WORKING PAPER SERIES

A hybrid solution approach for the 3L-VRP with simultaneous delivery and pickups

Henriette Koch/Andreas Bortfeldt/Gerhard Wäscher

Working Paper No. 5/2017

OTTO VON GUERICKE
UNIVERSITÄT
MAGDEBURG

FACULTY OF ECONOMICS
AND MANAGEMENT

Impressum (§ 5 TMG)

Herausgeber:
Otto-von-Guericke-Universität Magdeburg
Fakultät für Wirtschaftswissenschaft
Der Dekan

Verantwortlich für diese Ausgabe:
H. Koch, A. Bortfeldt, G. Wäscher
Otto-von-Guericke-Universität Magdeburg
Fakultät für Wirtschaftswissenschaft
Postfach 4120
39016 Magdeburg
Germany

http://www.fww.ovgu.de/femm

Bezug über den Herausgeber
ISSN 1615-4274

A hybrid solution approach for the 3L-VRP with simultaneous delivery and pickups

Henriette Koch* Andreas Bortfeldt* Gerhard Wäscher*†

Abstract

This paper deals with a special vehicle routing problem with backhauls where each customer receives items from a depot and, at the same time, returns items back to the depot. Moreover, time windows are assumed and three-dimensional loading constraints are to be observed, i.e. the items are three-dimensional boxes and packing constraints, e.g. regarding load stability, are to be met. The resulting problem is the vehicle routing problem with simultaneous delivery and pickup (VRPSDP), time windows, and three-dimensional loading constraints (3L-VRPSDPTW). This problem occurs, for example, if retail stores are supplied by a central warehouse and wish to return packaging material.

A particular challenge of the problem is to transport delivery and pickup items simultaneously on the same vehicle. In order to avoid any reloading effort during a tour, we consider two different loading approaches of vehicles: (i) loading from the back side with separation of the loading space into a delivery section and a pickup section and (ii) loading at the long side.

A hybrid algorithm is proposed for the 3L-VRPSDPTW consisting of an adaptive large neighbourhood search for the routing and different packing heuristics for the loading part of the problem. Extensive numerical experiments are conducted with VRPSDP instances from the literature and newly generated instances for the 3L-VRPSDPTW.

Keywords: vehicle routing, backhauls, three-dimensional loading constraints, large neighbourhood search

*Department of Management Science, Otto-von-Guericke-University Magdeburg, 39106 Magdeburg, Germany

†School of Mechanical, Electronic and Control Engineering, Beijing Jiaotong University, 100044 Beijing, China

1 Introduction

Retail stores usually receive their supplies from a central warehouse. Trucks start from the warehouse and deliver the requested articles to several outlets before they return to the warehouse again. This gives rise to various vehicle routing problems (VRPs), for example, variants which include backhauling, i.e. the return transportation of goods from the customers to the depot. This allows for organizing transportation more economically and more environmentally friendly. By increasing the capacity utilization of the trucks in use, it reduces the number of trips and the respective travel distances which are necessary to satisfy all transportation requests.

Whereas different variants of backhauling can be distinguished (cf. e.g. [35, 22]), we consider a specific one called VRP with simultaneous delivery and pickup (VRPSDP) where each client is assigned with linehaul (delivery) *and* backhaul (pickup) demands and is serviced by exactly one vehicle. This situation occurs, for example, when fresh-food supermarkets are provided with articles in cooling boxes or in the clothing sector where clothes are delivered to the shops in cardboard boxes. In both cases, the boxes have to be returned to the warehouse. Further applications include the delivery of household appliances and furniture and the pickup of bulky waste like defective electronic devices or damaged furniture. The transportation of automotive parts constitutes another example. Suppliers deliver components to original equipment manufacturers and pickup finished products from them in return.

In order to provide a more realistic modelling of the problem, we consider time windows for the customers and the depot. Furthermore, since the transported goods and/or the respective packaging materials are usually of a size that cannot be neglected when the trucks are being loaded, the goods are explicitly assumed to be three-dimensional (cuboid) items. That way, their spatial dimensions are taken into account in order to generate feasible loading plans that consider practically relevant packing constraints and can guarantee the feasibility of routes. A particular challenge of the problem is to transport delivery and pickup items simultaneously on the same vehicle. In order to avoid any reloading effort during a tour, we consider two different loading approaches of vehicles: (i) loading from the back side with separation of the loading space into a delivery section and a pickup section and (ii) loading at the long side. The resulting problem belongs to the group of vehicle routing problems with three-dimensional loading constraints (3L-VRPs) which

was introduced in [17].

We propose a hybrid algorithm for solving the three-dimensional VRPSDP. It consists of an adaptive large neighbourhood search (ALNS) which is based on an approach presented in [39] and which is extended by various operators. The ALNS is combined with conventional packing heuristics to ensure the feasibility of the obtained solutions with respect to the packing subproblem.

The remainder of this paper is organized as follows: In Section 2, the considered problem is described in detail. The relevant literature is discussed in Section 3. Followingly, the ALNS approach and the used packing heuristics are presented in Section 4. Section 5 contains the description, results and analysis of the numerical experiments. Finally, in Section 6, the paper concludes with a summary and an outlook to future research.

2 Problem description

Let $G = (N, E)$ be a weighted, undirected graph consisting of a node set $N = \{0, 1, \ldots, n\}$ which represents the depot ($\{0\}$), i.e. the warehouse, and n customers ($\{1, \ldots, n\}$), and an edge set $E = \{(i, j) : i, j \in N\}$. A cost c_{ij} ($c_{ij} \geq 0$) is assigned to each edge $(i, j) \in E$ ($i \neq j$). Moreover, time windows are considered. That is, a ready time RT_i, a due date DD_i, and a service time ST_i are assigned to each location i ($i \in N$). If a vehicle arrives at a customer location before the customer's ready time, the service cannot start until the ready time, i.e. the vehicle has to wait. On the other hand, a vehicle must not arrive after the due date of a customer or the depot, respectively. Each customer $i \in N \setminus \{0\}$ demands a set $I_i^L = \{1, \ldots, m_i^L\}$ of m_i^L linehaul items from the depot and returns a set $I_i^B = \{m_i^L + 1, \ldots, m_i^L + m_i^B\}$ of m_i^B backhaul items to the depot. Each item I_{ik} ($i = 1, \ldots, n; k = 1, \ldots, m_i^L + m_i^B$) has a known length l_{ik}, width w_{ik}, height h_{ik} and weight d_{ik}, and is assigned with a fragility flag f_{ik} indicating whether it is fragile ($f_{ik} = 1$) or not ($f_{ik} = 0$). A fleet of v_{max} identical vehicles is available. Each vehicle can carry a maximum weight D and has a loading space with a given length L, width W and height H.

A feasible packing plan P comprises information about the placement for one or more items and fulfils the following conditions: (P1) all items are placed entirely within the loading space, (P2) any two items that are simultaneously in one vehicle must not overlap, (P3) all items must be placed orthogonally to the loading space edges. Moreover, the

following additional packing constraints must be adhered to:

(PC1) **Fixed vertical orientation:** Each item has a fixed vertical orientation, i.e. the height dimension is fixed. The items can be turned by 90° on the horizontal plane, though.

(PC2) **Vertical stability:** Each item must be supported by a given percentage α by the top face of other items or the container floor.

(PC3) **Fragility:** The items are divided into fragile and non-fragile items. Whereas fragile items can be placed on top of any other item, non-fragile items can only be placed on top of other non-fragile items.

(PC4) **LIFO:** In order to load and unload the items solely by straight movements towards the door, the items must be arranged in a way that the loading and unloading at a certain station is not blocked by items that are to be delivered later or have already been picked up. That is, linehaul and backhaul items must not be placed in front or on top of each other. Moreover, a linehaul item that is delivered later must not be placed in front or on top of a linehaul item that is delivered earlier, and analogously for backhaul items.

It is further assumed that *any* reloading during the tour is forbidden. Therefore, the LIFO constraint is particularly challenging when a backhaul variant is considered where linehaul and backhaul items are transported simultaneously. Two different loading approaches are used here which allow for arranging the items in a way that reloading can be avoided. Firstly, double-decker vehicles are considered. These vehicles are rear-loaded and their loading space can be separated horizontally, so that linehaul and backhaul items can be transported in separate compartments. It is assumed here that both compartments are of identical size. Hence, a mixture of linehaul and backhaul items does not need be considered with respect to the LIFO policy. In the second variant, so-called tautliners are used. This kind of vehicle is not loaded from the rear but can be loaded (and unloaded) from one long side. The principle is depicted in Fig. 1. By loading linehaul (light grey) and backhaul (dark) items from the opposing sites of the loading space, the unloading of linehaul items generates spaces for backhaul items. At a given stage of a tour, L_{LH} represents the loading length (i.e. the maximum front edge) of all linehaul items currently in the vehicle. Analogously, L_{BH} represents the length needed for the backhaul items. The

sum of both lengths must not exceed the length of the loading space L in order to avoid the overlapping of any two items during the tour. Furthermore, the LIFO constraint is adjusted for this loading approach. It must not only be considered along the length but also along the width axis. In the example in Fig. 1, item 4 cannot be delivered later than item 1 (LIFO along length axis). Although the items are not unloaded along the length axis in this case, this aspect of the LIFO constraints ensures that the unloading of linehaul items successively creates space for backhaul items. Moreover, items 2 and 3 must not be delivered after item 1 either, because they would block the unloading of item 1 (LIFO along width axis).

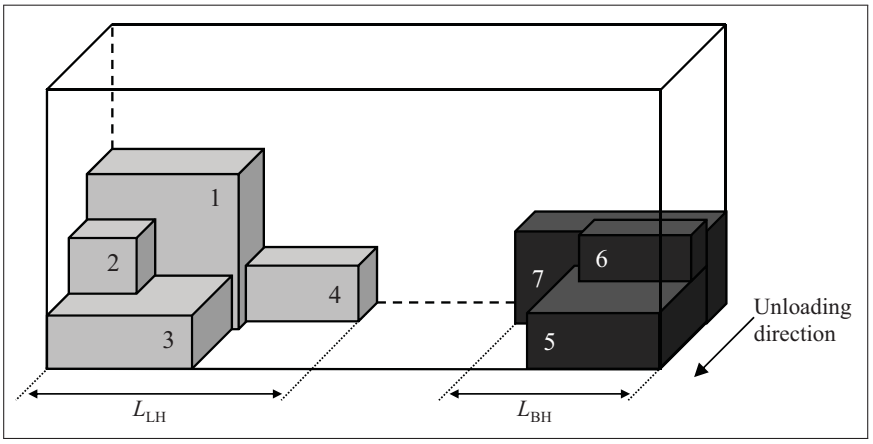

Figure 1: Side loading

Furthermore, a route R is defined as a sequence of locations $(0, i_1, \ldots, i_{n_r}, 0)$ which is feasible if (R1) it starts and ends at the depot, (R2) it contains each customer $i \in R \setminus \{0\}$ exactly once, (R3) the vehicle does not arrive after the due date DD_i of any location $i \in R$ and (R4) the sum of the weights of all items transported simultaneously does not exceed the vehicle capacity D. Each route R must be provided with two packing plans: one for all linehaul items at the beginning of the tour (P_L) and one for all backhaul items at the end of the tour (P_B). It is assumed that each vehicle travels only one tour. Let v be the number of used vehicles in a solution. Thus, a solution consists of a set of v triples $(R_t, P_{t,L}, P_{t,B})$ containing a route R_t for each vehicle t ($t = 1, \ldots, v$) and the corresponding packing plans $P_{t,L}$ and $P_{t,B}$. A solution is feasible if (S1) all routes R_t and packing plans $P_{t,L}, P_{t,B}$ ($t = 1, \ldots, v$) are feasible, (S2) each packing plan $P_{t,L}, P_{t,B}$ contains all of the respective linehaul or backhaul items (and no other) of all customers serviced in R_t ($t = 1, \ldots, v$), (S3) each customer $i \in N \setminus \{0\}$ is assigned to exactly one route, (S4) the number of used vehicles v does not exceed the number of available vehicles

v_{max}. Moreover, a feasible solution for the problem under application of the side loading approach must also fulfil the condition (S5) that the linehaul and backhaul items that are transported simultaneously on a route R_t ($t = 1, \ldots, v$) do not overlap.

A feasible solution is to be found that minimizes the total travel distance (TTD). The problem can be classified as a 3L-VRP with simultaneous delivery and pickup and time windows (3L-VRPSDPTW). By replacing the sets of three-dimensional items by one-dimensional demands, and the loading space of a vehicle by a one-dimensional capacity and by omitting the packing constraints, the 3L-VRPSDPTW will be reduced to the one-dimensional VRPSDPTW and to the VRPSDP if time windows are not considered either.

3 Literature review

In the following section, an overview of the relevant literature is given. Since there is – to the best of our knowledge – no scientific publication yet dealing with the 3L-VRPSDPTW the focus will be on the (one-dimensional) VRPSDP and variants of the 3L-VRP as a group of related problems.

3.1 Vehicle routing problem with simultaneous delivery and pickup

Initially, classical heuristic methods were proposed in the first place for solving the VRPSDP. E.g., cluster-first-route-second approaches were presented in [32] and [19]. Insertion heuristics were proposed in [41] and [12]. Later, the focus shifted towards metaheuristic solution approaches. A hybrid procedure consisting of tabu search (TS) and variable neighbourhood descent is proposed in [9]. Another TS algorithm was presented in [48]. ALNS for a large variety of vehicle routing problems, among them the VRPSDP, was proposed in [39]. For a detailed review of the literature regarding the VRPSDP until 2008 we refer to [35].

More recently, various metaheuristics were developed for the problem. Reactive TS algorithms in which the length of the tabu list is adapted dynamically during the search process were proposed in [56, 57]. Hybrid algorithms were presented in [60] (TS and guided local search), in [45] (variable neighbourhood search (VNS) and iterated local

search), and in [2] (local search (LS) and TS). Further approaches included, among others, ant colony optimization (ACO) [7], scatter search [30, 62], genetic algorithm (GA) [51, 62], evolutionary algorithm [15], or VNS [36].

In addition, approaches for the VRPSDP with time windows have been designed as well. [13] proposed a TS algorithm for identifying solutions that minimize the total travel distance and maximize customer satisfaction. [54] and [53] aimed at minimizing the number of used vehicles and the total travel distance by applying a GA and a simulated annealing approach, respectively. [63] tackled the problem with a hybrid algorithm integrating ACO and TS.

Moreover, several exact methods have been proposed as well for the VRPSDP. They include, for example, branch-and-price approaches [1, 10], branch-and-cut [46], or branch-and-cut-and-price [47]. Furthermore, [38] proposed an efficient MILP formulation with valid inequalities which is solved by using a standard LP/IP code (CPLEX).

3.2 Vehicle routing problem with three-dimensional loading constraints

The capacitated vehicle routing problem with three-dimensional loading constraints (3L-CVRP) including the above mentioned packing constraints was first presented in [17]. Subsequently, the problem was studied by various researchers (e.g. [50, 14, 5]). In addition, also variants of the problem have been studied. For example, the 3L-CVRP with time windows and the minimization of the number of vehicles were considered in [33] and [34]. The 3L-CVRP with a heterogeneous vehicle fleet was studied in [58].

Usually, a metaheuristic approach is applied to the routing subproblem, e.g., GA [33, 31], TS [17, 50, 55, 28, 59, 65, 49], or ACO [14]. On the other hand, the packing problem is usually tackled by rather simple heuristics because solving the packing problem is, in general, computationally expensive. The application of such heuristics, which are often based on deepest-bottom-left or touching area approaches, may result in low-quality solutions, though. More complex packing heuristics, e.g. those proposed in [64] (LS-based approach) or in [5] (tree search), have been applied as well and provided significant improvements of the best known solutions of benchmark instances.

As mentioned above, the 3L-VRPSDP has not been studied in the literature, yet. Only few variants of the VRP with backhauls have been considered in combination with three-

dimensional loading constraints so far. A 3L-VRP with clustered backhauls was dealt with in [6]. In this problem, the customers are either pure linehaul or pure backhaul customers and within one tour the linehaul customers must be visited before the backhaul customers. The authors applied an ALNS and a VNS to the routing and a tree search procedure to the packing problem. Moreover, the pickup and delivery problem with three-dimensional loading constraints was presented and solved in [4] and [29]. In this problem variant, goods are not transported between the depot and the customers, but from loading to unloading locations. A detailed overview of the developments in the literature regarding the 3L-VRP is provided in [37].

4 A hybrid solution approach

Both the VRP and the packing problem are NP-hard optimization problems. The 3L-VRPSDPTW, as a combination of both problems, is even more challenging and very difficult to solve (cf. e.g. [21]). Therefore, we propose a hybrid solution approach to the 3L-VRPSDPTW, consisting of a metaheuristic – more precisely an ALNS algorithm – for the routing subproblem and conventional packing heuristics for the packing problem.

4.1 Adaptive large neighbourhood search

Our ALNS is based on an approach presented in [39, 40]. The large neighbourhood search was originally proposed in [42, 43] for the CVRP and the VRPTW. The general framework is depicted in Fig. 2. In the first step of each iteration of the algorithm, customers are removed from the current solution by a removal heuristic. Then, these customers are reinserted by making use of an insertion heuristic. Different removal and insertion heuristics are available. In each iteration, the applied heuristics are selected randomly based on a roulette wheel selection principle and the number of customers to be removed is determined randomly within a given interval. The acceptance check of a new solution (line 8) is embedded in a simulated annealing framework. In the following, the different components of the algorithm are described in detail.

Initial solution

The savings heuristic [8] is used for the generation of the initial solution. This heuristic, however, does not include any means for controlling the number of tours. Thus, its

```
1: procedure ALNS(in: instance data, parameters, out: best solution s_best)
2:     construct initial solution s_init
3:     s := s_best := s_init
4:     while stopping criterion is not met do
5:         select number of customers to be removed n_rem ∈ [no_min, no_max]
6:         select removal heuristic rem_heur and insertion heuristic ins_heur
7:         determine next solution s_next := ins_heur(rem_heur(s, n_rem))
8:         check acceptance of s_next
9:         if s_next is accepted then
10:            s := s_next
11:            if f(s) < f(s_best) then s_best := s end if
12:        end if
13:        if segment end is reached then ▷ segment: pre-defined number of iterations
14:            update weights of insertion and removal heuristics
15:        end if
16:    end while
17: end procedure
```

Figure 2: Adaptive large neighbourhood search

application may result in more than v_{max} tours.

Penalization of the objective function value

In order to guide the search towards feasible solutions, penalty terms are applied. These terms aim (i) at reducing the number of used vehicles to at most v_{max} vehicles and (ii) at including all customers in the solution. Compliance with the constraints described in Section 2 is, in general, ensured within the algorithm. However, due to the initial solution procedure, solutions with more than v_{max} vehicles may occur and reinserting removed customers into the solution again may not be possible. The objective function value f of a solution s can then be determined as: $f(s) = f^*(s) + pen_v \cdot \max(0, v_{used} - v_{max}) + pen_{mc} \cdot n_{mc}$. Here, $f^*(s)$ provides the actual objective function value of s (total travel distance), pen_v and pen_{mc} represent the penalty terms, v_{used} and n_{mc} indicate the number of vehicles used in the solution and the number of missing customers, respectively.

Removal heuristics

Whereas only one removal heuristic was used in [42, 43], three different heuristics were used in [40] and five in [39]. Further removal heuristics were, e.g., presented in [11]. [39] observed that the implementation of a larger number of heuristics results in better solutions. Therefore, we decided to equip the algorithm with a large number of heuristics

while the adaptive mechanism of the ALNS provides the selection of the best performing heuristics. In total, 21 removal heuristics have been implemented which are described in Table 1. As can be seen, most of them originate from the literature, to which we refer for more details. In addition, some new operators have been developed and some have been modified. Below they will be described in greater detail.

Table 1: Removal heuristics

Name	**Description**	**Ref.**
Shaw removal	Removes customers that are related w.r.t. distance, demand, time windows.	[42]
Random removal	Removes random customers.	[40]
Worst removal	Removes customers that deteriorate the solution cost the most.	[40]
Cluster removal	Partitions a tour into two clusters and randomly removes one of the clusters.	[39]
Tour removal	Removes an entire randomly chosen tour.	[6]
Worst distance removal	Removes customers with high distances from their respective predecessor and successor.	[11]
Worst time removal	Removes customers with large differences between their ready time and the start of service.	[11]
Proximity-based Shaw removal	Removes customers that are related w.r.t. distance.	[11]
Time-based Shaw removal	Removes customers that are related w.r.t. time windows.	[11]
Demand-based Shaw removal	Removes customers that are related w.r.t. demand.	[11]
Neighbour graph removal	Is based on historical information regarding the best solution found so far in which a certain customer is visited before another customer.	[39]
Historical knowledge removal	Is based on historical information regarding the best position found so far for each customer.	[11]
Neighbourhood removal	Removes customers which deteriorate the average distance of a tour.	[11]
Node neighbourhood removal	Removes customers close to a randomly chosen customer.	[11]
Overlap removal	Removes customers that lead to the intersection of two tours.	*New*
Inner tour removal	Removes a tour that is completely surrounded by another and splits the surrounding tour into two.	*New*
Intersection removal	Removes customers that lead to intersections within a tour.	*New*
Tour pair removal	Removes two tours that are intersecting.	*New*
Least customer-tour removal	Removes the tour with least number of customers.	*New*
Largest distance-tour removal	Removes the tour with largest total distance.	*New*
Average distance-tour removal	Removes the tour with largest average distance.	*New*

Shaw removal: The Shaw removal heuristic aims at removing "related" customers from a solution. For the definition of the "relatedness" of two customers i and j, their distance (c_{ij}), their ready times (RT_i, RT_j) and their demand volumes are taken into account. Let $netvol_i$ be the net demand volume of customer i, i.e. the difference between its delivery

and pickup demand volume:

$$netvol_i = \sum_{k=1}^{m_i^L}(l_{ik} \cdot w_{ik} \cdot h_{ik}) - \sum_{k=m_i^L+1}^{m_i^L+m_i^B}(l_{ik} \cdot w_{ik} \cdot h_{ik}).$$

Thus, the net demand volume is greater than zero if the volume of the delivery goods is larger than the volume of the pickup goods, less than zero if the pickup volume exceeds the delivery volume, and equals zero if both volumes are equal. We also consider whether the customers are assigned to the same tour. Let $st_{ij} = -1$, if customer i and j are in the same tour, and $st_{ij} = 1$ otherwise [11]. The relatedness measure $relate_{ij}$ for two customers i and j ($i, j \in N \setminus \{0\}$) can then be determined as follows:

$$relate_{ij} = \omega_1 \cdot c_{ij}^* + \omega_2 \cdot |RT_i^* - RT_j^*| + \omega_3 \cdot |netvol_i^* - netvol_j^*| + \omega_4 \cdot st_{ij}.$$

Here, $\omega_1, \omega_2, \omega_3$, and ω_4 represent given weights for the different components. Moreover, the distances, ready times and net demands are normalized in the interval [0,1] which is indicated by the asterisk in the equation above. The smaller the relatedness measure between two customer is, the more related they are. The heuristic randomly selects one customer from the solution and removes another related customer until n_{rem} customers are removed. A determinism parameter p ($p \geq 1$) introduces some randomness so that not necessarily the customer with the highest relatedness value is removed. The higher the value of this parameter is, the smaller is the randomness. See [40] for further details regarding the procedure and the determinism parameter. This parameter is also applied in other removal heuristics that contain sortings of customers.

Overlap removal: This operator aims at removing intersections between tours. Intersections are determined in the following way: For each tour a rectangle is computed based on the minimum and maximum x- and y-coordinates of the customer locations within this tour. If the rectangles of two tours overlap, it is further tested whether two edges of the tours intersect. An example is illustrated in Fig. 3. The gray-shaded area represents the overlapping rectangles. Within this area, the edges (3,4) and (8,9) intersect. Thus, also the respective tours intersect.

First, all intersecting pairs of tours are identified as described above. Then, one pair is randomly selected and customers in the overlap area are removed from the solution.

This procedure is repeated until n_{rem} customers have been removed. If fewer than n_{rem} customers could be determined in that way, further customers are randomly selected and removed.

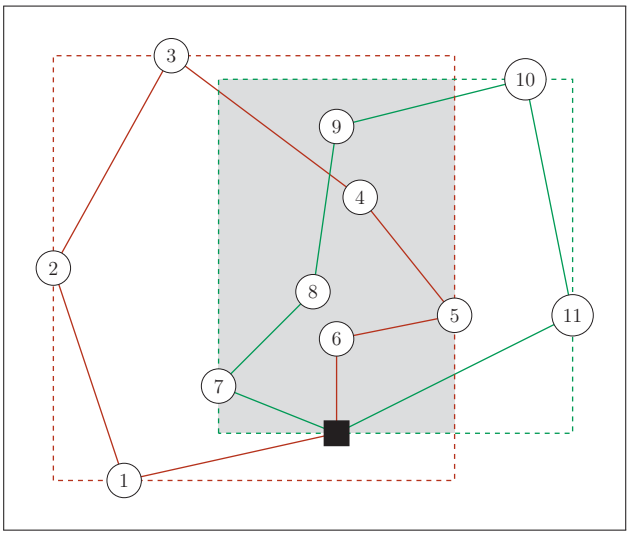

Figure 3: Overlap removal

Inner tour removal: This operator does not only remove customers from a solution, but it also breaks up tours. First, it is checked whether there are tours that are completely surrounded by another tour, as depicted in Fig. 4. This is done by determining the outer rectangle for each tour as above. If one of these rectangles lies completely within another one, the respective tour must be surrounded. The inner tour is then removed from the current solution. The other tour is divided into two tours, each of which receiving half the number of customers if the number of customers n_t in the tour is even. Otherwise, the first $\lceil \frac{n_t}{2} \rceil$ customers and the last $\lfloor \frac{n_t}{2} \rfloor$ customers form the two new tours.

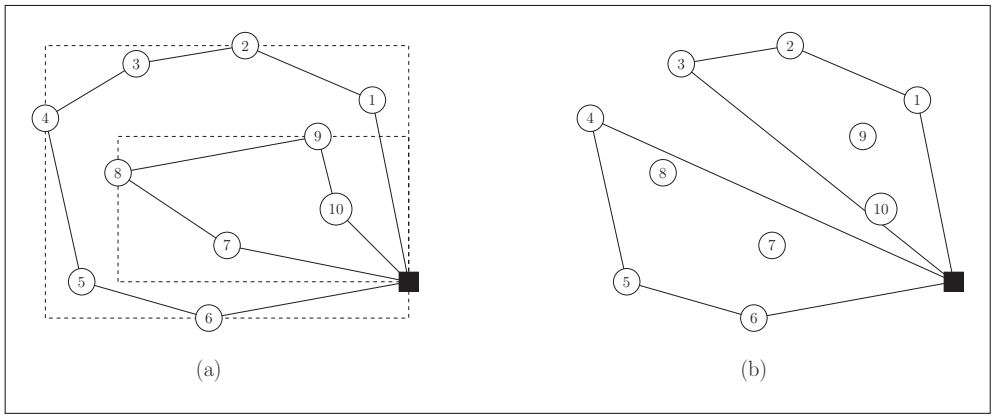

Figure 4: Inner tour removal; a) solution before and b) after Inner tour removal

Intersection removal: The Intersection removal operator checks whether two edges within

a tour intersect. In this case, the four customers (except for the depot) which form these edges are removed. An example is illustrated in Fig. 5. Here, the customers 2, 3, 5, and 6 would be removed.

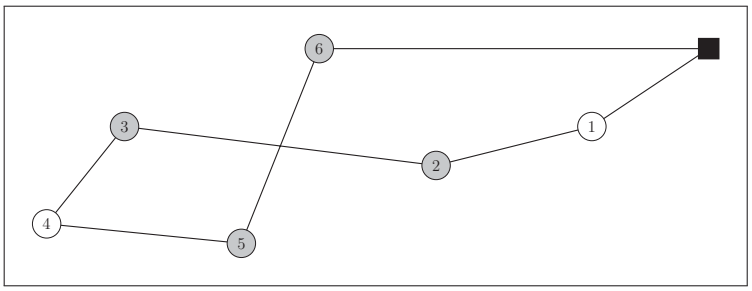

Figure 5: Intersection removal

Tour pair removal: First, intersecting tour pairs are determined as in the Overlap removal heuristic. The operator removes a randomly selected intersecting tour pair from the solution.

Tour removal variants: Unlike using a single random Tour removal operator as the one presented in [6], three additional variants are considered here. In the first one, the tour with the least number of customers is removed. Secondly, the longest tour is removed. Finally, in the third variant, the tour with the largest average distance between the customers in the tour and their corresponding successors is removed.

Insertion heuristics

The insertion heuristics have been adopted from [40]. The first heuristic is a basic Greedy procedure. In each iteration of the insertion process, the insertion costs are determined for each unassigned customer (who is not assigned to any tour in the solution) at every possible position in every tour. The customer who leads to the least increase in the travel distance is inserted into the solution at the respective minimum-cost position. Only feasible insertions are considered, i.e. time window, capacity and packing constraints must not be violated. This procedure is repeated until all customers have been inserted or no remaining unassigned customer can be inserted. Furthermore, two less myopic Regret-k heuristics have been implemented which take the k best insertions into account. Here, the Regret-2 and Regret-3 heuristic are used. In each iteration, the customer with the highest regret value is inserted. The regret value is given by the difference between the cost of

the best and the second best insertion (Regret-2), assumed that these insertions are made in different tours. Analogously, in the case of the Regret-3 heuristic, the regret value is defined by the difference between the cost of the best and the second best insertion plus the difference between the cost of the best and the third best insertion.
In addition, we also consider these heuristics in a randomized way in order to increase the diversification within the search. In these variants, a noise factor is added to the objective function value when calculating the insertion costs. The noise factor is randomly chosen from the interval $[-\eta \cdot c_{max}, \eta \cdot c_{max}]$, where η is a noise parameter and $c_{max} = \max_{i,j \in N} c_{ij}$ (see [40] for further details).
A customer who cannot be inserted into any route, remains on the list of "missing" customers and causes a penalization of the objective function value (see above). Variants with and without noise are looked upon as different heuristics. Hence, six different insertion heuristics have been implemented.

Acceptance check

As mentioned above, a simulated annealing approach is applied. The newly generated solution s_{next} is always accepted if it is better than the current solution s. In that case it is also checked whether the best solution found so far could be improved. If s_{next} is worse than the current solution, it is accepted with the probability $e^{-(f^*(s_{next})-f^*(s))/T}$ where T is the current temperature ($T > 0$). The value of the starting temperature for the first iteration is determined in such a way that a solution that is $w\%$ worse than the initial solution would be accepted with a probability of 0.5. w is a pre-specified parameter for controlling the start temperature (cf. [40]). After each iteration, the temperature is decreased according to a cooling rate γ $(0 < \gamma < 1)$: $T := T \cdot \gamma$.

Heuristic selection and weight adjustment

As in [40], the heuristics are selected based on a roulette wheel selection principle, i.e. their selection probabilities depend on their weights. Initially, all weights are set to 1. For the weight adjustment, a score and a counter are needed for each heuristic. Both are set to 0 at the beginning of each segment which lasts for a given number of iterations. In each iteration in which a heuristic was used, its counter is increased by 1. The score is increased by one of the following values:

- σ_1, if the remove-insert operation led to a new globally best solution,

- σ_2, if the remove-insert operation improved the current solution,
- σ_3, if the remove-insert operation led to a solution that is worse than the current solution but has not been accepted before or is as good as the current solution.

Note that updating of the score is slightly different to the approach in [40]. Unlike in [40], where σ_2 is only added to the score if the new solution improved the current solution and has not been accepted before, improvements of the current solution are always rewarded here. Moreover, the addition of σ_3 for solutions equally good as the current solution represents another modification.
At the end of each segment the weights of the heuristics are recalculated. Let Ω_{hj} be the weight of heuristic h in segment j. Taking into account the current score and counter of the heuristic h, its weight in the following segment $j+1$ is calculated as follows:

$$\Omega_{h,j+1} = \Omega_{hj} \cdot (1-r) + r \cdot \frac{score_h}{count_h}.$$

r is a reaction parameter controlling the impact of the recent performance of the heuristics.

Stopping criteria

The algorithm stops after a given number of iterations $iter_{max}$ or if no further improvement was found after a given number of iterations $iter_{no\_impr}$. In addition, a time limit t_{max} is used because some instance characteristics may lead to high computing times (see below).

4.2 Packing heuristics

Different packing heuristics have been integrated into the ALNS including simple construction heuristics and more complex local search (LS)-based approaches.

Construction heuristics

This group consists of comparatively simple heuristics. It includes the implementation of a deepest-bottom-left-fill (DBLF) heuristic for the three-dimensional packing problem which was presented in [25]. This procedure was based on the approach for the two-dimensional case (bottom-left (BL)) which was proposed in [3] and [20]. Initially, the items are sorted with respect to the customer sequence of a given tour. That is, the linehaul items of

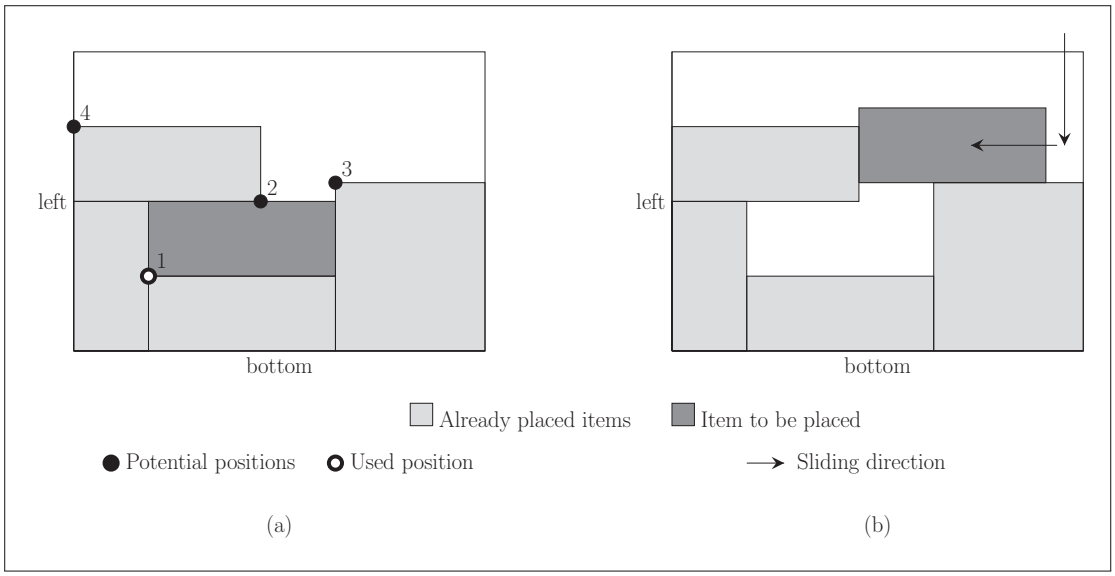

Figure 6: Comparison between BLF- and BL-approach: (a) BLF [20], (b) BL [23, 26]

the customers who are visited last are loaded into vehicle first. Loading for backhaul items follows an analogous rule. Ties are broken by the item fragility (non-fragile items first), breaking ties by non-increasing volume, breaking ties by non-increasing length and breaking ties by non-increasing width. Based on the resulting item sequence, the items are packed into the loading space where they are placed according to the following priorities: as far as possible (1) to the back, (2) to the bottom, and (3) to the left. In earlier implementations of the BL procedure, a sliding technique is used (e.g. in [23, 26]): The starting position of an item is the right upper edge. From there it is moved alternately as far as possible to the bottom and to the left. On the contrary, the DBLF approach applied here scans all positions that are available for placement (with both item orientations). These positions are sorted based on the placement priorities mentioned above. A comparison of both approaches is illustrated in Fig. 6 (for the two-dimensional packing problem). In Fig. 6(a) the available positions are numbered according to the bottom left-sequence. They are tested successively until a feasible placement is found. Since it is possible to fill the gaps created by other items (by placing the item in position 1 in the depicted example) the approach is called (deepest-)bottom-left-*fill*. Note that this may not be possible with the sliding technique (Fig. 6(b)). Moreover, a variant, called DBLF (stable), was implemented. In this variant, an item that has not been placed DBL-stably – i.e. as far as possible to the back, to the bottom, and to the left – is further moved towards such a position.

As a second construction heuristic, a touching area (TA) heuristic has been implemented

which is based on an approach first presented in [27] for the two-dimensional bin packing problem. The first item is placed in the deepest bottom-left corner of the loading space. All further items are placed in positions that maximize the touching area, i.e. the percentage of the item surfaces that touch previously placed items or the container walls. In contrast to the DBLF-algorithm, searching for a position does not stop as soon as one feasible position is found, but *all* potential positions and all possible orientations are evaluated. In addition, a variant that does not consider the container walls was implemented.

Local search and open space heuristic

This procedure was presented in [64] and is a LS-based approach. Two cases have to be distinguished: In the first case, the number of items to be packed is smaller than or equal to a given value (the authors propose 8). Then, all permutations of the item sequence are packed until one is identified in which the items could be packed feasibly or all have been tested. Packing is performed by an open space-based heuristic. We refer to [64] for further details. In the second case in which the number of items to be packed is larger than the given value, a procedure is applied which is illustrated in Fig. 7. Different item sequences are obtained by randomly swapping two items within the sequence. In contrast to the first case, only a limited number of permutations (which equals the number of items in the tour n_{items}) is tested here. The procedure *OpenSpaceHeuristic* returns the number of items that could be packed feasibly (p). The procedure terminates when all n_{items} items could be packed (cf. lines 6, 15) or when the number of iterations is exceeded (cf. lines 8, 20). Two different sorting rules are applied (line 2). In the first sorting rule, the items are sorted lexicographically based on the customer sequence in the tour, by fragility, by non-increasing base area, and finally by non-increasing volume. The second sorting rule is very similar. Merely, the length of the items is used as the third component instead of the base area.

Local search and construction heuristics

Finally, we also integrated the above-mentioned construction heuristics into the LS approach. That is, instead of the open space heuristic, a DBLF- or TA-heuristic is applied to the different item sequences within the LS. Moreover, the sorting rule which is described in the paragraph *Construction heuristics* is used.

```
1: procedure LOCAL_SEARCH_PACK
2:     for each sorting rule do
3:         I = sorted sequence of all items demanded by the customers in the route
4:         p := OpenSpaceHeuristic(I)
5:         if p = n_items then                          ▷ all n_items of the tour could be packed
6:             return TRUE
7:         else
8:             for k = 1 to n_items do
9:                 generate a new sequ. I* by swapping two randomly selected items in I
10:                p* := OpenSpaceHeuristic(I*)
11:                if p* ≥ p then
12:                    I := I*, p* := p
13:                end if
14:                if p = n_items then
15:                    return TRUE
16:                end if
17:            end for
18:        end if
19:    end for
20:    return FALSE
21: end procedure
```

Figure 7: LS-based packing procedure (case 2) [64]

In conclusion, the following nine packing heuristics are available:

- Firstly, each of the four construction heuristics – DBLF, DBLF (stable), TA-Walls, and TA-noWalls – can be applied individually.
- Moreover, each of these heuristics can be integrated into the LS framework.
- In addition, the LS procedure with the open space heuristic is also available.

In the hybrid solution approach, the ALNS can be combined with each of the packing heuristics. The following section describes the components of the ALNS into which a packing heuristic is integrated.

4.3 Integration of routing and packing

The packing procedure is applied in order to check whether (partial) routes that have been generated can be packed feasibly. This contains two aspects: On the one hand, the procedure determines two packing plans – one for the linehaul items and one for the backhaul items of a tour. On the other hand, in the case of the side loading approach, it tests whether linehaul and backhaul items would overlap at any stop of the route according to the generated packing plans. As these procedures are computationally expensive – about 95 % of the computation time of the hybrid routing and packing solution approach

is needed for packing – calling them as rarely as possible is desired. Packing checks are done in connection with the insertion heuristics and with the identification of a new globally best solution.

In order to highlight the usage of the packing procedure within the insertion heuristics, a general insertion procedure is described here (see Fig. 8). This procedure includes all insertion heuristics described above. By choosing the value $k = 1$ for the input parameter k, the Greedy insertion is called; for $k = 2$ or $k = 3$ if the respective Regret-k-heuristic is applied.

The set U comprises all customers that are not assigned to any tour in the temporary solution s_{next}, i.e. customers that have been removed in the current iteration or were not assigned to any tour before the current iteration. Initially, for each unassigned customer $i \in U$, the best feasible insertion for each tour is determined (lines 5-14). This is done by calling the procedure *select_ins* for each tour and accumulating the best insertions in the sets I_i ($i \in U$). The insertions of the unassigned customers are then sorted by non-decreasing insertion costs. In the next part (lines 15-37), the unassigned customers are successively inserted into the solution based on the insertion criteria of the respective heuristic. After an insertion was performed, the best insertions of the remaining customers in U into the tour, which was altered by the insertion, are updated (line 31). Again, the procedure *select_ins* is called for this purpose. The insertion procedure ends when all unassigned customers have been inserted or no more customers can be inserted into any tour.

The packing procedure can be viewed as part of the procedure *select_ins* which is outlined in Fig. 9. Only feasible insertions can be returned by the procedure. However, not *all* potential insertions are checked in detail for feasibility. First of all, only those tours are considered into which the customer can be inserted without exceeding the vehicle weight capacity by all linehaul items (at the beginning of the tour) and all backhaul items (at the end of the tour) (line 3). Here, d_t^L (d_t^B) represents the total weight of all linehaul (backhaul) items in tour t and d_i^L (d_i^B) represents the total weight of all linehaul (backhaul) items of customer i. The potential insertions into a tour are then sorted by non-decreasing insertion costs and successively tested for feasibility until a feasible insertion was identified or all potential insertions were tested unsuccessfully. Within these tests, the resulting routes are firstly tested with respect to the weight capacity and time window constraints at every stop of the tour. If these are satisfied, the packing procedure is called (line 15). The

procedure stops after a feasible insertion was found (line 16) or all potential insertions were tested unsuccessfully (line 21).

```
 1: procedure INSERTION(in: k, inout: temporary solution s_next)
 2:     U := set of unassigned customers in s_next
 3:     T := set of used tours in s_next (+ one empty tour, if not all vehicles are used)
 4:     ins_poss := FALSE                          ▷ indicates whether at least one insertion is possible
 5:     for each i ∈ U do
 6:         I_i := ∅                                ▷ set of best feasible insertions of customer i
 7:         for each t ∈ T do
 8:             {ins_it} := select_ins(i, t, s_next)   ▷ select the best feas. insertion into tour t
 9:             I_i := I_i ∪ {ins_it}
10:         end for
11:         if |I_i| > 0 then                       ▷ at least one insertion is possible for customer i
12:             sort I_i by non-decreasing insertion costs, ins_poss := TRUE
13:         end if
14:     end for
15:     while |U| > 0 do
16:         if ins_poss = FALSE then
17:             return                              ▷ terminate if no insertion possible
18:         end if
19:         if k = 1 then                                                          ▷ Greedy
20:             i := argmin_{i∈U} c(I_i(1))         ▷ I_i(1): cheapest insertion in I_i
21:         else                                                                   ▷ Regret-k
22:             for each i ∈ U do r_i := Σ_{j=1}^{k} (c(I_i(1)) − c(I_i(j))) end for
23:             i := argmax_{i∈U} r_i
24:         end if
25:         insert i at its minimum cost position into s_next
26:         t := tour into which i was inserted
27:         U := U \ {i}
28:         ins_poss := FALSE
29:         for each j ∈ U do                       ▷ update the best feasible insertion for tour t
30:             I_j := I_j \ {ins_jt}
31:             ins_jt := select_ins(j, t, s_next)
32:             I_j := I_j ∪ {ins_jt}
33:             if |I_i| > 0 then
34:                 sort I_i by non-decreasing insertion costs, ins_poss := TRUE
35:             end if
36:         end for
37:     end while
38: end procedure
```

Figure 8: Insertion heuristic

If customers are removed, the tours are not tested for packing feasibility. Hence, infeasible solutions may occur if no customer is reinserted into a tour altered by the removal. Therefore, if a new globally best solution is found, it is also checked for feasibility. It is only accepted if it is feasible with respect to the packing feasibility.

Moreover, a cache is used to reduce the packing effort. Routes are stored that have already been packed. However, for the LS-based packing procedures only those routes are saved that have been packed successfully. Due to the random component of the heuristics, it may happen that a route that could not be packed in one iteration can be packed feasibly

```
1: procedure SELECT_INS(in: customer i, tour t, solution s, out: best insertion ins)
2:     I := ∅                                              ▷ set of all insertions into tour t
3:     if d_t^L + d_i^L > D or d_t^B + d_i^B > D  then
4:         return ∅
5:     end if
6:     for p = 0 to n_t do
7:         c(ins_itp) := cost of inserting i into tour t at position p
8:         I := I ∪ {ins_itp}
9:     end for
10:    sort I by non-decreasing cost
11:    while |I| > 0 do
12:        ins := I(1)                                     ▷ element of I with the least cost
13:        R := R_t with i inserted at position p(ins)     ▷ R_t: Route t of s
14:        if R is feasible w.r.t. time windows and weight capacity at each stop then
15:            if R can be packed feasibly then            ▷ call of packing procedure
16:                return {ins}
17:            end if
18:        end if
19:        I := I \ {ins}
20:    end while
21:    return ∅
22: end procedure
```

Figure 9: Procedure *select_ins*

in another.

5 Computational experiments

In experiments, we tested the ALNS for the one-dimensional VRPSDP in order to evaluate its performance and competitiveness with other procedures. Moreover, we examined the hybrid algorithm combining the ALNS with a packing procedure for solving the 3L-VRPSDPTW.

In Section 5.1, one-dimensional instances are selected from the literature and newly generated 3L-VRPSDPTW instances are described.

In Section 5.2, parameter values are chosen for the proposed ALNS and the hybrid algorithm.

In Section 5.3, test results for the one-dimensional VRPSDP are presented and evaluated.

Test results for three variants of the hybrid algorithm for solving the 3L-VRPSDPTW are analysed in Section 5.4. They differ w.r.t the used packing heuristic.

The hybrid algorithm was implemented in C++ and the experiments were run on a Haswell system with up to 3.2 GHz and 16 GB RAM per core.

5.1 Problem instances

One-dimensional instances

For the one-dimensional tests of the ALNS, we used 14 VRPSDP instances introduced in [41] with the number of customers ranging from 50 to 199. The instances can be downloaded at: `https://www.kent.ac.uk/kbs/research/research-centres/clho/datasets.html`.[1]

However, the ALNS was originally developed for solving various VRP variants. Therefore, not only VRPSDP instances were used for determining appropriate parameter values for different problem classes. In addition, instances of [44] (VRP with time windows), [18] and [52] (VRP with clustered backhauls (VRPCB)), [16] (VRPCB with time windows), and [41] (VRP with mixed backhauls), were selected for parameter setting.

Three-dimensional instances

Since no 3L-VRP instances exist which include backhaul and time window aspects, new instances have been generated. For these instances, the customer locations have been determined randomly. The generation of time windows is based on [44]. The proportion of customers who have a time window (*time window density*) differs in each instance. A customer time window is generated in the following way: Its centre is selected randomly from the interval $[RT_0 + c_{0i}, DD_0 - c_{i0} - ST_i]$ (for the time windows of the depot, see Table 2). The time window width is a normally distributed random number with mean μ and standard deviation σ as listed below. If no time window is assigned to a customer i, the customer receives a "fictional" time window $(RT_i, DD_i) = (0, DD_0 - c_{i0} - ST_i)$. We further assume that the service times of all customers are identical. In order to evaluate the impact of different levels of item heterogeneity, we generated instances with different numbers of item types.

The generation of items is based on [17] where the item length (width, height) is uniformly randomly distributed in the interval $[0.2L,\ 0.6L]$ ($[0.2W,\ 0.6W]$, $[0.2H,\ 0.6H]$). This approach is also used for the item length and width for the new instances. However, since we consider the loading variant with a horizontally split loading space, the item heights must not be larger than $0.5H$. In addition, we also consider instances with small

[1] Note that we omit the 14 instances with drop times and maximum distance constraints since those problem aspects are not considered here and the instances are identical to the other 14 instances with respect to the other problem information.

items. They are generated with a length (width, height) of $[0.1L, 0.3L]$ ($[0.1W, 0.3W]$, $[0.1H, 0.3H]$). To each item type a weight is assigned that is uniformly distributed between $1[\frac{weight\ units}{vol.\ units}] \cdot$ item vol.$[vol.\ units]$ and $10[\frac{weight\ units}{vol.\ units}] \cdot$ item vol.$[vol.\ units]$. Moreover, at least one item type but at most 20 % of the item types is fragile. The dimensions of the loading space are fixed to $L = 60$, $W = 25$ and $H = 30$ and the weight capacity to $D = 200$ for all instances.

The number of available vehicles v_{max} was determined by means of the savings heuristic in order to ensure that a feasible solution exists.

The chosen values for the different instance features are given in Table 2.

Table 2: Instance characteristics

Feature	**Parameters**
Total number of items m	200
Number of customers n	20, 60, 100
Number of items per customer	5-15 for $n = 20$, 2-5 for $n = 60$, 1-3 for $n = 100$
Share of linehaul items	50 % , 80 %
Time window width	wide: $RT_0 = 0, DD_0 = 1000, \mu = 240, \sigma = 60$, narrow: $RT_0 = 0, DD_0 = 230, \mu = 60, \sigma = 10$
Time window density	25 %, 50 %, 75 %, 100 %
Sizes of items	large ($[0.2L, 0.6L], [0.2W, 0.6W], [0.2H, 0.5H]$), small ($[0.1L, 0.3L], [0.1W, 0.3W], [0.1H, 0.3H]$)
Number of different item types	3, 10, 100
Support parameter α	75 %

For each combination of the instance characteristics *number of customers*, *time window width*, *linehaul share*, *item size* and *number of item types* five instances were generated. However, instances with 20 customers and large items were omitted. Due to the relatively large number of items per customer, the items of two customers (let alone more than two) would not fit into a vehicle together. The resulting solutions would, thus, consist of single-stop tours. All in all, 300 instances were generated for the 3L-VRPSDPTW.

5.2 Parameter setting

50 one-dimensional VRP instances (of which ten were VRPSDP instances) were used for parameter tuning of the ALNS (see above). The settings from [40] were used as a basis. Different values were tested for the parameters successively while all others were fixed. The best settings were kept for each parameter. During the tests, the ALNS was applied five times to each instance and for each tested parameter combination. The final settings are given in Table 3. The same parameter values were used later for the three-dimensional

case. The only parameter which is specific to this case, is i_{max} which occurs in connection with the LS-based packing procedures. It determines up to which number of items in a route all item sequence permutations are packed (see secton 4.2). In this case, the parameter value was adopted from [64].

Table 3: Parameter settings

Parameter	**Description**	**Value**
$iter_{max}$	Max. number of iterations	25000
$iter_{no\_impr}$	Max. number of iterations without improvement	8000
t_{max}	Time limit [min]	15 for $n = 20$, 60 for $n \geq 60$
no_{min}, no_{max}	Interval for number of removed customers	$0.04n, 0.4n$
$\sigma_1, \sigma_2, \sigma_3$	Weight adjustment parameters	50, 10, 5
$\omega_1, \omega_2, \omega_3, \omega_4$	Shaw weights	6, 3, 2, 6
p	Determinism parameter	6
r	Reaction factor	0.8
seg	Segment length	100
η	Noise parameter	0.025
γ	Cooling rate	0.99975
w	Start temperature control parameter	5 %
pen_v	Penalty term for violation of the tour number restriction; $c_{max} = \max_{i,j \in N} c_{ij}$	$10 \cdot c_{max}$
pen_{mv}	Penalty term for missing customers	$1 \cdot c_{max}$
i_{max}	Max. number of items in a route up to which all item sequence permutations are packed	8

5.3 Results for the VRPSDP

Table 4 reports the results for the VRPSDP instances from [41]. Since it contains several random components, the ALNS was applied five times to each test instance.

In the columns 3-6, total travel distances (TTDs) are given. In columns 3 and 4, the average (*ALNS avg.*)/ best (*ALNS best*) TTDs of the ALNS are presented. In columns 5 and 6, the best known TTDs (*BKS*) and the best TTDs provided by [39] (*RP06*) are indicated. In columns 7 and 8, the solutions of our approach are compared to the best known solutions (*gap BKS*). The percentage gaps between the average/ best TTDs of the ALNS and the TTDs are given. In the following column (*gap RP06*), the percentage gaps between the best TTDs provided by the ALNS and the best TTDs achieved by the approach of [39] are listed.

The last column (t) contains the average computing times of our approach.

The results indicate a very good performance of the algorithm. Averaged over the instances, the gap between the average TTDs of the ALNS and the best known TTDs amounts to -0.18 %. This makes the proposed ALNS one of the best published solution approaches for the VRPSDP. Moreover, the tests show that our approach results in considerably better solutions than the original ALNS in [39] with an average gap between the best found TTDs of -4.26 % and improvements of up to 14.89 %. Some best known values could also be improved by up to 5.7 %.

Table 4: Results of the ALNS for 14 VRPSDP instances

		TTD				**gap BKS[%]**		**gap RP06[%]**	**t[s]**
Instance	**n**	*ALNS avg.*	*ALNS best*	*BKS*	*RP06*	*ALNS avg.*	*ALNS best*	*ALNS best*	*ALNS avg.*
CMT01X	50	471.24	470.48	466.75[1]	467	0.96	0.80	0.74	3.48
CMT01Y	50	461.68	461.24	458.96[2]	467	0.59	0.50	-1.23	4.11
CMT02X	75	686.82	684.29	668.77[2]	702	2.70	2.32	-2.52	16.18
CMT02Y	75	659.36	659.29	663.25[2]	685	-0.59	-0.60	-3.75	17.15
CMT03X	100	725.93	720.17	715.32[3]	727	1.48	0.68	-0.94	32.25
CMT03Y	100	712.49	702.73	719.00[4]	734	-0.91	-2.26	-4.26	27.72
CMT04X	150	855.12	854.17	852.46[5]	877	0.31	0.20	-2.60	126.80
CMT04Y	150	833.32	828.64	847.58[3]	854	-1.68	-2.24	-2.97	85.71
CMT05X	199	1,072.40	1,066.72	1,029.25[6]	1108	4.19	3.64	-3.73	218.94
CMT05Y	199	990.92	987.12	1,029.25[6]	1131	-3.72	-4.09	-12.72	260.04
CMT11X	120	831.38	829.84	833.92[6,7]	837	-0.31	-0.49	-0.86	75.85
CMT11Y	120	795.93	783.06	830.39[2]	920	-4.15	-5.70	-14.89	59.67
CMT12X	100	663.78	660.82	644.70[2]	683	2.96	2.50	-3.25	31.32
CMT12Y	100	630.52	628.12	659.52[2]	673	-4.40	-4.76	-6.67	29.69
Minimum						-4.40	-5.70	-14.89	3.48
Average						-0.18	-0.68	-4.26	70.64
Maximum						4.19	3.64	0.74	260.04

(References BKS: [1][15], [2][57], [3][24], [4][48], [5][60], [6][45], [7][61])

5.4 Results for the 3L-VRPSDPTW

Firstly, the best packing construction heuristic was determined. For this purpose, the four construction heuristics described in section 4.2 were applied to randomly generated routes of the new 3L-VRPSDPTW instances. The performance of the different heuristics was evaluated based on the share of routes that could be packed feasibly. In another test, the savings heuristic was applied to the same instances and the resulting travel distances were compared.

In the second step, three versions of the hybrid algorithm, based on the best construction heuristic, were applied to the 3L-VRPSDPTW instances.

Determination of the best packing construction heuristic

Results for random routes: As mentioned above, we wanted to determine the best packing construction heuristic first. The different heuristics were applied to more than 37,000 randomly generated routes of the above introduced three-dimensional instances with different levels of loading space volume utilization. For small items they ranged from 25 to 90 %, and for large items from 10 from 60 %.

Table 5 contains the results of these tests showing the shares of routes that could be packed feasibly in the respective intervals and with the respective packing heuristics. The results indicate that the DBLF heuristic is the dominant one with more feasibly packed routes in each interval and for both large and small items. The TA heuristic which also takes the container walls into account (*TA-Walls*) yields comparatively good results for routes with large items. However, both TA heuristics are clearly dominated by the DBLF heuristics in the case of small items.

Results for the savings heuristic: Furthermore, the savings heuristic was applied to the 3L-VRPSDPTW instances with the different packing construction heuristics. The results are presented in Table 6. The average percentage gaps between the obtained TTDs and the best ones found over all packing procedures are indicated for the instances with large and small items and for all instances. The obtained results support the insights of the first packing tests. In total, the best solutions were achieved with the DBLF heuristic. Although it is outperformed by the TA heuristics in the case of large items, it clearly dominates the TA heuristics in the case of small items. Therefore, we will use the *plain* DBLF approach for the further experiments.

Table 5: Results of the application of the packing heuristics to random routes

Vol. utilization[%]		**Share[%] of feasibly packed routes**								
		10-20	*20-30*	*30-40*	*40-50*	*50-60*	*60-70*	*70-80*	*80-90*	***Total***
Large items	DBLF	98.6	86.4	61.2	34.2	12.1				57.3
	DBLF (st.)	98.1	83.2	56.6	30.4	11.5				54.7
	TA-noWalls	97.9	80.1	52.8	28.1	10.3				52.6
	TA-Walls	98.4	84.6	57.6	29.8	10.8				55.0
Small items	DBLF		98.9	89.6	58.9	31.7	10.7	0.9	0.0	37.1
	DBLF (st.)		77.7	57.5	35.2	19.1	5.9	0.4	0.0	24.2
	TA-noWalls		55.9	35.9	18.5	7.3	1.8	0.2	0.0	14.1
	TA-Walls		44.0	26.3	9.8	2.8	0.3	0.0	0.0	9.4

Table 6: Results of the savings heuristic combined with different packing heuristics

	Gap[%] w.r.t. the best found TTDs		
Heuristic	*Large items*	*Small items*	*Total*
DBLF	1.93	0.30	0.68
DBLF (stable)	1.96	1.23	1.40
TA-noWalls	1.79	3.83	3.36
TA-Walls	0.80	7.51	5.71

Results for the hybrid algorithm

Since there are no benchmark results available for the 3L-VRPSDPTW instances, we compare the results of different versions of the hybrid algorithm each combining the ALNS with one of three packing procedures – namely the DBLF, the DBLF in combination with the LS approach, or the LS procedure of [64]. Furthermore, two alternative loading approaches (side loading and loading space partition) were applied. As before, the approach was applied five times to each instance.

The results are presented in Table 7. The first four columns refer to the instance characteristics and contain the number of customers (n), the item sizes ($Items$), the width of the time windows (TW), and the share of linehaul items (LH). The following columns present the results for the different packing heuristics: the DBLF-heuristic, the LS in combination with the DBLF procedure (LS_DBLF) and the LS approach of [64] (LS_OS). The loading approaches loading space partition (LSP) and side loading ($Side$) are compared. The columns gap_l represent the average percentage gaps between the obtained TTDs and the best TTDs for the respective loading variant, whereas the columns gap give the average percentage gaps between the obtained TTDs and the best found TTDs over both loading approaches.

Comparing the different packing procedures, the best results could be achieved with the LS_DBLF approach. The obtained travel distances deviate 0.89 % (LSP) and 0.88 % (Side) from the ones of best found solutions for the respective loading variants. With a gap of 1.54 % from the best TTDs, the results with the LS_OS for the loading space partition are also rather good. Interestingly enough, with the side loading approach the ALNS with the simple DBLF heuristic delivers, on average, better solutions than the LS_OS. However, this is most likely not because the DBLF heuristic delivers better packing results. The LS_OS needs much more computing time than the other approaches. Thus, less iterations can be run within the computing time limit leading to worse solutions

regarding the routing problem. In fact, the results indicate that the good quality of the LS-based heuristics do not necessarily offset their large requirements of computing times. The instances with the biggest effort in packing – because they often require many items to be packed within one tour – are the ones with 100 customers (thus, with few items per customer), small items and wide time windows. With respect to these instances, the best results were obtained with the ALNS in combination with the DBLF heuristic.

Considering the different loading approaches it can be stated that side loading produces better results than rear loading with separated loading space. With all compared packing heuristics the solutions obtained with the side loading approach deviate less from the best found solutions. These results could be expected since the side loading allows to take full use of the loading space. The differences are comparatively large for instances with large items. Small items could be packed easier into the separated loading space and, thus, rather good solutions could be achieved with this approach, too. Moreover, the side loading approach seems to be particularly beneficial if the share of delivery and pickup items is unequal since the compartments for linehaul and backhaul items are equally large in the case of the LSP. Hence, the largest deviations from the best found TTDs resulted for instances with large items and a share of 80 % linehaul items. However, there are instance classes in which the LSP approach actually led to better solutions than the side loading. Namely, this is the case for the ALNS combined with LS_OS and the instances with 100 customers and small items. Presumably, the same reasoning as above can be given here. The side loading approach requires more computing time than the LSP approach (see below) and, thus, less iterations of the ALNS can be run within the time limit. In combination with the implied difficulty of the instances this affects the results much more than in the case of other instance classes.

Table 8 shows an overview of the average computing times (t) and the average number of iterations that where run ($iter$) in the different instance classes and with the different packing heuristics. The three stopping criteria listed above have been applied. For instances with 20 customers, best solutions were found rather quickly which is why the computations stopped after not even 9,000 iterations on average. Naturally, the computing times increase with the number of customers due to the increasing difficulty of the underlying VRP. As mentioned above, instances with small items and/or wide time windows usually lead to solutions containing tours with many items to be packed. Therefore, these instance classes are solved within comparatively long computing times. Further-

more, the side loading approach is related to a higher computational effort than the LSP since the packing plans for every stop of a tour must be compared in order to ensure that they do not overlap. Hence, the computing times for the side loading approach is higher in most cases. Moreover, the general impact of the packing procedures on the computing times can be concluded as well. The time limit was never reached in the one-dimensional case (see Table 4). There, instances with 100 customers could be solved in about 30 seconds. In the three-dimensional case, though, the time limit of one hour was reached frequently.

Table 7: Results of the hybrid algorithm applied to 3L-VRPSDPTW instances

	Packing procedure			**DBLF**				**LS_DBLF**				**LS_OS**			
	Loading approach			**LSP**		**Side**		**LSP**		**Side**		**LSP**		**Side**	
n	Items	TW	LH	gap_l[%]	gap[%]	gap_l[%]	gap[%]	gap_l[%]	gap[%]	gap_l[%]	gap[%]	gap_l[%]	gap[%]	gap_l[%]	gap[%]
20	small	narrow	50%	0.64	1.43	1.16	1.50	0.07	0.84	1.16	1.50	0.00	0.76	0.27	0.60
			80%	1.83	2.17	1.58	1.81	0.22	0.56	1.04	1.28	0.30	0.64	0.47	0.69
		wide	50%	0.93	1.60	1.20	2.86	0.02	0.68	0.96	2.61	0.09	0.75	0.91	2.54
			80%	4.00	9.32	4.38	4.49	0.87	6.05	3.29	3.41	1.94	7.15	2.11	2.22
60	large	narrow	50%	11.29	22.67	4.06	4.73	1.52	12.17	0.82	1.47	0.74	11.36	10.85	11.56
			80%	14.87	46.68	10.63	10.63	2.63	31.49	1.78	1.78	0.65	28.99	2.08	2.08
		wide	50%	11.11	18.62	3.63	5.65	1.39	8.50	0.62	2.58	1.60	8.67	17.53	19.84
			80%	11.85	51.19	9.68	9.68	1.81	37.79	0.94	0.94	2.24	38.31	5.97	5.97
	small	narrow	50%	0.04	0.04	0.04	0.04	0.04	0.05	0.06	0.06	0.04	0.04	0.11	0.11
			80%	0.05	0.05	0.07	0.08	0.05	0.05	0.06	0.08	0.04	0.04	0.09	0.10
		wide	50%	0.48	0.48	0.31	1.24	0.31	0.32	0.34	1.26	0.65	0.66	1.18	2.12
			80%	0.91	1.56	0.37	1.34	0.64	1.28	0.37	1.33	2.16	2.81	4.90	5.91
100	large	narrow	50%	8.22	13.71	2.44	4.22	1.59	6.78	0.86	2.64	1.58	6.82	8.04	9.99
			80%	12.12	45.45	5.97	5.97	2.15	32.54	1.48	1.48	4.99	36.21	4.22	4.22
		wide	50%	6.60	20.86	2.68	3.13	1.21	14.85	1.05	1.48	3.19	17.14	15.83	16.31
			80%	10.28	47.13	6.02	6.02	1.08	34.88	1.20	1.20	5.78	41.11	8.12	8.12
	small	narrow	50%	0.16	0.18	0.17	0.19	0.15	0.18	0.19	0.21	0.21	0.24	0.27	0.29
			80%	0.29	0.30	0.27	0.28	0.19	0.20	0.22	0.22	0.21	0.22	0.27	0.28
		wide	50%	0.53	0.65	0.49	0.64	0.76	0.88	0.49	0.65	1.51	1.63	1.90	2.06
			80%	0.51	0.65	0.48	0.68	1.11	1.26	0.71	0.91	2.92	3.07	5.99	6.20
Minimum				0.04	0.04	0.04	0.04	0.02	0.05	0.06	0.06	0.00	0.04	0.09	0.10
Average				4.83	14.24	2.78	3.26	0.89	9.57	0.88	1.35	1.54	10.33	4.56	5.06
Maximum				14.87	51.19	10.63	10.63	2.63	37.79	3.29	3.41	5.78	41.11	17.53	19.84

Table 8: Average computing time and number of iterations

	Packing procedure			**DBLF**				**LS_DBLF**				**LS_OS**			
	Loading approach			**LSP**		**Side**		**LSP**		**Side**		**LSP**		**Side**	
n	Items	TW	LH	t[s]	iter	t[s]	iter	t[s]	iter	t[s]	iter	t[s]	iter	t[s]	iter
20	small	narrow	50%	3	8,228	16	8,244	19	8,486	16	8,263	86	8,241	65	8,937
			80%	7	8,381	52	8,803	303	7,993	78	8,807	450	5,704	289	7,005
		wide	50%	37	8,548	282	8,787	615	5,908	301	8,726	802	3,022	710	5,428
			80%	58	8,511	512	8,105	902	1,250	665	5,748	907	207	911	280
60	large	narrow	50%	19	20,024	83	18,738	167	20,440	221	19,835	3,181	9,968	3,046	12,610
			80%	15	18,127	76	19,349	206	20,386	364	20,227	3,600	6,926	3,127	14,670
		wide	50%	38	20,737	218	19,892	463	22,264	611	19,559	3,601	3,182	3,601	3,948
			80%	27	19,579	214	19,857	479	21,479	1,401	21,863	3,601	2,248	3,601	3,707
	small	narrow	50%	17	13,057	28	13,341	18	12,854	29	12,676	80	13,147	783	13,943
			80%	19	11,727	37	12,627	20	11,769	38	12,299	48	11,946	228	12,473
		wide	50%	597	12,720	1,430	14,016	1,887	10,829	1,375	13,270	2,712	7,444	2,649	11,106
			80%	1,034	11,986	2,173	12,006	3,297	5,286	2,452	11,577	3,582	1,639	3,612	1,083
100	large	narrow	50%	65	23,242	128	21,958	281	23,346	270	22,451	3,573	8,335	3,366	13,585
			80%	70	22,798	141	20,944	453	23,025	495	22,243	3,532	4,985	3,441	9,538
		wide	50%	168	23,290	582	21,053	1,088	23,311	1,980	22,380	3,602	1,301	3,601	1,851
			80%	138	23,052	510	22,111	1,130	22,430	2,491	21,491	3,602	1,430	3,602	2,075
	small	narrow	50%	60	20,447	82	19,551	60	19,659	91	19,538	318	19,798	2,967	18,581
			80%	62	18,197	105	19,383	66	18,534	110	18,596	192	18,477	1,115	17,975
		wide	50%	1,333	16,073	2,268	14,910	2,191	12,318	2,180	14,061	3,059	8,947	3,548	6,223
			80%	2,096	15,209	3,225	12,475	3,086	8,088	3,268	10,843	3,403	4,210	3,615	2,021
Minimum				3	8,228	16	8,105	18	1,250	16	5,748	48	207	65	280
Average				293	16,197	608	15,807	837	14,983	922	15,723	2,197	7,058	2,394	8,352
Maximum				2,096	23,290	3,225	22,111	3,297	23,346	3,268	22,451	3,602	19,798	3,615	18,581

6 Conclusions

In this paper, the 3L-VRPSDPTW was introduced which is an integrated routing and packing problem. The routing subproblem contains the VRP with simultaneous delivery and pickup, a VRP variant in which goods are delivered to the customers from a central depot, and – at the same time – goods are picked up from them. This approach allows for organizing transportations more economically by reducing the number of empty vehicle trips. The problem occurs, for example, in the retail sector if the delivery vehicles pickup packaging material, like empty cardboard or cooling boxes. For a more realistic modelling, the transported goods are assumed to be three-dimensional items which must be packed into the loading space observing several packing constraints in order to guarantee stability and accessibility of the load.

A hybrid solution approach consisting of an ALNS to solve the routing subproblem and a packing procedure to tackle the packing subproblem was presented. In contrast to previously presented variants of the ALNS, we work with a relatively large number of removal heuristics. The adaptive weight adjustment component of the ALNS aims to choose the best performing heuristics for each instance in the course of the search. Some new removal operators were developed for this purpose. The results for one-dimensional VRPSDP instances indicate that the proposed ALNS belongs to the best published approaches for this problem. Since it improved the solutions found by the original approach significantly, it can be concluded that the new operators and the usage of a large number of removal operators are very valuable additions to the ALNS. Moreover, different packing heuristics – simple construction heuristics and more complex LS-based procedures – were tested and integrated into the ALNS.

The hybrid algorithm was applied to 300 newly generated 3L-VRPSDPTW instances with three alternative packing procedures. The best results were obtained with a LS-based packing procedure into which a DBLF packing heuristic was integrated. However, although the LS-based packing procedures are in general able to solve a larger number of packing problems (i.e. to proof the feasibility of a larger number of routes), their comparatively large computational effort can also be a huge disadvantage compared to simple construction heuristics. In some problem classes, the ALNS combined with the simple DBLF led to the best results, especially in the case of instances where long routes with many items are possible. This is presumably because the LS-based packing procedures re-

quire so much computing time that considerably fewer ALNS iterations can be conducted within a given limit.

In order to avoid any reloading effort, we have considered two loading variants to handle the simultaneous transport of linehaul and backhaul items. Whereas the loading space partition approach is implemented comparatively easily, better solutions resulted from the more difficult side loading approach. The differences are particularly high for instances with large items or an uneven share of linehaul and backhaul items.

The integration of vehicle routing problems with backhauls and packing problems is an interesting topic for further research as there are different backhaul variants that could be considered. Furthermore, we focused on a problem in which the reloading during the tour is completely forbidden. The implementation of such reloading efforts into the 3L-VRP (with or without backhauls) could be a promising approach, though, in order to better utilize the vehicle capacities and, thus, save travel distance.

Acknowledgements We thank Lijun Wei (JiangXi University of Finance and Economics, China) for providing us with the program for the local search and the open space heuristic.

References

[1] Angelelli, E.; Mansini, R. (2002): The Vehicle Routing Problem with Time Windows and Simultaneous Pick-up and Delivery. In: A. Klose; M. G. Speranza; L. N. van Wassenhove, eds., *Quantitative Approaches to Distribution Logistics and Supply Chain Management*, pp. 249–267, Springer, Berlin, Heidelberg.

[2] Avci, M.; Topaloglu, S. (2016): A hybrid metaheuristic algorithm for heterogeneous vehicle routing problem with simultaneous pickup and delivery. In: *Expert Systems with Applications*, 53, pp. 160–171.

[3] Baker, B. S.; Coffman, Jr., E. G.; Rivest, R. L. (1980): Orthogonal Packings in Two Dimensions. In: *SIAM Journal on Computing*, 9, 4, pp. 846–855.

[4] Bartók, T.; Imreh, C. (2011): Pickup and Delivery Vehicle Routing with Multidimensional Loading Constraints. In: *Acta Cybernetica*, 20, 1, pp. 17–33.

[5] Bortfeldt, A. (2012): A hybrid algorithm for the capacitated vehicle routing problem with three-dimensional loading constraints. In: *Computers & Operations Research*, 39, 9, pp. 2248–2257.

[6] Bortfeldt, A.; Hahn, T.; Männel, D.; Mönch, L. (2015): Hybrid algorithms for the vehicle routing problem with clustered backhauls and 3D loading constraints. In: *European Journal of Operational Research*, 243, 1, pp. 82–96.

[7] Çatay, B. (2010): A new saving-based ant algorithm for the Vehicle Routing Problem with Simultaneous Pickup and Delivery. In: *Expert Systems with Applications*, 37, 10, pp. 6809–6817.

[8] Clarke, G.; Wright, J. W. (1964): Scheduling of Vehicles from a Central Depot to a Number of Delivery Points. In: *Operations Research*, 12, 4, pp. 568–581.

[9] Crispim, J.; Brandão, J. (2005): Metaheuristics Applied to Mixed and Simultaneous Extensions of Vehicle Routing Problems with Backhauls. In: *The Journal of the Operational Research Society*, 56, 11, pp. 1296–1302.

[10] Dell'Amico, M.; Righini, G.; Salani, M. (2006): A Branch-and-Price Approach to the Vehicle Routing Problem with Simultaneous Distribution and Collection. In: *Transportation Science*, 40, 2, pp. 235–247.

[11] Demir, E.; Bektaş, T.; Laporte, G. (2012): An adaptive large neighborhood search heuristic for the Pollution-Routing Problem. In: *European Journal of Operational Research*, 223, 2, pp. 346–359.

[12] Dethloff, J. (2001): Vehicle routing and reverse logistics: The vehicle routing problem with simultaneous delivery and pick-up. In: *OR Spektrum*, 23, 1, pp. 79–96.

[13] Fan, J. (2011): The Vehicle Routing Problem with Simultaneous Pickup and Delivery Based on Customer Satisfaction. In: *Procedia Engineering*, 15, pp. 5284–5289.

[14] Fuellerer, G.; Doerner, K. F.; Hartl, R. F.; Iori, M. (2010): Metaheuristics for vehicle routing problems with three-dimensional loading constraints. In: *European Journal of Operational Research*, 201, 3, pp. 751–759.

[15] García-Nájera, A.; Bullinaria, J. A.; Gutiérrez-Andrade, M. A. (2015): An evolutionary approach for multi-objective vehicle routing problems with backhauls. In: *Computers & Industrial Engineering*, 81, pp. 90–108.

[16] Gélinas, S.; Desrochers, M.; Desrosiers, J.; Solomon, M. M. (1995): A new branching strategy for time constrained routing problems with application to backhauling. In: *Annals of Operations Research*, 61, 1, pp. 91–109.

[17] Gendreau, M.; Iori, M.; Laporte, G.; Martello, S. (2006): A Tabu Search Algorithm for a Routing and Container Loading Problem. In: *Transportation Science*, 40, 3, pp. 342–350.

[18] Goetschalckx, M.; Jacobs-Blecha, C. (1989): The vehicle routing problem with backhauls. In: *European Journal of Operational Research*, 42, 1, pp. 39–51.

[19] Halse, K. (1992): Modeling and solving complex vehicle routing problems. *Ph.D. Thesis*, Technical University of Denmark, Lyngby.

[20] Hopper, E. (2000): Two-dimensional packing utilising evolutionary algorithms and other meta-heuristic methods. Master's thesis, University of Wales. Cardiff.

[21] Iori, M.; Martello, S. (2010): Routing problems with loading constraints. In: *TOP*, 18, 1, pp. 4–27.

[22] Irnich, S.; Toth, P.; Vigo, D. (2014): The Family of Vehicle Routing Problems. In: P. Toth; D. Vigo, eds., *Vehicle routing*, vol. 18 of *MOS-SIAM series on optimization*, pp. 1–33, SIAM, Philadelphia, Pa.

[23] Jakobs, S. (1996): On genetic algorithms for the packing of polygons. In: *European Journal of Operational Research*, 88, 1, pp. 165–181.

[24] Jun, Y.; Kim, B.-I. (2012): New best solutions to VRPSPD benchmark problems by a perturbation based algorithm. In: *Expert Systems with Applications*, 39, 5, pp. 5641–5648.

[25] Karabulut, K.; Inceoglu, M. M. (2005): A Hybrid Genetic Algorithm for Packing in 3D with Deepest Bottom Left with Fill Method. In: T. Yakhno, ed., *Advances in Information Systems*, vol. 3261 of *Lecture Notes in Computer Science*, pp. 441–450, Springer, Berlin/Heidelberg.

[26] Liu, D.; Teng, H. (1999): An improved BL-algorithm for genetic algorithm of the orthogonal packing of rectangles. In: *European Journal of Operational Research*, 112, 2, pp. 413–420.

[27] Lodi, A.; Martello, S.; Vigo, D. (1999): Heuristic and Metaheuristic Approaches for a Class of Two-Dimensional Bin Packing Problems. In: *INFORMS Journal on Computing*, 11, 4, pp. 345–357.

[28] Ma, H.-w.; Zhu, W.; Xu, S. (2011): Research on the Algorithm for 3L-CVRP with Considering the Utilization Rate of Vehicles. In: R. Chen, ed., *Intelligent Computing and Information Science*, vol. 134 of *Communications in Computer and Information Science*, pp. 621–629, Springer Berlin Heidelberg, Berlin, Heidelberg.

[29] Männel, D.; Bortfeldt, A. (2016): A hybrid algorithm for the vehicle routing problem with pickup and delivery and three-dimensional loading constraints. In: *European Journal of Operational Research*, 254, 3, pp. 840–858.

[30] Maquera, G.; Laguna, M.; Gandelman, D. A.; Sant'Anna, A. P. (2012): Scatter search applied to the vehicle routing problem with simultaneous delivery and pickup. In: *Trends in Developing Metaheuristics, Algorithms, and Optimization Approaches*, pp. 149–168.

[31] Miao, L.; Ruan, Q.; Woghiren, K.; Ruo, Q. (2012): A hybrid genetic algorithm for the vehicle routing problem with three-dimensional loading constraints. In: *RAIRO - Operations Research*, 46, 1, pp. 63–82.

[32] Min, H. (1989): The multiple vehicle routing problem with simultaneous delivery and pick-up points. In: *Transportation Research Part A: General*, 23, 5, pp. 377–386.

[33] Moura, A. (2008): A Multi-Objective Genetic Algorithm for the Vehicle Routing with Time Windows and Loading Problem. In: A. Bortfeldt; J. Homberger; H. Kopfer; G. Pankratz; R. Strangmeier, eds., *Intelligent Decision Support*, Gabler Edition Wissenschaft, pp. 187–201, Betriebswirtschaftlicher Verlag Dr. Th. Gabler / GWV Fachverlage GmbH Wiesbaden, Wiesbaden.

[34] Moura, A.; Oliveira, J. F. (2009): An integrated approach to the vehicle routing and container loading problems. In: *OR Spectrum*, 31, 4, pp. 775–800.

[35] Parragh, S. N.; Doerner, K. F.; Hartl, R. F. (2008): A survey on pickup and delivery problems. In: *Journal für Betriebswirtschaft*, 58, 1, pp. 21–51.

[36] Polat, O.; Kalayci, C. B.; Kulak, O.; Günther, H.-O. (2015): A perturbation based variable neighborhood search heuristic for solving the Vehicle Routing Problem with Simultaneous Pickup and Delivery with Time Limit. In: *European Journal of Operational Research*, 242, 2, pp. 369–382.

[37] Pollaris, H.; Braekers, K.; Caris, A.; Janssens, G. K.; Limbourg, S. (2015): Vehicle routing problems with loading constraints: State-of-the-art and future directions. In: *OR Spectrum*, 37, 2, pp. 297–330.

[38] Rieck, J.; Zimmermann, J. (2013): Exact Solutions to the Symmetric and Asymmetric Vehicle Routing Problem with Simultaneous Delivery and Pick-Up. In: *Business Research*, 6, 1, pp. 77–92.

[39] Ropke, S.; Pisinger, D. (2006): A unified heuristic for a large class of Vehicle Routing Problems with Backhauls. In: *European Journal of Operational Research*, 171, 3, pp. 750–775.

[40] Ropke, S.; Pisinger, D. (2006): An Adaptive Large Neighborhood Search Heuristic for the Pickup and Delivery Problem with Time Windows. In: *Transportation Science*, 40, 4, pp. 455–472.

[41] Salhi, S.; Nagy, G. (1999): A Cluster Insertion Heuristic for Single and Multiple Depot Vehicle Routing Problems with Backhauling. In: *The Journal of the Operational Research Society*, 50, 10, pp. 1034.

[42] Shaw, P. (1997): A new local search algorithm providing high quality solutions to vehicle routing problems. *Working Paper*, APES Group, Department of Computer Science, University of Strathclyde, Glasgow, Scotland.

[43] Shaw, P. (1998): Using Constraint Programming and Local Search Methods to Solve Vehicle Routing Problems. In: M. Maher; J.-F. Puget, eds., *Principles and Practice of Constraint Programming — CP98*, vol. 1520 of *Lecture Notes in Computer Science*, pp. 417–431, Springer Berlin Heidelberg, Berlin, Heidelberg.

[44] Solomon, M. M. (1987): Algorithms for the Vehicle Routing and Scheduling Problems with Time Window Constraints. In: *Operations Research*, 35, 2, pp. 254–265.

[45] Subramanian, A.; Drummond, L.; Bentes, C.; Ochi, L. S.; Farias, R. (2010): A parallel heuristic for the Vehicle Routing Problem with Simultaneous Pickup and Delivery. In: *Computers & Operations Research*, 37, 11, pp. 1899–1911.

[46] Subramanian, A.; Uchoa, E.; Ochi, L. S. (2010): New Lower Bounds for the Vehicle Routing Problem with Simultaneous Pickup and Delivery. In: P. Festa, ed., *Experimental algorithms*, vol. 6049 of *Lecture Notes in Computer Science*, pp. 276–287, Springer, Berlin.

[47] Subramanian, A.; Uchoa, E.; Pessoa, A. A.; Ochi, L. S. (2013): Branch-cut-and-price for the vehicle routing problem with simultaneous pickup and delivery. In: *Optimization Letters*, 7, 7, pp. 1569–1581.

[48] Tang Montané, F. A.; Galvão, R. D. (2006): A tabu search algorithm for the vehicle routing problem with simultaneous pick-up and delivery service. In: *Computers & Operations Research*, 33, 3, pp. 595–619.

[49] Tao, Y.; Wang, F. (2015): An effective tabu search approach with improved loading algorithms for the 3L-CVRP. In: *Computers & Operations Research*, 55, pp. 127–140.

[50] Tarantilis, C. D.; Zachariadis, E. E.; Kiranoudis, C. T. (2009): A Hybrid Metaheuristic Algorithm for the Integrated Vehicle Routing and Three-Dimensional Container-Loading Problem. In: *IEEE Transactions on Intelligent Transportation Systems*, 10, 2, pp. 255–271.

[51] Tasan, A. S.; Gen, M. (2012): A genetic algorithm based approach to vehicle routing problem with simultaneous pick-up and deliveries. In: *Computers & Industrial Engineering*, 62, 3, pp. 755–761.

[52] Toth, P.; Vigo, D. (1996): A Heuristic Algorithm for the Vehicle Routing Problem with Backhauls. In: L. Bianco; P. Toth, eds., *Advanced Methods in Transportation Analysis*, pp. 585–608, Springer Berlin Heidelberg, Berlin, Heidelberg.

[53] Wang, C.; Mu, D.; Zhao, F.; Sutherland, J. W. (2015): A parallel simulated annealing method for the vehicle routing problem with simultaneous pickup–delivery and time windows. In: *Computers & Industrial Engineering*, 83, pp. 111–122.

[54] Wang, H.-F.; Chen, Y.-Y. (2012): A genetic algorithm for the simultaneous delivery and pickup problems with time window. In: *Computers & Industrial Engineering*, 62, 1, pp. 84–95.

[55] Wang, L.; Guo, S.; Chen, S.; Zhu, W.; Lim, A. (2010): Two Natural Heuristics for 3D Packing with Practical Loading Constraints. In: B.-T. Zhang; M. A. Orgun, eds., *PRICAI 2010: trends in artificial intelligence*, vol. 6230 of *Lecture notes in computer science Lecture notes in artificial intelligence*, pp. 256–267, Springer, Berlin.

[56] Wassan, N. A.; Nagy, G.; Ahmadi, S. (2008): A heuristic method for the vehicle routing problem with mixed deliveries and pickups. In: *Journal of Scheduling*, 11, 2, pp. 149–161.

[57] Wassan, N. A.; Wassan, A. H.; Nagy, G. (2008): A reactive tabu search algorithm for the vehicle routing problem with simultaneous pickups and deliveries. In: *Journal of Combinatorial Optimization*, 15, 4, pp. 368–386.

[58] Wei, L.; Zhang, Z.; Lim, A. (2014): An Adaptive Variable Neighborhood Search for a Heterogeneous Fleet Vehicle Routing Problem with Three-Dimensional Loading Constraints. In: *IEEE Computational Intelligence Magazine*, 9, 4, pp. 18–30.

[59] Wisniewski, M. A.; Ritt, M.; Buriol, L. S. (2011): A tabu search algorithm for the capacitated vehicle routing problem with three-dimensional loading constraints. In: *XLIII Simposio Brasilero de Pesquisa Operacional*.

[60] Zachariadis, E. E.; Tarantilis, C. D.; Kiranoudis, C. T. (2009): A hybrid metaheuristic algorithm for the vehicle routing problem with simultaneous delivery and pick-up service. In: *Expert Systems with Applications*, 36, 2, pp. 1070–1081.

[61] Zachariadis, E. E.; Tarantilis, C. D.; Kiranoudis, C. T. (2010): An adaptive memory methodology for the vehicle routing problem with simultaneous pick-ups and deliveries. In: *European Journal of Operational Research*, 202, 2, pp. 401–411.

[62] Zhang, T.; Chaovalitwongse, W. A.; Zhang, Y. (2012): Scatter search for the stochastic travel-time vehicle routing problem with simultaneous pick-ups and deliveries. In: *Computers & Operations Research*, 39, 10, pp. 2277–2290.

[63] Zhang, T.; Chaovalitwongse, W. A.; Zhang, Y. (2014): Integrated Ant Colony and Tabu Search approach for time dependent vehicle routing problems with simultaneous pickup and delivery. In: *Journal of Combinatorial Optimization*, 28, 1, pp. 288–309.

[64] Zhang, Z.; Wei, L.; Lim, A. (2015): An evolutionary local search for the capacitated vehicle routing problem minimizing fuel consumption under three-dimensional loading constraints. In: *Transportation Research Part B: Methodological*, 82, pp. 20–35.

[65] Zhu, W.; Qin, H.; Lim, A.; Wang, L. (2012): A two-stage tabu search algorithm with enhanced packing heuristics for the 3L-CVRP and M3L-CVRP. In: *Computers & Operations Research*, 39, 9, pp. 2178–2195.

Otto von Guericke University Magdeburg
Faculty of Economics and Management
P.O. Box 4120 | 39016 Magdeburg | Germany

Tel.: +49 (0) 3 91 / 67-1 85 84
Fax: +49 (0) 3 91 / 67-1 21 20

www.fww.ovgu.de/femm

ISSN 1615-4274