WORKING PAPER SERIES

The split delivery vehicle routing problem with three-dimensional loading constraints

Andreas Bortfeldt/Junmin Yi

Working Paper No. 12/2018

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:
Andreas Bortfeldt
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

The split delivery vehicle routing problem with three-dimensional loading constraints

Andreas Bortfeldt[1] and Junmin Yi[2]
[1]Otto-von-Guericke-University Magdeburg, Germany, e-mail: andreas.bortfeldt@ovgu.de
[2]Xiamen University of Technology, China, e-mail: yijunmin@xmut.edu.cn

Abstract: The Split Delivery Vehicle Routing Problem with three-dimensional loading constraints (3L-SDVRP) combines vehicle routing and three-dimensional loading with additional packing constraints. In the 3L-SDVRP splitting deliveries of customers is basically possible, i.e. a customer can be visited in two or more tours. We examine essential problem features and introduce two problem variants. In the first variant, called 3L-SDVRP with *forced* splitting, a delivery is only split if the demand of a customer cannot be transported by a single vehicle. In the second variant, termed 3L-SDVRP with *optional* splitting, splitting customer deliveries can be done any number of times. We propose a hybrid algorithm consisting of a local search algorithm for routing and a genetic algorithm and several construction heuristics for packing. Numerical experiments are conducted using three sets of instances with both industrial and academic origins. One of them was provided by an automotive logistics company in Shanghai; in this case some customers per instance have a total freight volume larger than the loading space of a vehicle. The results prove that splitting deliveries can be beneficial not only in the one-dimensional case but also when goods are modeled as three-dimensional items.

Keywords: Vehicle routing, split delivery, 3D loading constraints, hybrid algorithm.

1 Introduction

In the distribution of goods and other logistic operations vehicle routing problems (VRP) are of eminent importance. In the Capacitated VRP (CVRP) as well as in numerous derived VRPs that address more complicated situations, each customer has to be visited only once. Thus, the entire demand must be delivered by one vehicle during one visit. In the Split Delivery VRP (SDVRP) this constraint is relaxed and the demand of a customer can be delivered by two or more vehicles. Both numerical experiments with solution methods and theoretical results prove that large savings in terms of travel distance can be obtained if splitting deliveries is allowed (see, e.g., Archetti and Speranza, 2012).

In this paper, we study whether splitting deliveries remains beneficial, if the SDVRP is extended to the SDVRP with three-dimensional (3D) loading constraints (3L-SDVRP). The CVRP with three-dimensional loading constraints (3L-CVRP) was introduced by Gendreau et al. (2006). Contrasting to the classical CVRP, customer demands are given by sets of rectangular shaped pieces (boxes) and the scalar capacity of vehicles is changed to a 3D loading space. Moreover, packing constraints frequently occurring in general cargo transportation concerning, for example, the stability of packed goods, can be included. A solution consists of routes and corresponding packing plans where each packing plan stows the boxes of all customers visited in the same route. Extending “pure” routing problems to vehicle routing problems with 3D loading constraints (3L-VRP) generally allows for a more realistic handling of real-word routing problems where general cargo is to be delivered or collected (Pollaris et al., 2015).

The 3L-SDVRP has been treated very rarely in the literature so far. Ceschia et al. (2013) proposed a local-search algorithm for solving the 3L-CVRP and 3L-SDVRP. While they achieved good results for the 3L-CVRP no distance savings could be obtained by permitting

split deliveries.

In the following, we will examine essential features and variants of 3L-SDVRP and propose a hybrid algorithm for solving the 3L-SDVRP. The algorithm should be able to generate good solutions with a simple structure of packing plans within short running times. We apply our algorithm to three sets of instances. The data of the first set stem from a Shanghai automotive logistics company. The second set of instances comes from Ceschia et al. (2013) with data originated from an Italian industrial partner. The third set is constructed using well-known benchmark instances for the CVRP and the container loading problem (CLP), respectively.

The paper is organized as follows. In Section 2, the relevant literature is reviewed and in Section 3, we identify essential features and variants of the 3L-SDVRP and formulate this problem with all details. In Section 4, we introduce a hybrid algorithm for the 3L-SDVRP. Section 5 is dedicated to the numerical experiments. Finally, in Section 6 we present concluding remarks and an outlook to future research.

2 Literature review

In this section, the literature concerning the SDVRP and 3L-VRPs is reviewed and existing approaches to the 3L-SDVRP are shortly described.

2.1 Vehicle routing problem with split deliveries

The SDVRP is defined, roughly spoken, as the classical CVRP except that splitting deliveries is allowed. Thus, a customer may be visited by several vehicles if beneficial. Moreover, customers can exist with large demands exceeding the vehicle capacity; obviously, this situation requires the possibility to split deliveries.

The SDVRP has been introduced by Dror and Trudeau (1989, 1990). The problem was extensively analyzed by, e.g., Archetti et al. (2008) and Archetti et al. (2011). Interesting results relate for example to worst-case analyses in order to quantify how much worse a solution without split deliveries can be compared to SDVRP solutions. Exact approaches were proposed amongst others by Dror et al. (1994), Belenguer et al. (2000), and Moreno et al. (2010). The Branch-and-Cut-and-Price algorithm by Archetti et al. (2014) and the method proposed by Ozbaygin et al. (2018) based on a new vehicle-indexed flow formulation belong currently to the strongest exact methods. Since only small instances can be solved using exact methods so far, the SDVRP is mostly tackled by metaheuristic methods. These include amongst others local search (Dror and Trudeau, 1989, 1990), tabu search (TS, Archetti et al., 2006; Berbotto et al., 2014), genetic algorithm (GA, Wilck and Cavalier, 2012), hybrid metaheuristics (Rajappa et al., 2016) and particle swarm optimization (Shi et al., 2018). For surveys of the SDVRP literature, the reader is referred to Archetti and Speranza (2012) and Irnich et al. (2014).

2.2 Vehicle routing problems with 3D loading constraints

Pollaris et al. (2015) survey the state of the art in the area of combined vehicle routing and loading problems (see also Iori and Martello, 2010).

The 3L-CVRP was introduced by Gendreau et al. (2006) and motivated by a real furniture distribution decision in Italy. Besides geometrical constraints, exact-one-visit condition and a weight constraint they considered four loading constraints on orientation, fragility, vertical

stability and last-in-first-out policy (LIFO). The 3L-CVRP with these constraints is called here the *Gendreau formulation* of 3L-CVRP.

Quite a few heuristic approaches for solving the 3L-CVRP in Gendreau formulation have been proposed in recent years. Most of the solution methods are hybrids consisting of a metaheuristic algorithm for routing and one or more relatively simple heuristic procedures for packing such as deepest-bottom-left-fill. Used metaheuristic strategies for routing are among others TS (Gendreau et al., 2006, Bortfeldt, 2012, Wisniewski et al., 2011, Zhu et al, 2012), a combination of TS and guided local search (Tarantilis et al., 2009), ant colony optimization (Fuellerer et al., 2010), a combination of GA and TS (Miao et al., 2012), adaptive variable neighborhood search (Wei et al., 2014), evolutionary local search (Zhang et al., 2017) and column generation technique-based heuristics (Mahvash et al., 2017). Some more elaborated and effective packing procedures were employed by Gendreau et al. (2006), Bortfeldt (2012), Tao and Wang (2015) and Zhang et al. (2017).

A model and exact approach of the 3L-CVRP was provided by Junqueira et al. (2013); this model has additional constraints compared to the Gendreau formulation and only small instances with nodes up to 15 and boxes up to 32 were solved. Hokama et al. (2016) proposed two Branch-and-Cut algorithms for 3L-CVRP variant with just LIFO constraint, but no fragility and stability constraints. Instances like E101-14s (Gendreau et al., 2006) with 100 nodes and 198 boxes were solved by the exact methods of Hokama et al. (2016) in which the branch and cut enumeration tree is pruned using different techniques.

Other features are also considered in combination with 3D loading constraints such as time windows by, e.g., Moura and Oliveira (2009), pallet loading by Zachariadis et al. (2012) and Zhang et al. (2017), backhauls by, e.g., Reil et al. (2018) and Koch et al. (2018), heterogeneous fleet by Pace et al. (2015), pick-up and delivery by, e.g., Bartók and Imre (2011) and Männel and Bortfeldt (2016).

2.3 Approaches to 3L-SDVRP

To the best of our knowledge, there are only four works that involve the possibility of splitting the customer's demands in a routing-packing problem context.

Moura and Oliveira (2009) proposed two methods for the vehicle routing problem with time windows and 3D loading constraints (3L-VRPTW). In the first method (called sequential method) the authors relax the condition that each customer has to be visited only once. Therefore we can view this method as an initial approach for solving the 3L-SDVRP.

Ceschia et al. (2013) deal with three problems. The first one is the 3L-CVRP in Gendreau formulation. The second problem is an extended 3L-CVRP where more difficult packing constraints, namely load bearing strength, robust stability, and reachability, are required (for details see Section 3). The third problem is the 3L-SDVRP with same packing constraints as in the second one. Ceschia et al. tackle all three problems by a local search approach that is single-staged and based on a composite strategy combining simulated annealing and large neighborhood search. Good results are achieved for well-known benchmark instances for the 3L-CVRP. To test their heuristic for the second and third problem they use 13 instances derived from practice with up to 129 customers, up to 8060 boxes and limited vehicle fleet. The local search approach does not reach feasible solutions for all instances of the extended 3L-CVRP. In case of 3D-SDVRP for all instances feasible solutions are provided, i.e. split delivery helped to achieve the missing feasible solutions. However, the results with split

delivery are worse than those without split delivery, which is in contrast to the classical SDVRP. Furthermore, relatively long running times of up to 10,000 seconds are reported.

Yi and Bortfeldt (2018) address the 3L-SDVRP with the same packing constraints as the 3L-CVRP in Gendreau formulation. It is assumed that the demand of some customers has a volume larger than the volume of one loading space. On the other hand only inevitable splits are allowed, i.e. serving a customer in two or more routes is only permitted if not all boxes can be packed into a single loading space. A hybrid heuristic consisting of a TS method for routing and several packing heuristics is developed. The heuristic works in two main steps. First small routes with one or two customers are generated in order to serve customers with large demand that does not fit in a single loading space. Then the residual problem is solved as 3L-CVRP. The heuristic is tested by a set of 11 instances from an automotive logistics company in Shanghai with up to 42 sites and 1549 boxes. Actually the goods have not to be delivered to the customer sites but have to be collected from them by homogeneous vehicles. Good results are achieved in small running times.

Finally, Li et al. (2018) propose a novel data-driven three-layer search algorithm to solve the 3L-SDVRP. They minimize the number of used vehicles with first and the total travel distance with second priority. Also, Li et al. add a packing material constraint and other process constraints for their industrial scenario in which an average delivery order contains more than 300 boxes to be distributed across the Pearl River Delta region. Their approach is organized in three parts for vehicle routing, cargo splitting and container loading. It is designed to learn from historic routing data and thus to improve both the efficiency and effectiveness of traditional meta-heuristics approaches. Unfortunately, very small instances with only six customers are used for testing their approach.

All in all, only limited research has been done on the 3L-SDVRP. In particular it has not been examined sufficiently whether splitting deliveries can be advantageous if three-dimensional loading constraints are required.

3 Essential features, problem variants and formulation of the 3L-SDVRP

We consider essential features and problem variants of the 3L-SDVRP before a complete problem formulation is given.

3.1 Essential features and problem variants of the 3L-SDVRP

The 3L-SDVRP is basically defined as the 3L-CVRP. However, it is no longer required that each customer is visited only once. There are two reasons to split the delivery of a customer: the splitting can be forced and it can be done optionally to gain a benefit. We consider both variants of splitting in detail.

Forced splitting

Forced splitting occurs if the load of a customer, given by a set of boxes, cannot be stowed in a single vehicle in terms of volume or weight. In the one-dimensional (1D) SDVRP splitting is forced if and only if the demand of a customer exceeds the volume or weight capacity of a vehicle. Thus, splitting only depends on the data of a given SDVRP instance. In the 3L-SDVRP the same applies with regard to the weight of a customer's demand. However, the volume of a customer's demand as reason for (forced) splitting is a little more difficult. Of course, if the total volume of a customer's demand is larger than the volume of the loading

space of a vehicle, splitting has to be done (such customers are called *big nodes* afterwards). Let β_i be the ratio of the demand's volume d_i of customer i and the loading space volume Q (in %, $i = 1,\ldots,n$). If β_i is less than 100%, the necessity of splitting depends not only on the instance data. For example, if $\beta_i = 85\%$ for customer i, a weaker packing algorithm might be unable to stow all boxes while a more elaborated packing algorithm can do and avoid splitting in this case. Hence the necessity of splitting in the 3D case does depend on the instance data but also on the packing algorithm employed. Correspondingly, the set of big nodes of an instance is in general a *proper* subset of the set of customers where splitting is inevitable.

In the problem variant 3L-SDVRP-f ("f" as *forced*), splitting customer deliveries is only allowed if the demand of a customer cannot be stowed in a single vehicle by means of the used packing algorithm. Of course, this definition is extended in a natural way to the case where the load of a customer does not fit in two vehicles etc., i.e. only the minimum number of demand splits per customer is permitted. The subtype 3L-SDVRP-f corresponds to a management point of view that can be outlined as follows: Splitting of demands requires organizational and working power effort. Hence splits are to be avoided whenever possible while the number of routes (v) and the total travel distance (*ttd*) are to be reduced by other means than by splitting demands.

Optional splitting

Optional splitting occurs if the load of a customer fits in a single vehicle (in terms of volume and weight) but nevertheless splitting is done because it is beneficial to serve the customer in two or more routes, i.e. the number of needed routes and the *ttd* can be reduced by splitting.

In the problem variant 3L-SDVRP-o ("o" as *optional*), splitting customer deliveries can be done any number of times. Again, the judgment whether a solution for a 3L-SDVRP instance is a 3L-SDVRP-o solution cannot be done without considering the used packing algorithm. The subtype 3L-SDVRP-o corresponds to a management point of view that can be sketched as follows: Use splitting as a means of reducing the total costs given by the *sum* of splitting costs and routing costs (based on *ttd* and/or number of routes). If costs can be saved, splitting deliveries should be applied. Of course, optional splitting is worthless if it leads to increasing routing costs.

Irnich et al. (2014) present several variants of the (1D-)SDVRP that are used for worst-case analyses. Some of them are quoted in Table 1 as we would like to find relations to useful variants of the 3L-SDVRP.

Table 1: Variants of (1D-)SDVRP following Irnich et al. (2014).

Variant	Description
SDVRP	Demands: $q_i \le Q$, $i = 1,\ldots,n$. Visits: the number of visits to a customer can be any positive integer.
$SDVRP^+$	Demands: at least one customer j has a demand $q_j > Q$ Visits: the number of visits to a customer can be any positive integer
VRP^+	Demands: at least one customer j has a demand $q_j > Q$. Visits: the number of visits to a customer i is given by $\lceil q_i / Q \rceil$ and the minimum delivery amount (MDA) per visit is given by (q_i mod Q)
H^{SDVRP+}	As $SDVRP^+$; in addition it is assumed that each demand $q_i > Q$ is first split according to $q_i = \lfloor q_i/Q \rfloor \cdot Q + (q_i \bmod Q)$. Customer i is then served $\lfloor q_i/Q \rfloor$ times with a full load Q in a direct trip. The remaining problem with demands (q_i mod Q) has to be solved as SDVRP.
H^{VRP+}	As H^{SDVRP+} but for the remaining demands the MDA is given as in VRP^+, i.e. the remaining problem has to be solved as CVRP.

In the SDVRP the vehicle fleet is mostly assumed to be homogeneous and unlimited and we will *only* consider such variants. We denote the number of customers by n, the vehicle capacity by Q and the customer demands by q_i, $i = 1,\ldots,n$. Based on the 1D-SDVRP variants indicated in Table 1 and the above definitions, four variants of the 3L-SDVRP are listed in Table 2. As indicated in Table 2 the 1D variants SDVRP and SDVRP$^+$ are integrated in the 3D case because of the "fuzzy" character of customers' demands that are to be split. The variants $H^{3L\text{-}SDVRP\text{-}o}$ and $H^{3L\text{-}SDVRP\text{-}f}$ represent heuristic approaches and specify that demands to be split are partially to be delivered by direct trips.

Table 2: Variants of 3L-SDVRP.

Variant	Description	Related 1D-SDVRP variant
3L-SDVRP-o	Optional splitting is allowed besides forced splitting.	SDVRP, SDVRP$^+$
3L-SDVRP-f	Only forced splitting is allowed and forced splitting is necessary for at least one customer.	VRP$^+$
$H^{3L\text{-}SDVRP\text{-}o}$	For each customer direct trips are to be planned (where necessary) until the residual demand can be packed in one vehicle. The residual problem has to be solved as 3L-SDVRP-o.	H^{SDVRP+}
$H^{3L\text{-}SDVRP\text{-}f}$	As $H^{3L\text{-}SDVRP\text{-}o}$ but the residual problem has to be solved as 3L-CVRP.	H^{VRP+}

3.2 Problem formulation

In this section, we describe the 3L-SDVRP more formally. We assume that n customers and a single depot (with index 0) are given. Each customer has a set I_i of boxes with known dimensions that are to be transported from the depot to the customer ($i = 1,...,n$). The set I_i includes m_i boxes I_{ik} ($k = 1,...,m_i$) and box I_{ik} has length l_{ik}, width w_{ik}, and height h_{ik} ($i = 1,...,n$, $k = 1,...,m_i$).

Let $V = \{0,1,..,n\}$ denote the set of all customers (also called nodes) including the depot. Let E be a set of undirected edges (i,j) connecting all node pairs ($0 \le i < j \le n$) and let $G = (V, E)$ be the resulting graph. Let a distance d_{ij} ($d_{ij} \ge 0$) be assigned to each edge (i,j) ($0 \le i < j \le n$), i.e. symmetric distances are given. Finally, there is an unlimited fleet of identical vehicles with a rectangular loading space with length L, width W, and height H. Each vehicle is rear-loaded.

The loading space of each vehicle is embedded in the first octant of a Cartesian coordinate system in such a way that the length, width and height of the loading space lie parallel to the x-, y-, and z-axis, respectively. The placement of a box I_{ik} in a loading space is given by the coordinates x_{ik}, y_{ik}, and z_{ik} of the box corner that is closest to the origin of the coordinates system; in addition, an orientation index o_{ik} indicates which of the possible spatial orientations is selected ($i = 1,...,n$, $k = 1,...,m_i$). A spatial orientation of a box is given by a one-to-one mapping of the three box dimensions and the three coordinate directions.

A packing plan P for a loading space comprises one or more placements and is regarded as feasible if the following three conditions hold: (FP1) each placed box lies completely within the loading space; (FP2) any two boxes that are placed in the same truck loading space do not overlap; (FP3) each placed box lies parallel to the surface areas of the loading space. Figure 1 shows a loading space with placed boxes.

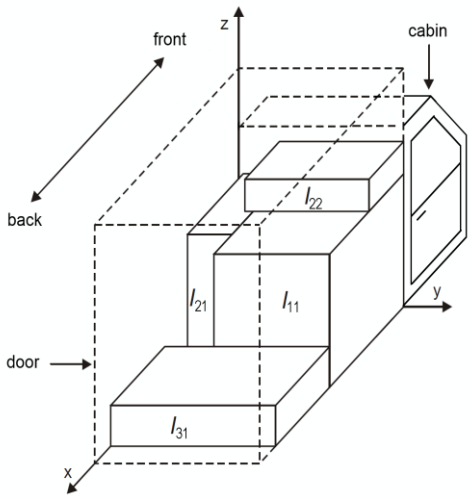

Figure 1: A loading space with placed boxes.

A route R is a sequence $(0, c_1,...,c_p, 0)$ that starts and ends at the depot. It is feasible if it includes $p \geq 1$ pairwise different customers $(0 < c_i \leq n, i = 1,...,n)$.

A solution of the 3L-SDVRP is a set of v ($v \geq 1$) pairs (R_μ, P_μ), where R_μ is a route and $P_{\mu,}$ is the corresponding packing plan ($\mu = 1,...,v$). A solution is called feasible if the following conditions are satisfied:

(F1) All routes R_μ and all packing plans $P_{\mu,}$ are feasible ($\mu = 1,...,v$).

(F2) Each customer occurs at least once in the routes R_μ ($\mu = 1,...,v$); in problem variant 3L-SDVRP-f the number of occurrences of a customer i is limited by the minimum number of vehicles needed to pack the set of boxes I_i ($i = 1,\ldots,n$); in problem variant 3L-SDVRP-o the number of occurrences of a customer i is not limited ($i = 1,\ldots,n$).

(F3) The packing plan $P_{\mu,}$ contains only placements of boxes that belong to customers visited in route R_μ ($\mu = 1,...,v$); each box I_{ik} ($i = 1,\ldots,n$, $k = 1,\ldots,m_i$) is packed in exactly one packing plan $P_{\mu,}$ ($\mu = 1,...,v$).

In addition, the weight constraint and following loading constraints are integrated:

(C1) Weight Limit: Each box I_{ik} has a positive weight db_{ik} ($i = 1,...,n$, $k = 1,\ldots,m_i$) and the total weight of all boxes in a packing plan P_μ must not exceed a maximum load weight D ($\mu = 1,...,v$).

(C2) LIFO Policy: Let c and c' be two customers and c is visited before c' in route μ ($\mu \in \{1,...,v\}$); let b and b' two boxes that belong to c and c', respectively. Then b' cannot be placed in packing plan P_μ between b and the rear of the vehicle or above b.

By this constraint it is ensured that all boxes of each customer can be unloaded by pure shifts in x-direction without moving other boxes.

(C3) Fixed Vertical Orientation: The height dimension of all boxes is fixed while horizontal 90° turns of boxes are allowed. Thus, only two of six values are allowed for the orientation index o_{ik} of a placement ($i = 1,...,n$, $k = 1,...,m_i$).

(C4) Vertical Stability: If a box is not placed on the vehicle floor, a certain percentage a of its base area has to be supported by other boxes.

(C5) Fragility: A fragility attribute f_{ik} ($i = 1,...,n$, $k = 1,...,m_i$) is assigned to each box. If a box is fragile ($f_{ik} = 1$) only other fragile boxes may be placed on its top surface, whereas both fragile and non-fragile boxes may be stacked on a non-fragile box ($f_{ik} = 0$).

Finally, the 3L-SDVRP consists of determining a feasible solution that meets the constraints (C1) – (C5) and minimizes firstly the number of routes and secondly (with lower priority) the total travel distance. The formulation applies to all variants of the 3L-SDVRP shown in Table 2, where obvious conditions are to be added for the variants $H^{3L\text{-}SDVRP\text{-}o}$ and $H^{3L\text{-}SDVRP\text{-}f}$.

As mentioned earlier, the constraints (C1) – (C5) were just adopted from the Gendreau formulation of the 3L-CVRP. For the convenience of the reader we quote here also the new loading constraints that were integrated by Ceschia et al. in their 3L-CVRP formulation (see Ceschia et al., 2013, pp. 1139-1141):

(C6) Load bearing strength: There is a maximum weight per unit area that a box can uphold depending on its type and its vertical orientation. This constraint is thought to replace constraint (C5).

(C7) Robust stability: A minimum supporting area has to be guaranteed for all items below the current one in the stack. It means that for a given item constraint (C4) is to require not only for the underlying item but also for all items below it. This constraint serves to replace constraint (C4).

(C8) Reachability: An item is considered reachable if the distance between it and a human operator or a forklift is less than or equal to a fixed length λ. It is supposed that the operator is placed as close as possible to items inside the vehicle, i.e. the position with the minimum length with respect to the current loading.

4 Hybrid algorithm

In this section, the hybrid algorithm for solving the 3L-SDVRP is described. It can be applied to both problem variants, namely the variant with forced splits and with optional splits.

4.1 Main steps of the hybrid algorithm

The hybrid algorithm works in two main steps, the packing step and the routing step, which are done strictly one after another and not in an interlocked fashion.

4.1.1 The packing step

In the packing step, 3D packing patterns are generated taking into account the entire demand of all customers. Each 3D packing pattern fits in a single loading space and consists of one or more vertical layers or walls that follow one another along the length of a loading space. The width and height of a layer are given by the width and height of the loading space, respectively. The layer length *llayer* is given by the difference of the largest and the smallest *x*-coordinate of the layer. Both coordinates have to be taken by at least one box of the layer. The pattern length *lpattern* is given as the sum of the lengths of all layers in the pattern. A packing pattern with four layers is depicted in Figure 2. For each customer one segment pattern (1C-SP) *must* and some further full load patterns (1C-FLPs) *can* be built. If multiple packing patterns are built the pattern with the lowest filling rate is taken as 1C-SP and the others are taken as 1C-FLPs. Together the patterns of a customer include all demanded boxes.

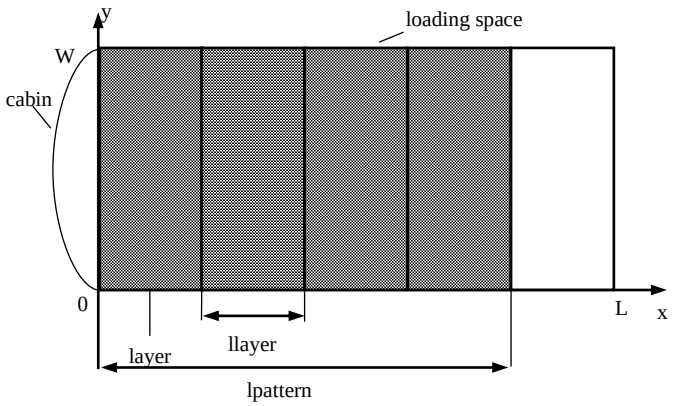

Figure 2: Loading space and packing pattern (top view).

Big nodes have at least one 1C-FLP and also other customers can have one or more 1C-FLPs, which are then used for direct trips (see below). A segment pattern fills a loading space in general only partly. The segment patterns of multiple customers can be combined in one loading space if and only if the sum of the pattern lengths of all related segment patterns does not exceed the loading space length *L*.

Often the filling rates of segment patterns are relatively low since only boxes of a single customer are available for packing. Therefore, additional segment patterns, which store all boxes of two chosen customers, are built and termed 2C-SP patterns. Again, a 2C-SP consists of vertical layers, but now in a layer boxes of two customers can be placed. Later the two individual segment patterns of two customers can be replaced by the related 2C-SP. This can be done if the customers are visited one after another and might be advantageous if the length of the 2C-SP is smaller than the sum of the pattern lengths of the related 1C-SPs. In Figure 3 the pattern types 1C-SP and 2C-SP are illustrated. The boxes in the left 1C-SP all belong to customer 1 while the boxes in the right 2C-SP belong to customers 3 and 2 and there is a mixed layer with boxes from both customers.

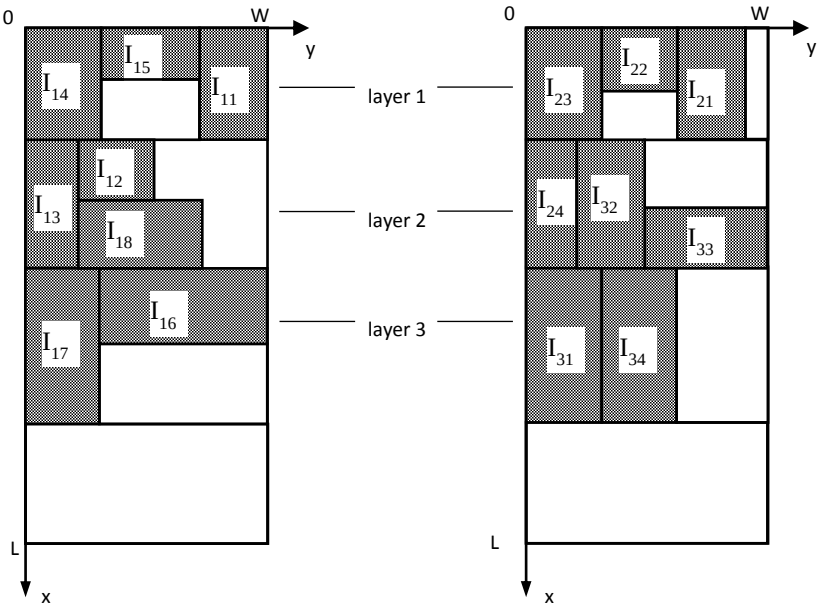

Figure 3: Pattern types 1C-SP and 2C-SP.

Since the customers of a 2C-SP have to be visited in direct succession only pairs of

customers are considered that are not too far apart from each other. For a customer c_1 only the *qnb*% of all customers c_2 are considered for 2C-SP (and called neighbors of c_1) which lead to the smallest insertion costs given as $d_{01}+d_{12}-d_{02}$ (0: depot, *qnb*: a parameter). A 2C-SP for the pair (c_1, c_2) is constructed in such a way that c_1 can be visited first and c_2 can be visited second regarding constraint (C2).

All generated patterns (1C-FLP, 1C-SP and 2C-SP) are made available to the routing step. Above all the following data are provided per pattern: 1) pattern type, 2) relevant customer(s), 3) pattern length, 4) number of layers and 5) length of each individual layer. In the generation of packing patterns the geometrical constraints (FP1)-(FP3) and the packing constraints (C3)-(C5) are observed. The routing step is mainly responsible for the weight constraint (C1) and the LIFO policy (C2). The generation of patterns is carried out by a genetic algorithm (1C-FLP, 1C-SP) and two construction heuristics (2C-SP).

4.1.2 The routing step

In the routing step first direct trips (depot→customer→depot) are constructed if there are full load patterns for one or more customers. Afterwards a reduced problem remains to be solved that can be characterized as follows:

- All boxes stowed in 1C-FLP and delivered by direct trips must not be considered any longer.
- Having packed all demanded boxes in packing patterns with a given pattern length only a one-dimensional VRP remains to be solved since the packing patterns (1C-SP, 2C-SP) can now be replaced by their pattern lengths.
- Thus, in this one-dimensional VRP two capacity restrictions are to be observed: the weight constraint (C1) and the length capacity constraint (LCC); the latter results from the packing step and requires that the sum of the pattern lengths of all 1C-SPs and 2C-SPs transported in the same route must not exceed the length L of the vehicle's loading space.

So far only *forced splits* have been implemented where appropriate and are represented for a given customer by his/her 1C-FLPs. Each of the 1C-SP and 2C-SP patterns can be accommodated in a single loading space. Hence, no further splits are necessary. Therefore, in the problem variant 3L-SDVRP-f only a one-dimensional CVRP remains to be solved in the routing step. This is done by a local search procedure. In problem variant 3L-SDVRP-o optional splits can be carried out and thus a one-dimensional SDVRP remains to be solved.

Optional splits are done as follows. Often the segment patterns of the customers of a route do not completely fill the length of a loading space. There is a residual free length that, however, cannot be filled by all layers of another customer *c* not being a member of the route yet. In such cases it can be tried at least to pack a (proper) subset of the layers and to include customer *c* in this route. Of course, customer *c* must also be a member of another route whose loading space must accommodate the remaining layers of *c*. Thus, a "split customer" is always the last customer in a route and the first customer in a second route. Sometimes one or more of such splits allow to save an entire route and/or enable some distance saving. To carry out optional splits the local search procedure is modified appropriately utilizing the layer structure of packing patterns.

4.1.3 Overview and characterization of the approach

The hybrid algorithm is overviewed in Figure 4. After packing and routing in the final step (not mentioned before) the 3L-SDVRP solution is completed, i.e. direct trips and all other routes of the best achieved routing solution are combined with related 3D packing patterns.

Algorithm 3L-SDVRP-H1 (**in**: problem data, parameters, **out**: best solution s_{best})
    // Packing step
    for each customer *i* **do**
        generate patterns for customer *i* (1C-FLPs where necessary, one 1C-SP) by GA
    endfor
    for selected customer pairs (*i*,*j*) **do**
        generate 2C-SP pattern for customer pair (*i*,*j*) by construction heuristics
    endfor
    // Routing step
    for each customer *i*
        generate as many direct trips $0 \rightarrow i \rightarrow 0$ as 1C-FLPs for customer *i* do exist
    endfor
    if only forced splits allowed **then**
        solve remaining CVRP by local search
    else // optional splits allowed
        solve remaining SDVRP by (modified) local search
    endif
    // Final step
    prepare solution s_{best} consisting of best achieved routing plan and related 3D packing patterns
end.

Figure 4: Overview of the hybrid algorithm.

We would like to characterize the approach as follows. The packing and the routing task of the 3L-SDVRP are solved in separate steps following the principle "Packing first, Routing second" that has been successfully used recently by Reil et al. (2018) and ensures small running times. The structure of the generated packing patterns is very simple. Boxes are placed in vertical layers and each layer accommodates only boxes of one or two customers. Finally, for both 3L-SDVRP variants the algorithm follows the heuristic approach where customers with large demands are served by direct trips before the residual problem is tackled, i.e. the hybrid algorithm belongs to the type $H^{3L\text{-}SDVRP\text{-}f}$ or $H^{3L\text{-}SDVRP\text{-}o}$ (see Table 1 and 2). In the following, both main steps of the hybrid algorithm are considered in greater detail.

4.2 Generating packing patterns

The generation of packing patterns is done in three steps. In a preprocessing step stacks of boxes are built from the boxes of all customers. Afterwards patterns for all customers are created by placing directly stacks instead of original boxes. Patterns of types 1C-FLP and 1C-SP are built in the second step while 2C-SP patterns are constructed in the third step.

4.2.1 Building stacks from boxes

Stacks are built from the original boxes and afterwards used *as boxes* to generate packing patterns in order to cope with large box sets in small running times. For each customer the boxes are transformed into stacks individually, i.e. a stack includes only boxes of a single customer. To handle a stack as a box in the pattern generation, the following data of a stack are provided: 1) the related customer, 2) the dimensions of the bounding cube of the stack, i.e. its length, width and height, 3) the weight, i.e. the sum of the weights of the included boxes, 4) the fragility, given by the fragility of the topmost (original) box. Only in the final step of the hybrid algorithm when the complete solution is prepared, all placed stacks are transformed back into placed original boxes. The procedure for transforming original boxes of a given customer into stacks is explained in the appendix.

4.2.2 Building patterns of types 1C-FLP and 1C-SP

For each customer 1C-FLP and 1C-SP patterns are separately generated by the procedure depicted in Figure 5.

```
Algorithm generate_1C_patterns(in: customer c, problem data, parameters, out: set of patterns P(c))
    B(c) := {boxes of customer c}
    P(c) := ∅
    do
        solve the container loading problem for container L, W, H and B(c)
            by means of GA CBGAS and get a pattern p
        P(c) := P(c) U {p}
        update B(c), i.e. remove all boxes placed in pattern p
    while ( B(c) ≠ ∅ )
    label the pattern p with lowest filling rate in P(c) as 1C-SP pattern
    label the other patterns (if any) as 1C-FLP patterns
end.
```

Figure 5: Building patterns for a single customer.

In each cycle of the loop a pattern is built, i.e. another vehicle loading space is filled, using the GA for container loading (called CBGAS) by Bortfeldt and Gehring (2001). The patterns are then marked as 1C-SP or 1C-FLP patterns. Here we want to outline only the main ideas and features of the GA:

Pattern structure: As mentioned above the GA generates patterns that consist of vertical layers following one after another along the length of the loading space (see Fig. 1). The length of each layer is given by the corresponding dimension of a so-called layer defining box (LDB) placed in a suitable spatial orientation.

Genetic operators: Complete solutions (patterns) are generated by means of problem specific genetic operators, namely a crossover and two variants of mutation. All operators first copy layers with high filling rates from parents to descendants. Since the descendents are mostly still incomplete then, new layers are generated afterwards by an internal construction heuristic that is also used to generate the initial population.

Generation of layers: A single layer is generated in two steps. First the layer is defined as the LDB and its spatial orientation is determined. Only large boxes are taken as LDBs. However, a variety of layers is reached by varying the LDB and its orientation. In the second step the layer is filled by the LDB and further boxes. Each box is placed in a corner of a cuboidal residual space and then three new residual spaces inside the one just filled and in front of, besides and above the placed box are constructed and filled afterwards if possible. Filling a layer is finished if no further residual space or no further box is available. To apply CBGAS to the 3L-SDVRP up to five of the largest residual spaces per layer that could not be filled are provided as a solution component in order to fill these spaces later by boxes of another customer. Only residual spaces are provided that are situated at the front side of its layer (nearer to the rear) and extend until the roof of the loading space.

Filling residual spaces: If a residual space is to be filled the most voluminous couple of boxes is determined that fits into the space. One of the boxes is immediately placed in the residual space. The other one is placed later in one of the derived residual spaces. This feature helps much to reach good quality solutions.

Fitness evaluation and selection: The fitness of a solution is derived from its filling rate while the selection of individuals for crosssover and mutation is done by a combination of

ranking selection and purely randomly selection.

Reproduction schema: generational replacement is applied, i.e. all individuals of an old generation are replaced by a complete new generation at once.

Observed constraints: The GA can be applied to the 3L-SDVRP as defined above, i.e. all relevant constraints can be observed by means of the GA.

4.2.3 Building patterns of type 2C-SP

For a customer c_1 and a neighboring customer c_2 two different procedures are applied to construct 2C-SPs that are based on the 1C-SPs for both customers. Seen from the rear first some layers of customer c_1 (c_1-layers for short) are chosen from the 1C-SP of c_1; then a (newly constructed) mixed layer with boxes from c_1 as from c_2 follows and the end of the pattern (nearer to the cabin) is formed by some layers of the 1C-SP for c_2. Figure 6 shows a 2C-SP generated by the first procedure on the left and a 2C-SP produced by the second procedure on the right.

The *first* procedure for generating 2C-SP is based on the provided residual spaces for the layers of the 1C-SPs. For each pair (c_1, c_2) one c_2-layer is selected and its residual spaces are taken to accommodate all the boxes of a c_1-layer to form a mixed layer. If this works the c_1-layer and its length are saved and the procedure terminates. For this purpose the c_1-layers are sorted by decreasing layer length and passed through in this order to make the saved layer length as large as possible. For each c_1-layer all c_2-layers are examined until all boxes of the current c_1-layer can be placed within residual spaces of the current c_2-layer or all c_2-layers were checked without success. Since the residual spaces of a used c_2-layer are always situated at the front of the layer (nearer to the rear) and this (mixed) layer follows directly the remaining c_1-layers the LIFO policy constraint (C2) is observed regarding the boxes of customers c_1 and c_2.

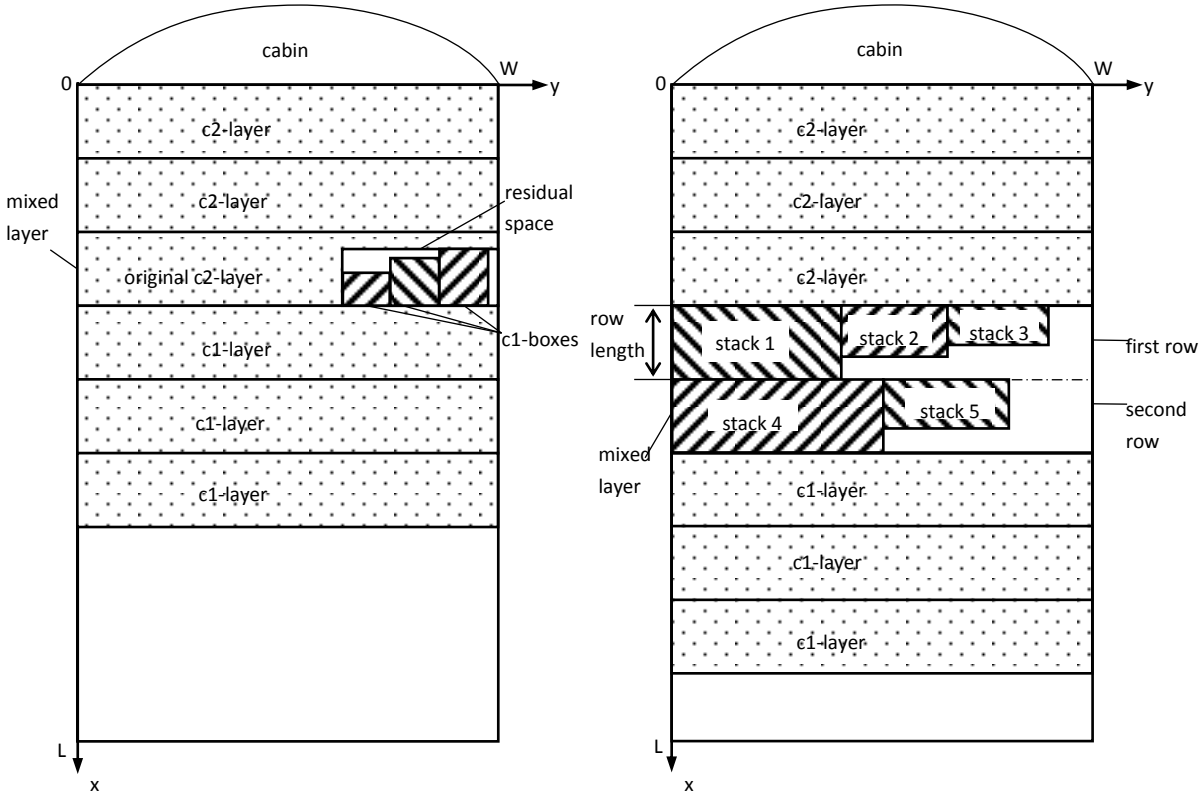

Figure 6: Two 2C-SPs generated by two different procedures.

In the *second* procedure for building 2C-SP first c_1-layers and c_2-layers of the related 1C-SPs are removed. The procedure is called two times; in the first call only the c_1-layer and the c_2-layer with worst filling rates are eliminated while in the second call two layers with the lowest filling rates are removed per customer. In any case the boxes of the removed layers are taken to build stacks where each stack includes boxes from c_1 as from c_2. The generation of stacks is done by a recursive procedure where each incomplete stack is extended by a next box in up to *maxcands* variants (*maxcands* is a parameter). The stacks, accommodating all boxes of the removed layers, are then placed in a new mixed layer that is again situated between the remaining c_1-layers (nearer to the rear) and the remaining c_2-layers (nearer to the cabin). More precisely, the stacks are placed in one or more rows running along the width of the loading space. In each stack boxes of customer c_1 are stowed above boxes of customer c_2; let h_{c2top} be the maximal height of the top face of boxes of customer c_2 in a stack. The stacks are sorted by decreasing height h_{c2top} before the rows are built in front of the first remaining c_2-layer. Through this the heights h_{c2top} of the stacks in the first row are as least as high as the heights h_{c2top} of the stacks in the second row etc. and constraint (C2) is met again.

At the end of all three trials for a customer pair (c_1,c_2), executed by the two 2C-SP procedures, only the 2C-SP with the largest saving δ is accepted if $\delta > 0$. The saving δ is defined as:

$$\delta(\text{2C-SP}(c_1,c_2)) := lpattern(\text{1C-SP}(c_1)) + lpattern(\text{1C-SP}(c_2)) - lpattern(\text{2C-SP}(c_1,c_2)). \quad (1)$$

Further details of the procedures for building 2C-SPs can be found in the appendix.

4.3 Routing algorithm

The routing algorithm is based on a representation of solutions as giant tours (cf. Vidal et al., 2014, p. 667) where any solution is given as a permutation of all n customers without explicit mention of visits to the depot. The giant tour representation allows the uniform handling of moves affecting single routes and pairs of routes. We will apply swap and shift moves for our local search. Moreover, this representation enables to manage both problem variants, 3L-SDVRP-f and 3L-SDVRP-o, in a uniform fashion. The routing algorithm is also characterized by the extensive use of diversification and post-optimization features. Subsequently the routing algorithm is described in a bottom-up fashion including the steps: (1) decoding a solution, (2) generating the initial solution, (3) determining the best neighbor of a current solution and (4) main algorithm of local search.

4.3.1 Decoding a solution

The decoding procedure gets as input a coded solution, i.e. a complete customer sequence. For each customer, the related 1D-SP pattern with its loading length, weight and all layers is needed. The 2C-SPs for customer pairs (c_1, c_2) must be also provided. Moreover, the parameter *max_split_costs* (see below) is required. The decoding procedure returns the complete solution with following features:

(1) the routes of the solution,
(2) all implemented customer pairs (CP), i.e. all pairs of consecutive customers for which a 2C-SP pattern is taken instead of two 1C-SP patterns,
(3) for problem variant 3L-SDVRP-o the customers whose delivery is split and the related layer distribution.

In our approach, a customer can be visited at most two times. If a customer is visited

twice then he/she must be the last customer in a route and the first customer in a second route. Besides the set of customers being visited twice a solution must include for each such customer the distribution of his/her cargo layers between both visits.

In the first step of decoding the CPs to be implemented are selected. For this purpose the potential CPs, i.e. all pairs of customers (c_1, c_2) in the coded solution for which a 2C-SP pattern exists are examined. Each customer can occur in multiple potential CPs while at most one pair per customer can be chosen for implementation. To maximize the total savings in terms of loading length a greedy approach is applied. The CP in the coded solution with the highest savings is selected first. Then the CP of the coded solution with the highest savings is selected whose customers do not already appear in the first pair, etc. The selection procedure for CPs is illustrated by an example in Figure 7.

Given customer sequence (n = 8):

2 7 1 4 5 3 8 6

Potential CPs, related savings (in length units) and selection:

c_1	c_2	savings δ	selection
1	4	12	yes
7	1	10	no (cust. 1 used before)
5	3	7	yes
3	8	5	no (cust. 3 used before)

Figure 7: Selection of CPs to be implemented (example).

In the second step of decoding the routes are determined and decisions on splitting deliveries are made where necessary. For that purpose the complete customer sequence is traversed (see Figure 8). Each cycle of the corresponding while-loop checks first, whether the current customer *c* at sequence place *ic* is the first of a CP or not. In the latter case, it is examined whether customer *c* fits into the current route *ir*, i.e. whether the 1C-SP of *c* fits into the loading space of route *ir* in terms of weight and loading length. If yes, route *ir* is assigned to customer *c* and the route is updated w.r.t. number of customers, needed loading length and packed weight. The decoding is continued with the same route *ir* and the next customer at place *ic*+1. If customer *c* does not fit completely into route *ir*, an attempt is made to split the delivery of *c*, i.e. to distribute the layers of the corresponding 1C-SP to the current route *ir* and next route *ir*+1. If this fails, the decoding is continued with the next route *ir*+1 while customer *c* remains to be processed. If at least one layer of *c* fits into the old route *ir* and splitting is not "too expensive" (see below), the routes *ir* and *ir*+1 are assigned to *c* who is labelled as a "split customer", i.e. *c* is the last customer in route *ir* and the first one in route *ir*+1. The routes *ir* and *ir*+1 are updated whereby the distribution of the *c*-layers to the routes has to be recorded, too. The decoding is continued with next customer and route *ir*+1 that has been already used for customer *c*.

A split delivery of a single customer *c* is tried as follows. First a travel cost checking is done in order to restrict the increase of travel distance caused by split deliveries. For a single customer it is checked whether the relative insertion costs of customer *c* in route *ir* do not exceed the parameter value *max_split_costs*, i.e.

$$(d_{cprev,c} + d_{c,0}) / d_{cprev,0} \leq max\_split\_costs \qquad (2)$$

where customer c_{prev} precedes *c*. If criterion (2) is not satisfied no splitting takes place.

```
Algorithm determine_routes (in: solution S, segment patterns, max_split_costs, out: S)
  initialize routes w.r.t. number of customers, needed loading length and packed weight
  // traverse complete customer sequence S.Cs
  ir := 1; ic := 1 // current route and current customer position
  while ic ≤ n do
    if current customer c = Cs(ic) does not belong to a customer pair then
      if c fits into current route ir then
        assign route ir to c and update route ir w.r.t. no. of customers, loading length, weight
        ic := ic + 1 // continue with next customer, route remains same
      else // c does not fit into route ir
        split delivery of c if necessary, distribute c-layers from 1C-SP(c) to current and next route
        if delivery of c could not be split then
          ir := ir + 1 // continue with next route, customer remains same
        else // at least one c-layer fits into route ir
          assign routes ir and ir+1 to c and assign attribute "split" to c
          update routes ir and ir+1 including distribution of c-layers to ir and ir+1
          ic := ic + 1 // continue with next customer
          ir := ir + 1 // continue with next route, i.e. second route of c
        endif
      endif
    else if customer c = Cs(ic) is first of customer pair (c,c') with c' = Cs(ic+1) then
      if (c,c') fits into current route ir then
        assign ir to c and c' and update route ir w.r.t. no. of customers, loading length, weight
        ic := ic + 2 // continue with next but one customer, route remains same
      else // (c,c') does not fit completely into current route ir
        split delivery of c or c' if necessary, distribute c/c'-layers from 2C-SP(c,c') to routes ir, ir+1
        if none of deliveries could be split then
          ir := ir + 1 // continue with next route, customers remain same
        else // at least first c-layer fits into route ir
          if delivery of c was split then
            assign routes ir and ir+1 to c and route ir+1 to c' and assign attribute "split" to c
          else // delivery of c' was split
            assign route ir to c and routes ir and ir+1 to c' and assign attribute "split" to c'
          endif
          update routes ir and ir+1 including distribution of c/c'-layers to ir and ir+1
          ic := ic + 2 // continue with next but one customer
          ir := ir + 1 // continue with next route
        endif
      endif
    endif
  endwhile
  no_of_routes := ir;
end.
```

Figure 8: Determining of routes as step of decoding procedure.

Note that by means of criterion (2) and value *max_split_costs* = 0 problem variant 3L-SDVRP-f is implemented, too. If criterion (2) is satisfied the layers of customer *c* are examined in ascending order of their loading lengths. Each layer that still fits into route *ir* is accepted for this route while the other layers are reserved for route *ir*+1. If no layer fits into route *ir*, the attempt to split the delivery of customer *c* remains also ineffective.

The second part of the loop deals with the case that the next customer is the first one of a CP (*c*,*c*') whose 2C-SP is to be used. The decoding procedure runs similarly to the above case of a single customer. An attempt to split the delivery of a customer pair is now done as follows. First all the layers of the related 2C-SP are examined in pattern order to ensure that LIFO constraint (C2) is met. If the last layer fitting in route *ir* is a pure *c*-layer the customer

whose delivery should be split is *c*.

Otherwise, if the last fitting layer is a mixed layer (with boxes from *c* and *c*') or a pure *c*'-layer the "split" customer is *c*'. In the former case a travel cost checking is done using again criterion (2); in the latter case the similar criterion (3) is applied:

$$(d_{c,c'} + d_{c',0}) / d_{c,0} \leq max\_split\_costs \quad (3)$$

Again, the attempt to split the delivery of one customer of the pair (*c*, *c*') remains ineffective if not even the first *c*-layer does fit into route *ir* or if criterion (2) or (3), respectively, is not met.

4.3.2 Generating the initial solution

The initial solution for the local search is determined by a randomized variant of the Savings heuristic. Multiple feasible 3L-SDVRP solutions are generated and the best among them is taken as initial solution. Splitting deliveries as well as using CPs (i.e. 2D-SP patterns) is *not* involved and a special coding schema is used. First a complete sorted savings list is produced. A coded solution is a binary string with the length of the savings list. A value 1 at position *j* means that the related savings is to implement if this does not lead to a violation of constraints while a value 0 at position *j* specifies that the related savings is not to implement even if this would not violate any constraint. A coded solution is decoded by examining the binary string and building the decoded solution from scratch. Two capacity constraints are observed, namely the weight and the length capacity constraint (LCC). For each of *nindiv* trials (*nindiv* is a parameter) first a coded solution is provided at random where the probability of value 1 per position is 0.5. This solution is then decoded, the number of tours and the *ttd* value are determined and the best solution found is updated if necessary.

4.3.3 Determining the best neighbor of a current solution

If the best neighbor s_{iter_best} of a current solution s_{curr} is determined within an iteration, swap and shift moves are applied at the same time as shown in Figure 9.

```
Algorithm determine_best_neighbour( in: current solution s_curr, parameter nbh_size,
                                    out: best neighbour solution s_iter_best)
    s_iter_best.nt = n + 1; s_iter_best.ttd = ∞
    for i := 1 to n do
        // determine range of moves
        j_min := MAX(1, i - nbh_size); j_max := MIN(n, i + nbh_size)
        // swap moves
        for j := j_min to j_max do
            s_next := s_curr
            swap customers at positions i, j in s_next
            decode solution s_next and update s_iter_best by s_next where necessary
        endfor
        // shift moves
        for j := j_min to j_max do
            s_next := s_curr
            shift customer at position i behind (i < j) or before (j < i) customer at position j in s_next
            decode solution s_next and update s_iter_best by s_next where necessary
        endfor
    endfor
end.
```

Figure 9: Determining the best neighbour solution.

For a given current solution and a first customer *C*(*i*) selected swap and shift moves with a second customer *C*(*j*) are tried. For each move the decoded solution s_{next} is provided and

solution s_{iter_best} is updated where necessary. The range of swap and shift moves is governed by the parameter *nbh_size*. We distinguish between small and large neighborhoods. In the former case *nbh_size* takes the value MAX(*n*/4, 3) while in the latter case *nbh_size* equals MAX(*n*, 3). By changing *nbh_size* during the search (see below) additional search paths are enabled and the search is diversified.

4.3.4 Main algorithm of local search

The entire local search is organized in multiple partial searches as depicted in Figure 10. Each partial search is characterized by two features. First, by the maximum admissible splitting costs that are taken into account when a solution is decoded (see 4.3.1). Second, by the range of swap and shift moves where small and large ranges alternate (see 4.3.3). Altogether $2 \cdot nps$ partial searches are carried out as the number of different values of the maximum splitting costs is given by parameter *nps* and there are two neighborhood sizes.

```
Algorithm local_search(in: problem data, parameters, out: best solution s_best)
    generate initial solution s_init and set s_best := s_init
    for ips := 1 to nps do      // outer loop: changing max. admissible splitting costs
      for inbh := 1 to 2 do     // inner loop: changing neighborhood size
        // specify max. admissible splitting costs
        max_split_costs := Max_split_costs[ips]
        // specify neighborhood size
        if inbh = 1 then nbh_size = MAX(n/4, 3) else nbh_size = MAX(n, 3) endif
        // partial search
        s_curr := s_best // starting from best solution so far
        for iter = 1 to niter do
            s_iter_best := determine_best_neighbour(s_curr, nbh_size)
            post-optimize s_iter_best by 2-opt
            if ips = 1 and n ≤ 100 then post-optimize siter_best by 3-opt endif
            update best solution s_best by s_iter_best where necessary
            s_curr := s_iter_best
        endfor
      endfor
    endfor
    post-optimize s_best by 3-opt
end.
```

Figure 10: Main algorithm of local search.

The possible values of the admissible maximum splitting costs are held in the vector *Max_split_costs*. At the first place the value 0 is recorded, saying that no optional splitting is allowed at all. Hence, the parameter *nps* is set to 1 for problem variant 3L-SDVRP-f and in this case only two partial searches are performed. Only for problem variant 3L-SDVRP-o further values of *Max_split_costs* on places 2,3,... are used. The values of *Max_split_costs* show an exponential growth so that large splitting costs are accepted at the end (see 5.1).

Each partial search is started from the best solution s_{best} achieved so far. Per iteration first the best neighbor of the current solution s_{curr} is calculated before the best neighbor solution is post-optimized by 2-opt and possibly by 3-opt. Both procedures are effective local search methods in routing (see, e.g., Helsgaun, 2009). Note that the decoding procedure has also to be integrated in the local search methods 2-opt and 3-opt. 3-opt is only called in the first two partial searches and if the number of customers does not exceed 100. At the end of an iteration the best solution s_{best} is updated if necessary and s_{curr} is set to s_{iter_best} for the next iteration. Finally, 3-opt is applied again, this time to the best found solution s_{best}.

5 Numerical experiments

In this section, we first describe the setting of the parameters used in our approach. Our 3L-SDVRP problem formulation follows the Gendreau 3L-CVRP formulation except the one-visit-per-customer condition. We test three sets of instances, namely the Shanghai instances, the Ceschia instances and the new B-Y instances. The first two sets are derived from real-world operations while the B-Y instances are generated based on well-known VRP and CLP benchmark instances. All instances are characterized by weakly heterogeneous box sets that are common in the context of factory inbound logistics. Note that the 3L-CVRP instances by Gendreau et al. (2006) have strongly heterogeneous box sets which often occur in retail and distribution logistics. We will describe the instance sets and compare our results with the best ones in the literature on the 3L-SDVRP. All experiments were run on a 3.40 GHz PC (AMD A10-5700 APU) with 8.0 GB RAM under Windows 8.1. The algorithm was coded using MS Visual C++ 2010 (Express Edition). Each instance of all sets was run ten times with different seed values of the random number generation. Our algorithm is denoted by SDVRLH2.

5.1 Parameter setting

The parameters for the GA that serves the generation of 1C-SP and 1C-FLP patterns are chosen unchanged as in Bortfeldt and Gehring (2001). Further parameters for packing and routing are set as depicted in Table 3 (*n* stands for the number of customers).

The number of partial searches *nps* (for a given neighborhood size) and the vector *Max_split_costs* are determined as follows. First the minimum and maximum relative insertions costs ric_{min} and ric_{max} over all pairs of customers are calculated according to ric_{min} = MIN $\{(d_{ij} + d_{j0}) / d_{i0} \mid 1 \leq i, j \leq n\}$ and ric_{max} = MAX $\{(d_{ij} + d_{j0}) / d_{i0} \mid 1 \leq i,j \leq n\}$. Then we set *Max_split_costs*(1) = 0 and *Max_split_costs*(2) = $2 \cdot ric_{min}$. On each of the following places 3,4,… the preceding value is doubled until the value ric_{max} is exceeded. Then the last value is replaced by ric_{max}. The number of occupied places of *Max_split_costs* gives the number of partial searches *nps*.

The specified parameter setting was determined in pre-tests of limited size and is used for all numerical experiments.

Table 3: Parameters for packing and routing.

Parameter	Meaning	Value
qnb	Percentage of neighboring customers (in %)	30
maxcands	Maximum no. of variants for next stack placement	if $n \leq 50$: 3 else: 2
nindiv	No. of trials for generating the initial solution	10
nbh_size (small)	Small neighborhood size for swap / shift moves	MAX(n/4, 3)
nbh_size (large)	Large neighborhood size for swap / shift moves	MAX(n, 3)
niter	No. of iterations in a partial search	100
niter_wimpr	No. of iterations without improvement; a partial search is terminated if *niter_wimpr* iterations are carried out without yielding a new best solution s_{best}	20
maxtime	Time limit for entire search (in s)	if $n < 50$: 240 else if $n < 200$: 1800 else: 3600
maxtime-2opt	Time limit for single 2-opt run (in s)	120

5.2 Experiments with Shanghai instances

Our data come from the cargo collecting operations in and around Shanghai area by a Shanghai automotive logistics company, which serves many car makers in metropolitan Shanghai and all over China. Both the data of routing aspects or packing aspects are from real-world, not "man-made".

There are 3849 order lines in the data table, each with a part number, part name, supplier, pick-up node address, area code, package dimensions (length, width and height), and required number of each part. In summary, there are 191 suppliers with 200 pickup nodes that are located around Shanghai area with each node-pair distance within 120 km, and they are initially grouped into 16 areas. These parts in all have 634 package dimensions ($h \times w \times l$) with the smallest box 6×9×12 cm^3 and the largest box 190×155×225 cm^3. For each pickup node, there are 24 items on average to pick up and the highest item number is 270 boxes.

For our Shanghai dataset, based on the areas, we divided the whole data table into 11 first-level instances with node addresses ranging from 5 to 46 (Yi and Bortfeldt, 2018). The box number of each instance ranges from 73 to 1459. In the second level, these 11 instances are combined to 3 big ones by area amalgamation. Instance Sha12 accumulates all the nodes and boxes of Sha2, 5 and 6 instances, but with unified bigger vehicle, similarly Sha13 gathers Sha1, 4, 7 and 10, Sha14 gathers Sha3, 8, 9 and 11. Finally, in the third level, Sha15 combines all eleven instances (Sha1-11) together with total 200 nodes, 634 box types and 4776 boxes. These 15 instances are summarized in Table 4.

A unique character of the Shanghai instances is that there are *big nodes* with $\beta > 100\%$, in Figure 11 the distribution of β values of the 200 nodes in instance Sha15 is depicted. In practice, these big nodes usually are secondary collecting points that receive parts or components from suppliers far from Shanghai area. To pick up demand from big nodes, split delivery has to be applied. However, whether to split other nodes leads our problem to both 3L-SDVRP-f and 3L-SDVRP-o categories.

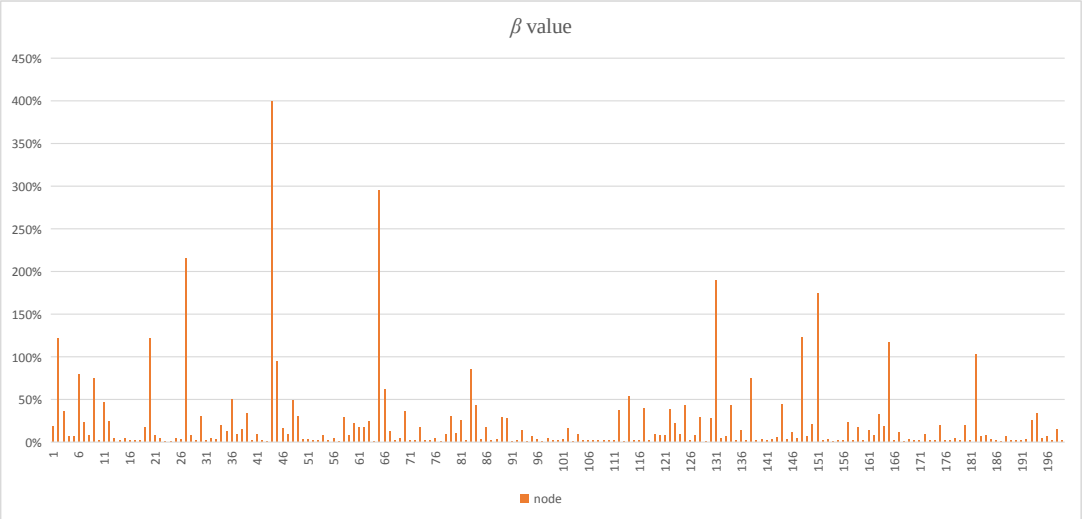

Figure 11: Distribution of β values (in %) of the 200 nodes in instance Sha15.

Table 4: Summary of the Shanghai instances.

Instance	Nodes (n)	Box types	Items (M)	Vehicle type	Lower bound LB on vehicles *	Items per box type	Items per customers
Sha1	5	26	261	MV	2	10.0	52.2
Sha2	8	50	167	SV	6	3.3	20.9
Sha3	10	17	73	SV	3	4.3	7.3
Sha4	12	33	204	BV	3	6.2	17.0
Sha5	12	59	228	MV	4	3.9	19.0
Sha6	15	56	228	MV	4	4.1	15.2
Sha7	16	79	439	BV	7	5.6	27.4
Sha8	18	51	303	MV	6	5.9	16.8
Sha9	27	98	734	CV	8	7.5	27.2
Sha10	31	134	590	BV	9	4.4	19.0
Sha11	46	185	1549	CV	16	8.4	33.7
Sha12	35	149	623	MV	10	4.2	17.8
Sha13	64	234	1494	BV	19	6.4	23.3
Sha14	101	311	2659	CV	26	8.5	26.3
Sha15	200	634	4776	CV	42	7.5	23.9

*: for calculation see equation (4) below

There are four vehicle types for different instances in Table 4: SV with $H \times W \times L$ dimensions 175×180×415 cm^3, weight capacity 2.5 t; MV: 242×217×640 cm^3, 8t; BV: 240×240×860 cm^3, 15 t; CV: 270×235×1202 cm^3, 26.46 t, respectively. However, for each instance, there is only one vehicle type, i.e. an unlimited homogeneous fleet. The cost to apply a new vehicle is quite high, thus the objective priority is given to the number of vehicles (tours) v, then to travel distance ttd. The lower bound LB on vehicles is calculated according to

$$LB = \left\lceil \sum_{i=1}^{n} \beta_i \right\rceil = \left\lceil \sum_i d_i / Q \right\rceil = \left\lceil \sum_{i=1}^{n} \sum_{k=1}^{m_i} h_{ik} w_{ik} l_{ik} \Big/ HWL \right\rceil \quad (4)$$

In each instance, the pickup node addresses are located on a public e-map. The distance between any pair of nodes is gained from the Baidu e-map route (map.baidu.com). For the Shanghai instances the Gendreau formulation of the 3L-SDVRP is assumed, i.e. the constraints (C1) – (C5) have to be observed. The percentage a of stability constraint (C4) is set to 75% and the same is done for the other instance sets.

The results for the Shanghai instances are presented in Table 5. For the problem variants with forced splitting and optional splitting the average number of tours v, the average ttd and the average run time rt (in seconds) as well as the best number of tours and best ttd over ten runs are indicated. In the last two lines sums are given for v and averages for ttd and rt.

It is worth comparing our results with those of Yi and Bortfeldt (2018). In this paper, only results for the first 11 Shanghai instances and for variant 3L-SDVRP-f were provided and each instance was run only once. With our new algorithm SDVRLH2, the results are improved in terms of number of vehicles v by 4.4% and in terms of ttd by 7.3% (if the best values for 3L-SDVRP-f are compared, see line "Total-11"). Also SDVRLH2 needs on average 6 seconds less run time.

Comparing the results for the problem variants 3L-SDVRP-f and 3L-SDVRP-o, we can see that the number of vehicles v was improved for 3L-SDVRP-o by 1.1% and the ttd was reduced by 0.4% if the best values are compared. With optional splits the run time is about 23% higher though. There are at least eight of 15 instances with better results (in **bold**) than those of 3L-SDVRP-f. However, only a small improvement was achieved by optional splits.

Table 5: Results for Shanghai instances.

Instance	SDVRLH2, forced splitting					SDVRLH2, optional splitting					Yi & Bortfeldt, 2018		
	Avg			*Best*		*Avg*			*Best*				
	v	*ttd*	*rt* (*s*)	*v*	*ttd*	*v*	*ttd*	*rt* (*s*)	*v*	*ttd*	*v*	*ttd*	*rt* (*s*)
Sha1	3.0	562.6	0.1	3	562.6	3.0	562.6	0.3	3	562.6	3	582.2	2
Sha2	9.0	2813.2	11.8	9	2813.2	9.0	2786.6	11.5	9	**2786.6**	10	2907.0	13
Sha3	4.0	397.7	0.4	4	393.0	4.0	396.5	1.1	4	393.0	4	369.2	42
Sha4	4.0	352.2	2.7	4	349.7	4.0	350.3	3.0	4	349.7	4	372.0	13
Sha5	6.0	1451.3	7.5	6	1447.6	6.0	1451.3	7.8	6	1447.6	6	1493.9	10
Sha6	8.0	486.1	10.6	8	481.3	8.0	482.1	11.2	8	**479.6**	7	620.0	19
Sha7	12.0	1708.3	16.0	12	1694.8	12.0	1707.5	16.8	12	1694.8	11	1701.4	28
Sha8	8.5	376.9	5.1	8	360.9	8.0	357.7	5.7	8	**352.7**	9	387.7	11
Sha9	13.1	955.1	12.2	13	942.6	13.0	944.3	16.9	13	**928.7**	15	1063.8	17
Sha10	14.0	1610.2	15.8	14	1590.1	14.0	1605.4	20.5	14	1590.1	15	1946.2	53
Sha11	27.0	518.2	93.2	27	512.6	26.2	523.2	150.7	**26**	521.5	29	581.8	33
Sha12	17.0	2972.7	35.4	17	2942.1	17.0	2968.1	48.1	17	2942.1			
Sha13	28.0	3174.6	424.8	28	3128.2	27.7	3147.9	594.1	**27**	3097.3			
Sha14	45.1	1538.5	883.0	45	1481.8	44.2	1574.4	1802.3	**44**	1522.5			
Sha15	72.1	4860.3	3620.5	71	4867.8	71.9	4867.0	3625.9	71	**4795.4**			
Total	270.7	1585.2	342.6	269	1571.2	268.0	1581.7	421.1	266	1564.3			
Total-11			15.9	108	1013.5						113	1093.2	21.9

The mean volume utilization of the loading spaces over the 15 instances amounts to 57.4% in variant 3L-SDVRP-f and to 57.8% in variant 3L-SDVRP-o; this can be considered a good utilization if the packing constraints are taken into account.

5.3 Experiments with instances from Ceschia et al.

Ceschia et al. (2013) present a set of industrial instances which exhibit a high variability in terms of the number of customers, the number of box types and the number of boxes. There are 13 instances with customers ranging from 11 to 129, boxes ranging from 254 to 8060, boxes per customer ranging from 1 to 217. These boxes are industrial goods with dimensions ranging from 10×1×11 to 239×100×230 (h×w×l, cm^3). For all instances the available number of vehicles is fixed (limited fleet), and some instances have a heterogeneous fleet. In their paper, the authors provide results for three experiments. First the instances are solved as 3L-CVRP problems in Gendreau formulation. Second the instances are treated again as 3L-CVRP instances but this time the constraints (C6) – (C8) are required (instead of (C5) and (C4), see 3.2). In the third experiment the same constraints are required as in the second one but this time splitting deliveries is allowed.

Since we want to minimize the number of used vehicles as well as the total travel distance we have modified the Ceschia instances. We have kept the same nodes and boxes but have changed to an unlimited homogeneous fleet. For the original instances SD-CSS5, SD-CSS7, SD-CSS8 and SD-CSS11 with heterogeneous fleet we have chosen the smallest vehicle type (dimensions 268×247×1362 cm^3) for our modified instances. We consider only the Gendreau formulation of 3L-CVRP and 3L-SDVRP, i.e. constraints (C6) – (C8) are not required.

The modified Ceschia instances are treated as 3L-SDVRP-f and 3L-SDVRP-o problems. The results are summarized in Table 6 that is organized as Table 5. Direct trips as a result of forced splitting only occur for instance SD-CSS3, this is consistent with Table 2 of Ceschia et al. (2013). Moreover, for a comparison the results of Ceschia et al. for their original (not modified) instances with Gendreau formulation are also listed; since splitting is not allowed in

this case, there is no feasible solution of instance SD-CSS3 and this instance is *not* taken into account in the comparison (see line Total-12).

Compared to Ceschia et al. (2013) better results are reached for problem variant 3L-SDVRP-f. The mean improvement regarding the number of vehicles amounts to 6.3% while the mean *ttd* improvement is 0.9% if the best 3L-SDVRP-f solutions are compared (see Total-12). Moreover, the run times of SDVRLH2 are about one order of magnitude lower than the run times reported by Ceschia et al.

A comparison of our results for the problem variants 3L-SDVRP-f and 3L-SDVRP-o proves that for the (modified) Ceschia instances optional splits lead to better solutions. The improvement in terms of vehicle numbers is 2.7% and the *ttd* improvement stands at 3.1% when the best solutions are compared. For 10 of 13 instances the best solution could be improved (marked bold in Table 6). The comparison of the reached average solution quality does confirm this outcome. The average volume utilizations of the loading spaces in the best solutions amount to 44.1% (3L-SDVRP-f) and 45.6% (3L-SDVRP-o) and can again be seen as satisfactory (see 5.2). The increase of run times is about 19.6% with variant 3L-SDVRP-o.

Table 6: Results for (modified) Ceschia instances.

Instance	SDVRLH2, forced splitting					SDVRLH2, optional splitting					Ceschia et al., 2013		
	Avg		*Best*			*Avg*		*Best*			*Best*		
	v	*ttd*	*rt* (*s*)	*v*	*ttd*	*v*	*ttd*	*rt* (*s*)	*v*	*ttd*	*v*	*ttd*	*rt* (*s*)
SD-CSS1	5.0	5543.6	0.2	5	5467.4	5.0	5238.5	0.9	5	**4941.8**	5	5084.1	351.2
SD-CSS2	17.0	13044.5	3.0	17	13044.5	14.0	11929.8	8.4	**14**	**11620.2**	13	11879.9	4709.1
SD-CSS3	24.0	16327.6	21.1	24	16154.0	23.0	16302.1	36.4	**23**	**15800.9**	-	-	-
SD-CSS4	14.1	12958.5	13.3	14	12700.3	14.0	12826.7	27.0	14	**12655.5**	12	11175.2	5646.8
SD-CSS5*	17.0	14513.8	19.5	17	14513.8	17.0	14513.8	33.5	17	14513.8	12	10451.0	10000.0
SD-CSS6	17.9	15850.9	36.7	17	16021.3	17.0	15883.7	70.2	17	**15332.5**	32	21487.6	1109.7
SD-CSS7*	13.2	12330.1	47.3	13	12084.8	13.0	12206.6	73.0	13	**11804.4**	10	10339.7	6986.1
SD-CSS8*	25.0	19747.2	44.3	25	19474.4	23.0	19709.8	123.3	**23**	**19225.8**	36	21908.7	4650.8
SD-CSS9	20.6	16638.1	133.7	20	16585.7	20.0	16512.9	220.5	20	**16015.0**	23	17258.0	1694.6
SD-CSS10	18.4	16380.4	205.1	18	16079.4	18.0	16432.6	273.8	18	**15874.5**	18	11865.1	9210.1
SD-CSS11*	21.1	18871.3	1825.5	21	18593.2	21.1	18871.3	1822.7	21	18593.2	13	24843.1	1443.5
SD-CSS12	45.8	37144.5	1135.0	45	36793.9	44.0	37261.3	1803.6	**44**	**36311.5**	48	34256.1	1734.7
SD-CSS13	25.0	21977.5	1782.0	25	21338.6	25.0	22047.5	1802.2	25	21694.3	31	26342.9	5490.1
Total	264.1	17025.2	405.1	261	16834.7	254.1	16902.8	484.3	254	16491.0			
Total-12	240.1	17083.4	437.1	237	16891.4						253	17241.0	4418.9

*: instance with only one vehicle type, i.e. with the smallest type of the corresponding original instance.

5.4 Experiments with B-Y instances

To further test our algorithm, we have introduced a new set of 20 B-Y instances which are constructed by combining 5 CVRP instances from Christofides et al. (1979) and 200 CLP instances by Bischoff & Ratcliff (1995).

We have chosen the first 5 CVRP instances C1-C5 with *n* ranging between 50 and 199. To generate 3L-SDVRP instances we have added boxes and loading spaces using the CLP instances by Bischoff and Ratcliff (1995) with 3 (test case BR1) and 20 box types (test case BR7). We use these two test cases in order to ensure two strongly different levels of box heterogeneity. For the loading spaces the dimensions of a 20 feet container are taken. The weight related data are derived from the demands and capacities of the CVRP instances. Each fifth box of a customer is set fragile.

For one-dimensional SDVRP instances the largest savings are obtained if the average customer demand is just above half of the vehicle capacity and the variance of the customer

demands is low (Archetti and Speranza, 2012, p. 9). Therefore, we have constructed for ten instances customer box sets where the ratio β of the total box volume and the loading space volume is just above 50% for each individual customer. For the other ten instances we have generated box sets where β is circa 20% per customer, thus a tour has mostly less than five customers. A summary of the 20 B-Y instances is given in Table 7 (*M*, *bt* stand for the number of boxes and the number of box types, respectively, β-mean represents the mean value of β over all customers).

Table 7: Summary of the B-Y instances.

Instance	*n*	*M*	*bt*	β-mean (%)	Instance	*n*	*M*	*bt*	β-mean (%)
B-Y1	50	3535	3	51.8	B-Y2	50	1295	3	19.0
B-Y3	50	3377	20	50.9	B-Y4	50	1442	20	22.0
B-Y5	75	5218	3	50.3	B-Y6	75	2135	3	21.6
B-Y7	75	5116	20	52.4	B-Y8	75	2054	20	20.5
B-Y9	100	6861	3	50.1	B-Y10	100	2854	3	21.4
B-Y11	100	6866	20	52.5	B-Y12	100	2680	20	20.7
B-Y13	150	10356	3	51.2	B-Y14	150	4508	3	20.8
B-Y15	150	10082	20	52.1	B-Y16	150	3904	20	20.7
B-Y17	199	14230	3	51.5	B-Y18	199	5790	3	20.3
B-Y19	199	13058	20	51.3	B-Y20	199	5797	20	22.2

The results for the B-Y instances are indicated in Table 8 that is organized similar to Table 5. Additional columns include the mean filling rates *fr* of the vehicle loading spaces as percentages. In the last three lines sums are presented for the vehicle numbers *v* and averages for all other quantities. The lines Total-50 and Total-20 include the best *v, ttd* and *fr* values for the instance subgroups where β is circa 50% and 20%, respectively.

Table 8: Results for B-Y instances.

Instance	SDVRLH2, forced splitting						SDVRLH2, optional splitting					
	Avg			*Best*			*Avg*			*Best*		
	v	*ttd*	*rt* (*s*)	*v*	*ttd*	*fr (%)*	*v*	*ttd*	*rt* (*s*)	*v*	*ttd*	*fr (%)*
B-Y1	50.0	2402.2	40.4	50	2402.2	51.8	37.0	2081.6	152.1	**37**	**2081.6**	70.1
B-Y2	16.0	1035.9	59.3	16	1019.4	59.3	15.5	1055.3	96.0	**15**	1044.2	63.2
B-Y3	50.0	2402.2	41.0	50	2402.2	50.9	37.0	2071.0	156.2	**37**	**2071.0**	68.7
B-Y4	19.1	1181.0	54.6	19	1163.9	57.9	18.0	1184.6	94.7	**18**	**1149.6**	61.1
B-Y5	75.0	3630.5	270.8	75	3630.5	50.3	54.0	2961.2	760.5	**54**	**2961.2**	69.8
B-Y6	25.9	1582.8	323.6	25	1573.5	64.8	25.0	1570.7	448.6	25	**1541.4**	64.8
B-Y7	75.0	3630.5	271.3	75	3630.5	52.4	57.0	3215.8	838.5	**57**	**3215.8**	69.0
B-Y8	27.1	1661.5	346.8	27	1643.6	56.9	26.0	1645.0	503.5	**26**	**1606.8**	59.0
B-Y9	100.0	4989.8	1064.1	100	4989.8	50.1	71.0	4014.9	1802.6	**71**	**4014.9**	70.5
B-Y10	34.7	2133.5	1817.3	34	2077.3	63.1	34.7	2133.5	1814.8	34	2077.3	63.1
B-Y11	100.0	4989.8	1060.7	100	4989.8	52.5	76.0	4386.7	1803.1	**76**	**4386.6**	69.1
B-Y12	35.5	2181.1	1342.4	35	2167.8	59.1	34.0	2173.6	1762.0	**34**	**2151.3**	60.8
B-Y13	149.0	7341.9	320.6	149	7341.9	51.6	109.0	6043.4	1811.7	**109**	**6022.4**	70.5
B-Y14	49.3	2951.6	1818.9	49	2915.4	63.7	48.0	2942.1	1802.8	**48**	**2888.3**	65.0
B-Y15	150.0	7360.5	320.6	150	7360.5	52.1	117.6	6477.7	1804.6	**117**	**6451.1**	66.7
B-Y16	53.4	3081.0	1594.5	53	3049.4	58.5	51.9	3089.8	1804.1	**51**	3177.8	60.8
B-Y17	198.0	9558.3	1098.2	198	9558.3	51.8	148.0	8462.3	1808.1	**148**	**8456.4**	69.3
B-Y18	65.3	3689.1	1969.6	64	3665.6	63.0	65.6	3676.7	1810.8	65	3647.1	62.1
B-Y19	199.0	9607.8	1106.2	199	9607.8	51.3	154.0	9298.8	1808.9	**154**	**9296.4**	66.2
B-Y20	75.2	4190.0	1875.9	74	4156.4	59.6	74.6	4298.5	1805.6	**73**	4351.4	60.4
Total	1547.5	3980.0	839.8	1542	3967.3	56.0	1253.9	3639.2	1234.5	1249	3629.6	65.5
Total-50				1146	5591.4	51.5				860	4895.7	69.0
Total-20				396	2343.2	60.6				389	2363.5	62.0

The main outcome here is that much better results are reached by optional splits. The improvements are 19% for the number of vehicles, 9.5 %-points for the volume utilization and 8.5% for the total travel distance if the best solutions are compared. An improvement has been reached in 18 of 20 instances (marked bold in Table 8). Again, the comparison of average solutions confirms this result. The average run time for the variant with optional splits is about 47% higher than for the variant with forced splits. However, with circa 20 minutes on average the run times remain moderate for the variant with optional splits, too. As expected forced splits proved to be not necessary for any instance.

For the subgroup with $\beta \approx 20\%$ the improvements are relatively small and the *ttd* values are even slightly worse if optional splits are admissible. However, in the other subgroup with $\beta \approx 50\%$ large improvements are made by optional splits (see line Total-50). For example, the average volume utilization grows by 17.5 %-points. Thus, the results for the 3D case are in line with the above quoted conclusion by Archetti and Speranza (2012).

6 Conclusion

In this work, we deal with the vehicle routing problem with split delivery and three dimensional loading constraints (3L-SDVRP) which is similar to the 3L-CVRP formulation by Gendreau et al. (2006) except for the only-one-visit assumption. Two problem variants are introduced, namely the 3L-SDVRP with forced splits and optional splits, respectively. In the former variant only indispensable splits are allowed while splitting deliveries is totally free in the latter variant.

We propose an algorithm that is able to cope with both problem variants. The method follows the principle "Packing first, routing second", thus the packing and routing task are solved in two strictly separate steps. In the packing step patterns for one or two customers are constructed that consist of vertical layers. This ensures simple packing plans which are easy to implement in practice. Forced splits are realized by direct trips to single customers and related packing patterns that fill nearly the complete loading space. The packing is carried out by a well-known GA and some constructive heuristics. The routing is done by a local search algorithm that is based on a giant tour representation. Optional splits utilize the layer structure of customer packing patterns.

Our algorithm is applied to three different sets of instances with industrial as well as academic origin. Comparisons to existing 3L-SDVRP methods prove a good quality of the results. Moreover, they are calculated in relatively short running times. The results with optional splits are generally better than the results calculated only with forced splits. This is especially true for instances with higher box volume / vehicle volume ratio per customer where still no direct tours are necessary though. Thus, it has been proven that by optional splits tours and travel distance can be saved also in the context of routing problems with three-dimensional loading constraints. Future work should enable the hybrid algorithm to cope with further loading constraints that occur in practice, for example the axle weight or load bearing strength constraint.

Acknowledgments

This research has support from NSFC research grant 71371162 and Fujian Fumin Foundation.

References

Archetti, C., Bianchessi, N., & Speranza, M. G. (2011). A column generation approach for the split delivery vehicle routing problem. *Networks*, *58*, 241-254.

Archetti, C., Bianchessi, N., & Speranza, M. G. (2014). Branch-and-cut algorithms for the split delivery vehicle routing problem. *European Journal of Operational Research*, *238*(3), 685-698.

Archetti, C., Savelsbergh, M. W. P., & Speranza, M. G. (2008). To split or not to split: that is the question. *Transportation Research Part E: Logistics and Transportation Review, 44*, 114–123.

Archetti, C., & Speranza, M. G. (2012). Vehicle routing problems with split deliveries. *International transactions in operational research*, *19*(1-2), 3-22.

Archetti, C., Speranza, M. G., & Hertz, A. (2006). A tabu search algorithm for the split delivery vehicle routing problem. *Transportation science*, *40*(1), 64-73.

Bartók, T., & Imre, C. (2011). Pickup and Delivery Vehicle Routing with Multidimensional Loading Constraints. *Acta Cybernetica*, *20*, 17-33.

Belenguer, J. M., Martinez, M. C., & Mota, E. (2000). A lower bound for the split delivery vehicle routing problem. *Operations Research, 48*, 801–810.

Berbotto, L., García, S., & Nogales, F. J. (2014). A randomized granular tabu search heuristic for the split delivery vehicle routing problem. *Annals of Operations Research*, *222*(1), 153-173.

Bischoff, E. E., & Ratcliff, M. S. W. (1995). Issues in the development of approaches to container loading. *Omega*, *23*(4), 377-390.

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

Bortfeldt, A., & Gehring, H. (2001). A hybrid genetic algorithm for the container loading problem. *European Journal of Operational Research*, *131*(1), 143-161.

Ceschia, S., Schaerf, A. & Stützle, T. (2013). Local search techniques for a routing packing problem. *Computers & Industrial Engineering 66*, 1138-1149.

Christofides, N., Mingozzi, A., & Toth, P. (1979). The vehicle routing problem. In N. Christofides, A. Mingozzi, P. Toth, & C. Sandi (Eds.), *Combinatorial optimization* (pp. 315–338). Chichester: John Wiley & Sons Ltd.

Dror, M., Laporte, G., & Trudeau, P. (1994). Vehicle routing with split deliveries. *Discrete Applied Mathematics*, *50*(3), 239-254.

Dror, M., & Trudeau, P. (1989). Savings by split delivery routing. *Transportation Science*, *23*(2), 141-145.

Dror, M., & Trudeau, P. (1990). Split delivery routing. *Naval Research Logistics (NRL)*, *37*(3), 383-402.

Fuellerer, G., Doerner, K. F., Hartl, R. F., & Iori, M. (2010). Metaheuristics for vehicle routing problems with three-dimensional loading constraints. *European Journal of Operational Research*, *201*(3), 751-759.

Gendreau, M., Iori, M., Laporte, G., & Martello, S. (2006). A tabu search algorithm for a routing and container loading problem. *Transportation Science*, *40*(3), 342-350.

Helsgaun, K. (2009): General k-opt submoves for the Lin–Kernighan TSP heuristic. *Mathematical Programming Computation*, 1 (2-3).

Hokama, P., Miyazawa, F. K., & Xavier, E. C. (2016). A branch-and-cut approach for the vehicle routing problem with loading constraints. *Expert Systems with Applications*, *47*, 1-13.

Iori, M., & Martello, S. (2010). Routing problems with loading constraints. *Top*, *18*(1), 4-27.

Irnich, S., Schneider, M., & Vigo, D. (2014). Chapter 9: Four Variants of the Vehicle Routing Problem. In *Vehicle Routing: Problems, Methods, and Applications, Second Edition*, 241-271. Society for Industrial and Applied Mathematics.

Junqueira, L., Oliveira, J. F., Carravilla, M. A., & Morabito, R. (2013). An optimization model for the vehicle routing problem with practical three- dimensional loading constraints. International Transactions in Operational Research, 20(5), 645-666.

Koch, H., Bortfeldt, A., & Wäscher, G. (2018). A hybrid algorithm for the vehicle routing problem with backhauls, time windows and three-dimensional loading constraints. *OR Spectrum 40*, 1029-1075.

Li, X., Yuan, M., Chen, D., Yao, J., & Zeng, J. (2018). A Data-Driven Three-Layer Algorithm for Split Delivery Vehicle Routing Problem with 3D Container Loading Constraint. *In KDD '18: The 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, August 19–23, 2018, London, United Kingdom. ACM, New York, NY, USA, 528-536.

Mahvash, B., Awasthi, A., & Chauhan, S. (2017). A column generation based heuristic for the capacitated vehicle routing problem with three-dimensional loading constraints. *International Journal of Production Research*, *55*(6), 1730-1747.

Männel, D., & Bortfeldt, A. (2016). A hybrid algorithm for the vehicle routing problem with pickup and delivery and three-dimensional loading constraints. *European Journal of Operational Research*, *254*(3), 840-858.

Miao, L., Ruan, Q., Woghiren, K., & Ruo, Q. (2012). A hybrid genetic algorithm for the vehicle routing problem with three-dimensional loading constraints. *RAIRO-Operations Research*, *46*(1), 63-82.

Moreno, L., Poggi de Aragão, M., & Uchoa, E. (2010). Improved lower bounds for the split delivery vehicle routing problem. *Operations Research Letters, 38*, 302–306.

Moura, A., & Oliveira, J. F. (2009). An integrated approach to the vehicle routing and container loading problems. *OR Spectrum*, *31*(4), 775-800.

Ozbaygin, G., Karasan, O., & Yaman, H. (2018). New exact solution approaches for the split delivery vehicle routing problem. *EURO Journal on Computational Optimization*, *6*(1), 85-115.

Pace, S., Turky, A., Moser, I., & Aleti, A. (2015). Distributing fibre boards: a practical application of the heterogeneous fleet vehicle routing problem with time windows and three-dimensional loading constraints. *Procedia Computer Science*, *51*, 2257-2266.

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. *OR Spectrum, 37*(2), 297-330.

Rajappa, G. P., Wilck, J. H., & Bell, J. E. (2016). An Ant Colony Optimization and Hybrid Metaheuristics Algorithm to Solve the Split Delivery Vehicle Routing Problem. *International Journal of Applied Industrial Engineering*, *3*(1), 55-73.

Reil, S., Bortfeldt, A., & Mönch, L. (2018). Heuristics for vehicle routing problems with backhauls, time windows, and 3D loading constraints. *European Journal of Operational Research*, *266*(3), 877-894.

Shi, J., Zhang, J., Wang, K., & Fang, X. (2018). Particle Swarm Optimization for Split Delivery Vehicle Routing Problem. *Asia-Pacific Journal of Operational Research*, *35(2)*, 1840006.

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

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. *IEEE Transactions on Intelligent Transportation Systems*, *10*(2), 255-271.

Vidal, T., Crainic, T.G., Gendreau, M., & Prins, C. (2014). A unified solution framework for multi-attribute vehicle routing problems. *European Journal of Operational Research*, *234*(3), 658-673.

Wei, L., Zhang, Z., & Lim, A. (2014). An adaptive variable neighborhood search for a heterogeneous fleet vehicle routing problem with three-dimensional loading constraints. IEEE Computational Intelligence Magazine *9*(4), 18-30.

Wilck, J.H., & Cavalier, T.M. (2012). A Genetic Algorithm for the Split Delivery Vehicle Routing Problem. *American Journal of Operations Research, 2*, 207-216.

Wisniewski, M., Ritt, M., & Buriol, L.S. (2011). A Tabu Algorithm for the Capacitated Vehicle Routing Problem with Three-dimensional Loading Constraints. *Anais do XLIII Simpósio Brasileiro de Pesquisa Operacional*. Ubatuba, Brazil, 1502-1511.

Yi, J., & Bortfeldt, A. (2018). The Capacitated Vehicle Routing Problem with Three-Dimensional Loading Constraints and Split Delivery – A Case Study. In *Operations Research Proceedings 2016*, 351-356, Springer, Cham.

Zachariadis, E. E., Tarantilis, C. D., & Kiranoudis, C. T. (2012). The pallet-packing vehicle routing problem. *Transportation Science*, *46*(3), 341-358.

Zhang, D., Cai, S., Ye, F., Si, Y.W., & Nguyen, T.T. (2017). A hybrid algorithm for a vehicle routing problem with realistic constraints. *Information Sciences*, *394/395*, 167-182.

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. *Computers & Operations Research*, *39*(9), 2178-2195.

Appendix

1) Procedure for building stacks from original boxes for a given customer

The procedure of stack building for a given customer *c* is organized in two steps. First stacks are built from boxes of only one box type (of customer *c*) where all boxes have the same spatial orientation which must always observe constraint (C3). Assume that *nb* boxes of a box type with height *h* are available. Then *nbst* = *H*/*h* boxes can be stacked at most and the procedure yields *q* = *nb*/*nbst* (*q* ≥ 0) stacks of maximal height *nbst*·*h* and at most one stack for the remaining *r* = *nb* % *nbst* boxes (% is the modulo operator, *r* ≥ 0).

In the second step mixed stacks (with one or more box types of customer *c*) are built from the pure stacks (with one box type) resulting from the first step. In what follows we will speak of stacks instead of mixed stacks and of boxes instead of pure stacks. Stacks are then generated one by one until the set of boxes is exhausted. If a new stack is to be generated the still available box with the largest base area is taken as the lowest box. As the next box of a stack among all suitable boxes the one with the largest base area is taken. A box is suitable as the next box for a stack if it is not yet stacked, can be placed regarding the height *H* and its placement meets the constraints (C1), (C4) and (C5). In general the next box is placed on the stack so that its length and width run parallel to the length and width, respectively, of the previous box. With regard to vertical stability (C4), however, the box is rotated by 90° in the horizontal plane where necessary. With each new box the bounding cube of the stack is adapted if necessary. If a stack cannot be extended because there is no suitable box, it is considered complete. For a complete stack the maximum number of copies is determined (with regard to the box set of the customer *c*) and implemented.

The stacks available at the end of the two stage procedure are taken as boxes in the following generation of patterns.

2) First procedure for generating 2D-SPs

In Figure 12 the first procedure for generating a 2D-SP based on two 1C-SP for the customers c_1 and c_2 is depicted. For the sake of brevity some details are omitted.

```
Algorithm generate_2C_SP_1(in: customer c1, customer c2, problem data, parameters,
                                out: success, pattern 2C-SP)
    success := 0 // no feasible 2C-SP pattern for c1 and c2
    // pass c1-layers in decreasing order of layer-depths and try to disintegrate each c1-layer
    for each c1-layer l1 do
        // try each c2-layer as accomodating layer
        for each c2-layer l2 do
            provide residual spaces in l2 and sort them by increasing volume
            no_generated_placements := 0
            for each box b in l1 do
                determine placement of b in a residual space rs with spatial orientation ov
                if (placement was found) then
                    update residual space rs
                    no_generated_placements := no_generated_placements + 1
                endif
            endfor
            if number of generated placements = number of boxes in l1 then
                success := 1 // feasible 2C-SP pattern for c1 and c2 found
                create 2C-SP from remaining c2-layers, generated single mixed layer and
```

```
                    remaining c1-layers  (seen from cabin)
                goto end
            endif
        endfor
    endfor
end.
```

Figure 12: First procedure for generating 2C-SP.

For determining a placement of a box *b* of a c_1-layer the residual spaces provided for a c_2-layer are went through in sorting sequence. The first residual space where the box fits is taken and the box is placed in the corner of the residual space that is nearest to origin. The spatial orientation *ov* of the box is chosen preferably in that way that its larger horizontal side dimension runs parallel to the smaller horizontal dimension of the residual space. After a placement for a box has been determined the related residual space is updated. The old residual space *rs* is replaced by a new residual space within the old one where the placed box is taken into account of course. Here two cases are to be differentiated. If the placed box is not fragile the new residual space is created above the box just placed. If it is fragile the new residual space is created besides or in front of the placed box. The variant with greater volume of the new residual space is taken.

3) Second procedure for generating 2D-SPs

The second procedure for generating 2D-SPs works in three steps. In the first step, some layers with worst volume utilization of both involved customer c_1 and c_2 are disintegrated and their boxes are collected in set *Bfree*. Let *llrem* be the total length of all removed layers. In the second step, stacks are constructed from all boxes of *Bfree*. In the third step, a new layer with length *llnew* is formed by these stacks. At the same time, the length reduction *llrem – llnew* is checked; only if it is greater than zero the procedure has been successful in the given trial. Subsequently the generation of stacks and their placement in a new layer are described more thoroughly.

Generation of stacks

Stacks are generated by means of the algorithm depicted in Figure 13.

```
Algorithm generate_stacks(in: customer c1, customer c2, free boxes Bfree, problem data, parameters,
                                out: set of stacks Stacks)
    Stacks := ∅ // initialize Stacks
    // create stacks as long as Bfree is not empty
    while |Bfree| > 0 do
        // create best possible stack Stbest for set Bfree by recursive procedure
        Stbest := gen_one_stack(c1, c2, Bfree)
        Stacks : = Stacks U {Stbest}
        update Bfree
    endwhile
end.
```

Figure 13: Algorithm generate_stacks.

The stacks are generated one by one until the box set *Bfree* is empty. Per cycle a best stack *Stbest* is constructed for the given set *Bfree* by the recursive procedure *gen_one_stack*() which works as follows:

1) Per procedure call a current stack (initially an empty stack) is extended by a new box

on top in different variants. The maximum number of variants is given by the parameter *maxcands*. The boxes in *Bfree* are sorted by their base area in descending order. The first *maxcands* feasible boxes that follow the old box on top in the sorting sequence are chosen as new top box of the current stack. In the case of an empty stack the candidate boxes are the first *maxcands* feasible boxes.

2) A feasible candidate for a new top box must satisfy the following criteria:
   - the height of the extended stack must not exceed the height of the loading space,
   - the constraints (C3) – (C5) are to be observed,
   - a stack might include boxes from customers c_1 and c_2; however, to satisfy the LIFO constraint (C2) no box of c_2 must be packed above a box of c_1.

   If one of the criteria is not satisfied the respective box is skipped.
3) The boxes of a stack are oriented in that way that their smaller horizontal dimensions run parallel to each other. This principle is omitted, though, if the vertical stability constraint (C4) would be violated then.
4) A stack is complete, if no further box cannot be placed on top. In this case it is checked whether the stack is a new best stack *Stbest*. The best stack is the complete stack with maximal filling rate. It is determined as quotient of the stowed box volume and the volume of the bounding cube of the stack. Note that the bounding cube is extended up to the roof of the loading space.
5) The generation of the best stack for a given set *Bfree* terminates if there is no stack that could be extended anymore.

Building a new layer by stacks

The stacks of set *Stacks*, written here as *St*[*i*], $i = 1,\ldots,nst$, are placed in one or more rows that run parallel to the width of the loading space. All stacks together form a new layer of length *llnew*. This layer follows the remaining c_2-layers and precedes the remaining c_1-layers (seen from cabin). A reduction of the pattern length is achieved if and only the difference *llrem* – *llnew* is positive.

A stack *St*[*i*], $i = 1,\ldots,nst$, has the following components: (1) *hd_min* and *hd_max*: minimum and maximum horizontal dimension of related bounding cube; (2) h_{c2top}: the maximal height of the top face of boxes of customer c_2 in a stack; note that $h_{c2top} = 0$, if there are no c_2-boxes and $h_{c2top} = H$ if there are no c_1-boxes; (3) *ov*: spatial orientation; *ov* is set to 0 if *hd_min* runs parallel to the length of the loading space (i.e. parallel to x-axis) and 1 otherwise; (4) *rc_x* and *rc_y*: x- and y-coordinate of the reference corner (i.e. corner of the stack nearest to origin). Note that only the components (1) and (2) are input data.

The algorithm for building a new mixed layer by stacks is shown in Figure 14. First some initializations are made. The coordinates *y_curr* and *x_curr* of the reference corner of the next placement are set to zero and to *x_max_c2*, i. e. the maximum x-coordinate of all remaining c_2-layers; by this the first row of stacks follows immediately to the last c_2-layer. Note that the x-coordinates of all placements in a row coincide. The length of the next row *lrow* and the maximum x-coordiante of a stack *x_max* are also set to zero.

The stacks *St*[*i*], $i = 1,\ldots,nst$, are sorted by h_{c2top} in decreasing order and are then placed in this sorting sequence. This sorting ensures that the LIFO restriction is met; seen from the vehicle rear, it can never happen that a c_2-box is placed above or before a c_1-box.

To place a stack first its feasible orientation variants are calculated. The reference corner

of a stack is always set to the point (*x_curr*, *y_curr*). An orientation variant of a stack (0 or 1, see above) is possible if the available width $W - y\_curr$ is not exceeded. Moreover, the difference of the maximum x-coordinate of the stack and *x_max_c2* must be smaller than *llrem* (see above). We can differentiate four cases:

Case 1: The stack can only be placed in orientation variant 0. Then the stack is placed accordingly and the variables *y_curr*, *lrow* and *x_max* are updated.

Case 2: The stack can only be placed in orientation variant 1. Case 2 is handled as case 1.

Case 3: Both orientation variants are feasible. In this case orientation variant 1 is chosen if this does not lead to an increase of *lrow*; otherwise variant 0 is selected. Preferring orientation variant 1 contributes to a better utilization of the space of the current row. Again, the variables *y_curr*, *lrow* and *x_max* are updated accordingly.

Case 4: None of the orientation variants is feasible. In this case placements are continued at the left end of the next row and the stack not yet placed is now placed there. The row length *lrow* is set again to zero. In the exceptional case that a placement in the next row at the left end (*y_curr* = 0) is also not feasible a positive length reduction cannot be achieved for the given set of stacks and the procedure terminates. Otherwise the length reduction is returned at the end.

```
Algorithm place_stacks(in: stacks St[i], i = 1,…,nst, maximum x-coordinate of remaining
                              c2-layers x_max_c2, total length of removed layers llrem, problem data,
                              out: length reduction llrem – llnew, set of placed stacks St[i], i = 1,…,nst)
    x_curr := x_max_c2 // x-coordinate of reference corner of next placement
    y_curr := 0 // y-coordinate of reference corner of next placement
    x_max := 0 // max. x-coordinate of placed stacks
    lrow := 0 // length (x-dimension) of current row
    sort stacks St[i], i = 1,…,nst, w.r.t. h_c2top in descending order

    // place all stacks in sorting sequence
    for i := 1 to nst 0 do
        // check feasible orientation variants
        ov_0 := y_curr + St[i].hd_max ≤ W and x_curr + St[i].hd_min < llrem + x_max_c2
        ov_1 := y_curr + St[i].hd_min ≤ W and x_curr + St[i].hd_max < llrem + x_max_c2
        // examine cases
        if ov_0 and (not ov_1 or St[i].hd_max > lrow) then
            St[i].rc_x := x_curr; St[i].rc_y := y_curr; St[i].ov := 0
            y_curr := y_curr + St[i].hd_max
            lrow := MAX(lrow, St[i].hd_min); x_max := MAX(x_max, x_curr + St[i].hd_min)
        else if ov_1 and (not ov_0 or St[i].hd_max ≤ lrow) then
            St[i].rc_x := x_curr; St[i].rc_y := y_curr; St[i].ov := 1
            y_curr := y_curr + St[i].hd_min
            lrow := MAX(lrow, St[i].hd_max); x_max := MAX(x_max, x_curr + St[i].hd_max)
        else // ov_0 = 0 and ov_1 = 0
            if y_curr = 0 then return 0 endif // stack i not to place, length reduction ≤ 0!
            x_curr := x_curr + lrow; y_curr := 0; lrow := 0; i := i - 1 // stack i not yet placed!
        endif
    endfor
    llnew := x_max – x_max_c2 // length of new mixed layer
    return llrem – llnew // positive length reduction!

end.
```

Figure 14: Algorithm place_stacks.

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