[EURO Journal on Computational Optimization 9 (2021) 100003](https://doi.org/10.1016/j.ejco.2020.100003)

Contents lists available at [ScienceDirect](http://www.ScienceDirect.com/)

EURO Journal on Computational Optimization

journal homepage: [www.elsevier.com/locate/ejco](http://www.elsevier.com/locate/ejco)

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

Franziska Theurich [a](#_bookmark0),[1](#_bookmark3),[∗](#_bookmark2), Andreas Fischer [b](#_bookmark1), Guntram Scheithauer [b](#_bookmark1)

a *Fraunhofer IVI, Zeunerstraße 38, Dresden 01069, Germany*

b *Faculty of Mathematics, Technische Universität Dresden, Dresden 01062, Germany*

a r t i c l e i n f o a b s t r a c t

*Keywords:*

Vehicle Routing Problem Railway maintenance scheduling Customer costs

Branch-and-bound

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.

# 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 limi- tations 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 over- all maintenance costs, the aim of our ballast tamping scheduling prob- lem is to compute a maintenance plan that minimizes the sum of travel and penalty costs.

To this end, in [Heinicke et al. (2013)](#_bookmark48) a new variant of the Vehi- cle Routing Problem (VRP) is developed which is called *Vehicle Rout- ing 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)](#_bookmark49) 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 so-

∗ Corresponding author

*E-mail address:* [franziska.theurich@ivi.fraunhofer.de](mailto:franziska.theurich@ivi.fraunhofer.de) (F. Theurich).

1 Formerly F. Heinicke.

lution 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 strate- gies based on a non-linear formulation in order to calculate the execu- tion times of the jobs directly from a feasible schedule; we present sev- eral lower bounds for the customer cost part of the objective function; and we give suitable lower bounds for the travel costs. By means of com- putational tests, we analyze which lower bounds lead to the best per- formance of the branch-and-bound methods. The needed upper bounds are calculated based on heuristics proposed in [Heinicke et al. (2013)](#_bookmark48). Fi- nally, we compare the performance of the developed branch-and-bound approaches against solving a corresponding MILP by means of a com- mercial solver (CPLEX). These computational results show significant improvements for our approach.

Branch-and-bound is a common principle to solve complex com- binatorial problems. An introduction to the method can be found in [Balas (1977)](#_bookmark41). 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 objec- tive 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.

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

2192-4406/© 2020 The Authors. Published by Elsevier Ltd on behalf of Association of European Operational Research Societies (EURO). This is an open access article under the CC BY-NC-ND license (<http://creativecommons.org/licenses/by-nc-nd/4.0/>)

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)](#_bookmark50). 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 so- lution. The resulting problem is to find a minimal spanning tree for the modified cost matrix. [Lucena (1990)](#_bookmark52) 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

* its working duration *ai* ,
* the travel costs *dij* from job *i* ∈ *Na* to job *j* ∈ *Na* , *j* ≠ *i*, and
* the travel time *rij* from job *i* ∈ *Na* to job *j* ∈ *Na* , *j* ≠ *i*.

All values are supposed to be non-negative integers and it is assumed that

* *ai >* 0 for all jobs *i* ∈ *N*,
* *𝑎*s*𝑘* = *𝑎*z*𝑘* = 0 and *𝑐*s*𝑘* = *𝑐*z*𝑘* = 0 for all *k* ∈ *M*, and
* both the travel costs and the travel times satisfy the triangle inequal-

ity.

(*Nk* )*k* ∈ *M* of *N* into *m* non-empty subsets and, for each *k* ∈ *M*, a permu- A feasible solution or *schedule S* of the VRP-CC consists of a partition

tation Π*𝑘* (*𝑁𝑘* ) = (*𝜋𝑘, 𝜋𝑘,* … *, 𝜋𝑘* ) defining the *route* of machine *k* which

1 2

|*𝑁𝑘* |

neglects the constraint that each city has to be visited. The resulting

is s*𝑘* ↦ *𝜋𝑘* ↦ *𝜋𝑘* ↦ ⋯ ↦ *𝜋𝑘*

↦ z*𝑘* with *𝜋𝑘* ∈ *𝑁𝑘* . For short, a schedule

problem is solved with dynamic programming. In [Fisher et al. (1997)](#_bookmark46) a VRP with time windows is solved by a branch-and-bound algorithm with Lagrangian relaxation and compared with a K-tree approach. The La- grangian 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)](#_bookmark42) 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 deci- sion 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

*𝑁𝑘 𝑖*

is represented by *S* ≔ (*Nk* , Π*k* (*Nk* ))*k* ∈ *M* .

1 2

| |

In difference to many VRPs, which only (or mainly) consider the

travel costs *gd*(*S*), in the VRP-CC an additional cost components *gc*(*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:

(PPP) min *𝑔*(*𝑆* ) = *𝑔𝑑* (*𝑆* ) + *𝑔𝑐* (*𝑆* )

( ( ))

s*.*t*. 𝑆* ∶= *𝑁𝑘,* Π*𝑘 𝑁𝑘 ,*

(*𝑁𝑘* )*𝑘*∈*𝑀*

is a

*𝑘*∈*𝑀*

partition of *𝑁* into exactly *𝑚* non-empty subsets*,*

problem. The dissertation of [Kohl (1995)](#_bookmark53) shows different decomposi- tion methods for the VRP with time windows, which lead to shortest path problems with time windows and capacity constraints as lower

Π*𝑘* (*𝑁𝑘* )is apermutation of the elements of *𝑁𝑘, 𝑘* ∈ *𝑀.*

(1)

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)](#_bookmark54), a branch-and-bound approach for a variant

Whereas the travel costs of *S* depend on the particular order of the

jobs, namely

−1

of the TSP resulting from the Location Routing Problem is shown. The

*𝑔𝑑* (*𝑆*) ∶=

∑ (*𝑑*s *𝜋𝑘* +

|*𝑁*∑*𝑘* |

*𝑑𝜋𝑘 𝜋𝑘*

+ *𝑑𝜋𝑘*

z )*,* (2)

objective function of the subproblems is estimated using a relaxation

*𝑘*∈*𝑀*

*𝑘* 1

*𝑖*=1

*𝑖 𝑖*+1

|*𝑁𝑘* |

that leads to a constrained assignment problem. To generate new sub- problems, the solution of the relaxation is checked for violations of the constraints that are relaxed.

*𝑘*

Among the lower bounds for the travel costs of the TSP, two main re- laxations are useful for our branch-and-bound approach. Firstly, remov- ing the subtour elimination constraints yields an assignment problem. Secondly, relaxing the node degree constraints leads to a particular min- imum 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)](#_bookmark43) and an enhancement of the bound obtained by 1-trees is presented in [Held and Karp (1971)](#_bookmark51). In [Section 4.2](#_bookmark35), 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](#_bookmark12), we propose two branching strategies and discuss their pros

and the customer costs of *S* are determined by the day *𝑡𝑑* ∈ ℕ on which job *i* ∈ *N* is executed resulting from the particular route Π*k* (*Nk* ) to which

job *i* belongs:

*𝑖*

*𝑔𝑐* (*𝑆*) ∶= *𝑐𝑖 𝑡𝑑 .* (3)

*𝑖*

∑

*𝑖*∈*𝑁*

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 *𝑢𝑖* ∶= *𝑢* − *𝑎𝑖* . In this case, the job has to

be postponed to the next day. For a correct scheduling, we calculate the

*execution time ti* 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: *𝑡𝑖* = (*𝑡𝑑 , 𝑡𝑚*).

*𝑖 𝑖*

and cons. Different lower bounds for customer costs and travel costs are provided and compared in [Section 4](#_bookmark24). The computational performance of the branch-and-bound algorithms for the non-linear model are inves- tigated in [Section 5](#_bookmark39). This includes a comparison with the application of CPLEX to a MILP model. The conclusions in [Section 6](#_bookmark47) complete the

presentation.

The first component *𝑡𝑑* ∈ ℕ represents the execution day and is the base for calculating the customer costs. The second component *𝑡𝑚* ≥ 0 is the execution minute on day *𝑡𝑑* that is restricted by the working shift, see

of any job *i* ∈ *N* must be bounded by *ai* ≤ *u*. Since travel time and condition [(6)](#_bookmark9) below. To avoid infeasibility of an instance, the duration

*𝑖*

*𝑖*

*𝑖*

working duration are in minutes, the execution time *𝑡𝑖* = (*𝑡𝑑 , 𝑡𝑚*) of job *i*

*𝑖 𝑖*

is converted into minutes by the function *𝜉* ∶ ℕ × ℝ → ℝ with

# The Vehicle Routing Problem with Customer Costs

*𝜉*(*𝑡𝑖* ) ∶= *ℎ 𝑡𝑑* + *𝑡𝑚,* (4)

*𝑖 𝑖*

Let *𝑁* = {1*,* 2*,* … *, 𝑛*} be a set of jobs that have to be scheduled to a set of machines *𝑀* = {1*,* 2*,* … *, 𝑚*}. Each machine *k* ∈ *M* has a start

depot s*k* and an end depot z*k* modeled as artificial jobs. The depot sets

where *h* ≔ 1440 represents the minutes per day. The execution time pair

*ti* of any job *i* ∈ *N* with job *pi* ∈ *N* ∪ *Ns* as predecessor in the schedule

has to fulfill:

are given as *Ns* ≔ {s*k* : *k* ∈ *M*} and *Nz* ≔ {z*k* : *k* ∈ *M*}. Furthermore, let *Na* ≔ *N* ∪ *Ns* ∪ *Nz* be the set of all jobs and depots with cardinality

*𝜉*(*𝑡𝑖* ) ≥ *𝜉*(*𝑡*

*𝑝𝑖*

) + *𝑎*

*𝑝𝑖*

+ *𝑟*

*𝑝𝑖 𝑖*

(5)

*𝑛𝑎* ∶= *𝑁𝑎* | = *𝑛* + 2*𝑚*. Each job *i* ∈ *Na* is characterized by

|

*𝑖*

* its customer cost coeﬃcient *ci* ,

and

0 ≤ *𝑡𝑚* ≤ *𝑢𝑖 .* (6)

Condition [(5)](#_bookmark8) ensures that the job *i* does not start before its predecessor

job *pi* is completed and the machine has traveled to job *i*. By restriction

(*𝑡𝑑 , 𝑡𝑚* ) = *𝑡*0 *, 𝑘* ∈ *𝑀* (21)

[(6)](#_bookmark9), the compliance with the working shift is guaranteed. For each job

*ℎ 𝑡𝑑* + *𝑡𝑚* − *ℎ 𝑡𝑑* − *𝑡𝑚*

*𝑖 𝑖 𝑗 𝑗*

s*𝑘*

s*𝑘*

*𝑖*

*ij*

*𝑡𝑑* ≤ *𝑡𝑑* ≤ *𝑑*

*, 𝑡𝑑* ∈ (ℕ)*, 𝑖* ∈ *𝑁*

*i* ∈ *N*, the arrival time *𝑡𝑎* is the time when the machine reaches the

*𝑖*

*ij*

*𝑠*

*𝑧*

location of job *i* and is calculated as

+ (n + *𝑎* + *𝑟* ) ⋅ *𝑥*

≤ n*, 𝑖* ∈ *𝑁* ∪ *𝑁, 𝑗* ∈ *𝑁*

∪ *𝑁, 𝑖* ≠ *𝑗* (22)

*𝑡𝑎* ∶= *𝜉*(*𝑡*

) + *𝑎*

+ *𝑟*

*,* (7)

0 *𝑖*

max *𝑖 𝑎*

*𝑖 𝑝𝑖*

*𝑝𝑖*

*𝑝𝑖 𝑖*

0 ≤ *𝑡𝑚* ≤ *𝑢𝑖 , 𝑡𝑚* ∈ (ℝ)*, 𝑖* ∈ *𝑁𝑎* (24)

*𝑖 𝑖*

(23)

where *𝑡𝑝𝑖* is the execution time of predecessor job *pi* which needs to be

already computed. Moreover we define the function *𝜁* ∶ ℝ × ℕ → ℕ × ℝ

to convert a time *t*, given in minutes, into a pair (*𝑡𝑑 , 𝑡𝑚*) that satisfies the

working shift condition [(6)](#_bookmark9):

( ) ⎧⎪(⌊ *𝑡* ⌋*, 𝑡* − ⌊ *𝑡* ⌋ ⋅ *ℎ*) if *𝑡* − ⌊ *𝑡* ⌋ ⋅ *ℎ* ≤ *𝑢 ,*

*𝑖*

In (MILP), the binary variables *xij* are 1, if and only if job *i* is the pre- decessor of *j* in the solution. The constraints [(11)](#_bookmark14) and [(12)](#_bookmark15) ensure that each job has one predecessor and one successor. The trips between the depots are regulated by the [Eqs. (13)](#_bookmark16)–[(15)](#_bookmark17). The integer variable *yi* repre- sents the machine that executes job *i*. This is necessary to guarantee the

*𝜁 𝑡, 𝑢𝑖* ∶=

⎨

*ℎ*

*ℎ*

*ℎ*

⎪⎩

*ℎ*

(⌊ *𝑡* ⌋ + 1*,* 0) otherwise*.* (8)

correct machine. Inequality [(19)](#_bookmark20) ensures that the jobs between a start depot and an end depot are allocated to the same machine as the depots

Then, the execution times of the jobs are calculated based on the

correct order of the depots. [Eqs. (17)](#_bookmark18) and [(18)](#_bookmark19) allocate the depots to the

*𝑘*

itself. The execution time of a job *i* ∈ *Na* is given by the decision variables

*𝑡𝑑* and *𝑡𝑚,* with the same meaning as before. By inequality [(22)](#_bookmark10), with a

schedule *𝑆* = (*𝑁𝑘,* Π (*𝑁𝑘* ))*𝑘*∈*𝑀* . For the start depots, we define the start *𝑖 𝑖*

*𝑑 𝑚*

*𝑑 𝑚*

time *𝑡*0 = (0*, 𝑢*) and set

*𝑡**𝜋𝑘* ∶= *𝑡*s*𝑘* = *𝑡*0*, 𝑘* ∈ *𝑀.* (9)

0

Thus, each route starts at the end of day 0 in the start depot. The execu- tion time of the jobs are calculated in the order defined by the permu- tations based on the arrival time as

*𝜋*

*𝑖*

suﬃciently large n*,* it is guaranteed that *ℎ 𝑡𝑗* + *𝑡𝑗* ≥ *ℎ 𝑡𝑖* + *𝑡𝑖* + *𝑎𝑖* + *𝑟𝑖𝑗*

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*.

# The branching strategies

*𝑡 𝑘*

*𝜋*

*𝑖*

*𝜁 𝑡𝑎 , 𝑢*

*𝜋𝑘*

*𝑖*

∶= (

*𝑘* ) = *𝜁* (*𝜉*(*𝑡*

*𝑘*

*𝑖*−1

*𝜋*

) + *𝑎*

*𝑘*

*𝑖*−1

*𝜋*

+ *𝑟*

*𝑘 𝑘*

*𝑖*−1 *𝑖*

*𝜋 𝜋*

*, 𝑢*

*𝑘 ,*

*𝑖*

*𝜋*

)

The definition of the branching strategies relies on the formulation (PPP). Thus, we do not branch in respect to binary or integer vari- ables from a MILP formulation, but we create new subproblems by in-

*𝑖* = 1*,* 2*,* … *, 𝑁𝑘 , 𝑘* ∈ *𝑀.* (10)

| |

This discrete non-linear model, described by *S*, the objective function *g(S)* [(1)](#_bookmark4) with travel costs [(2)](#_bookmark5) and time-dependent customer costs [(3)](#_bookmark6), and the calculation of the execution times via the [Eqs. (9)](#_bookmark11) and [(10)](#_bookmark13), is very in- tuitive. The advantages of this model are the low memory requirements,

a solution can be stored in an array of length *𝑛* + 2 *𝑚,* and that the exe-

cution 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)](#_bookmark49), different linearizations of VRP-CC (i.e., MILP formulations) were devel- oped 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-n lineariza- tion of time constraints. Therefore, we use this model for comparison and call it here (MILP). In this model, the routes are arranged one af- ter another so that one tour visiting all jobs and depots represents a schedule.

(MILP) min ∑ ∑ *𝑑ij 𝑥ij* + ∑ *𝑐𝑖 𝑡𝑑*

*𝑖*

tegrating a further job to a partial solution. A partial solution *𝑆̃* is de- fined by the set of not yet planned jobs *𝑁̃* that will also be called un- planned jobs, and the uncompleted schedule (*𝑁̃𝑘 ,* Π*𝑘* (*𝑁̃𝑘* ))*𝑘*∈*𝑀 ,* whereby

*𝑁̃* ∪ *𝑘*∈*𝑀 𝑁̃𝑘* = *𝑁* . Let S denote the set of all open partial solutions.

(⋃ )

At the beginning, we set S ∶= {(*𝑁,* (∅*,* ())*𝑘*∈*𝑀* )}. In a branching step, we select one open partial solution *𝑆̃* ∈ S and create a set of new partial solutions by integrating one job *𝑖* ∈ *𝑁̃* 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 applica- tion 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](#_bookmark39).

* 1. *Append a job on a partial solution*

With the first branching strategy, we create a set of new partial so- lutions 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 par- tial solution can be determined very eﬃciently since only the execution

*𝑖*∈*𝑁𝑎 𝑗*∈*𝑁𝑎* ⧵{*𝑖*}

*𝑖*∈*𝑁*

time of the appended job has to be calculated to add the resulting travel

s.t.

∑

*𝑗*∈*𝑁𝑎* ⧵{*𝑖*}

*𝑥ij*

= 1*, 𝑖* ∈ *𝑁𝑎*

(11)

costs and customer costs of this job.

In detail, the branching strategy Append works as follows: Let *𝑆̃* ∈ S

be a partial solution. We create a set of new partial solutions by append-

*𝑗*∈*𝑁*∑*𝑎* ⧵{*𝑖*}

*𝑥ji* = 1*, 𝑖* ∈ *𝑁𝑎* (12)

ing each unplanned job at the end of certain routes. To avoid repeated solutions (which would be generated very often otherwise), we fill the

*𝑥*z*𝑚* s1 = 1 (13)

routes one after the other. To this end, index *𝑙* ∶= *𝑙*(*𝑆̃*) ∈ *𝑀* denotes the

*𝑥*z*𝑘* s*𝑘*+1

= 1*, 𝑘* ∈ *𝑀* ⧵ {*𝑚*} (14)

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

*𝑥*s*𝑘* z*𝑙* = 0*, 𝑘, 𝑙* ∈ *𝑀* (15)

*𝑥ij* ∈ {0*,* 1}*, 𝑖, 𝑗* ∈ *𝑁𝑎* (16)

of that, we call the routes 1*,* 2*,* … *, 𝑙* − 1 closed and the routes *𝑙, 𝑙* + 1*,* … *, 𝑚*

open. Note, that the routes *𝑙* + 1*,* … *, 𝑚* of the partial solution are empty.

*𝑦*s*𝑘*

*𝑦*z

= *𝑘, 𝑘* ∈ *𝑀* (17)

= *𝑘, 𝑘* ∈ *𝑀* (18)

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

*𝑘*

*𝑦* − *𝑦*

*𝑖*

*𝑗*

*ij*

*𝑠*

*𝑧*

≤ (1 − *𝑥* ) *𝑚, 𝑖* ∈ *𝑁* ∪ *𝑁, 𝑗* ∈ *𝑁*

∪ *𝑁* (19)

of the first partial solution is set to *𝑙*((*𝑁,* (∅*,* ())

*̃*

If *𝑙* = *𝑙*(*𝑆* ) *>* 0*,* we create for each unplanned job *𝑖* ∈ *𝑁* one new par-

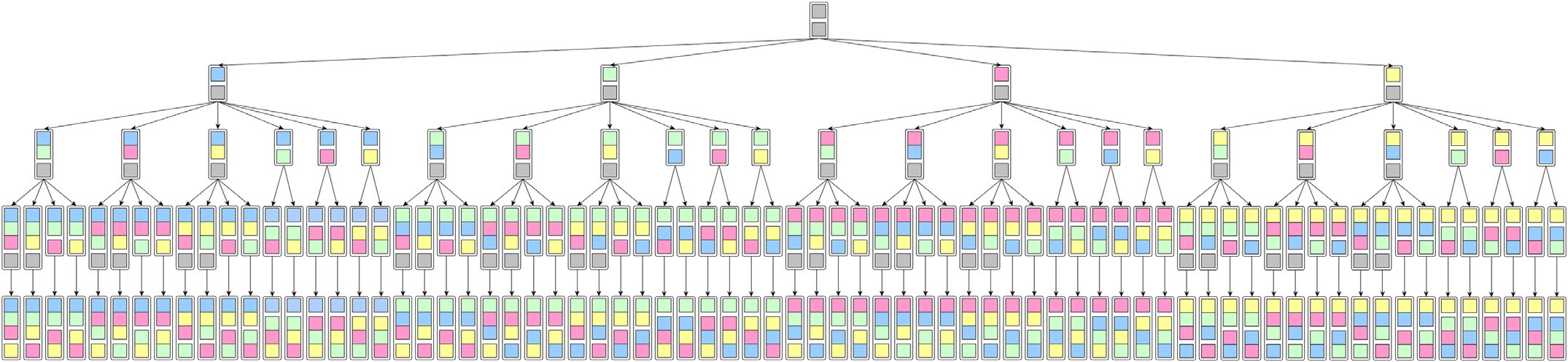
*𝑘*∈*𝑀*

)) = 0.

*̃*

*𝑦𝑖* ∈ {1*,* 2*,* … *, 𝑚*}*, 𝑖* ∈ *𝑁𝑎* (20)

tial 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

*𝑆𝑖,𝑙* = (*𝑁̃* ⧵ {*𝑖*}*,* (*𝑁̃ 𝑙* ∪ {*𝑖*}*,* (*𝜋𝑙 ,* … *, 𝜋𝑙 ̃*

1 |*𝑁𝑙* |

*, 𝑖*))*,* (*𝑁̃ 𝑘 ,* Π*𝑘* (*𝑁̃ 𝑘* )) )*.*

* 1. *Include a job into a partial solution*

With the second branching strategy, we generate new partial solu- tions by including a certain job inside the routes. This strategy is called

And, if *𝑙* = *𝑙*(*𝑆̃*) *< 𝑚,* we create for each unplanned job *𝑖* ∈ *𝑁̃* one new partial solution by initializing route *𝑙* + 1 with this job, which leads to

*𝑘*∈*𝑀* ⧵{*𝑙*}

*𝑆* = (*𝑁̃* ⧵ {*𝑖*}*,* ({*𝑖*}*,* (*𝑖*))*,* (*𝑁̃ ,* Π*𝑘* (*𝑁̃* )) )

*𝑖,𝑙*+1

*𝑘*

*𝑘*

*𝑘*∈*𝑀* ⧵{*𝑙*+1}

*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 *𝑆̃* ∈ S

With it, we create *𝑁̃* or 2 *𝑁̃* new partial solutions.

| | | |

be a partial solution. To create a set of new partial solutions, we select

We illustrate the branching strategy Append by means of a small ex- ample with four jobs and two routes in [Fig. 1](#_bookmark21). 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 col-

routes. Let *𝑖* ∈ *𝑁̃* be this job. Then, we generate the following new partial one unplanned job that will be included into each position of the current

solutions:

*𝑆𝑗,𝑙* ∶= (*𝑁̃* ⧵ {*𝑖*}*,* (*𝑁̃𝑙* ∪ {*𝑖*}*,* (*𝜋𝑙 ,* … *, 𝜋𝑙 , 𝑖, 𝜋𝑙 ,* … *, 𝜋𝑙* ))*,* (*𝑁̃𝑘 ,* Π*𝑘* (*𝑁̃𝑘* ))*𝑘*∈*𝑀*⧵{*𝑙*})

*̃*

ored boxes for the jobs. An empty route is symbolized by a gray square.

+ 1}*, 𝑙* ∈ *𝑀.*

1 *𝑗*−1 *𝑗*

|*𝑁𝑙* |

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

| *𝑙*| | *𝑙*|

*𝑗* ∈ {1*,* 2*, ..., 𝑁̃ , 𝑁̃*

With it, we create *𝑁*

– *𝑁̃*

+ *𝑚* new partial solutions, because the job *i*

second job was appended to route two, only two new branches are gen- erated 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 un- balances 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 ap- pend the fourth job – always to route two since all routes must be used. Thus, there are many branching nodes created in the first branching lev- els 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 level 0, we have to create *n* branches, one for each job. In level *𝜅*, at In the first branching levels, much more branches are created than later: most 2(*𝑛* − *𝜅*) branches are produced, because *𝑛* − *𝜅* 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 be- cause of filling the routes one after the other, which could be seen in the small example shown in [Fig. 1](#_bookmark21). 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.

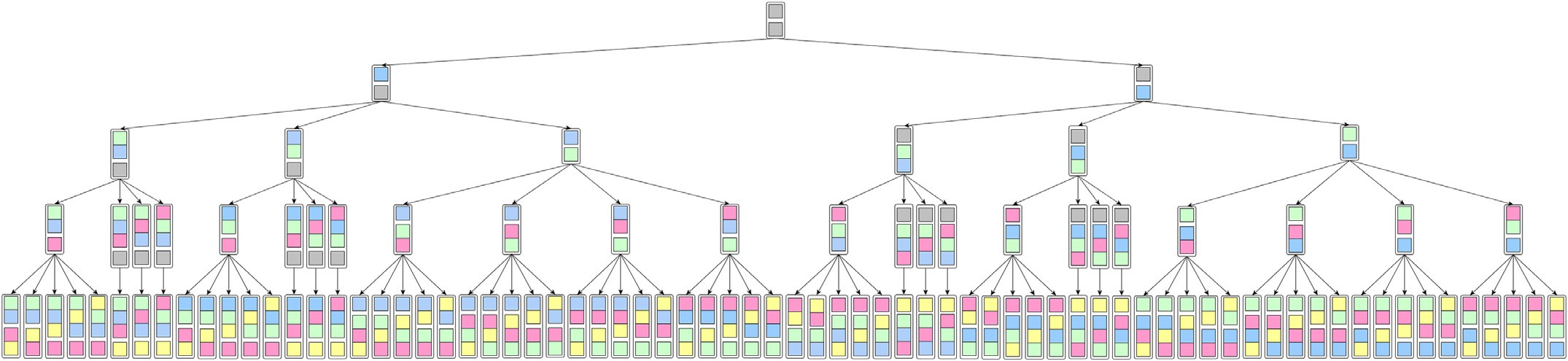
can be included after each already planned job, which leads to *𝑁* − *𝑁̃*

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](#_bookmark22) with four jobs and two routes. In branching level 0, two branches are generated: In- cluding the first job to route one and to route two. For each of both partial solutions, three new branches are generated including the sec- ond 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 solu- tion, 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 par- tial 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, elim- inating 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 cal- culation effort because some execution times have to be recalculated to obtain the costs of a partial solution. On the other hand, the calcula- tion 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](#_bookmark39), we analyze whether better lower bounds or a more balanced search tree leads to faster computation times.

# 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) pre- sented in [Section 2](#_bookmark7). 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 ,* the travel cost part is called LB*𝑑* . However,

The solution can be found easily by sorting the jobs appropriately. Then, to obtain a lower bound, we solve the problem described by [(28)](#_bookmark23)– [(30)](#_bookmark25).

LB*𝑐* ∶= min *𝑐𝑖 𝜏𝑖* (28)

s

∑

*𝑖*∈*𝑁𝑐*

s.t. *𝜏𝑖* ∈ *𝑇 , 𝑖* ∈ *𝑁𝑐* (29)

{*𝑖* ∈ *𝑁𝑐 𝜏𝑖* = *𝑡*} ≤ *𝜂 𝑡* ∈ *𝑇* (30)

| | |

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

LP LP

as we will see, lower bounds are needed that are more appropriate.

In the first subsection, we propose some lower bounds for the cus- tomer cost part and compare their quality and computation time. For the most suitable lower bound on customer costs, we present needed modi- fications to apply it in both branching strategies. In [Section 4.2](#_bookmark35), we out- line 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.

* 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

|

*𝑐* |. We set *𝑇*

* + 1. *Lower bound from minimum weighted sum bin packing problems*

The first lower bound LB*𝑐* is very simple. Instead of using a fixed

s

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](#_bookmark44)). In con- trast 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)](#_bookmark56) 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)](#_bookmark45), where the

items are aﬄicted with a bin-dependent profit and the usage of bins

by *N**c*

∑

of days for job execution taking into account that *𝑡*0 = (0*,* 480).

≔ {*i* ∈ *N*| *ci >* 0} with *𝑛𝑐*

= *𝑁*

= {1*,* 2*,* …} to be the set

leads to costs.

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

*4.1.1. Trivial lower bound for customer costs*

The idea of this bound is very simple: we calculate how many jobs

(BP1) LB*𝑐*

∶= min

s

*𝑖*∈*𝑁𝑐 𝑥it*

)

*𝑖*∈*𝑁𝑐*

∑*𝑡*∈*𝑇*

*𝑐𝑖 𝑡 𝑥it*

can be done at most per day and select the most expensive jobs for the

BP1

*.*t*. 𝑡*∈*𝑇 𝑥it* = 1*,*

*𝑖* ∈ *𝑁𝑐*

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

first day, the next most expensive jobs for the second day, and so on. At

needed per job for execution and to travel to the next one, which is

*𝑎𝑖* + *𝑟𝑖 ,* with *ri* ≔ min {*rij* | *j* ∈ *Nc, j* ≠ *i*} is the minimal time needed to

where

∑ (

*𝑎𝑖* + *𝑟𝑖*

≤ *mu* + *𝑅𝑚, 𝑡* ∈ *𝑇*

*𝑥it* ∈ {0*,* 1}*, 𝑖* ∈ *𝑁𝑐 , 𝑡* ∈ *𝑇*

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

∑ | | |

*𝑅𝑚* ∶= max{ *𝑟𝑖 𝐼 ⊆ 𝑁𝑐 , 𝐼* = *𝑚*}*.* (25)

*𝑖*∈*𝐼*

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

min (*𝑎𝑖* + *𝑟𝑖* ) ≤ *𝑚𝑢* + *𝑅𝑚* (26)

||

*𝐼 ⊆𝑁𝑐 𝐼*

and

*𝜂*

{ = } ∑

|

*𝑖*∈*𝐼*

* *ri* ≔ min {*rij* | *j* ∈ *Nc, j* ≠ *i*} is again the minimal travel time from job

*i* ∈ *Nc* to any other job in *Nc* ,

* *Rm* is the time reserved for the trips outside the working shift, see

[(25)](#_bookmark26), and

* *𝑥𝑖𝑡* = 1 if and only if job *i* is allocated to day *t* ∈ *T*.

time available for maintenance and travel per day *𝑚𝑢* + *𝑅𝑚* . The time In (BP1), each bin corresponds to one day. The bin size is set to the

needed for a job *i* to execute it and travel to the next one is at least

*𝑎𝑖* + *𝑟𝑖* which is the size of the job in the bin packing problem. In the

*mu* + *𝑅 <* min ∑(*𝑎* + *𝑟* ) (27)

*𝑐* || |

*𝑚*

{*𝐼⊆𝑁 𝐼* =*𝜂*+1}

*𝑖*∈*𝐼*

*𝑖*

*𝑖*

objective function, the total customer cost of job *i* is calculated based on

the day *t* with *𝑥𝑖𝑡* = 1.

To improve the bound, we define a second bin packing problem

*𝑖*∈*𝑁𝑐*

(BP2) where for each machine an own bin per day is defined.

≤ ∑ *𝑥*

*𝑎* + ∑ *𝑦*

*𝑟* + *𝑟𝑗𝑏*

(33)

(BP2) LB*𝑐*

*𝑖*∈*𝑁𝑐*

∶= min ∑

*𝑖*∈*𝑁𝑐*

∑*𝑡*∈*𝑇*

∑*𝑘*∈*𝑀*

*𝑐𝑖 𝑡 𝑥*

*itk*

≤ *𝑢* + *𝑟𝑗𝑏*

∀*𝑏* ∈ *𝐵* (34)

s*.*t*.* ∑*𝑡*∈*𝑇* ∑*𝑘*∈*𝑀*( *𝑥itk* =)1*, 𝑖* ∈ *𝑁𝑐*

BP2

max

*𝑖𝑏*

*𝑖*∈*𝑁𝑐 𝑥itk*

*𝑎𝑖* + *𝑟𝑖*

≤ *𝑢* + *𝑟𝑘*

*, 𝑡* ∈ *𝑇 , 𝑘* ∈ *𝑀*

Inequality [(32)](#_bookmark31) shows the transition from (BP3) to (BP2). The formula

*itk* ∈ {0*,* 1}*, 𝑖* ∈ *𝑁𝑐 , 𝑡* ∈ *𝑇 , 𝑘* ∈ *𝑀*

max

*𝑥*

day and machine. Each bin associated to route *k* ∈ *M* has a capacity of Problem (BP2) is a bin packing problem with *m* · |*T*| bins, one per

*𝑢* + *𝑟𝑘* where *𝑟𝑘* is the *k*th largest value of the set {*ri* | *i* ∈ *Nc* }. Thus,

max

max

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

on the left side equals to the size of the items of a bin in (BP3), which is

additional summand *ri* . This additional summand is not larger than *𝑟𝑗𝑏* smaller than the size of the items of the bin in (BP2) because this has an

max

and with it, inequality [(33)](#_bookmark27) is true. The fact that in (BP3) the bin size is limited to *u*, leads to inequality [(34)](#_bookmark28). Thus, the variables *xib* of a feasible solution of (BP3) lead to a feasible solution of (BP2). The contrary is not necessarily true, because in (BP2) we have

*𝑖*∈*𝑁𝑐*

*𝑖𝑏*

*𝑖*

*𝑖*

max

jobs in the bin packing problem is *𝑎𝑖* + *𝑟𝑖* 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

∑ *𝑥*

(*𝑎* + *𝑟* ) ≤ *𝑢* + *𝑟𝑘 .*

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

we have to ensure that *𝑖*∈*𝑁𝑐 𝑥𝑖𝑏* (*𝑎𝑖* + *𝑟𝑖* ) − max*𝑖*∈*𝑁𝑐 𝑥𝑖𝑏 𝑟𝑖* ≤ *𝑢,* because To transform a feasible solution of (BP2) into a feasible solution of (BP3),

in (BP3) one travel item is removed and the bin size is decreased by

∑

maximal trip length. The following bin packing problem (BP3) models

≥ LB

*𝑘*

max

*𝑟*

. This is not guaranteed for each feasible solution, because normally

this by considering working duration *a* and minimal possible travel time

max*𝑖*∈*𝑁 𝑥𝑖𝑏 𝑟𝑖 < 𝑟𝑘*

. Thus we get LB*𝑐*

≥ LB*𝑐* .

*i 𝑐*

BP2

max

BP3

BP2

*ri* as separated items.

Second, we proof that LB*𝑐*

*𝑐*

BP1

. Because of

(BP3) LB*𝑐*

max

∶= min ∑*𝑖*∈*𝑁𝑐*

∑*𝑏*∈*𝐵*

*𝑐𝑖 𝑡𝑏 𝑥ib*

∑ ∑ (*𝑎𝑖* + *𝑟𝑖* )*𝑥𝑖𝑡𝑘* ≤ ∑ (*𝑢* + *𝑟𝑘*

) = *𝑚𝑢* + *𝑅𝑚* ∀*𝑡* ∈ *𝑇 ,*

s*.*t*.* ∑*𝑖*∈*𝑁 𝑥ib 𝑎𝑖* + ∑*𝑖*∈*𝑁 𝑦ib 𝑟𝑖* ≤ *𝑢, 𝑏* ∈ *𝐵*

BP3

∑ ∑

*𝑘*∈*𝑀 𝑖*∈*𝑁𝑐*

*𝑘*∈*𝑀*

*𝑐 𝑐*

*𝑖*∈*𝑁𝑐 𝑦ib* + 1*, 𝑏* ∈ *𝐵*

∑ ∈ *𝑥ib* = 1*, 𝑖* ∈ *𝑁𝑐*

∑

*𝑖*∈*𝑁𝑐 𝑥ib* ≤

each feasible solution of (BP1) is feasible for (BP2). Let *𝑁̂* (*𝑡*) ∶= {*𝑖* ∈ each feasible solution of (BP2) is also feasible for (BP1). However, not

*𝑏 𝐵*

*𝑏*∈*𝐵 𝑦ib* ≤ 1*, 𝑖* ∈ *𝑁𝑐*

*𝑁𝑐* | *𝑥*

*𝑖𝑡*

= 1} denote the set of jobs executed at day *t* ∈ *T*. To obtain a

*̂*

*𝑦ib* − *𝑥ib* ≤ 0*, 𝑖* ∈ *𝑁𝑐 , 𝑏* ∈ *𝐵*

feasible solution of (BP2), we have to find a partition of *𝑁* (*𝑡*) into m

*𝑥* ∈ {0*,* 1}*, 𝑦* ∈ {0*,* 1}*, 𝑖* ∈ *𝑁 , 𝑏* ∈ *𝐵*

subsets *𝑁̂ 𝑘* (*𝑡*)*, k* ∈ *M*, with ∑*𝑖*∈*𝑁̂* (*𝑡*) (*𝑎𝑖* + *𝑟𝑖* ) ≤ *𝑢* + *𝑟𝑘* which is not nec-

where

*ib ib*

*𝑐* essarily possible. Thus, we have *𝑘*LB*𝑐*

*𝑐*

BP1

≥ LB

.

max

Finally, it remains to show, that LB*𝑐* ≥ LB*𝑐* . Because of

BP2

BP1

s

*𝜂* for each *t* ∈ *T*. Thus each feasible solution of (BP1) satisfies the con-

s

BP1

s

* *B* denotes the set of bins (one per day and machine),

∑*𝑖*∈*𝑁𝑐 𝑥𝑖𝑡* (*𝑎𝑖* + *𝑟𝑖* ) ≤ *𝑢* + *𝑅𝑚* and the definition of *𝜂* we obtain ∑*𝑖*∈*𝑁𝑐 𝑥𝑖𝑡* ≤

*b* | *b* |

* *t*

is the day of bin *b* with *t*

∈ *T* and {*𝑏* ∈ *𝐵 𝑡*

* *𝑟𝑖* = min{*𝑟𝑖𝑗 𝑗* ∈ *𝑁𝑐 , 𝑗* ≠ *𝑖*} is the minimal travel time from job *i* ∈ *Nc*

to any other job in *Nc* ,

| *𝑏* |

straints used to calculate LB*𝑐* and we get LB*𝑐*

≥ LB*𝑐* . □

* *𝑥𝑖𝑏* = 1 if and only if the execution of job *i* is allocated to bin *b*, and

= *𝑡*} = *𝑚, 𝑡* ∈ *𝑇 ,*

* *𝑦𝑖𝑏* = 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 *ai* and the travel item *ri* . 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 (*yib* ≤ *xib* ).

Finally, note that LB*𝑐 ,* LB*𝑐 ,* LB*𝑐* and LB*𝑐* are indeed lower

* + 1. *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 in- stances, 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 op- timal solution with the optimal costs minimizing only the travel costs *gd*, 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 coef- ficient of each job by 10 and multiply the resulting value with 100. With

s BP1

BP2

BP3

bounds. Moreover the following result is satisfied.

**Theorem 1.** *For any instance of VRP-CC, we have* LB*𝑐*

≥ LB*𝑐* ≥

it, about 99% of the optimal total costs are customer costs. For most of the instances dominated by customer costs, the optimal solution value

*𝑐*

LB

s

BP1

≥ LB*𝑐 .*

BP3

BP2

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,

**Proof.** [Theorem 1](#_bookmark30) First, we show that LB*𝑐*

is not worse than LB*𝑐* .

where the optimal solution value could not be determined.

Let {*x*

} and {*y*

}, *i* ∈ *N* , *b* ∈ *B*

BP3

BP2

The bounds are compared in quality measured by *𝑞* ∶= bound value in

*ib ib*

*c* , be a feasible solution of (BP3). We

*𝑔𝑐* (*𝑆* ∗ )

have to show that this solution can be transformed into a solution of

to a job executed on the next day. For all *t* ∈ *T*, there exists a mapping same costs of (BP2). At first, for each machine we have at most one trip

from bin *b* to one route *jb* in (BP2) so that

percent, where *gc*(*S*∗) is the customer cost part belonging to an opti- mal 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

*𝑖* ∈∑*𝑁𝑐*

*𝑥ib* = 1*, 𝑦ib* = 0

⋃

*𝑗𝑏*

*𝑖* max

*𝑟* ≤ *𝑟*

∀*𝑏*

∈ *𝐵* with *𝑡𝑏*

= *𝑡*

distances to the best bound Δ ∶= 1 − bound value in percent in-

stead of the quality. Further, we compare the computational effort since

max obtained bound value

for the branch-and-bound method, lower bounds are needed that can be calculated fast. We measure the computational effort as the average

and

*𝑏*∈*𝐵 𝑡* =*𝑡*

{*𝑗𝑏* } = *𝑀.* (31)

computation time. The lower bounds based on bin packing problems are

computed with CPLEX. If no optimal solution was found within the time

{ *𝑏* }

*𝑖*

*𝑖𝑏*

*𝑖*

max

|

Then, we have

∑ *𝑥𝑖𝑏 𝑎𝑖* + ∑ *𝑦𝑖𝑏 𝑟𝑖* ≤ ∑ *𝑥𝑖𝑏* (*𝑎𝑖* + *𝑟𝑖* ) (32)

*𝑖*∈*𝑁*

*𝑖*∈*𝑁*

*𝑖*∈*𝑁*

*𝑐*

*𝑐*

*𝑐*

limit of 60 s, we use the minimal possible objective function value of the remaining unexplored nodes.

In [Table 1](#_bookmark32), we compare different customer cost bounds: the bound

LP

s

of the LP-relaxation LB*𝑐 ,* the trivial bound LB*𝑐 ,* and the bounds based

**Table 1**

Comparison of different customer cost bounds.

LP s

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| LB*𝑐* | | LB*𝑐* | LB*𝑐* | LB*𝑐* | LB*𝑐*  BP3 |
| 100 instances with customer costs for a part of jobs | | | | | |
| Avg *q* | 86% | 94% | 97% | 97% | 97% |
| *q* ≥ 95% | 48% | 70% | 86% | 87% | 88% |
| *q* ≥ 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* ≥ 95% | 25% | 56% | 96% | 98% | 99% |
| *q* ≥ 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 |

BP1

BP2

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* for a partial solution *𝑆̃* = (*𝑁̃,* (*𝑁̃𝑘,* Π*𝑘*(*𝑁̃𝑘*))*𝑘*∈*𝑀* )*,* we

s

have to take into account all jobs with customer costs. The jobs, that are

calculated based on the partial solution, which is *𝑡𝑑 , 𝑖* ∈ *𝑁* ∖*𝑁̃ .* Using the allocated to a route in the partial solution, have a day for their execution branching strategy Include, we calculate the lower bound LB*𝑐*ep*,*include as

*𝑖*

follows:

LB*𝑐*ep*,*include ∶= min *𝑐𝑖𝜏𝑖* (35)

∑

*𝑖*∈*𝑁𝑐*

s.t. *𝜏𝑖* ∈ *𝑇 , 𝑖* ∈ *𝑁𝑐* (36)

|{*𝑖* ∈ *𝑁𝑐* | *𝜏𝑖* = *𝑡*}| ≤ *𝜂, 𝑡* ∈ *𝑇* (37)

*𝑖*

BP3

*𝑡𝑑* ≤ *𝜏𝑖, 𝑖* ∈ *𝑁𝑐* ⧵ *𝑁̃*

(38)

on bin packing problems LB*𝑐*

BP1

*𝑐*

BP2

*,* LB

*,* and LB*𝑐*

. For the two sets of

zero customer costs gets an execution day *𝜏i* in the relaxation [(35)](#_bookmark33)– The constraint [(38)](#_bookmark34) ensures that each already allocated job with non-

small instances, we provide the average bound quality *q*, how often the obtained bound quality *q* exceeds 95% and 90%, and the average com-

average of the distances Δ in percent, the average computation time in putation time in milliseconds. For the 25 large instances, we show the

milliseconds, and how many instances were not solved within 60 s.

The LP-relaxation leads to the worst bounds. The analysis of the re- sults 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

with a commercial solver. The trivial bound LB*𝑐* performs well. In both estimation of the customer costs is a key problem by solving the VRP-CC

s

kinds of small instances, the quality is in average about 94% and 92%,

see that LB*𝑐* is in average 9% worse than the best-obtained bound. The respectively. Analyzing the bounds obtained for the large instances, we

s

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*

or LB*𝑐*

are better than LB*𝑐*

*,*

[(38)](#_bookmark34), which is not earlier than the execution day of the current partial solution. This optimization problem can be solved by suitably ordering the jobs.

* 1. *Lower bounds for travel costs*

Lower bounds for travel costs are well known from research concern- ing the TSP, see [Reinelt (1994)](#_bookmark57). 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 ma-

trix *𝐷* = (*𝑑𝑖𝑗* )*𝑖,𝑗*∈*𝑁𝑎 ,* see [Heinicke et al. (2015)](#_bookmark49). To this end, the travel

costs between the depots z*k* and s*𝑘*+1*, 𝑘* = 1*,* … *, 𝑚* − 1*,* as well as z*m* and

s1 are set to zero, and all the other travel costs between depots are set to infinity.

* + 1. *The two-neighbor bound*

A simple lower bound for the travel costs is the so-called 2-neighbor bound ([Reinelt, 1994](#_bookmark57)):

LB*𝑑* ∶= 1 (∑ min{ + | ∈ ∪ }

2N

2

*𝑖*∈*𝑁*

*𝑑𝑗𝑖*

*𝑑𝑖𝑘 𝑗*

*𝑁*

*𝑁𝑠, 𝑘*

*𝑁*

*𝑁𝑧, 𝑗* ≠ *𝑖, 𝑘* ≠ *𝑖, 𝑘* ≠ *𝑗*

∈

∪

computation time to calculate the bounds differs significantly. LB*𝑐* which can be seen comparing how often the quality exceeds 95%. The

BP2

BP3

BP1

BP1

is calculated with the smallest computation effort, much more time is

and the most time is required for LB*𝑐*

. Especially for

+ ∑ min{*𝑑𝑖𝑗* | *𝑗* ∈ *𝑁* } + ∑ min{*𝑑𝑗𝑖*| *𝑗* ∈ *𝑁* })

BP2

needed for LB*𝑐*

*𝑖*∈*𝑁𝑠*

*𝑖*∈*𝑁𝑧*

BP3

For each job *i* ∈ *N*, the costs to its two nearest neighbors are summed up.

bound LB*𝑐* could be calculated within the time limit of 60 s for each of the large instances, the increase of computation time is substantial. The

BP1

the 25 large instances. In 9 of these instances, it was not possible to op- timally solve (BP2) within the time limit, and the calculation of bound

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

*𝑐*

LB

BP3

requires in 17 instances more than 60 s. Because of these unsolved

*𝑐*

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.

instances, the average Δ-value of LBBP3 is not zero, in difference to the

theoretical statement.

Because the bin packing problems are NP-hard, we also tested their

to the trivial bounds LB*𝑐* . Moreover, since the LP relaxation of (BP1) LP relaxations. It turned out that the LP relaxation bounds are similar

s

* + 1. *Lower bound from assignment problem*

Another lower bound can be derived from the assignment problem:

(AP)LB*𝑑*AP ∶= min ∑*𝑖*∈*𝑁* ∑*𝑗*∈*𝑁 𝑑ij 𝑥ij*

*𝑎*

*𝑎*

and (BP2) can be calculated fast with a suitable ordering of the jobs, we tested their application in the branch-and-bound methods. The per-

s*.*t*.*

*𝑗*∈*𝑁𝑎* ⧵{*𝑖*} *𝑥ij* = 1 *𝑖* ∈ *𝑁𝑎*

*𝑗*∈*𝑁𝑎* ⧵{*𝑖*} *𝑥ji* = 1 *𝑖* ∈ *𝑁𝑎*

∑

formance is comparable to that of LB*𝑐* . Altogether, we decide to use

s

only the bound LB*𝑐* in our branch-and-bound methods because its good

*𝑥ij*

∈ {0*,* 1} *𝑖, 𝑗* ∈ *𝑁𝑎*

s The result of (AP) is in general a set of cycles. To exclude cycles of

quality and very short computation time.

length 1, we set *𝑑*

*𝑖𝑖*

= ∞ for each *i* ∈ *Na*. The assignment problem leads

*4.1.4. Application to branch-and-bound strategy Append*

By applying bound LB*𝑐* to the branch-and-bound strategy Append,

s

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*,* … *, 𝑚* are

day *𝜂*. For each day *t* ∈ *T*, we calculate the number of executable jobs empty. Because of that, we adapt the number of jobs executable per

removing subtour elimination constraints. It can be solved in U(*𝑛*3) by to a lower bound for the TSP, because it is derived from the TSP by

means of a good implementation of the Hungarian method, as shown in [Munkres (1957)](#_bookmark58).

[Christofides (1972)](#_bookmark43) provides an improvement of the assignment bound. Normally, the solution of (AP) contains small cycles. He devel- oped an iterative method to add the costs necessary to connect the cy- cles. The algorithm starts with the distance matrix. For this matrix, the

**Table 2**

Comparison of different travel cost bounds.

LP

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | LB*𝑑* | 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* ≥ 95% | 1% | 0% | 1% | 8% | 0% 0% | 7% | 26% | |
| *q* ≥ 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* ≥ 95% | 3% | 0% | 3% | 24% | 0% 1% | 45% | 91% | |
| *q* ≥ 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 result- ing 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.

* + 1. *Minimal spanning tree based lower bound*

In the paper of [Held and Karp (1971)](#_bookmark51), a lower bound for the travel

base for the bound LB*𝑑*MST . Therefore, we need a symmetric distance costs of TSP is presented that uses minimal spanning trees, which is the

customer costs for a part of the jobs from Section [4.1.3](#_bookmark29); secondly, 100 small instances without customer costs, which are generated from the

previous set of instances by set *𝑐𝑖* = 0 for each job *i* ∈ *N*; and thirdly, the

25 large instances with 75 up to 100 jobs from Section [4.1.3](#_bookmark29).

As before, we compare the quality *𝑞* ∶= bound value in percent, where *gd*(*S*∗) is the travel cost value of an optimal solution, or the distance to the best bound Δ ∶= 1 − bound value ; and the computational

*𝑔𝑑* (*𝑆*∗)

max obtained bound value

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.

costs of the LP-relaxation of (MILP) LB*𝑑* , the 2-neighbor bound LB*𝑑*2N *,* In [Table 2](#_bookmark36), the different travel cost bounds are compared: the travel 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.

LP

matrix. To ensure that, we set *𝑑̃*

*𝑖𝑗*

= min{*𝑑*

*𝑖𝑗*

*, 𝑑*

*𝑗𝑖*}.

The LP-relaxation leads to an acceptable lower bound for the travel

Let *C*∗ be the minimal costs of a TSP and *W*∗ be the weight of a

set {2*,* 3*,* … *, 𝑛𝑎*} and two distinct edges of minimal weight incident to minimum-weight 1-tree, which is a minimal spanning tree with vertex 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)](#_bookmark59) or [Kruskal (1956)](#_bookmark55). Then, using a suitable implementation and data structure, the complexity of the al-

gorithms can be stated as U(*𝑛*2) and U(*𝑛*2 log *𝑛*)*,* respectively, because we

have U(*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

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 com-

putation time for LB*𝑑* is much higher than for the other bounds. The

LP

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 av- erage 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 millisec-

is done by setting the edge weight to *𝑤𝑖𝑗* = *𝑑̃*

+ *𝜋* + *𝜋* where *𝜋* and *𝜋*

ond in the small instances and in average 10 ms in the large instances.

*𝑖𝑗*

*𝑖 𝑗 i j*

are the penalties of vertices *i* and *j*. Let *W*∗(*𝜋*) be the minimal weight of an 1-tree with penalty vector *𝜋*, then we have *𝐶*∗ ≥ *𝑊* ∗(*𝜋*) − 2 ∑*𝑖*∈*𝑁 𝜋𝑖.*

Thus, the best lower bound is given by max *𝑊* ∗(*𝜋*) − 2 ∑ *𝜋 𝑎*

penalty vector *𝜋* is changed iteratively. Let *di* be the degree of vertex

*𝜋*

*𝑖*∈*𝑁𝑎*

*𝑖*. The

*i* in the last calculated 1-tree, then the penalty for the next iteration

is set to *𝜋𝑚*+1 = *𝜋𝑚* + *𝑡𝑑𝑖* where *t* is an appropriately chosen step size.

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 min-

imum spanning trees are comparable to the assignment bounds. The

*𝑖 𝑖*

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.

* + 1. *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

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

ilar to the quality of LB*𝑑*APC ; and with 1000 iterations, the quality is the 2-neighbor bound. With 100 iterations, the bound quality is sim-

bound LB*𝑑*MST with 1000 iterations is very close to the optimal costs. further improved. In the instances dominated by the travel costs, the

In the large instances, the bounds based on minimum spanning trees

25 large instances, LB*𝑑*MST with 1000 iterations leads to the best lower are in average a bit better than the assignment bounds. In 24 of the

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 × 108 | 2.4 × 108 | 7.4 × 107 | 9 × 107 | 3.8 × 107 | 1.5 × 107 |  |
| 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 × 109 | 2.9 × 108 | 2.4 × 108 | 4.1 × 108 | 1.2 × 108 | 6.5 × 107 |  |
| 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 bet- ter performance of the assignment bounds because they consider the asymmetries in distance matrices that occur in the solution process.

* + 1. *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, *jl* 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 *jl* that works as start depot of the route *l*,
      * the start depots of the routes *𝑙* + 1*,* … *, 𝑚,* and
      * the end depots of the routes *𝑙, 𝑙* + 1*,* … *, 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](#_bookmark35), we set

the travel costs between the depots z*k* and s*𝑘*+1*, 𝑘* = *𝑙, 𝑙* + 1*,* … *, 𝑚* − 1*,* as

well as z*m* and *jl* to zero. The other travel costs between depots including

*jl* 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.

* + 1. *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* + 2*𝑚*. Let *𝑁𝑝* ∶=

* + 1. *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](#_bookmark37), 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 us- age 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 algo- rithm. In the strategy Include, the assignment bound leads to a reduc- tion 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 In- clude which are taken into account in the assignment bound. In both

⋃*𝑘*∈*𝑀*

*𝑁̃𝑘*

be the set of the already planned jobs. Between these jobs,

strategies, the improvement of the assignment bound by Christofides

leads to a better performance of the branch-and-bound, and the effect is

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 *dii* 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:

∞*,* if *𝑖* = *𝑗* ∈ *𝑁𝑎*

⎧⎪

*𝑑ij ,* if *𝑖* ∈ *𝑁 𝑝* ∪ *𝑁𝑠, 𝑗* ∈ *𝑁 𝑝* ∪ *𝑁𝑧* and *𝑖* is the predecessor of *𝑗*

∞*,* if *𝑖* ∈ *𝑁 𝑝* ∪ *𝑁 , 𝑗* ∈ *𝑁 𝑝* ∪ *𝑁* and *𝑖* is not the predecessor of *𝑗*

⎪ *𝑝 𝑠 𝑧 ̃*

*𝑑ij ,* if *𝑖* ∈ *𝑁* or *𝑖* ∈ *𝑁𝑠* and *𝑗* ∈ *𝑁*

⎪

*ij ij 𝑧*

*𝑑̃* =

*𝑑 ,* if *𝑖* ∈ *𝑁̃* and *𝑗* ∈ *𝑁 𝑝* ∪ *𝑁*

⎨⎪ *̃*

*𝑑ij ,* if *𝑖, 𝑗* ∈ *𝑁, 𝑖* ≠ *𝑗*

∞*,* if *𝑖* ∈ *𝑁𝑠, 𝑗* ∈ *𝑁𝑧*

⎪

0*,* if *𝑖* = z*𝑘, 𝑗* = s*𝑘*+1 (*𝑚𝑜𝑑 𝑚*) and *𝑘* ∈ *𝑀*

∞ if *𝑖* = z*𝑘, 𝑗* = s*𝑙, 𝑘, 𝑙* ∈ *𝑀* and *𝑙* ≠ *𝑘* + 1 (*𝑚𝑜𝑑 𝑚*)

⎩

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 prob- lem 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.

much higher in the strategy Append. There, the number of analyzed par- tial 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 im-

prove the computation time even though the number of analyzed par- tial 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 en-

tries 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 solu- tions 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.

# 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 us-

**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 *rc* of jobs with non-zero customer cost coeﬃcient

*rc <* 25% 17 0.07 0.04 0.41

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ing best-first search. We parallelize the solution process by creating all | 25% ≤ *rc <* 50% | 29 | 11.17 | 2.25 | 0.61 |
| partial solutions up to a given depth. The partial solutions with the best | 50% ≤ *rc <* 75% | 33 | 23.49 | 0.42 | 0.53 |
| lower bounds are analyzed first in hope to improve the upper bound | 75% ≤ *rc* | 21 | 28.76 | 4.66 | 13.86 |
| faster and to obtain a better solution in case of premature break caused Grouped by number *n*  *n* ∈ {10, 11, 12} 29 3.03 0.01 0.01 | | | | | |
| by reaching the time limit. At the beginning, an upper bound is calcu- | *n* ∈ {13, 14, 15} | 30 | 12.19 | 0.04 | 0.61 |
| lated using a greedy heuristic ([Heinicke et al., 2013](#_bookmark48)). We do the test | *n* ∈ {16, 17, 18, 19} | 41 | 30.50 | 4.30 | 7.67 |

on a PC with an Intel○R 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

Grouped by increase in travel costs caused by considering customer costs

Larger than 25% 37 39.76 2.45 3.27

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Less than 5% | 32 | 0.07 | 0.06 | 0.68 |
| Between 5% and 25% | 31 | 7.45 | 2.75 | 6.15 |

a depth of *𝑑* = 3: therewith, in case of *n* ≥ 4, *m* ≥ 3, we have 4 *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 computa- tion 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 cal- culating a lower bound for the customer cost part. Thus, the more jobs with customer costs are fixed, the better the lower bounds for the cus- tomer 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 *𝑑* (*𝑚* + *𝑗* − 1). In experiments, we found out that we should

*𝑗*=1

have more than 500 threads for a good parallelization. We increase *d*

ample, in case of 15 jobs and 3 routes, we set *𝑑* = 5 and obtain 2520 until the number of defined branches will be larger than 500. For ex-

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)](#_bookmark49), it was found out that (MILP) could be faster solved with CPLEX, if some addi- tional 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)](#_bookmark49). 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 con- sidering 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)](#_bookmark49). 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)](#_bookmark49).

100 small benchmark instances with non-zero customer cost coeﬃcient for a part of the jobs, see Section [4.1.3](#_bookmark29). We compare the average compu- tation 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 ap- proaches, 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 ef- fort. 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 in- stance 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 vari- ants 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.

In [Table 5](#_bookmark38), 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* ∶= 1 {*𝑗* ∈ *𝑁 𝑐 >* 0} ⋅ 100%. Each group contains between 17 and 33

| | |

In [Table 4](#_bookmark40), we show the performance of the two branch-and-bound algorithms and of solving (MILP) with CPLEX. For the tests, we use the

**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  Analyzed partial solutions | 73%  8.8 × 1012 | 98% 98%  7.4 × 107 2.35 × 108 |  |
| Fastest solved | 27% | 31% 52% |  |

*𝑐 𝑛 𝑗*

instances. This grouping shows that solving instances with higher cus-

tomer cost ration *rc* 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 *rc <* 50%

are in average faster solved using strategy Include, but for instances with *rc* ≥ 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 cus- tomer costs. Therefore, we solve the instance with objective function

*𝑔*(*𝑆*) = *𝑔𝑑* (*𝑆*) + *𝑔𝑐* (*𝑆*)*,* with the travel costs part *gd*(*S*), and call the solu-

tion *S*∗. Additionally we solve the problem with *𝑔̃*(*𝑆*) = *𝑔𝑑* (*𝑆*) and obtain the solution *Sd*. The increase in travel costs is then given by *𝑔𝑑* (*𝑆* ∗ ) − 1.

*𝑔𝑑* (*𝑆 𝑑* )

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 han- dle 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](#_bookmark40). 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.

# 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 bench- mark instances and compared the performance of both with the appli- cation of the commercial solver CPLEX to a linearization of the prob- lem. We found out that the branch-and-bound approaches could better handle the time-dependent customer costs. Both branch-and-bound al- gorithms solved these instances with significantly less computation time than the commercial solver.

There are still some points for further research. A more detailed anal- ysis about the influence of the instance structure to the performance of the different approaches is needed to select the most suitable solu-

tion 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 re- search 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,](http://refhub.elsevier.com/S2192-4406(20)30003-4/sbref0001) [E.K., 1983. An exact algorithm for the time-constrained traveling salesman prob- lem. Oper. Res. 31 (5), 938–945.](http://refhub.elsevier.com/S2192-4406(20)30003-4/sbref0001)

[Balas, E., 1977. Branch and bound/implicit enumeration. Ann. Discret. Math. 5, 185–191](http://refhub.elsevier.com/S2192-4406(20)30003-4/sbref0002).

[Baldi,](http://refhub.elsevier.com/S2192-4406(20)30003-4/sbref0003) [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.](http://refhub.elsevier.com/S2192-4406(20)30003-4/sbref0003)

[Christofides,](http://refhub.elsevier.com/S2192-4406(20)30003-4/sbref0005) [N., 1972. Bounds for the travelling-salesman problem. Oper. Res. 20 (5), 1044–1056.](http://refhub.elsevier.com/S2192-4406(20)30003-4/sbref0005)

[Ebstein,](http://refhub.elsevier.com/S2192-4406(20)30003-4/sbref0007) [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.](http://refhub.elsevier.com/S2192-4406(20)30003-4/sbref0007)

[Fisher,](http://refhub.elsevier.com/S2192-4406(20)30003-4/sbref0009) [M.L., Jörnsten, K.O., Madsen, O.B.G., 1997. Vehicle routing with time windows: two optimization algorithms. Oper. Res. 45 (3), 488–492.](http://refhub.elsevier.com/S2192-4406(20)30003-4/sbref0009)

[Gavish,](http://refhub.elsevier.com/S2192-4406(20)30003-4/sbref0010) [B., Srikanth, K., 1986. An optimal solution method for large-scale multiple trav- eling salesmen problems. Oper. Res. 34 (5), 698–717.](http://refhub.elsevier.com/S2192-4406(20)30003-4/sbref0010)

[Heinicke,](http://refhub.elsevier.com/S2192-4406(20)30003-4/sbref0012) [F., Simroth, A., Scheithauer, G., Fischer, A., 2015. A railway maintenance scheduling problem with customer costs. EURO J. Transp. Logist. 4 (1), 113–137.](http://refhub.elsevier.com/S2192-4406(20)30003-4/sbref0012)

[Heinicke,](http://refhub.elsevier.com/S2192-4406(20)30003-4/sbref0013) [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).](http://refhub.elsevier.com/S2192-4406(20)30003-4/sbref0013) [Pp 304–309.](http://refhub.elsevier.com/S2192-4406(20)30003-4/sbref0013)

[Held,](http://refhub.elsevier.com/S2192-4406(20)30003-4/sbref0014) [M., Karp, R.M., 1971. The traveling-salesman problem and minimum spanning trees: Part II. Math. Program. 1 (1), 6–25.](http://refhub.elsevier.com/S2192-4406(20)30003-4/sbref0014)

[Kohl,](http://refhub.elsevier.com/S2192-4406(20)30003-4/sbref0015) [N., 1995. Exact Methods for the Time Constrained Routing and Related Scheduling Problems. University of Denmark.](http://refhub.elsevier.com/S2192-4406(20)30003-4/sbref0015)

[Kruskal,](http://refhub.elsevier.com/S2192-4406(20)30003-4/sbref0016) [J.B., 1956. On the shortest spanning subtree of a graph and the traveling salesman problem. Proc. Am. Math. Soc. 7 (1), 48–50.](http://refhub.elsevier.com/S2192-4406(20)30003-4/sbref0016)

[Laporte,](http://refhub.elsevier.com/S2192-4406(20)30003-4/sbref0017) [G., Nobert, Y., Taillefer, S., 1988. Solving a family of multi-depot vehicle routing and location-routing problems. Transp. Sci. 22 (3), 161–172.](http://refhub.elsevier.com/S2192-4406(20)30003-4/sbref0017)

[Lucena,](http://refhub.elsevier.com/S2192-4406(20)30003-4/sbref0018) [A., 1990. Time-dependent traveling salesman problem – the delivery man case.](http://refhub.elsevier.com/S2192-4406(20)30003-4/sbref0018)

[Network 20, 753–763.](http://refhub.elsevier.com/S2192-4406(20)30003-4/sbref0018)

[Munkres,](http://refhub.elsevier.com/S2192-4406(20)30003-4/sbref0019) [J., 1957. Algorithms for the assignment and transportation problems. J. Soc.](http://refhub.elsevier.com/S2192-4406(20)30003-4/sbref0019)

[Ind. Appl. Math. 5 (1), 32–38.](http://refhub.elsevier.com/S2192-4406(20)30003-4/sbref0019)

[Prim,](http://refhub.elsevier.com/S2192-4406(20)30003-4/sbref0020) [R.C., 1957. Shortest connection networks and some generalizations. Bell Syst. Tech.](http://refhub.elsevier.com/S2192-4406(20)30003-4/sbref0020)

[J. 36 (6), 1389–1401.](http://refhub.elsevier.com/S2192-4406(20)30003-4/sbref0020)

[Reinelt,](http://refhub.elsevier.com/S2192-4406(20)30003-4/sbref0021) [G., 1994. The Traveling Salesman: Computational Solutions for TSP Application.](http://refhub.elsevier.com/S2192-4406(20)30003-4/sbref0021)

[Springer-Verlag, Berlin Heidelberg.](http://refhub.elsevier.com/S2192-4406(20)30003-4/sbref0021)

[Scheithauer,](http://refhub.elsevier.com/S2192-4406(20)30003-4/sbref0022) [G., 2018. Introduction to Cutting and Packing Optimization. Springer Inter- national Publishing.](http://refhub.elsevier.com/S2192-4406(20)30003-4/sbref0022)