	2.3	2.8
Optimally solved	96%	95%	98%	98%	98%	98%
Analyzed partial solutions	5.0×10^8	2.4×10^8	7.4×10^7	9×10^7	3.8×10^7	1.5×10^7
Fastest solved	43%	21%	44%	47%	6%	0%
Include						
Avg time (min)	7.2	3.8	3.3	4.6	7.1	8.5
Optimally solved	94%	97%	98%	96%	92%	89%
Analyzed partial solutions	1.5×10^9	2.9×10^8	2.4×10^8	4.1×10^8	1.2×10^8	6.5×10^7
Fastest solved	45%	52%	72%	33%	11%	3%

Because of the similar results, we decided to test all bounds besides the LP-relaxation in the branch-and-bound approaches. We expect a better performance of the assignment bounds because they consider the asymmetries in distance matrices that occur in the solution process.

4.2.5. Application to branch-and-bound strategy Append

In the following, we show how the distance matrix is defined for a partial solution constructed in the branch-and-bound algorithm with strategy Append. In this partial solution, j_l denotes the last job appended to route l. Then we have to consider

- the jobs which are not allocated in the partial solution,
- the last appended job j_l that works as start depot of the route l,
- the start depots of the routes $l+1, \dots, m$, and
- the end depots of the routes $l, l+1, \dots, m$.

With it, the size of the travel cost matrix decreases with increasing depth in the branch-and-bound tree. As mentioned in Section 4.2, we set the travel costs between the depots z_k and s_{k+1}, $k = l, l+1, \dots, m-1$, as well as z_m and j_l 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 could not be considered in the travel cost bounds based on minimum spanning trees. To avoid cycles of length 1 in the solution of the assignment problem, the cost value d_{ii} of each job i is also set to infinity.

4.2.6. Application to branch-and-bound strategy Include

By defining the travel cost matrix for a partial solution in the branch-and-bound tree with strategy Include, we have to take into account all jobs. Thus, the size of the travel cost matrix is always $n + 2m$. Let $N^p := \bigcup_{k \in M} \tilde{N}_k$ be the set of the already planned jobs. Between these jobs, only the travel costs of consecutive jobs have to be considered. Because currently not planned jobs could be allocated at each position in the current routes, for these jobs we have to consider the travel costs to all other jobs. The costs d_{ii} are again set to infinity. For the depots, we have to apply the same rules as before to transform the m routes into one large route. In summary, we use the following travel costs:

$$\tilde{d}_{ij} = \begin{cases} \infty, & \text{if } i = j \in N_a \\ d_{ij}, & \text{if } i \in N^p \cup N_s,\ j \in N^p \cup N_z \text{ and } i \text{ is the predecessor of } j \\ \infty, & \text{if } i \in N^p \cup N_s,\ j \in N^p \cup N_z \text{ and } i \text{ is not the predecessor of } j \\ d_{ij}, & \text{if } i \in N^p \text{ or } i \in N_s \text{ and } j \in \tilde{N} \\ d_{ij}, & \text{if } i \in \tilde{N} \text{ and } j \in N^p \cup N_z \\ d_{ij}, & \text{if } i, j \in \tilde{N},\ i \neq j \\ \infty, & \text{if } i \in N_s,\ j \in N_z \\ 0, & \text{if } i = z_k,\ j = s_{k+1} \pmod m \text{ and } k \in M \\ \infty & \text{if } i = z_k,\ j = s_l,\ k, l \in M \text{ and } l \neq k+1 \pmod m \end{cases}$$

Note that the resulting travel cost matrix contains more asymmetries than a cost matrix obtained with the strategy Append. Because of that, we expect that the travel cost bounds based on the assignment problem will be better than the bounds based on the minimum 1-trees, and that the difference in the performance of both kinds of bounds is more pronounced in this branch-and-bound strategy.

4.2.7. Comparison of performance in branch-and-bound application

For the branch-and-bound approaches, we need fast and good lower bounds. The better the lower bound, the more branches can be discarded which reduces the calculation effort. Contrariwise, a high computation time of the lower bound can increase the total calculation effort, since many lower bounds have to be calculated within the branch-and-bound method. For the comparison, we used the same 100 small benchmark instances with non-zero customer cost coefficient for a part of the jobs as before and limited the computation time to one hour.

In Table 3, we show the results of our experiments. We compare the average computation time, the percentage of optimally solved instances, the average number of analyzed partial solutions, and how often the usage of a lower bound leads to the best computation time with a tolerance of 100 ms. The results show that the simple lower bound for travel costs is not suitable for both branching strategies. Because of the worse bound qualities, less branches are discarded and more partial solutions have to be analyzed.

Analyzing the performance of $LB^{d_{AP}}$ and $LB^{d_{APC}}$, we see that in the strategy Append, the assignment bound leads to similar computation times as the simple bound even though only a half of partial solutions is analyzed. Thus, the bound quality is in the application better, but the higher calculation effort results in a similar performance of the algorithm. In the strategy Include, the assignment bound leads to a reduction of the average computation time by one-half compared to the simple bound because of a stronger reduction of the number of analyzed partial solutions. The reason for the better quality could be the higher number of asymmetric entries in the cost matrices generated using strategy Include which are taken into account in the assignment bound. In both strategies, the improvement of the assignment bound by Christofides leads to a better performance of the branch-and-bound, and the effect is much higher in the strategy Append. There, the number of analyzed partial solutions is significantly reduced which leads to a strong reduction of the computation time.

In contrast, more iterations in the calculations of $LB^{d_{MST}}$ do not improve the computation time even though the number of analyzed partial solutions is significantly reduced. In the strategy Append, a small increase in the computation time from 2.1 min to 2.8 min can be seen even through the number of analyzed partial solutions is decreased to its one-sixth. There, the bounds $LB^{d_{APC}}$ and $LB^{d_{MST}}$ with 10 iterations are all about the same. With $LB^{d_{APC}}$, the average computation time is a bit smaller, but using $LB^{d_{MST}}$ with 10 iterations shows a larger number of fastest solved instances. In the strategy Include, where the cost matrices is of constant size and contains more asymmetric entries than in the strategy Append, $LB^{d_{MST}}$ leads to a worse performance than the assignment bounds. The increase in the number of iteration results in a similar reduction in the amount of analyzed partial solutions as with strategy Append, but to a strong increase in computation time from 4.6 min to 8.5 min. A reason could be the constant size of the distance matrix during the branch-and-bound process with strategy Include.

As consequence of these results, we only use LB^{d}_{APC} in both branch-and-bound approaches for our competitive tests with standard software.

5. The branch-and-bound method

For both branch-and-bound strategies, we use a depth-first search to avoid the storage of many partial solutions, which is necessary by using best-first search. We parallelize the solution process by creating all partial solutions up to a given depth. The partial solutions with the best lower bounds are analyzed first in hope to improve the upper bound faster and to obtain a better solution in case of premature break caused by reaching the time limit. At the beginning, an upper bound is calculated using a greedy heuristic (Heinicke et al., 2013). We do the test on a PC with an Intel® 2.8 GHz Quad-Core, GB RAM and Windows 7 (64-bit). The branch-and-bound methods are implemented in Java.

With branching strategy Append, we create at first all branches until a depth of $d = 3$: therewith, in case of $n \geq 4, m \geq 3$, we have $4\frac{n!}{(n-3)!}$ branches which are analyzed in a parallelized way. For example with $n = 15$ and $m = 3$, we create 10,920 branches.

For strategy Include, we sort the jobs by decreased customer cost coefficients, which leads to a significant improvement of the computation time compared to use an unsorted job list. The reason is that the unplanned jobs can be included at each position in each route by calculating a lower bound for the customer cost part. Thus, the more jobs with customer costs are fixed, the better the lower bounds for the customer cost part. Using the branching strategy Include, we create at first all branches until a depth d, which is chosen dependent on the number of jobs and machines. Therefore, we calculate the number of branches which is $\prod_{j=1}^{d}(m + j - 1)$. In experiments, we found out that we should have more than 500 threads for a good parallelization. We increase d until the number of defined branches will be larger than 500. For example, in case of 15 jobs and 3 routes, we set $d = 5$ and obtain 2520 branches which are analyzed in parallelized way.

In the following, we compare both branch-and-bound applications with solving the VRP-CC using the commercial solver CPLEX to the linearized formulation (MILP). In Heinicke et al. (2015), it was found out that (MILP) could be faster solved with CPLEX, if some additional subtour-elimination-constraints are iteratively added. Therefore, we used this approach for our experiments here. To further improve the performance of CPLEX, we start the calculation with an initial feasible solution, which is the same as used as upper bound in the branch-and-bound methods. Beside this, we use the default settings of CPLEX.

In this paper, other instances are used as for the calculations in Heinicke et al. (2015). These instances have a stronger influence of customer cost into the solution structure. Comparing the travel costs of an optimal solution of a VRP-CC instance with the optimal travel costs of a comparable instance without customer costs shows that considering customer costs leads in average to an increase in travel costs by 23% in the new instances, but only to 4% in the instances used for Heinicke et al. (2015). Because of that, the influence of customer costs is higher and the computation times for solving instances using CPLEX increased compared to the results in Heinicke et al. (2015).

In Table 4, we show the performance of the two branch-and-bound algorithms and of solving (MILP) with CPLEX. For the tests, we use the 100 small benchmark instances with non-zero customer cost coefficient for a part of the jobs, see Section 4.1.3. We compare the average computation time, the number of instances that are solved within one hour, the number of analyzed partial solutions and how often an approach leads to the best computation time of these three methods with a tolerance of 100 ms. Comparing the average computation time, CPLEX has the worst performance and the strategy Append the best. This is confirmed by the number of optimally solved instances. For both branch-and-bound approaches, in two instances the solution was not found within one hour, but 27 instances were not solved within one hour using CPLEX. It should be mentioned that in 18 of these instances, the obtained solution was an optimal one, but the optimality was not proved. Also the number of analyzed partial solutions shows that CPLEX has a higher calculation effort. In CPLEX, in average more than 8.8 trillion nodes are analyzed, but our branch-and-bound algorithms analyze in average 74 million partial solutions with strategy Append and 235 million partial solutions with strategy Include. Noteworthy is how often an approach solves an instance as fastest. In about one-third of the instances, CPLEX leads to the fastest computation time; in another one-third, branch-and-bound with strategy Append was the fastest method, and in more than the half of the instances, branch-and-bound with strategy Include leads to fastest computation of an optimal solution. Remarkably, in 11 instances, the difference in computation time between both branch-and-bound variants was smaller than 100 ms, so that both are counted to be the fastest method; but there was no instance, where CPLEX and one of the branch-and-bound approaches were both fast. Because of that, we additionally analyze the performance of the approaches in some special groupings of the 100 small instances with customer costs for a part of the jobs.

Table 4
Comparison of using the branch-and-bound algorithms and solving (MILP) with CPLEX.

	CPLEX	Branch-and-bound with strategy	
		Append	Include
Avg time (min)	17.0	1.8	3.3
Optimally solved	73%	98%	98%
Analyzed partial solutions	8.8×10^{12}	7.4×10^{7}	2.35×10^{8}
Fastest solved	27%	31%	52%

Table 5
Comparison of average computation time (in minutes) of the three approaches applied to different kinds of instances.

	Instances	CPLEX	Branch-and-bound with strategy	
			Append	Include
Grouped by ratio r_c of jobs with non-zero customer cost coefficient				
$r_c < 25\%$	17	0.07	0.04	0.41
$25\% \leq r_c < 50\%$	29	11.17	2.25	0.61
$50\% \leq r_c < 75\%$	33	23.49	0.42	0.53
$75\% \leq r_c$	21	28.76	4.66	13.86
Grouped by number n				
$n \in \{10, 11, 12\}$	29	3.03	0.01	0.01
$n \in \{13, 14, 15\}$	30	12.19	0.04	0.61
$n \in \{16, 17, 18, 19\}$	41	30.50	4.30	7.67
Grouped by increase in travel costs caused by considering customer costs				
Less than 5%	32	0.07	0.06	0.68
Between 5% and 25%	31	7.45	2.75	6.15
Larger than 25%	37	39.76	2.45	3.27

In Table 5, we show the average computation times in minutes of the three approaches for different groupings of the instances and for each group the number of instances belonging to it. At first, we grouped the instances by the ratio of jobs with non-zero costumer cost coefficient $r_c := \frac{1}{n}|\{j \in N \mid c_j > 0\}| \cdot 100\%$. Each group contains between 17 and 33 instances. This grouping shows that solving instances with higher customer cost ration r_c with CPLEX takes much more time than solving these instances with one of the branch-and-bound methods. Comparing both branch-and-bound approaches shows that instances with $r_c < 50\%$ are in average faster solved using strategy Include, but for instances with $r_c \geq 75\%$ the strategy Append takes significantly less time to obtain a proved optimal solution.

Next, we analyze the influence of the number of jobs. Therefore, we split the instances into three groups by the number of jobs. As expected, instances with more jobs are in average harder to solve than instances with less jobs. In each group, both branch-and-bound variants are in average better than solving the MILP-formulation with the commercial solver. Thereby, strategy Append performs better than strategy Include.

Finally, we group the instances by the influence of costumer costs represented by the increase in travel costs caused by considering customer costs. Therefore, we solve the instance with objective function $g(S) = g^d(S) + g^c(S)$, with the travel costs part $g^d(S)$, and call the solution S^*. Additionally we solve the problem with $\tilde{g}(S) = g^d(S)$ and obtain the solution S^d. The increase in travel costs is then given by $\frac{g^d(S^*)}{g^d(S^d)} - 1$. We differ between an increase less than 5%, between 5% and 25%, and larger than 25%. Each of these groups contains approximately one-third of the 100 instances. As it can be seen, again solving (MILP) with CPLEX is much more sensitive regarding the customer costs. Instances with an increase in travel costs less than 5% are solved fast with CPLEX and with the branch-and-bound approach using strategy Append. An increasing influence of the customer costs leads to significantly larger computation times for CPLEX. Our branch-and-bound approach with strategy Append solves the instances with an increase in travel costs of more than 5% in average within 2.6 min which is in average faster than with strategy Include. With it, the strategy Append has in each of these three groups in average the best performance.

This analysis shows that the branch-and-bound methods seem to handle customer costs significantly better than solving (MILP) with CPLEX. The strategy Include is in many cases faster than Append, because it solves more than the half of the instances as fastest, as shown in Table 4. In contrast, comparing the average computation times shows that the branch-and-bound approach with strategy Append is the better way to solve VRP-CC instances because it is more robust against the structure of an instance.

6. Conclusion

We presented two branch-and-bound algorithms to solve the VRP-CC efficiently. Therefore, we developed two branching strategies based on a non-linear formulation of the problem, where the execution times of the jobs are calculated directly from the schedule. For both branching strategies, we discussed the pros and cons.

To improve the algorithms, we designed some new lower bounds for the customer cost part of the objective function and discussed their usability for our branch-and-bound methods. We found out that a simple and fast bound for the customer costs is the most suitable one. Further, we summarized some lower bounds for the costs of the TSP, which are used as lower bounds for the travel cost part of the objective function, and analyzed their performance in the branch-and-bound algorithms.

Finally, we tested our branch-and-bound algorithms on 100 benchmark instances and compared the performance of both with the application of the commercial solver CPLEX to a linearization of the problem. We found out that the branch-and-bound approaches could better handle the time-dependent customer costs. Both branch-and-bound algorithms solved these instances with significantly less computation time than the commercial solver.

There are still some points for further research. A more detailed analysis about the influence of the instance structure to the performance of the different approaches is needed to select the most suitable solution method for a certain instance. For a further improvement of the branch-and-bound methods, even better bounds for the customer cost part should be developed. Additionally, it could be analyzed whether other bounds for the travel cost part improve the performance of the branch-and-bound approaches. Another interesting topic of further research concerns lower bounds for the whole problem taking into account travel costs and customer costs.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

We would like to thank the anonymous reviewers for there comments that helped to improve the paper.

References

Baker, E.K., 1983. An exact algorithm for the time-constrained traveling salesman problem. Oper. Res. 31 (5), 938–945.
Balas, E., 1977. Branch and bound/implicit enumeration. Ann. Discret. Math. 5, 185–191.
Baldi, M.M., Manerba, D., Perboli, G., Robero, T., 2019. A generalized bin packing problem for parcel delivery in last-mile logistics. EURO J. Oper. Res. 274 (3), 990–999.
Christofides, N., 1972. Bounds for the travelling-salesman problem. Oper. Res. 20 (5), 1044–1056.
Ebstein, L., Levin, A., 2007. Minimum weighted sum bin packing. In: Kaklamanis, C., Skutella, M. (Eds.), Proceedings of the International Workshop on Approx and Online Algorithms. Springer, Berlin, Heidelberg, pp. 218–231.
Fisher, M.L., Jörnsten, K.O., Madsen, O.B.G., 1997. Vehicle routing with time windows: two optimization algorithms. Oper. Res. 45 (3), 488–492.
Gavish, B., Srikanth, K., 1986. An optimal solution method for large-scale multiple traveling salesmen problems. Oper. Res. 34 (5), 698–717.
Heinicke, F., Simroth, A., Scheithauer, G., Fischer, A., 2015. A railway maintenance scheduling problem with customer costs. EURO J. Transp. Logist. 4 (1), 113–137.
Heinicke, F., Simroth, A., Tadei, R., Baldi, M.M., 2013. Job order assignment at optimal costs in railway maintenance. In: Proceedings of the 2nd International Conference on Operations Research and Enterprise Systems (ICORES 2013). Pp 304–309.
Held, M., Karp, R.M., 1971. The traveling-salesman problem and minimum spanning trees: Part II. Math. Program. 1 (1), 6–25.
Kohl, N., 1995. Exact Methods for the Time Constrained Routing and Related Scheduling Problems. University of Denmark.
Kruskal, J.B., 1956. On the shortest spanning subtree of a graph and the traveling salesman problem. Proc. Am. Math. Soc. 7 (1), 48–50.
Laporte, G., Nobert, Y., Taillefer, S., 1988. Solving a family of multi-depot vehicle routing and location-routing problems. Transp. Sci. 22 (3), 161–172.
Lucena, A., 1990. Time-dependent traveling salesman problem – the delivery man case. Network 20, 753–763.
Munkres, J., 1957. Algorithms for the assignment and transportation problems. J. Soc. Ind. Appl. Math. 5 (1), 32–38.
Prim, R.C., 1957. Shortest connection networks and some generalizations. Bell Syst. Tech. J. 36 (6), 1389–1401.
Reinelt, G., 1994. The Traveling Salesman: Computational Solutions for TSP Application. Springer-Verlag, Berlin Heidelberg.
Scheithauer, G., 2018. Introduction to Cutting and Packing Optimization. Springer International Publishing.