Computers & Operations Research 149 (2023) 105988

Contents lists available at ScienceDirect

Computers and Operations Research

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

# Effective loading in combined vehicle routing and container loading problems

Corinna Krebs [a,*], Jan Fabian Ehmke [b], Henriette Koch [a]

[a] Otto-von-Guericke University, Department of Management Science, Universitaetsplatz 2, 39106 Magdeburg, Germany
[b] University of Vienna, Department of Business Decisions and Analytics, Oskar-Morgenstern-Platz 1, 1090 Vienna, Austria

ARTICLE INFO

*MSC:*
90-04
90-05

*Keywords:*
3D loading
Vehicle routing problem
Deepest-Bottom-Left-Fill
3L-VRPTW

ABSTRACT

This paper addresses more effective loading within the 3L-VRPTW, which is a combination of the vehicle routing problem with time windows and 3D loading constraints. We use a hybrid algorithm consisting of an outer Adaptive Large Neighborhood Search tackling the routing problem in combination with an inner Deepest-Bottom-Left-Fill algorithm solving the container loading problem. We propose and compare three new variants for the Deepest-Bottom-Left-Fill algorithm which differ in the representation and storage of available possible placement positions and the shift of items. The possible placement positions can be determined either by available points so that (1) a non-overlapping check between the items is necessary, (2) by available free spaces, or (3) by using a Rectangle Tree (RTree), where items and their positions are stored in a tree. For computational studies, two well-known instance sets are used. The algorithms are evaluated and compared concerning their solution quality and performance. Hereby, the algorithm with free spaces receives the best results with the smallest runtime. Moreover, the impact of different loading constraints is analyzed showing that the LIFO and minimal supporting area constraints have significant effects on the total travel distance. Several new best solutions were found. All results are validated and published at GitHub.

## 1. Introduction

The efficient usage of loading space of containers is of great economic importance, especially in the area of container ship loading, pallet loading, warehouse management, and aircraft freight management. One approach solving this complex problem is the Deepest-Bottom-Left-Fill (DBLF) algorithm introduced by Karabulut and İnceoğlu (2005). The DBLF algorithm is usually embedded in the optimization of superior problems like container loading problems or the vehicle routing problem (namely 3L-CVRP). It has even been adapted to the field of additive manufacturing (Araújo et al., 2019). The advantages of the DBLF-algorithm are clear: it is simple and has a good trade-off between runtime and solution quality. However, when the number of items becomes large, DBLF shows its weakness: Whenever a current item is to be placed, a non-overlapping check with all already placed items inside the loading space has to be carried out. This might need to be repeated for several possible positions until a feasible position for the current item is found. Consequently, the original non-overlapping check proposed by Karabulut and İnceoğlu (2005) has a high computational complexity.

Being a subproblem of a superior optimization problem, the DBLF algorithm has to be implemented as efficiently as possible. When combined with the vehicle routing problem with time-windows (3L-VRPTW), Koch et al. (2019) showed that a combination of an Adaptive Large Neighborhood Search (ALNS) for the routing problem and the DBLF algorithm for the load optimization is competitive. We build on this solution framework in this paper and investigate the performance of three new DBLF implementation variants in the context of the 3L-VRPTW. We do so by analyzing a large number of different loading constraints sets. In particular, we investigate efficient ways of the non-overlapping check (I); we evaluate the DBLF algorithm solely and in conjunction with the combined container loading and vehicle routing problem with time windows (3L-VRPTW) (II), we analyze the impact of each loading constraint on the objective value (III) and we provide new best-solutions for two well-known instance sets (IV).

Concerning (I), we implement three algorithms: one is inspired by current industry practice; one is based on academic literature, namely Karabulut and İnceoğlu (2005); and one is a new variant where a non-overlapping check is not necessary. In the first algorithm, coming from the industrial environment, a Rectangle Tree (RTree) is used. RTree structures are common to evaluate spatial data quickly, e.g. for data analysis or collision detection in the field of three-dimensional computer graphics and game development. In an RTree, the objects

* Corresponding author.
*E-mail addresses:* Corinna.Krebs@ovgu.de (C. Krebs), Jan.Ehmke@univie.ac.at (J.F. Ehmke), Henriette.Koch@ovgu.de (H. Koch).

https://doi.org/10.1016/j.cor.2022.105988
Received 30 December 2021; Received in revised form 2 August 2022; Accepted 5 August 2022
Available online 24 August 2022

and their positions are stored in a tree structure enabling a fast check for non-overlapping of the current item with all already placed items. The idea of using an RTree in the context of the DBLF algorithm was first proposed by Allen et al. (2011) and is further developed in this paper by exploring more loading constraints. The second one is based on Koch et al. (2019) and uses the non-overlapping check as proposed in Karabulut and İnceoğlu (2005). In the last approach, free available spaces are determined so that a non-overlapping check is not necessary to place items.

Concerning (II), we evaluate the three DBLF algorithm variants in the context of the 3L-VRPTW. While in classical packing problems packing algorithms focus on the maximization of the volume, combining packing and vehicle routing requires the quick provision of good solutions for a wide variety of routes and customer sequences. Hence, our DBLF algorithm variants need to be evaluated regarding effective interaction with routing algorithms, esp. with handling a large number of routes.

Concerning (III), we will present the results of three computational studies. In the first one, the DBLF algorithm variants are called with predefined routes to enable comparison with the same computing resources. In the second study, the combined problem is solved by a hybrid algorithm as shown by Koch et al. (2018). The third study deals with a sensitivity analysis of the loading constraints. For the second and the third study, an outer ALNS determines a set of routes. Then, for this set of routes, one of the DBLF algorithm variants is called to create a feasible packing plan for each route. The computational tests are conducted based on two well-known instance sets from the literature. We compare each variant concerning the solution quality (total travel distance), the runtime, and, in the case of the second study, also in the number of iterations. In the third study, we analyze the impact of the loading constraints on the total travel distance, e.g. the LIFO policy, the load capacity of vehicles, the consideration of support and fragility of items. To summarize, this paper presents improving factors for the DBLF-algorithm and the impact of different loading constraints.

The paper is organized as follows. In Section 2, the relevant literature is reviewed. The 3L-VRPTW is formulated in Section 3. In Section 4, the hybrid algorithm and the three DBLF algorithm variants are described. Section 5 presents computational experiments. Finally, conclusions are drawn in Section 6.

## 2. Literature review

This paper presents three variants of the DBLF algorithm. These are evaluated in the context of the 3L-VRPTW, which is a combination of the vehicle routing problem with time windows (VRPTW) and 3D loading constraints. First, relevant literature for the DBLF algorithms is reviewed. Then, research on the related 3L-VRPTW is discussed.

### 2.1. Deepest-Bottom-Left-Fill algorithms

The DBLF algorithm has its origin in Baker et al. (1980), who developed the Bottom-Left algorithm in the context of the 2D-strip packing problem. In this problem, two-dimensional items are packed into a container where one dimension is unknown and must be minimized. Their idea is to place items first into the lowest possible position and then move the items as left as possible along their vertical position. Hopper and Turton (2001) use this approach to solve the 2D container loading problem. To avoid large empty spaces, they modify the algorithm so that available holes are filled (Bottom-Left-Fill). To cover 3D container loading problems, Karabulut and İnceoğlu (2005) extend the Bottom-Left-Fill method and introduce the DBLF algorithm, which places items first in the deepest position. As in Karabulut and İnceoğlu (2005), the DBLF principle has often been implemented with genetic algorithms to solve 3D container loading problems (e.g. in Kang et al. (2012), Moon and Nguyen (2013), Feng et al. (2015) and Jamrus and Chien (2016)) or implemented to solve 3D strip packing problems (e.g. Allen et al. (2011) and Wauters et al. (2013)). The DBLF principle has not only been used for 3D container loading problems, but also in combination with vehicle routing problems (e.g. Ma et al. (2011), Zhu et al. (2012), Wu et al. (2013), Koch et al. (2018, 2019), and Krebs et al. (2021)). In this problem field, the objective is to minimize the total travel distance so that the DBLF algorithms need to provide good solutions quickly rather than maximizing the volume, for example. Concerning best practices in container loading problems, we recommend (Silva et al., 2019). In Araújo et al. (2019), the practicality of the DBLF principle is even analyzed for the area of additive manufacturing.

The DBLF principle has been extended by several variants, e.g. by adding further loading constraints. In Kang et al. (2012), the DBLF principle is embedded into a genetic algorithm and used in the context of the 3D Bin Packing Problem. Hereby, the items are allowed to be rotated in all ways. Moon and Nguyen (2013) tackle the 3D Bin Packing Problem by combining the DBLF principle with a greedy heuristic and a genetic algorithm. Additionally, they ensure a balanced loading w.r.t. the x- and $y$-axis of the loading space. Krebs and Ehmke (2021a) deal with the modeling of axle weights for different vehicle types. Hereby, the DBLF principle is used with an ALNS for solving the combined vehicle routing and container loading problem. The same algorithm is applied in Krebs et al. (2021) for a comprehensive study of new loading constraints (e.g. the reachability of items and realistic load distribution of stacked items). This algorithm is also used in Krebs and Ehmke (2021b), who focus on vertical stability constraints and introduce a new approach enabling stable packing. To extend the research of the impacts of loading constraints on objective values, we evaluate this aspect for the basic loading constraints as introduced by Gendreau et al. (2006) in this paper.

Another variant of the DBLF algorithm considers its complexity: In the implementation by Karabulut and İnceoğlu (2005), the non-overlapping of items is ensured by checking the current item with all already placed items. To reduce the complexity of the algorithm, Allen et al. (2011) store the positions of the placed items in an RTree enabling an efficient non-overlapping check of items. Their algorithm achieved several new best-known solutions. Therefore, the potential of RTree representations is further explored in this paper (see "DBLF with RTree" algorithm) in the context of more loading constraints.

### 2.2. 3L-VRPTW

Gendreau et al. (2006) introduce the combination of the vehicle routing and 3D container loading problem (namely 3L-CVRP). This problem involves the optimal planning of routes to deliver goods to customers that are located in a depot. This is accomplished by a fleet of vehicles having a certain load capacity. Gendreau et al. (2006) solve the vehicle routing problem (VRP) with an "outer" tabu search, which determines the routes. The loading problem is tackled by an iteratively invoked "inner" tabu search. The 3L-VRPTW is a problem variant of the 3L-CVRP, where the depot and the customer locations have time windows. In Pace et al. (2015), a heuristic based on simulated annealing and an iterated local search is proposed to solve the VRPTW. Since they examine the distribution of fiber boards, a specialized loading heuristic and a balanced loading constraint are necessary. The latter is also adopted by Mak-Hau et al. (2018), who develop a mixed-integer linear program for a simplified version of the 3L-VRPTW and a heterogeneous fleet. Zhang et al. (2017) solve the 3L-VRPTW with a hybrid approach where the routing heuristic is based on a tabu search and an artificial bee colony algorithm. The loading heuristic is a combination of a personification heuristic and simulated annealing. They combine two well-known instance sets provided by Gendreau et al. (2006) and Solomon (1987).

As the combination of (meta-)heuristics has proven to be an efficient way to deal with the 3L-VRPTW, in this paper, we use the approach by Krebs et al. (2021) to test the effectiveness of different DBLF algorithm variants. In this hybrid algorithm, an outer ALNS combined with Simulated Annealing solves the routing problem, while an inner DBLF algorithm generates a packing plan for the created set of routes.

## 3. Problem description

In the following, we present a problem description for the 3L-VRPTW, which we use to evaluate the different DBLF algorithm variants proposed in this paper. A complete mathematical formulation is provided in Appendix A. Adapting the convention by Koch et al. (2018), the 3L-VRPTW is specified as follows: Let $G = (N, E)$ be a complete, directed graph, where $N$ is the set of $n$+1 nodes including the depot (node 0) and $n$ customers to be served (node 1 to $n$), and $E$ is the edge set connecting each pair of nodes. Each edge $e_{i,j} \in E$ $(i \neq j, i, j = 0, \dots, n)$ has an associated routing distance $d_{i,j}$ $(d_{i,j} > 0)$. The demand of customer $i \in N \setminus \{0\}$ consists of $c_i$ cuboid items. Let $m$ be the total number of all demanded items. Moreover, time windows are defined thanks to the following three times at each node $i$: the ready time $RT_i$, which is the earliest possible start time of service, the due date $DD_i$, the latest possible start time, and the service time $ST_i$, which specifies the needed time to (un-)load all $c_i$ items of a customer $i$.

Each item $I_{i,k}$ $(k = 1, \dots, c_i)$ is defined by mass $m_{i,k}$, length $l_{i,k}$, width $w_{i,k}$, and height $h_{i,k}$. The items are delivered by at most $v_{max}$ available, homogeneous vehicles. Each vehicle has a maximum load capacity $D$ and a cuboid loading space defined by length $L$, width $W$ and height $H$. It is assumed that each vehicle has a constant speed of 1 distance unit per time unit. If a vehicle arrives at an edge before its ready time, it has to wait until the ready time is reached.

Let $v_{used}$ be the number of used vehicles in a solution. A solution is a set of $v_{used}$ pairs of routes $R_v$ and packing plans $PP_v$, whereby the route $R_v$ $(v = 1, \dots, v_{used})$ represents an ordered sequence of at least one customer, and $PP_v$ is a packing plan containing the position within the loading space for each item included in the route.

A solution is feasible if

(S1) All routes $R_v$ and packing plans $PP_v$ are feasible (see below);
(S2) The number of used vehicles $v_{used}$ does not exceed the number of available vehicles $v_{max}$;
(S3) Each customer is visited exactly once;
(S4) Each packing plan $PP_v$ contains all $c_i$ items of all customers $i$ included in the corresponding route ($i \in R_v$).

A route $R_v$ must meet the following routing constraints:

(C1) Each route starts and terminates at the depot and visits at least one customer;
(C2) The vehicle does not arrive after the due date $DD_i$ of any location $i$.

Each packing plan $PP_v$ of a route $R_v$ must obey the following loading constraints:

(L1) *Geometry*: The items must be packed within the vehicle ensuring non-overlapping;

(L2) *Orthogonality*: The items can only be placed orthogonally inside a vehicle;

(L3) *Rotation*: The items can be rotated 90° only on the width-length plane;

(L4) *Load Capacity*: The sum of masses of all included items of a vehicle does not exceed the maximum load capacity $D$;

(L5) *LIFO*: No item is placed above or in front of item $I_{i,k}$, which belongs to a customer served after customer $i$;

(L6) *Minimal Supporting Area*: Each item has a supporting area of at least a percentage $\alpha$ of its base area;

(L7) *Fragility*: No non-fragile items are placed on top of fragile items.

The 3L-VRPTW aims at determining a feasible solution minimizing the total travel distance $ttd$ and meeting all above constraints.

## 4. Hybrid solution approach

To evaluate our DBLF algorithm variants in the context of the 3L-VRPTW, we employ a hybrid solution approach consisting of a routing algorithm for creating feasible routes and an embedded container loading algorithm, which generates feasible packing plans $PP_v$ for the generated set of routes $R_v$. As a routing algorithm, we use the ALNS proposed by Koch et al. (2018). For the packing algorithm, one of the three investigated DBLF algorithm variants is used, which are presented in this section. We reference the line numbers of the algorithms in square brackets. Generally, we consider a solution to be feasible if all loading and routing constraints are obeyed.

### 4.1. Routing algorithm

As a routing algorithm, we employ the ALNS proposed by Koch et al. (2018) adapted to the 3L-VRPTW without backhauls. The algorithm is shown in Algorithm 1 and briefly described in the following, where the corresponding line numbers are given in square brackets. For further details, we refer to the original paper.

**Algorithm 1** Adaptive Large Neighbourhood Search

**Input:** Instance data, parameters
**Output:** best feasible solution $s_{best}$

```
1: construct initial feasible solution s_init using Savings Heuristic
2: s_best := s_init
3: s_curr := s_init
4: do
5:     select removal operator rem
6:     select number of customers to be removed n_rem
7:     select insertion operator inst
8:     determine next feasible solution s_next := inst(rem(s_curr, n_curr))
9:     for each route R_v in s_next do
10:        feasible := true
11:        if Deepest-Bottom-Left-Fill(R_v) not feasible then
12:            feasible := false
13:            break
14:        end if
15:    end for
16:    check acceptance of s_next using Simulated Annealing
17:    if feasible AND s_next is accepted then
18:        s_curr := s_next
19:        if f(s_curr) < f(s_best) then
20:            s_best := s_next
21:        end if
22:    end if
23:    if it_p reached then
24:        update selection probabilities for insertion and removal
       heuristics
25:    end if
26: while one stopping criterion is not met
```

Initially, a set of routes $s_{init}$ is constructed [1] by the Savings Heuristic developed by Clarke and Wright (1964). Hereby, routing and loading constraints must be ensured. Consequently, the DBLF algorithm is called to check the loading feasibility. The next feasible solution is determined by removing a randomized number of customers from the routes using removal heuristics [5-6]. The removed customers are reinserted using insertion heuristics [7-8]. The removal and insertion heuristics are described in detail in Koch et al. (2018). The new set of routes is checked for whether it fulfills all routing constraints, and whether a feasible packing plan can be created with the DBLF algorithm [9-15].

Infeasible solutions are always discarded. A feasible and superior solution (i.e. less total travel distance) is always accepted as the current solution. An inferior feasible solution may be accepted depending on the result of a Simulated Annealing Heuristic [16-17] developed by Kirkpatrick et al. (1983). Hereby, a worse but feasible solution is accepted with the probability $e^{-(f(s_{next}-f(s))/T)}$. $T$ represents the current temperature ($T > 0$). The starting temperature is determined as proposed by Ropke and Pisinger (2006), i.e. a $w$% worse solution compared to the initial solution would be accepted with a probability of 0.5. After each iteration, the new temperature is calculated by multiplying the current temperature $T$ by a cooling rate $\gamma$.
The current solution $s_{curr}$ is then used to generate the next solution [18]. The success of the removal and insertion heuristics is evaluated after a defined number of iterations $it_p$ to update the selection probabilities of these heuristics [23-25].
The algorithm stops if a time limit is reached or a total number of iterations is executed or the solution does not improve within a defined number of iterations [26].

### 4.2. Packing algorithm

In this section, three DBLF algorithm variants are presented, which are based on the DBLF algorithm proposed by Karabulut and İnceoğlu (2005). The basic concept is to place the items as far as possible to the back (first priority), to the bottom (second priority), and to the left (third priority) of the loading space. For all algorithms, we assume that the point of origin of a Cartesian coordinate system is located in the deepest, bottommost, leftmost point of the loading space. The driver's cab is located behind it accordingly. The length, width, and height of the loading space are parallel to the x, y, and $z$-axis. The position of an item $I_{i,k}$ is defined by $(x_{i,k}, y_{i,k}, z_{i,k})$ of the corner which is closest to the point of origin. First, the items of each customer are sorted by means of the following priorities:

1. fragility flag $f_{i,k}$ (non-fragile first)
2. volume (larger volume first)
3. length $l_{i,k}$ (longer first)
4. width $w_{i,k}$ (wider first).

Then, the items are added to the loading sequence $IS$ reversed to the customer's visiting order. This sequence is used as an input parameter for the DBLF algorithm. Since each customer is visited exactly once (S3), all items of a customer must be placed in the loading space. If no feasible position for an item can be found, the route is revised, and a new one must be searched by the ALNS.

#### 4.2.1. Deepest-Bottom-Left-Fill with points

The following DBLF algorithm variant is the same as in Koch et al. (2019). We formalize this variant in Algorithm 2. The algorithm is rather close to the original proposed by Karabulut and İnceoğlu (2005) with the difference that the rotation of items is allowed here. In the set $SP$, all possible placement points for an item are stored. The first point in the set is the origin of the loading space [2]. Then, for each item $I_p$ of the packing sequence $IS$ and for each allowed orientation, a feasible position is searched [3-5]. For this purpose, the points of $SP$ are successively tested as a possible position for the current item $I_p$. Based on the position, the item $I_p$ is tried to slide further to the back, bottom, and left (see Fig. 1).

After sliding, for this current position for $I_p$, it is checked whether there is an intersection with any already placed item [6-10]. If this is not the case and thus the Geometry constraint (L1) is met, the position for item $I_p$ is checked whether fulfilling the other loading constraints [11]. If the position is feasible, the position is saved for the item $I_p$ [12], and the used point $po$ is removed from $SP$ [13]. Then, three new points are created as new possible placement points [14]: the bottom-right-back, the top-left-back, and the bottom-left-front points of the placed item (see Fig. 2).

**Algorithm 2** Deepest-Bottom-Left-Fill with Points

**Input:** Instance data, smallest dimensions $l_{min}$, $h_{min}$
**Output:** Packing Plan $PP_v$, feasibility of $R_v$

```
1: initialize sorted sequence of unpacked items IS
2: initialize set of unique available points SP
3: for each item I_p ∈ IS do
4:     for each point po ∈ SP do
5:         for each permitted rotation do
6:             for each placed item I_q ∈ IS do
7:                 if I_p and I_q overlap then
8:                     continue with next orientation or point po
9:                 end if
10:            end for
11:            if position is feasible w.r.t. all loading constraints then
12:                save placement for I_p
13:                erase point po
14:                create and include new points in SP
15:                sort SP based on DBL
16:                for each point pi ∈ SP do
17:                    if pi and I_p overlap then
18:                        erase point pi
19:                    end if
20:                end for
21:                break
22:            end if
23:        end for
24:    end for
25:    if no feasible position found then
26:        return false
27:    end if
28: end for
29: return true, PP_v
```

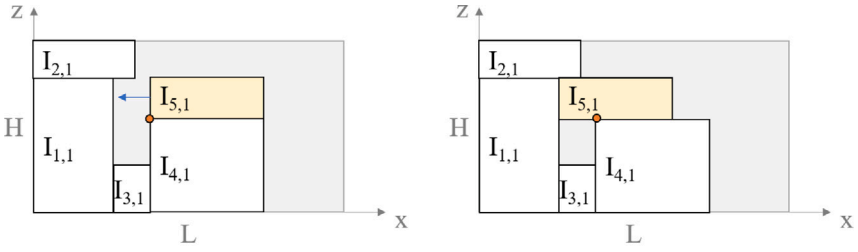
Only points are included in $SP$ providing that any item of the instance could be placed there. Thus, the shortest length or width $l_{min}$ and smallest height $h_{min}$ of any item of the instance are searched once and the points are checked whether their distances to the loading space walls are sufficient. After inserting the new points, the entire set $SP$ is sorted based on the DBL policy [15]. In the last step, all possible points of $SP$ are tested concerning whether they are covered by the placed item $I_p$. In this case, the points are removed from the $SP$ [16-20].

#### 4.2.2. Deepest-Bottom-Left-Fill with spaces

As the previous approach shows, each item must be checked for non-overlapping with all already placed items. If there is an overlap, the next position for the current item is selected and the non-overlapping check is executed again. Due to this high complexity, the idea of the DBLF with Spaces was born. In this approach shown in Algorithm 3, the possible placement positions are stored as available free spaces instead of possible points. Thus, the effort for checking the non-overlapping between items and the vehicle is reduced.

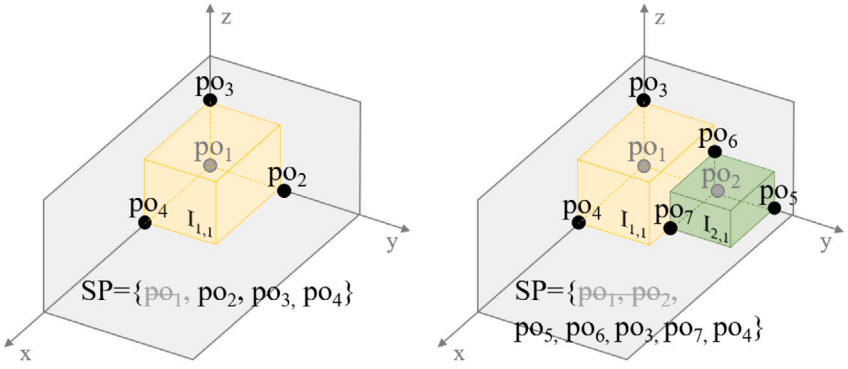
Let $S$ be the set of unique cuboids representing available free spaces for placing items. The general procedure is comparable to the previous one: The set $SP$ is initialized with the entire loading space [2], sorted by the DBL-rule [11], and for each item $I_p$ of the sequence $IS$, a possible position must be found [3]. Thus, each space $sp$ of the set and each allowed rotation are tested as possible item positions until a feasible position is found, obeying all loading constraints [4-7]. In contrast to DBLF with Points, the items are not further slid. Moreover, instead of three new points, up to six spaces are created based on the current position of the placed item $I_p$.

The first space is created by the front edge (*Front Space*) of item $I_p$. Starting from the front edge (minimum x-value), the maximum x-value is determined by expanding the space along the $x$-axis until



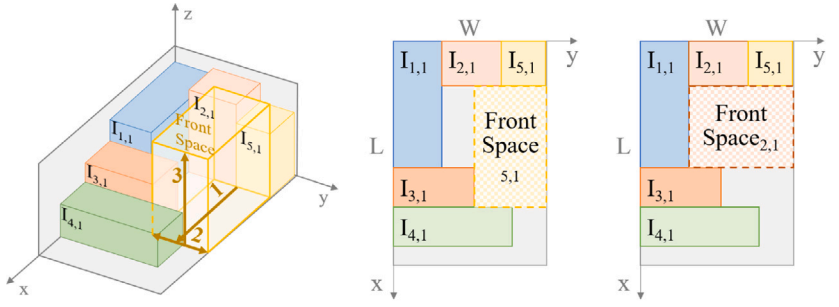

(a) Point *po* for item $I_{1,5}$ (b) Sliding item to new position

**Fig. 1.** Sliding item $I_{5,1}$ based on current position.




(a) After first placed item $I_{1,1}$ (b) After second placed item $I_{2,1}$

**Fig. 2.** Elements of set $SP$.




(a) Front Space for $I_{5,1}$ (b) Front Space for $I_{5,1}$ (c) Front Space for $I_{2,1}$

**Fig. 3.** Front Space Creation.

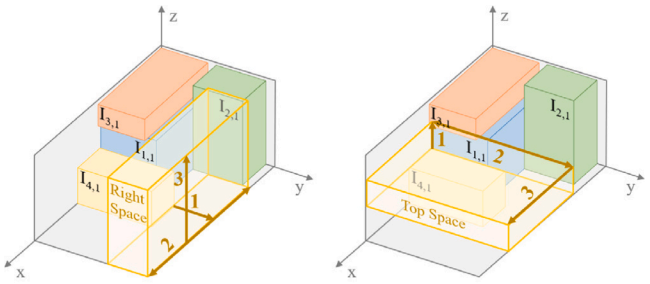
reaching the vehicle door or another item. In the next step, the search for the minimum and maximum z-values is carried out, which are determined by the vehicle (floor or ceiling) or other items (underlying or overhanging ones). Then, the minimum and maximum y-values are searched by extending to the vehicle wall or to an item (see Fig. 3a, b). The order in which the dimensions of the space are determined is decisive for the resulting space. For example, if the *Front Space* would expand along the *y*-axis first, the *Front Space* in Fig. 3c occurs which corresponds to the *Front Space* of $I_{2,1}$ applying the current definition.

The minimum y-value for the *Right Space* is defined by the item's right edge (see Fig. 4a). Then, the maximum y-value is searched, which is defined either by an item or the vehicle wall. In the next step, the

**Algorithm 3** Deepest-Bottom-Left-Fill with Spaces

**Input:** Instance data
**Output:** Packing Plan $PP_v$, feasibility of $R_v$

initialize sorted sequence of items $IS$
initialize set of unique available spaces $SP$
**for each** item $I_p \in IS$ **do**
  **for each** space $sp \in SP$ **do**
    **for each** permitted rotation **do**
      **if** item $I_p$ fits in space $sp$ **then**
        **if** position is feasible w.r.t. all loading constraints **then**
          save placement for $I_p$
          erase space $sp$
          create and include new spaces
          sort $SP$ based on DBL
          get $l_{min}$ and $h_{min}$ of unplaced items $\in IS$
          **for each** space $si \in SP$ **do**
            update space $si$
            **if** $si$ too small **then**
              erase space $si$
            **end if**
            **break**
          **end for**
        **end if**
      **end if**
    **end for**
  **end for**
  **if** no feasible position found **then**
    **return** false
  **end if**
**end for**
**return** true, $PP_v$

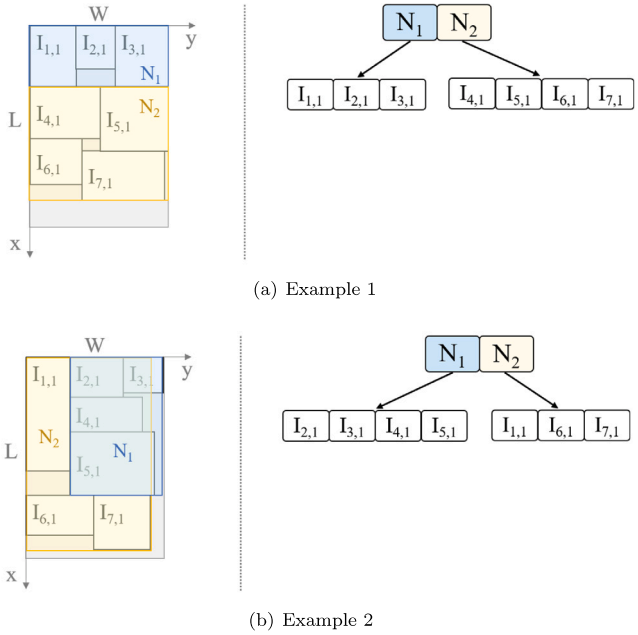



(a) Right Space for $I_{4,1}$ (b) Top Space for $I_{4,1}$

**Fig. 4.** Right and Top Space Creation.

minimum and maximum values for the $z$-axis and then for the $x$-axis are searched. For the *Top Space*, the top edge determines the minimum z-value. Then, the minimum and maximum values along the $y$-axis and $x$-axis are searched (see Fig. 4b). In addition, further three spaces are created if they are unique: (1) Another Front and (2) Right Space, where the minimum z-value represents the bottom edge of item $I_p$; (3) another Top Space, where the minimum x-value is the deepest edge of item $I_p$.

fter the feasible placement of an item $I_p$, the used space $sp$ is removed from the set [9], and the new spaces are inserted in the set $SP$ [10]. To ensure that the available spaces of the set $S$ represent empty volumes, the dimensions of all spaces are checked w.r.t. intersection with item $I_p$. If one or more spaces intersect with item $I_p$, then, these




(a) Example 1

(b) Example 2

**Fig. 5.** Packed items in vehicle and created RTree.

spaces are decreased so that no intersection occurs [14]. Therefore, if an item can be placed within an available space, it is guaranteed that the item does not overlap with other items or with the vehicle's walls (Geometry constraint (L1)). In contrast to the approach by Karabulut and İnceoğlu (2005) and the previous one, a non-overlapping check between each item is not necessary, which decreases the complexity. As in the previous approach, the possible placements must not exceed minimum dimensions. In contrast to the previous approach, the shortest length or width and height are not determined by the smallest items of the instance but rather by the smallest dimensions of unplaced items of the route. Therefore, the shortest length or width $l_{min}$ and height $h_{min}$ of any unplaced item of the route are searched [12], and all spaces which have smaller dimensions than the minimum one are removed [15-17].

#### 4.2.3. Deepest-Bottom-Left-Fill with RTree

Regarding common approaches for collision-detection, e.g. in game development, tree structures are the preferred choice. Thus, in this approach, we use a so-called RTree for the non-overlapping check (Guttman, 1984). Hereby, the tree contains geometric objects. The main idea is to group geometries that are close to each other inside of the tree. At each level of the tree, there is a fixed number of objects. Therefore, an RTree represents a balanced tree. At the leaf level, each geometry describes a single object. In the next higher level of the tree, a minimum bounding geometry is defined by its underlying objects. An RTree can be used to execute queries, e.g. to ensure non-overlapping of geometries: Since all objects lie within the minimal bounding geometry, an object that does not intersect with the bounding geometry can also not intersect with any of the contained underlying objects.

In this third approach and also as proposed by Allen et al. (2011), an RTree is used to store items and their positions and to check for non-overlapping of the current item with all already placed items. In Fig. 5, we show two examples for packed items in a vehicle and the corresponding RTree representation. As described, the leaves of the RTree represent the already placed items while the nodes are the minimum bounding rectangles of the underlying items. The number of objects per node is here set to 4. As Fig. 5b shows, the minimum bounding rectangles are allowed to overlap.

In this approach, we extended the DBLF algorithm by Allen et al. (2011) by considering the rotation and fragility of items. Moreover, to speed up the algorithm, positions where an item cannot be placed are removed (see below). The algorithm is summarized in Algorithm 4.

**Algorithm 4** Deepest-Bottom-Left-Fill with RTree

**Input:** Instance data
**Output:** Packing Plan $PP_v$, feasibility of $R_v$

initialize sorted sequence of unpacked items $IS$
initialize set of unique available points $SP$
**for each** item $I_p \in IS$ **do**
  **for each** point $po \in SP$ **do**
    **for each** permitted rotation **do**
      **if** $I_p$ does not overlap (RTree Query) **then**
        **if** position is feasible w.r.t. all loading constraints **then**
          save placement for $I_p$
          add item $I_p$ in RTree
          erase point $sp$
          create and include new points in set $SP$
          sort $SP$ based on DBL
          get $l_{min}$ and $h_{min}$ of unplaced items $\in IS$
          **for each** point $pi \in SP$ **do**
            **if** $pi$ too small **then**
              erase space $pi$
            **end if**
          **end for**
          **break**
        **end if**
      **end if**
    **end for**
  **end for**
  **if** no feasible position found **then**
    **return** false
  **end if**
**end for**
**return** true, $PP_v$

Similar to DBLF with Points, the possible placement points are stored in the set $SP$, and three new points are created based on the currently placed item [11]. As in the previous two DBLF algorithms, for each item $I_p$ [3], every available placement position [4] and allowed rotation are tested [5] until a feasible position for the current item $I_p$ is found [7]. If the non-overlapping check by the RTree is positive and also the other loading constraints are fulfilled, the placement of item $I_p$ is saved [8], and the item with its position is additionally stored in the RTree [9]. In contrast to the other algorithms, no sliding techniques are used. After saving the feasible position of the item, three new placement points are created based on the current item $I_p$ (see DBLF with Points) and added to the set $SP$ [11]. The set is sorted according to the DBL order [12]. Then, the minimum dimensions $l_{min}$ and $h_{min}$ of all unplaced items are searched as in DBLF with Spaces [13]. The purpose of the minimum dimensions is to remove points whose distance to the vehicle walls is too small to place any item [14-18].

#### 4.2.4. Comparison

The differences between the DBLF algorithm variants are summarized in Table 1. In DBLF with Points, a non-overlapping check between a possible item position and all placed items is necessary. In contrast to that, this is not necessary for DBLF with Spaces, since the dimensions of the space represent empty volumes and are adapted after each item placement. In DBLF with RTree, the items and their positions are stored in the tree, which is then used to ensure the non-overlapping condition. Moreover, in each algorithm, the smallest dimensions are searched to remove impossible placement positions. In the case of DBLF with Spaces and with RTree, the dimensions of the possible placement must be larger and higher than the smallest dimensions of all unplaced items in the route. The dimensions are determined after each placement of an item. In the DBLF with Points, the smallest dimensions are searched once in all items of the instance.

**Table 1**
Comparison of DBLF approaches.

| | DBLF with | | |
|---|---|---|---|
| | Points | Spaces | RTree |
| Non-overlapping check | Each item with all placed items | Spaces ensure non-overlapping | Using Tree |
| Sliding of items | After selecting placement point | While space creation | None |
| Smallest Dimensions | all items of Instance | unplaced items | unplaced items |

**Table 2**
Algorithm Parameters.

| Category | Description | Value |
|---|---|---|
| Stopping Criterion | Maximal number of iterations | 25,000 |
| Stopping Criterion | Maximal number of iterations without improvement | 8,000 |
| Stopping Criterion | Time limit [min] | 60 |
| ALNS | Number of iterations for updating probabilities for removal and insertion operators $it_p$ | 100 |
| ALNS, Simulated Annealing | Cooling Rate $\gamma$ | 0.99975 |
| ALNS, Simulated Annealing | Start Temperature Parameter $w$ | 5% |
| RTree | Number of Leaves per Node | 16 |
| Minimal Supporting Ratio | $\alpha$ | 0.75 |

## 5. Computational experiments

This section presents the computational studies. In particular, we compare the three DBLF algorithm variants concerning solution quality and performance. Moreover, the impact of the loading constraints per DBLF algorithm variant is analyzed. Hereby, the instance sets by Zhang et al. (2017) as well as our 3L-VRPTW instance set[1] are used. Each instance and each constraint set are tested 15 times. Out of these 15 runs per instance and constraint set, the best one is determined (lowest total travel distance, then lowest runtime), and the average result is presented as well. In Appendix B, we provide summarized result tables. Moreover, to ensure full transparency and traceability, we published all results on GitHub.[2] These are additionally validated with our solution validator.[3]

The algorithms are implemented in C++ as a single-core application and are compiled using the VC++ 2019 version, v14.26 compiler. The experiments were executed on several i7-2600 quad-cores with 3.4 GHz and 16 GB RAM. The operating system is Windows 10. The RTree is implemented by using the well-known Boost library, 1.73 version.[4]

[1] See http://open-science.ub.ovgu.de/xmlui/bitstream/handle/684882692/59/Overview.zip
[2] See https://github.com/CorinnaKrebs/Results
[3] See https://github.com/CorinnaKrebs/SolutionValidator
[4] See https://www.boost.org/users/history/version_1_73_0.html

**Table 3**
Comparison of DBLF Algorithms with predefined routes.

| | Number of Routes | Success Rate [%] DBLF with | | | avg. time [s] DBLF with | | |
|---|---|---|---|---|---|---|---|
| | per Range | Points | Spaces | RTree | Points | Spaces | RTree |
| **30–40** | 43.970 | **98.51** | 98.43 | 96.37 | 2.28 | **1.18** | 5.03 |
| **40–50** | 57.826 | 94.93 | **95.61** | 86.57 | 5.00 | **2.87** | 13.56 |
| **50–60** | 48.648 | 88.04 | **88.21** | 69.14 | 20.10 | **12.48** | 37.56 |
| **60–70** | 29.908 | **83.90** | 83.10 | 60.74 | 62.66 | **34.41** | 82.36 |
| **70–80** | 14.103 | 79.64 | **81.17** | 58.25 | 142.16 | **66.75** | 225.51 |
| **80–90** | 3.534 | 60.98 | **62.85** | 37.95 | 699.01 | **272.61** | 608.16 |
| **Total** | 197.989 | 84.33 | **84.90** | 68.17 | 155.20 | **65.05** | 162.03 |

### 5.1. Parameters

The necessary parameters for the hybrid algorithm are listed in Table 2. Regarding the parameters for the routing heuristic, we performed a preliminary study to tune the parameters. As the evaluation showed, the best results were obtained by the parameters as described in Koch et al. (2018) and therefore, these parameters are kept. The *Number of Leaves per Node* was received experimentally.

### 5.2. Comparison of DBLF algorithms

In the following, we evaluate the solution quality and the performance of the DBLF algorithm variants. Two computational experiments are conducted. The first evaluates the algorithm variants based on predefined routes with given volume ranges. In the second, the algorithms are tested in the context of the hybrid algorithm.

#### 5.2.1. Predefined routes

For the following computational experiments, predefined routes for each instance of our instance set are created. The corresponding volume of all dispatched items in the route is calculated and set in relation to the vehicle loading space. Each route is tested once by each algorithm including all loading constraints. In Table 3, the results are presented, including the success rate, which is the total number of successfully packed routes related to the total number of routes. Moreover, the average runtime per volume range is presented.

As expected, the higher the volume utilization rate within the vehicle loading space, the lower the success rate of the packing process. Surprisingly, the industrial approach (DBLF with RTree) achieves the worse results. The overall highest success rate is achieved by DBLF with Spaces, closely followed by DBLF with Points. However, the average runtime of the DBLF with Points is around 2.4 times the average runtime of the DBLF with Spaces. This shows the lower complexity (and thus the better concept of the non-overlapping check) of the DBLF with Spaces compared to the traditional approach (DBLF with Points). The success rate is an indicator of the efficiency of sliding items: In DBLF with RTree, the items are not slid, and this approach achieves significantly worse results than the other approaches containing sliding techniques. Concerning the performance of the DBLF with RTree, the average runtime is almost 2.5 times the average runtime of DBLF with Spaces. Further analysis shows that the tree must be adapted after each successful placement of an item causing high runtimes. To conclude, the DBLF with Spaces achieves merely the highest success rate per volume range along with the smallest runtime.

#### 5.2.2. Combination with vehicle routing problem

Table 4 gives an overview of the results per instance set for each DBLF algorithm. All loading constraints described in Section 3 are considered. Per instance set, we show the average total travel distance (*ttd*), the average runtime in seconds (*time*), and the average number of iterations (*iterations*) calculated based on all instances. Since each instance is tested 15 times, the results are further grouped by best out of 15 runs and the calculated average of the 15 runs. Moreover, we count the number of instances in which a DBLF algorithm was the only one to find the best result along with the total number of best results found (# Best results found).

**Table 4**
Results for DBLF Algorithms — all Constraints.

| | | DBLF with | | | Benchmark |
|---|---|---|---|---|---|
| | | Points | Spaces | RTree | |
| Results for our Instances | | | | | |
| best | ∅ *ttd* | 1,167.34 | **1,158.36** | 1,263.25 | |
| | ∅ time [s] | 2,174.97 | **2,133.41** | 2,963.55 | |
| | ∅ iterations | 11,187.05 | 10,227.22 | 3,212.00 | |
| avg. | ∅ *ttd* | 1,195.68 | **1,172.33** | 1,288.66 | |
| | ∅ time [s] | 2,182.33 | **2,136.27** | 2,970.96 | |
| | ∅ iterations | 11,065.92 | 10,306.46 | 3,204.22 | |
| # Best results found | | 94/363 | **230/497** | 5/180 | |
| Results for Zhang et al. (2017) Instances | | | | | |
| best | ∅ *ttd* | 783.20 | **777.07** | 854.18 | 965.74 |
| | ∅ time [s] | 499.08 | **476.46** | 1,911.17 | |
| | ∅ iterations | 17,399.59 | 17,176.70 | 7,161.26 | |
| avg. | ∅*ttd* | 788.41 | **781.51** | 869.51 | 971.35 |
| | ∅ time [s] | 501.28 | **468.89** | 1,946.59 | 1,163.16 |
| | ∅ iterations | 17,590.99 | 17,099.25 | 7,857.70 | |
| # Best results found | | 5/7 | **19/20** | 1/2 | |

Generally, the same findings as before can be drawn: DBLF with Spaces dominates the other DBLF algorithms, which is reflected by the smaller total travel distance, better performance (smaller runtime), and a higher number of best results found. This highlights the importance of the reduced complexity of the non-overlapping check. Compared to DBLF with Spaces, the total travel distance obtained with DBLF with Points is longer. In the case of our instance set, the total travel distance increases by about 2%, while for Zhang et al. (2017) instances, it is < 1% on average. Therefore, it can be concluded that sliding the possible placement positions during their creation (DBLF with Spaces) leads to better results than choosing one position and sliding the items afterward (DBLF with Points). In general, the solutions are achieved with an approximately 5% increase in runtime when using DBLF with Points.

Surprisingly, the results for DBLF with RTree are generally worse. On average, the total travel distance increases by up to 10% compared to DBLF with Spaces. In general, the runtime is also significantly higher, e.g. for our instances, it is approx. 39%, and for Zhang et al. (2017), it is even up to 3.2 times higher than with DBLF with Spaces. As explained before, the higher runtime is caused by the necessary adaptions of the structure of the RTree after each successful placement of an item. The check for overlap itself is much faster compared to the other algorithms. Due to these adaptions of the RTree, the time per iteration is higher and thus, fewer iterations can be conducted within the time limit. Another reason for the longer total travel distance lies in the missing sliding techniques used to place items in the deepest, leftmost, bottommost position. Still, the DBLF with RTree finds the best results for 182 of 627 instances (29%), and for 6 instances, DBLF with RTree is the only algorithm finding the best solution at all. Consequently, this approach has potential once the high runtime for adapting the tree structure is reduced. This could be achieved by other tree structures in future work.

In comparison to the benchmark by Zhang et al. (2017), DBLF with Spaces achieves solutions that save approximately 20% of total travel distance, on average, in 40% of the time. When comparing the best results with the average results for DBLF with Spaces, the total travel distance increases by up to 1.5%, on average, depending on the instance. Moreover, also the runtime varies by several additional percent. Consequently, there is still room for improvement to make the algorithm more stable in finding the best solutions. Nevertheless, DBLF with Spaces achieves the best results and has the best performance.

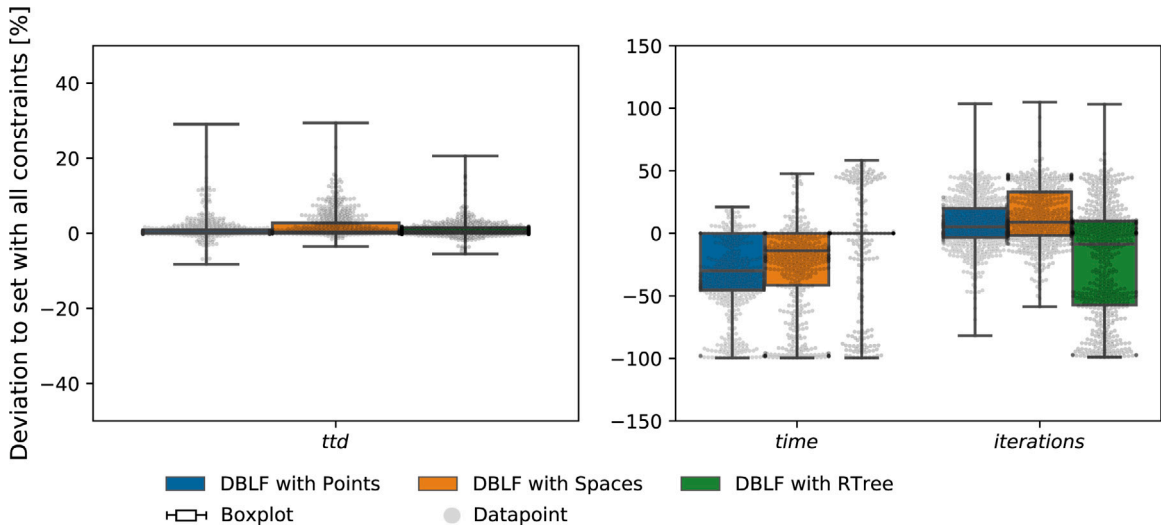



**Fig. 6.** Comparison of DBLF Results — Without Rotation.

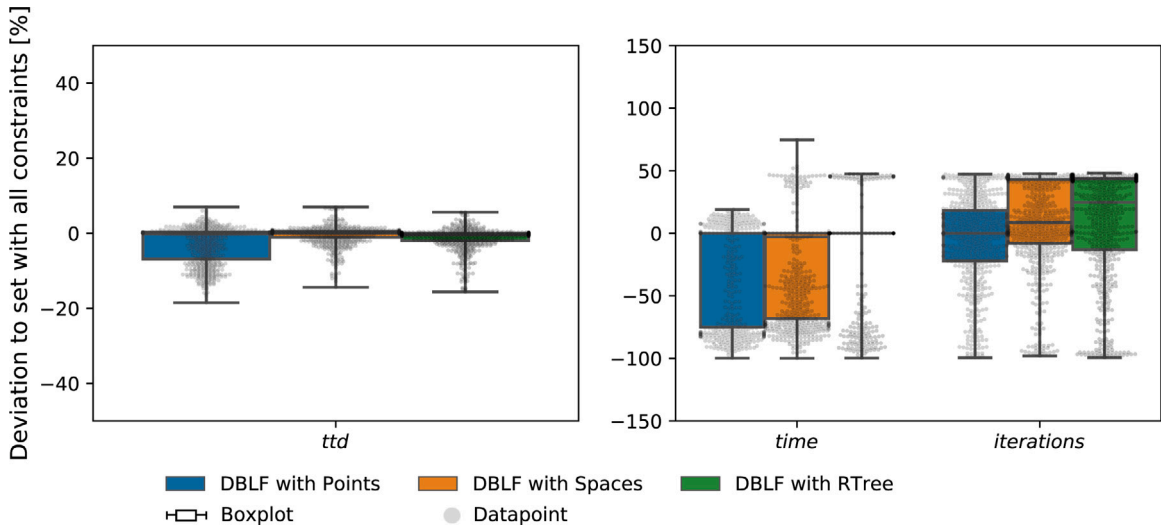



**Fig. 7.** Comparison of DBLF Results — Without Load Capacity.

### 5.3. Impact of loading constraints

Above, each instance set was tested 15 times for each DBLF algorithm variant while all loading constraints are fulfilled. In this section, we want to evaluate the sensitivity of the DBLF algorithm variants concerning the loading constraints. Therefore, we analyze the impact of the loading constraints for each variant by removing one loading constraint and testing each instance of our instance set again 15 times. As before, we report results concerning the total travel distance ($ttd$), the runtime ($time$), and the number of iterations. The relative deviation of the results where one loading constraint is excluded to the results considering all constraints with the same DBLF algorithm variant is calculated. These deviations are further visualized by boxplots showing the minimum, 1st quartile ($Q_1$), median, 3rd quartile ($Q_3$), and maximum values of the relative results. Each boxplot is overlapped by a swarm plot indicating the distribution of the data points.

In general, the number of performed iterations varies greatly for all results. On the one hand, since the performance increases, more iterations can be carried out when disregarding one constraint. Then, also the runtime increases while the total travel distance does not necessarily improve. On the other hand, the number of performed iterations along with the runtime can decrease because the solution space is larger and thus, a good solution can be found faster and the algorithm terminates earlier.

Fig. 6 visualizes the results for the instances when ignoring the Rotation constraint (L3). Consequently, the solution space is smaller without allowing the rotation of items. However, for most instances, the total travel distance remains mostly the same independent of the DBLF algorithm, as $Q_1$ to the median are around zero. For one-quarter of the instances ($Q_3$), the total travel distance increases by several percent — especially for DBLF with Spaces by up to 2.6%. In the case of DBLF with Spaces, the median runtime reduction is around 16%, in the case of DBLF with Points even 32%. In the case of DBLF with RTree, the runtime remains unchanged. Unlike as stated in Kang et al. (2012), who tackled the pure 3D container loading problem, we can conclude that

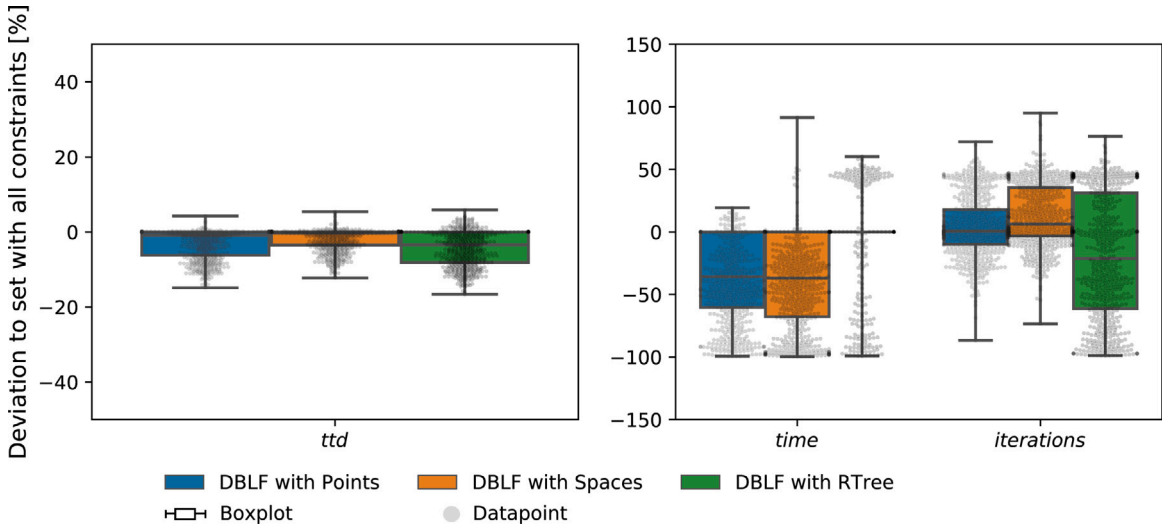



**Fig. 8.** Comparison of DBLF Results — Without LIFO.

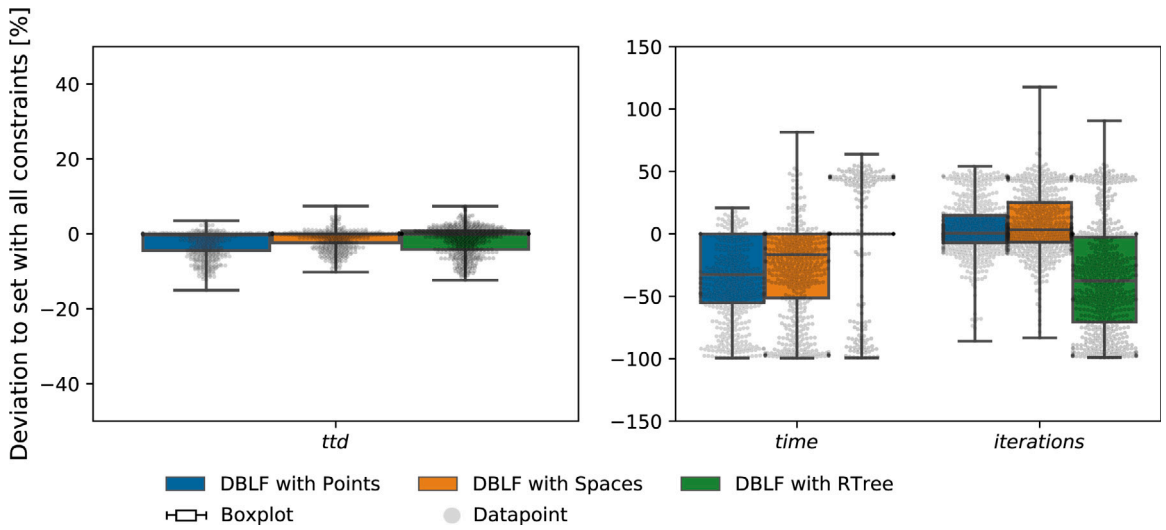



**Fig. 9.** Comparison of DBLF Results — Without Minimal Supporting Area.

the Rotation constraint has merely only small impacts on the objective value of the 3L-VRPTW.

In Fig. 7, the results obtained without taking the load capacity (L4) into account are shown. Since the range between median to $Q_3$ is around zero, the constraint has merely no effect on the total travel distance. For DBLF with Points, for one quarter ($Q_3$), the total travel distance decreases by up to 7.4%. In the case of DBLF with Spaces and DBLF with Points, the runtime decreases significantly ($Q_1$ up to −75%, $Q_3 = 0$); for DBLF with RTree, the runtime remains similar. The number of iterations varies widely for all algorithms ($Q_1 = -19\%$, $Q_3 = 43\%$). For our instances with a high number of customers, items, and item types, the solutions are generated faster with significantly fewer iterations. To conclude, the load capacity constraint has merely small impacts on the objective values and highly positive effects on the runtime.

Fig. 8 illustrates the results without considering the LIFO policy (L5). The total travel distance improves significantly for almost every instance ($Q_3 = 0$, $Q_1$ up to -8.3%). The runtime decreases significantly for DBLF with Points ($Q_1 \approx -62\%$) and especially for DBLF with Spaces ($Q_1 \approx -68\%$). For the DBLF with RTree, there are only isolated improvements in runtime ($Q_1$ to $Q_3 = 0$). To summarize, the LIFO constraint has a significant impact on the total travel distance and the performance.

Fig. 9 shows the impact of disregarding the minimal supporting area constraint (L6). The total travel distance remains the same (median and $Q_3$ are around zero) or decreases ($Q_1 \approx -4.8\%$). The runtime shows the same tendencies: $Q_1$ is −56% for DBLF with Points and −54% for

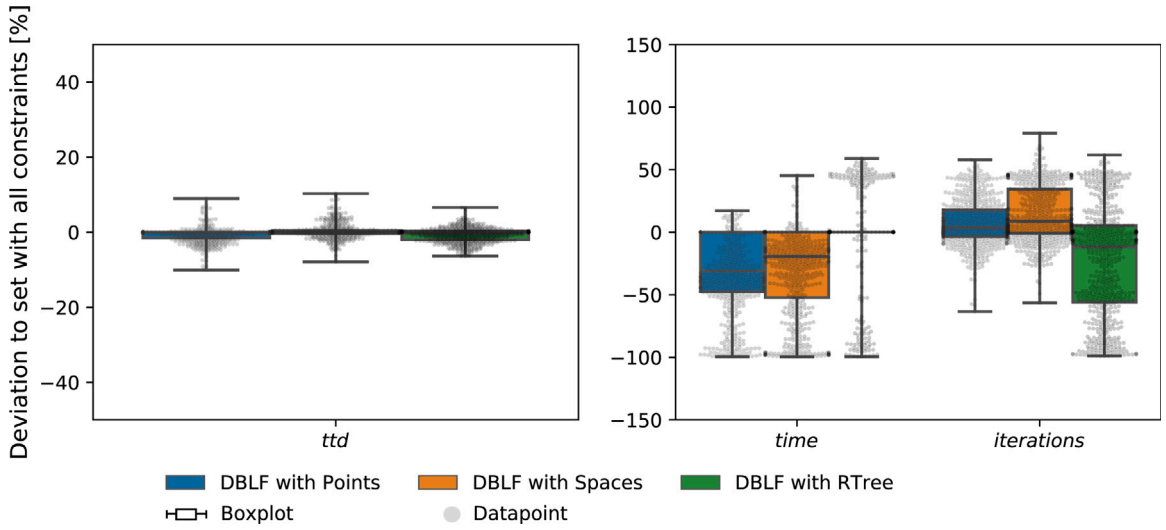



**Fig. 10.** Comparison of DBLF Results — Without Fragility.

DBLF with Spaces. Further analysis shows that improvements of the total travel distance are especially achieved for instances with a high number of different item types since the constraint is more restrictive for these instances than for instances with homogeneous items.

Fig. 10 visualizes the results without the consideration of an item's fragility (L7). The total travel distance remains unchanged for most instances or improves slightly ($Q_1 \approx -2\%$, $Q_3 \approx 0.4\%$). The runtime decreases significantly ($Q_3 = 0$, $Q_1$ up to 56%) for DBLF with Points and DBLF with Spaces. In the case of DBLF with RTree, there is no effect on the runtime.

In summary, disregarding the rotation of items and the fragility of items have rather small impacts on the total travel distance. However, load capacity, minimal supporting area, and the LIFO constraint show significant effects. When disregarding a constraint, the runtime for DBLF with Points and with Spaces decreases significantly. To conclude, when designing loading algorithms, the focus should be on the fulfillment of the LIFO policy and minimal supporting area to improve the performance. For our future work, we plan to improve the DBLF with Spaces in this respect.

## 6. Conclusion

The DBLF algorithm introduced by Karabulut and İnceoğlu (2005) is a widely-used loading algorithm and the basis of this paper. In the context of the 3L-VRPTW, the DBLF algorithm tackles the packing problem and finds feasible packing plans for a set of routes. Comparing the effectiveness of the investigated DBLF algorithm variants in this paper, our experiments show that DBLF with Spaces clearly finds superior results (shortest total travel distance) with the best performance (smallest runtime). Moreover, it outperforms the current benchmark by Zhang et al. (2017). Surprisingly, DBLF with RTree has a relatively high runtime caused by the necessary frequent restructuring of the tree. Therefore, other tree structures in combination with sliding techniques are worth to be examined further.

We also analyzed the impact of loading constraints per DBLF algorithm variant. The load capacity, the LIFO policy, and the minimal supporting area influence the total travel distance strongly. Therefore, based on the DBLF with Spaces variant, our future work will focus on the design and implementation of loading algorithms which are specialized in the fast fulfillment of these loading constraints. Despite the increased solution space, the results show that disregarding the rotation of items improves the total travel distance only slightly for most instances. Although the non-consideration of the fragility of items only has a minor impact on the total travel distance, for security reasons, this should be investigated further. As future work, we want to extend the problem w.r.t. a heterogeneous fleet.

## CRediT authorship contribution statement

**Corinna Krebs:** Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Data curation, Writing – original draft, Writing – review & editing, Visualization, Project administration. **Jan Fabian Ehmke:** Methodology, Validation, Resources, Writing – review & editing, Supervision. **Henriette Koch:** Software.

## Acknowledgments

This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors.

## Appendix A. Mathematical formulation

Based on Koch (2018), this section presents a mathematical formulation for the 3L-VRPTW. First, we introduce the decision variables followed by the formulations for the vehicle routing problem. The last subsection deals with the formulation of the loading constraints.

### A.1. Decision variables

There are four decision variables for the mathematical formulation. The first, shown in Eq. (A.1), represents the routing decision, i.e. that a vehicle $v$ connects customer $i$ with customer $j$ in period $t$. In period $t$, the vehicle $v$ has left node $i$ and has not yet reached the next node

$j$. Consequently, in period $t = 0$, the vehicle $v$ has left the depot and drives to the first customer.

$$\varepsilon_{i,j}^{t,v} = \begin{cases} 1, & \text{if vehicle } v \text{ drives directly from node } i \\ & \text{to node } j \text{ in period } t, \\ 0, & \text{otherwise} \end{cases} \tag{A.1}$$

The next decision variable (Eq. (A.2)) represents the decision for a position (x, y, z) of an item $I_{i,k}$ demanded by customer $i$, placed inside of vehicle $v$ in period $t$.

$$\pi_{x,y,z}^{i,k,t,v} = \begin{cases} 1, & \text{if the item } I_{i,k} \text{ of customer } i \text{ in period } t \\ & \text{inside of vehicle } v \text{ is placed with its minimal} \\ & \text{corner point at position (x, y, z),} \\ 0, & \text{otherwise} \end{cases} \tag{A.2}$$

The third decision variable, Eq. (A.3), describes the decision for the rotation of item $I_{i,k}$ along the length–width plane (see Rotation constraint (L3)).

$$\sigma_{i,k} = \begin{cases} 1, & \text{if the item } I_{i,k} \text{ is not rotated, e.g. the length } l_{i,k} \\ & \text{is parallel to the x-axis,} \\ 0, & \text{otherwise} \end{cases} \tag{A.3}$$

The last decision variable (Eq. (A.4)) determines whether a point (x, y, z) is occupied by an item $I_{i,k}$. This is necessary for the formulation of several loading constraints.

$$\rho_{x,y,z}^{i,k,v} = \begin{cases} 1, & \text{if an item } I_{i,k} \text{ in vehicle } v \text{ occupies the point (x, y, z)} \\ 0, & \text{otherwise} \end{cases} \tag{A.4}$$

### A.2. Vehicle routing

This section deals with the mathematical formulation of the vehicle routing part. If applicable, we link the formulas to the constraints introduced before. Note that constraint S1, ensuring the feasibility of a solution, is guaranteed through the mathematical formulation as a whole.

The objective function is to minimize the total travel distance ($ttd$), see Eq. (A.5).

$$\min ttd = \sum_{i \in N} \sum_{j \in N} \sum_{t \in N} \sum_{v=0}^{v_{used}} (d_{i,j} \cdot \varepsilon_{i,j}^{t,v}) \tag{A.5}$$

Eq. (A.6) corresponds to the solution constraint (S2), so that at least one vehicle and at most the number of available vehicles ($v_{max}$) are dispatched.

$$v_{used} \in [1, v_{max}] \tag{A.6}$$

The next equations deal with the creation of routes. Eq. (A.7) and the following equations ensure that each customer is left exactly once, corresponding to (S3). Eq. (A.8) ensures the connectivity of each route. Eq. (A.9) prevents the situation in which the vehicle travels to the same node again after leaving it. Each customer is visited and left by the same vehicle (Eq. (A.10)). The depot is left at most once in period 0 (Eq. (A.11)) and not later (Eq. (A.12)) in every route. These equations cover constraint (C1), ensuring that each route starts and ends at the depot and visiting at least one customer.

$$\sum_{j \in N} \sum_{t \in N} \sum_{v=0}^{v_{used}} \varepsilon_{i,j}^{t,v} = 1 \quad \forall i \in N \setminus \{0\} \tag{A.7}$$

$$\sum_{j \in N} \sum_{t \in N \setminus \{0\}} \sum_{v=0}^{v_{used}} (t \cdot \varepsilon_{i,j}^{t,v}) - \sum_{j \in N} \sum_{t \in N} \sum_{v=0}^{v_{used}} (t \cdot \varepsilon_{i,j}^{t,v}) = 1 \quad \forall i, t \in N \setminus \{0\} \tag{A.8}$$

$$\varepsilon_{i,i}^{t,v} = 0 \quad \forall i, t \in N, \forall v \tag{A.9}$$

$$\sum_{j \in N} \varepsilon_{i,j}^{t+1,v} - \sum_{j \in N} \varepsilon_{j,i}^{t,v} = 0 \quad \forall i \in N \setminus \{0\}, \forall t \in N \setminus \{n\}, \forall v \tag{A.10}$$

$$\sum_{j \in N \setminus \{0\}} \varepsilon_{0,j}^{0,v} \leq 1 \quad \forall v \tag{A.11}$$

$$\varepsilon_{0,j}^{t,v} = 0 \quad \forall j \in N \setminus \{0\}, \forall t \in N \setminus \{n\}, \forall v \tag{A.12}$$

The following equations ensure the consideration of the time windows. Hereby, $start_i^v$ is the time when the vehicle $v$ starts the unloading process at the customer $i$. $M_1$ is a sufficiently large number.[5] In Eq. (A.13), the start time for each route is set to zero. Eq. (A.14) ensures that the unloading of the items at customer $i$ does not start before the arrival at the customer $i$. Eq. (A.15) deals with the waiting times, resulting when the vehicle $v$ arrives at a customer $i$ before its ready time $RT_i$, which are considered also in the following calculations. Eq. (A.16) guarantees that the start of the unloading process at customer $i$ does not begin before the ready time $RT_i$. Due to Eq. (A.15), the unloading process at customer $i$ is not allowed to start after the due date $DD_i$. This corresponds to constraint (C2).

$$d_{0,i} - start_i^v \leq M_1 \cdot (1 - \varepsilon_{0,i}^{0,v}) \quad \forall i \in N \setminus \{0\}, \forall v \tag{A.13}$$

$$start_i^v + ST_i + d_{i,j} - start_j^v \leq M_1 \cdot (1 - \varepsilon_{i,j}^{t,v}) \quad \forall i \in N \setminus \{0\}, \forall j \in N, \forall v, \forall t \in N \setminus \{0\} \tag{A.14}$$

$$start_j^v \geq start_i^v + ST_i + d_{i,j} \quad \forall i \in N \setminus \{0\}, \forall j \in N, \forall v \tag{A.15}$$

$$start_i^v \geq RT_i \quad \forall i \in N, \forall v \tag{A.16}$$

$$start_i^v \leq DD_i \quad \forall i \in N, \forall v \tag{A.17}$$

### A.3. Container loading

This section deals with the mathematical formulation for the container loading part. We first introduce the required variables. After that, the formulation of the loading constraints follows. Please note that we assume integer values for positions and dimensions. Moreover, note that an item is positioned in its the rearmost, leftmost bottom corner point. These conditions guarantee the orthogonal packing of items, as expected in the Orthogonality constraint (L2). The following equations deal with the possible positions of an item $I_{i,k}$ inside of a vehicle loading space. To prevent overlapping, an item's dimensions (length, width, height) reduces the possible positions. Eq. (A.18) represents the available positions along the $x$-axis for an item $I_{i,k}$. As the rotation of an item $I_{i,k}$ along the length–width plane is allowed, the possible placement x-coordinates are reduced accordingly. The same applies to the possible y-coordinates, as stated in Eq. (A.19). The possible z-coordinates are only reduced by the item's height $h_{i,k}$ (see Eq. (A.20)).

$$X_{i,k} = \{0, 1, 2, \ldots, L - \min(l_{i,k}, w_{i,k})\} \tag{A.18}$$

$$Y_{i,k} = \{0, 1, 2, \ldots, W - \min(l_{i,k}, w_{i,k})\} \tag{A.19}$$

$$Z_{i,k} = \{0, 1, 2, \ldots, H - h_{i,k}\} \tag{A.20}$$

Using the rotation decision variable (Eq. (A.3)), the length and width of an item $I_{i,k}$ can be further described as:

$$l'_{i,k} = \sigma_{i,k} \cdot l_{i,k} + (1 - \sigma_{i,k}) \cdot w_{i,k} \tag{A.21}$$

$$w'_{i,k} = (1 - \sigma_{i,k}) \cdot l_{i,k} + \sigma_{i,k} \cdot w_{i,k} \tag{A.22}$$

The placement position ($x_{i,k}$, $y_{i,k}$, $z_{i,k}$,) of an item $I_{i,k}$ can be determined by using the placement decision variable as shown in

[5] E.g. $M_1 \geq \max_{i \in N} DD_i + \max_{i \in N} ST_i + \max_{i,j \in E} c_{i,j} - \min_{i \in N} RT_i$.

Eqs. (A.23)–(A.25):

$$x_{i,k} = \sum_{t\in N\setminus\{n\}} \sum_{v=0}^{v_{used}} \sum_{x\in X_{i,k}} \sum_{y\in Y_{i,k}} \sum_{z\in Z_{i,k}} x \cdot \pi_{x,y,z}^{i,k,t,v} \tag{A.23}$$

$$y_{i,k} = \sum_{t\in N\setminus\{n\}} \sum_{v=0}^{v_{used}} \sum_{x\in X_{i,k}} \sum_{y\in Y_{i,k}} \sum_{z\in Z_{i,k}} y \cdot \pi_{x,y,z}^{i,k,t,v} \tag{A.24}$$

$$z_{i,k} = \sum_{t\in N\setminus\{n\}} \sum_{v=0}^{v_{used}} \sum_{x\in X_{i,k}} \sum_{y\in Y_{i,k}} \sum_{z\in Z_{i,k}} z \cdot \pi_{x,y,z}^{i,k,t,v} \tag{A.25}$$

The occupancy of a point (x', y', z') by an item $I_{i,k}$ placed in point (x, y, z) is determined in the following way (see Eq. (A.26)):

$$\rho_{x',y',z'}^{i,k,v} = \sum_{t\in N\setminus\{n\}} \sum_{\{x\in X_{i,k}|x'-l'_{i,k}+1\le x\le x'\}} \sum_{\{y\in Y_{i,k}|y'-w'_{i,k}+1\le y\le y'\}} \sum_{\{z\in Z_{i,k}|z'-h_{i,k}+1\le z\le z'\}} \pi_{x,y,z}^{i,k,t,v} \tag{A.26}$$

Eq. (A.27) guarantees that for every item $I_{i,k}$ of a customer $i$ exactly one position is found inside of a vehicle $v$. Through Eq. (A.28), it is guaranteed that all items of customer $i$ are packed in the vehicle $v$. These two equations ensure the constraint (S4).

$$\sum_{k=1}^{c_i} \sum_{t\in N\setminus\{n\}} \sum_{v=0}^{v_{used}} \sum_{x\in X_{i,k}} \sum_{y\in Y_{i,k}} \sum_{z\in Z_{i,k}} \pi_{x,y,z}^{i,k,t,v} = 1 \qquad \forall i \in N\setminus\{0\} \tag{A.27}$$

$$\sum_{k=1}^{c_i} \sum_{x\in X_{i,k}} \sum_{y\in Y_{i,k}} \sum_{z\in Z_{i,k}} \pi_{x,y,z}^{i,k,t,v} = c_i \cdot \sum_{i\in N} \varepsilon_{i,j}^{t,v} \qquad \forall i \in N\setminus\{0\}, \forall t \in N\setminus\{n\}, \forall v \tag{A.28}$$

Eq. (A.29) ensures the non-overlapping of items by excluding the position points based on the set rotation.

$$\sum_{\{x\in X_{i,k}|x>L-l'_{i,k}\}} \sum_{y\in Y_{i,k}} \pi_{x,y,z}^{i,k,t,v} + \sum_{x\in X_{i,k}} \sum_{\{y\in Y_{i,k}|y>W-w'_{i,k}\}} \pi_{x,y,z}^{i,k,t,v} = 0$$
$$\forall i \in N\setminus\{0\}, \forall c_i, \forall v, \forall t \in N\setminus\{n\}, \forall z \in Z_{i,k} \tag{A.29}$$

Eq. (A.30) presents an alternative formulation for Eq. (A.29) where each point inside of the vehicle loading space is allowed to be occupied at most once. Combined with the previous definitions, this ensures the Geometry constraint (L1).

$$\sum_{i\in N\setminus\{0\}} \sum_{k}^{c_i} \rho_{x,y,z}^{i,k,v} \le 1 \qquad \forall v, \forall x \in X_{i,k}, \forall y \in Y_{i,k}, \forall z \in Z_{i,k} \tag{A.30}$$

Eq. (A.31) prevents exceeding of the vehicle loading space volume. To this end, the volumes of the loads for each customer $i$ and each item $I_{i,k}$ are added up, and the sum must be smaller than the available loading space volume of the vehicle $v$. Eq. (A.32) corresponds to the load capacity constraint (L4), ensuring that the load capacity of vehicle $v$ is not exceeded. Hereby, for each customer $i$ and each item $I_{i,k}$, the load mass is added up. The sum must be smaller than the load capacity of the vehicle $v$.

$$\sum_{i\in N\setminus\{0\}} \sum_{k=1}^{c_i} \sum_{j\in N} \sum_{t\in N\setminus\{0\}} \varepsilon_{i,j}^{t,v} \cdot l_{i,k} \cdot w_{i,k} \cdot h_{i,k} \le L \cdot W \cdot H \qquad \forall v \tag{A.31}$$

$$\sum_{i\in N\setminus\{0\}} \sum_{k=1}^{c_i} \sum_{j\in N} \sum_{t\in N\setminus\{0\}} \varepsilon_{i,j}^{t,v} \cdot m_{i,k} \le D \qquad \forall v \tag{A.32}$$

Eqs. (A.33) and (A.34) ensure the LIFO constraint (L5). The first one guarantees the LIFO policy along the $x$-axis. It prevents positions where an item $I_{i,k}$, placed in point (x, y, z) and delivered in period $t$, would be behind another item $I_{j,q}$ placed in point (x', y', z') and delivered in period $u$, a period later than $t$ ($u > t$). Consequently, Eq. (A.33) sums the items $I_{j,q}$ delivered later than period $t$ and being placed in front of item $I_{i,k}$ delivered in period $t$. The sum must be smaller than a sufficiently large number[6] $M_3$.

$$\sum_{j\in N\setminus\{0\}} \sum_{q}^{c_j} \sum_{\{u\in N|t<u<n\}} \sum_{\{x'\in X_{j,q}|x'\ge x+l'_{i,k}\}} \sum_{\{y'\in Y_{j,q}|y-w'_{j,q}+1\le y'\le y+w'_{i,k}-1\}} \sum_{\{z'\in Z_{j,q}|z-h_{j,q}+1\le z'\le z+h_{i,k}-1\}} \cdot \pi_{x',y',z'}^{j,q,u,v} \le (1-\pi_{x,y,z}^{i,k,t,v}) \cdot M_3$$
$$\forall i \in N\setminus\{0\}, \forall c_i, \forall v, \forall t \in N\setminus\{n\}, \forall x \in X_{i,k}, \forall y \in Y_{i,k}, \forall z \in Z_{i,k} \tag{A.33}$$

The same procedure as shown in Eq. (A.33) applies for Eq. (A.34) concerning the $z$-axis. Consequently, it prevents placements where an item $I_{i,k}$ placed in point (x, y, z) and delivered in period $t$ is underneath another item $I_{j,q}$ placed in point (x', y', z') and delivered in period $u$, a period later than $t$ ($u > t$).

$$\sum_{j\in N\setminus\{0\}} \sum_{q}^{c_j} \sum_{\{u\in N|t<u<n\}} \sum_{\{x'\in X_{j,q}|x-l'_{j,q}+1\le x'\le x+l'_{i,k}-1\}} \sum_{\{y'\in Y_{j,q}|y-w'_{j,q}+1\le y'\le y+w'_{i,k}-1\}} \sum_{\{z'\in Z_{j,q}|z'\ge z+h_{i,k}\}} \cdot \pi_{x',y',z'}^{j,q,u,v} \le (1-\pi_{x,y,z}^{i,k,t,v}) \cdot M_3$$
$$\forall i \in N\setminus\{0\}, \forall c_i, \forall v, \forall t \in N\setminus\{n\}, \forall x \in X_{i,k}, \forall y \in Y_{i,k}, \forall z \in Z_{i,k} \tag{A.34}$$

The minimal supporting area constraint (L6) is formulated in Eq. (A.35). Hereby, the sum of all points which lay directly underneath of item $I_{i,k}$ and are occupied by another item is determined. This sum must be equal or greater than the base area of item $I_{i,k}$ multiplied by the support parameter $\alpha$.

$$\sum_{j\in N\setminus\{0\}} \sum_{q}^{c_j} \sum_{\{x'\in X_{j,q}|x\le x'\le x+l'_{i,k}-1\}} \sum_{\{y'\in Y_{j,q}|y\le y'\le y+w'_{i,k}-1\}} \rho_{x',y',z-1}^{j,q,v} \ge \alpha \cdot l_{i,k} \cdot w_{i,k} \cdot \pi_{x,y,z}^{i,k,t,v}$$
$$\forall i \in N\setminus\{0\}, \forall c_i, \forall v, \forall t \in N\setminus\{n\}, \forall x \in X_{i,k}, \forall y \in Y_{i,k}, \forall z \in Z_{i,k}\setminus\{0\} \tag{A.35}$$

Eq. (A.36) corresponds to the fragility constraint (L7), where non-fragile items cannot be placed on top of fragile items. However, fragile items are allowed to stack on top of every item. Note that the fragility flag for fragile items is 1. Suppose that item $I_{i,k}$ is placed on top of another item $I_{j,q}$. Then, the top surface of item $I_{j,q}$ touches the bottom of item $I_{i,k}$. The corresponding fragility flags of the underlying items are added up. This is expressed by the left side of the equal sign. The other side expresses the number of allowed fragile items placed underneath item $I_{i,k}$. If item $I_{i,k}$ is fragile, it is $M_2$ (sufficient large number.[7]) If item $I_{i,k}$ is non-fragile, no fragile items are allowed to be placed underneath of item $I_{i,k}$ (= zero).

$$\sum_{j\in N\setminus\{0\}} \sum_{q}^{c_j} \sum_{\{u\in N|t\le u<n\}} \sum_{\{x'\in X_{j,q}|x-l'_{j,q}+1\le x'\le x+l'_{i,k}-1\}} \sum_{\{y'\in Y_{j,q}|y-w'_{j,q}+1\le y'\le y+w'_{i,k}-1\}} f_{j,q} \cdot \pi_{x',y',z-h_{j,q}}^{j,q,u,v} \le (1-(1-f_{i,k}) \cdot \pi_{x,y,z}^{i,k,t,v}) \cdot M_2$$
$$\forall i \in N\setminus\{0\}, \forall c_i, \forall v, \forall t \in N\setminus\{n\}, \forall x \in X_{i,k}, \forall y \in Y_{i,k}, \forall z \in Z_{i,k}\setminus\{0\} \tag{A.36}$$

## Appendix B. Result Tables

See Tables B.5–B.23.

[6] E.g. $M_3 \ge \sum_{i\in N\setminus\{0\}} \cdot c_i$.

[7] E.g. $M_2 \ge \sum_{i\in N\setminus\{0\}} \cdot c_i$.

**Table B.5**
Results for DBLF with Points — All Constraints.

| Instances | | | ∅ ttd | | ∅ time [s] | | ∅ iterations | |
|---|---|---|---|---|---|---|---|---|
| n | m | Item types | best | avg. | best | avg. | best | avg. |
| 20 | 200 | 3 | 408.62 | 409.13 | 589.39 | 592.61 | 7,844.40 | 7,937.79 |
| | | 10 | 417.03 | 417.88 | 1,804.29 | 1,804.56 | 5,215.70 | 5,264.47 |
| | | 100 | 422.22 | 422.61 | 1,805.44 | 1,828.12 | 7,482.55 | 7,482.71 |
| | 400 | 3 | 440.27 | 443.71 | 2,071.40 | 2,096.03 | 5,626.25 | 5,703.95 |
| | | 10 | 459.59 | 466.71 | 2,464.40 | 2,471.42 | 3,589.65 | 3,697.46 |
| | | 100 | 482.60 | 489.77 | 2,678.65 | 2,674.07 | 3,746.65 | 4,061.67 |
| 60 | 200 | 3 | 1,004.94 | 1,014.13 | 1,525.79 | 1,540.68 | 15,011.75 | 14,440.42 |
| | | 10 | 1,030.97 | 1,055.00 | 1,654.37 | 1,665.33 | 13,653.45 | 13,415.70 |
| | | 100 | 1,069.41 | 1,097.51 | 1,615.52 | 1,628.51 | 14,002.48 | 13,892.59 |
| | 400 | 3 | 1,341.82 | 1,367.85 | 1,957.30 | 1,962.52 | 11,383.23 | 10,963.66 |
| | | 10 | 1,355.07 | 1,404.86 | 2,372.11 | 2,381.11 | 10,748.58 | 11,044.82 |
| | | 100 | 1,473.33 | 1,496.02 | 2,893.46 | 2,903.03 | 10,405.80 | 10,332.49 |
| 100 | 200 | 3 | 1,194.90 | 1,207.99 | 2,164.28 | 2,162.31 | 15,070.53 | 14,427.46 |
| | | 10 | 1,271.18 | 1,297.21 | 2,265.07 | 2,270.37 | 14,603.18 | 14,534.88 |
| | | 100 | 1,358.49 | 1,388.50 | 2,109.06 | 2,109.28 | 15,031.83 | 14,604.46 |
| | 400 | 3 | 1,628.59 | 1,668.93 | 2,672.88 | 2,683.39 | 10,368.45 | 9,981.93 |
| | | 10 | 1,726.92 | 1,793.91 | 2,830.23 | 2,831.52 | 11,148.18 | 11,043.06 |
| | | 100 | 1,739.25 | 1,818.47 | 2,857.75 | 2,863.59 | 9,625.73 | 10,233.30 |
| Average | | | 1,167.34 | 1,195.68 | 2,174.97 | 2,182.33 | 11,187.05 | 11,065.92 |

**Table B.6**
Results for DBLF with Points — w/o Rotation.

| Instances | | | ∅ ttd | | ∅ time [s] | | ∅ iterations | |
|---|---|---|---|---|---|---|---|---|
| n | m | Item types | best | avg. | best | avg. | best | avg. |
| 20 | 200 | 3 | 408.91 | 409.16 | 467.79 | 465.98 | 8,068.85 | 8,104.62 |
| | | 10 | 417.60 | 418.24 | 1,803.15 | 1,803.36 | 5,285.35 | 5,328.53 |
| | | 100 | 422.22 | 423.37 | 948.70 | 977.51 | 7,951.25 | 8,002.39 |
| | 400 | 3 | 442.50 | 447.04 | 1,942.43 | 1,955.39 | 5,339.65 | 5,525.85 |
| | | 10 | 463.51 | 470.18 | 2,409.62 | 2,398.18 | 3,586.60 | 3,616.47 |
| | | 100 | 486.00 | 492.11 | 2,262.74 | 2,271.69 | 3,967.65 | 4,084.61 |
| 60 | 200 | 3 | 1,010.53 | 1,027.45 | 1,012.94 | 1,020.00 | 14,247.93 | 14,318.14 |
| | | 10 | 1,035.79 | 1,069.02 | 1,110.80 | 1,117.08 | 14,100.85 | 13,604.02 |
| | | 100 | 1,078.05 | 1,112.55 | 913.13 | 919.39 | 14,172.00 | 13,603.87 |
| | 400 | 3 | 1,375.90 | 1,398.23 | 1,560.48 | 1,557.57 | 10,968.15 | 11,355.55 |
| | | 10 | 1,396.63 | 1,461.18 | 1,232.17 | 1,236.06 | 10,673.05 | 10,638.23 |
| | | 100 | 1,492.83 | 1,584.12 | 1,249.35 | 1,254.54 | 9,807.60 | 9,829.57 |
| 100 | 200 | 3 | 1,201.26 | 1,213.04 | 1,828.87 | 1,841.72 | 14,844.18 | 15,025.90 |
| | | 10 | 1,267.13 | 1,286.97 | 1,993.51 | 2,005.06 | 16,028.05 | 15,163.26 |
| | | 100 | 1,350.88 | 1,374.65 | 1,880.68 | 1,895.77 | 16,103.13 | 16,258.16 |
| | 400 | 3 | 1,654.60 | 1,695.88 | 2,100.85 | 2,104.17 | 10,762.50 | 10,885.11 |
| | | 10 | 1,734.00 | 1,806.92 | 2,080.25 | 2,083.28 | 12,169.68 | 11,688.64 |
| | | 100 | 1,744.42 | 1,832.04 | 2,115.35 | 2,121.38 | 10,841.28 | 10,981.78 |
| Average | | | 1,177.49 | 1,212.81 | 1,599.71 | 1,606.14 | 11,454.54 | 11,378.90 |

**Table B.7**
Results for DBLF with Points — w/o LIFO.

| Instances | | | ∅ ttd | | ∅ time [s] | | ∅ iterations | |
|---|---|---|---|---|---|---|---|---|
| n | m | Item types | best | avg. | best | avg. | best | avg. |
| 20 | 200 | 3 | 408.21 | 408.49 | 369.68 | 377.67 | 7,970.65 | 8,078.55 |
| | | 10 | 413.89 | 414.39 | 1,446.72 | 1,478.37 | 5,393.75 | 5,274.66 |
| | | 100 | 419.71 | 420.49 | 786.30 | 795.92 | 6,811.95 | 6,939.82 |
| | 400 | 3 | 436.60 | 437.60 | 1,076.99 | 1,089.22 | 5,740.00 | 5,735.48 |
| | | 10 | 445.59 | 447.90 | 2,258.86 | 2,270.90 | 3,613.65 | 3,612.29 |
| | | 100 | 461.28 | 465.67 | 2,256.39 | 2,260.58 | 3,951.80 | 3,849.71 |
| 60 | 200 | 3 | 989.50 | 997.84 | 917.76 | 926.67 | 13,610.85 | 14,036.38 |
| | | 10 | 975.26 | 1,007.24 | 1,022.84 | 1,039.90 | 12,728.98 | 12,780.77 |
| | | 100 | 1,002.12 | 1,036.89 | 786.19 | 812.06 | 14,241.28 | 13,886.50 |
| | 400 | 3 | 1,326.42 | 1,338.36 | 1,508.52 | 1,535.82 | 10,937.93 | 11,288.85 |
| | | 10 | 1,289.83 | 1,335.88 | 1,446.92 | 1,454.15 | 10,145.23 | 10,558.46 |
| | | 100 | 1,387.14 | 1,454.60 | 1,202.02 | 1,209.00 | 10,169.03 | 10,461.06 |
| 100 | 200 | 3 | 1,178.14 | 1,187.66 | 1,716.21 | 1,721.82 | 14,832.25 | 14,630.52 |
| | | 10 | 1,209.19 | 1,226.96 | 1,908.55 | 1,905.60 | 15,465.08 | 14,883.87 |
| | | 100 | 1,279.41 | 1,301.27 | 1,794.60 | 1,811.47 | 15,813.83 | 15,527.56 |
| | 400 | 3 | 1,597.43 | 1,631.16 | 2,102.26 | 2,112.30 | 10,625.73 | 10,646.79 |
| | | 10 | 1,641.24 | 1,705.68 | 2,227.68 | 2,237.08 | 10,877.38 | 10,823.70 |
| | | 100 | 1,617.09 | 1,693.67 | 2,233.12 | 2,238.92 | 9,911.18 | 9,525.06 |
| Average | | | 1,119.03 | 1,147.63 | 1,530.94 | 1,542.74 | 11,073.31 | 11,052.98 |

**Table B.8**
Results for DBLF with Points — w/o Minimal Supporting Area.

| Instances | | | ∅ ttd | | ∅ time [s] | | ∅ iterations | |
|---|---|---|---|---|---|---|---|---|
| n | m | Item types | best | avg. | best | avg. | best | avg. |
| 20 | 200 | 3 | 408.81 | 409.39 | 515.87 | 516.96 | 7,986.95 | 8,079.97 |
| | | 10 | 416.85 | 417.59 | 1,692.97 | 1,700.14 | 5,355.60 | 5,405.89 |
| | | 100 | 420.15 | 421.47 | 365.73 | 381.90 | 7,562.85 | 7,675.31 |
| | 400 | 3 | 439.78 | 443.21 | 1,831.21 | 1,851.53 | 5,425.75 | 4,959.65 |
| | | 10 | 455.23 | 461.09 | 2,425.73 | 2,430.72 | 3,652.40 | 3,598.25 |
| | | 100 | 469.13 | 473.62 | 2,257.86 | 2,263.88 | 4,029.20 | 3,884.22 |
| 60 | 200 | 3 | 998.22 | 1,010.48 | 1,039.74 | 1,054.92 | 13,874.70 | 13,725.37 |
| | | 10 | 992.12 | 1,020.05 | 1,062.12 | 1,071.11 | 13,505.08 | 13,594.86 |
| | | 100 | 1,013.58 | 1,041.18 | 836.14 | 826.08 | 15,485.28 | 14,444.97 |
| | 400 | 3 | 1,325.77 | 1,351.19 | 1,568.54 | 1,582.66 | 11,164.83 | 10,931.38 |
| | | 10 | 1,307.44 | 1,354.69 | 1,394.26 | 1,401.98 | 10,140.78 | 10,652.61 |
| | | 100 | 1,378.41 | 1,437.85 | 1,214.58 | 1,218.45 | 11,356.30 | 10,524.39 |
| 100 | 200 | 3 | 1,184.56 | 1,196.98 | 1,838.95 | 1,840.67 | 15,812.93 | 14,968.25 |
| | | 10 | 1,226.99 | 1,248.10 | 1,954.39 | 1,971.51 | 14,817.60 | 14,577.91 |
| | | 100 | 1,292.19 | 1,311.62 | 1,647.78 | 1,691.51 | 16,394.00 | 16,253.25 |
| | 400 | 3 | 1,612.72 | 1,649.00 | 2,094.65 | 2,100.07 | 10,799.80 | 10,445.15 |
| | | 10 | 1,659.36 | 1,717.74 | 2,140.47 | 2,144.94 | 11,507.38 | 11,435.09 |
| | | 100 | 1,631.82 | 1,699.09 | 2,181.10 | 2,189.20 | 9,621.73 | 10,236.14 |
| Average | | | 1,128.54 | 1,156.74 | 1,567.83 | 1,577.71 | 11,432.45 | 11,239.40 |

**Table B.9**
Results for DBLF with Points — w/o Fragility.

| Instances | | | ∅ ttd | | ∅ time [s] | | ∅ iterations | |
|---|---|---|---|---|---|---|---|---|
| n | m | Item types | best | avg. | best | avg. | best | avg. |
| 20 | 200 | 3 | 408.78 | 409.15 | 475.00 | 489.61 | 8,067.60 | 8,297.08 |
| | | 10 | 416.90 | 417.53 | 1,701.20 | 1,707.18 | 5,397.30 | 5,402.51 |
| | | 100 | 420.52 | 422.51 | 475.05 | 502.02 | 8,003.25 | 7,819.21 |
| | 400 | 3 | 439.78 | 443.47 | 1,776.81 | 1,792.58 | 5,340.65 | 5,576.86 |
| | | 10 | 457.73 | 464.42 | 2,419.38 | 2,422.82 | 3,725.75 | 3,619.69 |
| | | 100 | 475.95 | 480.84 | 2,320.47 | 2,332.22 | 4,205.45 | 4,342.82 |
| 60 | 200 | 3 | 1,003.60 | 1,021.28 | 982.46 | 992.24 | 13,325.60 | 13,826.13 |
| | | 10 | 1,014.12 | 1,046.33 | 1,103.21 | 1,116.92 | 13,990.50 | 13,863.84 |
| | | 100 | 1,052.08 | 1,085.80 | 839.47 | 846.12 | 14,271.80 | 13,931.99 |
| | 400 | 3 | 1,332.43 | 1,356.79 | 1,554.15 | 1,567.07 | 11,188.58 | 10,968.21 |
| | | 10 | 1,337.50 | 1,402.06 | 1,302.63 | 1,309.05 | 10,768.55 | 10,501.32 |
| | | 100 | 1,440.80 | 1,524.92 | 1,148.59 | 1,155.42 | 10,370.58 | 10,101.56 |
| 100 | 200 | 3 | 1,186.87 | 1,198.90 | 1,858.05 | 1,869.46 | 15,294.35 | 14,697.74 |
| | | 10 | 1,248.84 | 1,269.29 | 2,053.31 | 2,053.75 | 15,392.05 | 15,459.34 |
| | | 100 | 1,321.72 | 1,345.33 | 1,818.43 | 1,824.73 | 15,991.73 | 16,102.71 |
| | 400 | 3 | 1,620.74 | 1,663.94 | 2,113.01 | 2,129.61 | 11,210.15 | 10,619.06 |
| | | 10 | 1,696.96 | 1,764.17 | 2,154.56 | 2,154.60 | 11,632.85 | 11,561.21 |
| | | 100 | 1,686.34 | 1,772.67 | 2,215.36 | 2,213.42 | 10,776.08 | 10,560.68 |
| Average | | | 1,150.12 | 1,184.70 | 1,581.81 | 1,590.37 | 11,438.85 | 11,314.86 |

**Table B.10**
Results for DBLF with Points — w/o Load Capacity.

| Instances | | | ∅ ttd | | ∅ time [s] | | ∅ iterations | |
|---|---|---|---|---|---|---|---|---|
| n | m | Item types | best | avg. | best | avg. | best | avg. |
| 20 | 200 | 3 | 406.97 | 409.69 | 567.68 | 568.90 | 7,213.55 | 7,127.09 |
| | | 10 | 416.99 | 417.86 | 1,452.03 | 1,452.44 | 5,416.75 | 5,405.27 |
| | | 100 | 422.22 | 423.87 | 1,237.98 | 1,241.07 | 7,689.45 | 7,208.60 |
| | 400 | 3 | 439.77 | 441.31 | 1,826.25 | 1,839.18 | 3,963.70 | 3,861.44 |
| | | 10 | 459.42 | 465.21 | 2,474.89 | 2,484.19 | 3,421.30 | 2,999.56 |
| | | 100 | 481.63 | 487.48 | 2,697.56 | 2,686.80 | 3,029.95 | 2,645.97 |
| 60 | 200 | 3 | 995.66 | 999.06 | 1,172.50 | 1,174.24 | 13,933.75 | 13,537.70 |
| | | 10 | 1,013.66 | 1,022.58 | 1,322.13 | 1,323.84 | 14,631.98 | 14,582.10 |
| | | 100 | 1,049.25 | 1,056.89 | 1,488.99 | 1,482.64 | 15,312.65 | 15,034.73 |
| | 400 | 3 | 1,338.54 | 1,348.82 | 1,860.44 | 1,860.67 | 9,351.63 | 8,966.77 |
| | | 10 | 1,355.94 | 1,370.59 | 2,039.40 | 2,043.99 | 11,258.18 | 11,056.39 |
| | | 100 | 1,477.76 | 1,494.71 | 2,340.28 | 2,348.89 | 11,541.08 | 11,169.88 |
| 100 | 200 | 3 | 1,180.56 | 1,186.65 | 1,471.07 | 1,476.04 | 14,569.43 | 14,232.62 |
| | | 10 | 1,250.06 | 1,258.45 | 1,745.19 | 1,748.69 | 15,638.30 | 15,661.47 |
| | | 100 | 1,337.27 | 1,347.34 | 1,710.06 | 1,704.67 | 16,601.08 | 16,115.42 |
| | 400 | 3 | 1,626.05 | 1,647.85 | 2,546.67 | 2,552.41 | 9,204.53 | 9,110.51 |
| | | 10 | 1,732.93 | 1,755.37 | 2,640.72 | 2,643.01 | 10,400.23 | 10,108.76 |
| | | 100 | 1,735.57 | 1,765.63 | 2,637.99 | 2,631.00 | 9,477.10 | 9,964.76 |
| Average | | | 1,160.45 | 1,171.78 | 1,873.58 | 1,875.09 | 11,152.48 | 10,944.34 |

**Table B.11**
Results for DBLF with Spaces — All Constraints.

| Instances | | | ∅ ttd | | ∅ time [s] | | ∅ iterations | |
|---|---|---|---|---|---|---|---|---|
| n | m | Item types | best | avg. | best | avg. | best | avg. |
| 20 | 200 | 3 | 408.37 | 408.71 | 570.65 | 572.66 | 7,831.90 | 7,624.40 |
| | | 10 | 416.65 | 417.69 | 1,801.83 | 1,801.96 | 4,581.00 | 4,837.80 |
| | | 100 | 421.79 | 422.36 | 1,711.44 | 1,728.29 | 5,777.45 | 5,678.51 |
| | 400 | 3 | 441.07 | 442.01 | 1,928.87 | 1,948.83 | 4,968.50 | 5,031.05 |
| | | 10 | 459.86 | 463.82 | 2,971.65 | 2,956.95 | 2,996.15 | 2,892.58 |
| | | 100 | 484.24 | 487.10 | 3,147.70 | 3,164.31 | 2,505.60 | 2,369.15 |
| 60 | 200 | 3 | 994.88 | 999.09 | 1,196.22 | 1,200.62 | 13,226.43 | 13,479.38 |
| | | 10 | 1,012.49 | 1,018.56 | 1,852.71 | 1,836.65 | 13,072.60 | 13,059.84 |
| | | 100 | 1,044.51 | 1,050.06 | 1,854.31 | 1,846.06 | 14,036.33 | 13,790.60 |
| | 400 | 3 | 1,334.01 | 1,342.53 | 1,997.95 | 2,013.29 | 10,258.13 | 10,420.38 |
| | | 10 | 1,348.19 | 1,373.07 | 2,389.56 | 2,377.26 | 9,676.28 | 9,558.20 |
| | | 100 | 1,471.60 | 1,487.80 | 2,702.07 | 2,698.91 | 9,433.98 | 9,324.15 |
| 100 | 200 | 3 | 1,184.89 | 1,193.46 | 1,542.37 | 1,578.03 | 14,592.98 | 14,883.33 |
| | | 10 | 1,254.88 | 1,268.18 | 2,169.92 | 2,168.66 | 13,863.38 | 14,688.41 |
| | | 100 | 1,336.98 | 1,351.89 | 2,070.42 | 2,068.13 | 15,547.10 | 15,463.70 |
| | 400 | 3 | 1,623.81 | 1,649.47 | 2,585.43 | 2,591.33 | 8,961.93 | 9,340.71 |
| | | 10 | 1,722.33 | 1,760.55 | 2,751.19 | 2,755.60 | 9,140.43 | 9,027.38 |
| | | 100 | 1,730.86 | 1,769.52 | 2,822.94 | 2,822.98 | 7,268.48 | 7,344.16 |
| Average | | | 1,158.36 | 1,172.33 | 2,133.41 | 2,136.27 | 10,227.22 | 10,306.46 |

**Table B.12**
Results for DBLF with Spaces — w/o Rotation.

| Instances | | | ∅ ttd | | ∅ time [s] | | ∅ iterations | |
|---|---|---|---|---|---|---|---|---|
| n | m | Item types | best | avg. | best | avg. | best | avg. |
| 20 | 200 | 3 | 408.72 | 408.90 | 452.12 | 458.25 | 7,743.55 | 7,606.63 |
| | | 10 | 417.73 | 418.43 | 1,600.41 | 1,621.40 | 5,154.85 | 4,903.65 |
| | | 100 | 422.22 | 423.99 | 581.24 | 601.45 | 6,076.55 | 6,145.43 |
| | 400 | 3 | 441.99 | 444.38 | 1,829.69 | 1,846.42 | 4,644.60 | 4,944.77 |
| | | 10 | 463.15 | 468.81 | 2,930.18 | 2,950.96 | 2,825.00 | 2,779.67 |
| | | 100 | 486.25 | 490.85 | 3,097.23 | 3,107.85 | 2,595.85 | 2,640.54 |
| 60 | 200 | 3 | 1,007.42 | 1,024.58 | 788.50 | 785.25 | 14,231.63 | 14,045.66 |
| | | 10 | 1,034.21 | 1,065.49 | 947.70 | 955.44 | 14,259.68 | 13,465.95 |
| | | 100 | 1,072.97 | 1,107.22 | 794.92 | 803.29 | 14,048.58 | 14,293.42 |
| | 400 | 3 | 1,367.66 | 1,389.25 | 1,385.63 | 1,402.57 | 10,867.20 | 10,759.32 |
| | | 10 | 1,391.70 | 1,455.47 | 1,177.77 | 1,180.26 | 9,961.60 | 9,745.82 |
| | | 100 | 1,487.46 | 1,575.47 | 1,169.42 | 1,177.72 | 10,185.63 | 9,462.70 |
| 100 | 200 | 3 | 1,195.44 | 1,204.05 | 1,437.06 | 1,446.24 | 15,351.18 | 15,266.00 |
| | | 10 | 1,260.62 | 1,273.53 | 1,943.74 | 1,957.20 | 15,131.15 | 14,856.15 |
| | | 100 | 1,343.13 | 1,357.84 | 1,864.35 | 1,874.03 | 15,846.53 | 15,761.38 |
| | 400 | 3 | 1,648.48 | 1,690.96 | 1,799.21 | 1,798.69 | 10,590.55 | 10,681.94 |
| | | 10 | 1,736.23 | 1,803.30 | 1,901.75 | 1,906.36 | 10,466.95 | 10,608.23 |
| | | 100 | 1,742.13 | 1,823.30 | 2,005.08 | 2,015.79 | 8,572.33 | 8,650.50 |
| Average | | | 1,173.83 | 1,206.54 | 1,497.37 | 1,506.40 | 10,935.55 | 10,807.16 |

**Table B.13**
Results for DBLF with Spaces — w/o LIFO.

| Instances | | | ∅ ttd | | ∅ time [s] | | ∅ iterations | |
|---|---|---|---|---|---|---|---|---|
| n | m | Item types | best | avg. | best | avg. | best | avg. |
| 20 | 200 | 3 | 408.21 | 408.48 | 329.31 | 330.97 | 7,506.25 | 7,648.59 |
| | | 10 | 413.95 | 414.63 | 856.68 | 870.78 | 5,099.60 | 5,100.00 |
| | | 100 | 419.98 | 420.81 | 535.98 | 545.40 | 5,097.40 | 5,247.48 |
| | 400 | 3 | 436.58 | 436.84 | 1,233.23 | 1,246.37 | 5,269.55 | 5,371.57 |
| | | 10 | 444.93 | 447.24 | 2,437.06 | 2,444.79 | 2,970.10 | 3,041.40 |
| | | 100 | 461.97 | 466.56 | 2,548.12 | 2,563.00 | 2,606.25 | 2,656.50 |
| 60 | 200 | 3 | 988.53 | 996.97 | 575.95 | 588.24 | 13,838.25 | 13,490.59 |
| | | 10 | 972.46 | 1,004.84 | 807.19 | 813.71 | 13,136.53 | 13,296.18 |
| | | 100 | 1,001.45 | 1,035.51 | 598.99 | 614.22 | 13,959.30 | 13,516.68 |
| | 400 | 3 | 1,320.57 | 1,330.34 | 1,206.76 | 1,234.29 | 10,539.65 | 10,470.26 |
| | | 10 | 1,285.76 | 1,332.08 | 1,366.33 | 1,367.25 | 10,105.55 | 10,155.68 |
| | | 100 | 1,388.06 | 1,453.33 | 1,139.85 | 1,136.29 | 10,238.88 | 9,819.24 |
| 100 | 200 | 3 | 1,173.70 | 1,180.70 | 1,259.46 | 1,269.14 | 14,282.20 | 14,273.39 |
| | | 10 | 1,207.79 | 1,218.37 | 1,678.45 | 1,687.20 | 15,030.15 | 14,455.12 |
| | | 100 | 1,277.65 | 1,292.84 | 1,539.75 | 1,551.51 | 16,147.00 | 15,636.21 |
| | 400 | 3 | 1,593.60 | 1,629.46 | 1,595.55 | 1,603.93 | 10,873.03 | 10,807.45 |
| | | 10 | 1,642.82 | 1,707.21 | 1,783.92 | 1,792.05 | 10,738.83 | 10,623.41 |
| | | 100 | 1,618.75 | 1,696.49 | 2,108.03 | 2,108.13 | 8,539.70 | 8,479.40 |
| Average | | | 1,117.60 | 1,145.03 | 1,308.70 | 1,317.78 | 10,780.24 | 10,637.09 |

**Table B.14**
Results for DBLF with Spaces — w/o Minimal Supporting Area.

| Instances | | | ∅ ttd | | ∅ time [s] | | ∅ iterations | |
|---|---|---|---|---|---|---|---|---|
| n | m | Item types | best | avg. | best | avg. | best | avg. |
| 20 | 200 | 3 | 408.65 | 409.39 | 437.47 | 441.51 | 7,529.15 | 7,557.73 |
| | | 10 | 416.33 | 417.51 | 1,520.85 | 1,559.53 | 5,134.50 | 4,970.59 |
| | | 100 | 420.34 | 421.34 | 228.30 | 245.78 | 5,689.30 | 5,730.79 |
| | 400 | 3 | 440.58 | 441.25 | 2,002.58 | 2,012.66 | 3,926.50 | 3,949.37 |
| | | 10 | 458.05 | 462.06 | 2,879.00 | 2,901.52 | 3,108.95 | 2,831.01 |
| | | 100 | 471.97 | 475.01 | 2,601.71 | 2,608.84 | 2,653.40 | 2,489.50 |
| 60 | 200 | 3 | 995.29 | 1,008.50 | 819.68 | 824.65 | 14,436.88 | 14,013.88 |
| | | 10 | 989.98 | 1,016.94 | 967.64 | 972.97 | 12,824.20 | 12,990.43 |
| | | 100 | 1,010.01 | 1,035.21 | 721.39 | 730.13 | 14,302.30 | 13,939.84 |
| | 400 | 3 | 1,322.57 | 1,345.48 | 1,481.23 | 1,506.78 | 10,727.40 | 10,417.03 |
| | | 10 | 1,302.24 | 1,350.97 | 1,342.05 | 1,355.23 | 10,526.20 | 10,382.11 |
| | | 100 | 1,378.11 | 1,433.78 | 1,161.19 | 1,164.13 | 10,451.55 | 10,227.31 |
| 100 | 200 | 3 | 1,177.41 | 1,186.67 | 1,513.40 | 1,512.08 | 15,165.43 | 14,610.00 |
| | | 10 | 1,226.13 | 1,236.55 | 1,902.86 | 1,913.58 | 14,862.10 | 14,648.62 |
| | | 100 | 1,287.70 | 1,301.14 | 1,691.43 | 1,690.43 | 15,522.55 | 15,801.54 |
| | 400 | 3 | 1,605.99 | 1,644.84 | 1,813.54 | 1,824.48 | 10,526.28 | 10,321.45 |
| | | 10 | 1,663.03 | 1,716.63 | 1,921.66 | 1,924.27 | 10,701.13 | 10,610.26 |
| | | 100 | 1,633.08 | 1,696.28 | 2,084.52 | 2,101.14 | 8,536.35 | 8,554.64 |
| Average | | | 1,126.63 | 1,152.42 | 1,483.70 | 1,493.65 | 10,840.22 | 10,685.44 |

**Table B.15**
Results for DBLF with Spaces — w/o Fragility.

| Instances | | | ∅ ttd | | ∅ time [s] | | ∅ iterations | |
|---|---|---|---|---|---|---|---|---|
| n | m | Item types | best | avg. | best | avg. | best | avg. |
| 20 | 200 | 3 | 408.56 | 410.11 | 419.81 | 422.88 | 7,293.25 | 7,758.01 |
| | | 10 | 416.89 | 417.44 | 1,499.46 | 1,524.56 | 4,809.90 | 5,010.31 |
| | | 100 | 421.21 | 422.61 | 269.33 | 285.52 | 5,968.30 | 5,881.23 |
| | 400 | 3 | 440.05 | 441.23 | 1,751.98 | 1,765.15 | 5,387.20 | 5,341.13 |
| | | 10 | 459.32 | 464.01 | 2,806.26 | 2,817.79 | 2,973.35 | 2,989.39 |
| | | 100 | 476.45 | 480.99 | 2,918.19 | 2,929.88 | 3,014.30 | 2,787.38 |
| 60 | 200 | 3 | 1,000.50 | 1,016.45 | 715.47 | 724.41 | 14,121.63 | 14,104.52 |
| | | 10 | 1,011.92 | 1,042.87 | 844.95 | 844.39 | 13,806.13 | 13,398.73 |
| | | 100 | 1,046.86 | 1,076.19 | 696.98 | 695.86 | 14,420.10 | 14,311.00 |
| | 400 | 3 | 1,327.26 | 1,348.91 | 1,454.52 | 1,453.03 | 10,668.05 | 10,753.20 |
| | | 10 | 1,334.51 | 1,394.52 | 1,207.79 | 1,213.79 | 9,325.65 | 9,257.27 |
| | | 100 | 1,438.99 | 1,517.72 | 1,096.30 | 1,099.78 | 9,474.13 | 9,479.32 |
| 100 | 200 | 3 | 1,181.27 | 1,191.09 | 1,375.64 | 1,377.47 | 14,913.50 | 14,943.72 |
| | | 10 | 1,245.55 | 1,256.90 | 1,920.51 | 1,931.45 | 15,433.88 | 14,995.97 |
| | | 100 | 1,318.60 | 1,332.38 | 1,775.83 | 1,778.25 | 16,679.58 | 16,142.49 |
| | 400 | 3 | 1,614.25 | 1,658.45 | 1,761.90 | 1,769.93 | 11,119.53 | 10,730.44 |
| | | 10 | 1,691.75 | 1,757.89 | 1,948.17 | 1,942.49 | 10,705.50 | 10,627.61 |
| | | 100 | 1,685.57 | 1,761.36 | 2,037.85 | 2,050.86 | 8,690.75 | 8,650.82 |
| Average | | | 1,147.22 | 1,178.19 | 1,444.56 | 1,450.31 | 10,938.77 | 10,818.59 |

**Table B.16**
Results for DBLF with Spaces — w/o Load Capacity.

| Instances | | | ∅ ttd | | ∅ time [s] | | ∅ iterations | |
|---|---|---|---|---|---|---|---|---|
| n | m | Item types | best | avg. | best | avg. | best | avg. |
| 20 | 200 | 3 | 406.08 | 409.12 | 446.32 | 446.63 | 7,412.45 | 7,401.08 |
| | | 10 | 416.80 | 417.77 | 1,449.79 | 1,450.19 | 5,280.60 | 5,038.72 |
| | | 100 | 422.40 | 423.89 | 1,299.14 | 1,300.57 | 5,947.80 | 6,019.43 |
| | 400 | 3 | 437.21 | 440.89 | 1,946.63 | 1,944.76 | 3,811.80 | 4,061.39 |
| | | 10 | 460.34 | 465.80 | 2,667.39 | 2,672.47 | 3,143.75 | 3,048.27 |
| | | 100 | 484.64 | 488.58 | 3,118.45 | 3,121.29 | 2,711.60 | 2,776.69 |
| 60 | 200 | 3 | 993.82 | 996.09 | 919.31 | 923.67 | 14,213.73 | 13,495.38 |
| | | 10 | 1,010.14 | 1,018.80 | 1,366.22 | 1,368.87 | 13,807.43 | 13,671.16 |
| | | 100 | 1,043.73 | 1,053.38 | 1,292.82 | 1,293.85 | 14,825.48 | 14,343.13 |
| | 400 | 3 | 1,331.68 | 1,338.69 | 1,870.96 | 1,874.00 | 9,689.85 | 9,531.50 |
| | | 10 | 1,350.31 | 1,363.80 | 2,081.61 | 2,101.58 | 10,670.30 | 10,801.72 |
| | | 100 | 1,473.23 | 1,489.41 | 2,398.97 | 2,394.19 | 11,181.73 | 10,997.51 |
| 100 | 200 | 3 | 1,173.16 | 1,179.29 | 1,170.69 | 1,181.61 | 14,161.98 | 14,151.12 |
| | | 10 | 1,242.86 | 1,253.01 | 1,539.73 | 1,547.17 | 15,963.08 | 15,836.05 |
| | | 100 | 1,332.48 | 1,341.65 | 1,722.82 | 1,714.17 | 15,706.48 | 15,489.47 |
| | 400 | 3 | 1,621.09 | 1,638.17 | 2,550.46 | 2,551.55 | 8,972.85 | 8,454.31 |
| | | 10 | 1,720.00 | 1,747.77 | 2,647.00 | 2,642.66 | 8,751.63 | 8,474.87 |
| | | 100 | 1,729.44 | 1,763.16 | 2,610.94 | 2,607.27 | 7,972.75 | 8,259.76 |
| Average | | | 1,155.71 | 1,167.08 | 1,842.36 | 1,844.57 | 10,671.42 | 10,511.92 |

**Table B.17**
Results for DBLF with RTree — All Constraints.

| Instances | | | ∅ ttd | | ∅ time [s] | | ∅ iterations | |
|---|---|---|---|---|---|---|---|---|
| n | m | Item types | best | avg. | best | avg. | best | avg. |
| 20 | 200 | 3 | 408.91 | 411.15 | 1,171.58 | 1,175.36 | 6,049.70 | 6,203.83 |
| | | 10 | 418.90 | 422.45 | 1,766.07 | 1,769.97 | 4,227.85 | 4,281.71 |
| | | 100 | 423.33 | 427.00 | 2,305.45 | 2,313.50 | 4,182.85 | 4,176.75 |
| | 400 | 3 | 443.15 | 449.27 | 2,482.88 | 2,517.98 | 3,688.00 | 3,446.39 |
| | | 10 | 465.89 | 477.33 | 2,604.76 | 2,604.60 | 2,980.70 | 2,865.67 |
| | | 100 | 492.93 | 506.85 | 2,492.67 | 2,498.13 | 3,202.80 | 3,116.92 |
| 60 | 200 | 3 | 1,036.32 | 1,050.28 | 2,931.78 | 2,943.80 | 3,951.33 | 3,852.55 |
| | | 10 | 1,102.85 | 1,131.68 | 2,992.54 | 2,997.29 | 2,825.28 | 2,750.34 |
| | | 100 | 1,165.00 | 1,190.18 | 3,301.62 | 3,307.65 | 2,180.15 | 2,315.36 |
| | 400 | 3 | 1,391.19 | 1,417.44 | 2,856.99 | 2,870.60 | 3,980.13 | 4,199.43 |
| | | 10 | 1,465.84 | 1,522.38 | 2,779.39 | 2,792.66 | 5,581.85 | 5,761.70 |
| | | 100 | 1,607.81 | 1,680.72 | 2,304.35 | 2,318.99 | 6,661.90 | 6,834.69 |
| 100 | 200 | 3 | 1,242.08 | 1,259.40 | 3,282.33 | 3,289.11 | 3,686.55 | 3,568.99 |
| | | 10 | 1,394.36 | 1,405.74 | 3,517.30 | 3,521.70 | 2,220.23 | 2,103.43 |
| | | 100 | 1,512.91 | 1,532.77 | 3,485.58 | 3,487.72 | 1,404.75 | 1,409.09 |
| | 400 | 3 | 1,739.93 | 1,759.22 | 3,389.67 | 3,395.15 | 2,872.33 | 2,576.32 |
| | | 10 | 1,950.96 | 1,982.12 | 3,600.00 | 3,600.00 | 422.03 | 422.33 |
| | | 100 | 2,012.89 | 2,050.93 | 3,600.00 | 3,600.00 | 227.48 | 223.41 |
| Average | | | 1,263.25 | 1,288.66 | 2,963.55 | 2,970.96 | 3,212.00 | 3,204.22 |

**Table B.18**
Results for DBLF with RTree — w/o Rotation.

| Instances | | | ∅ ttd | | ∅ time [s] | | ∅ iterations | |
|---|---|---|---|---|---|---|---|---|
| n | m | Item types | best | avg. | best | avg. | best | avg. |
| 20 | 200 | 3 | 408.85 | 409.70 | 1,205.88 | 1,215.16 | 5,159.90 | 5,366.50 |
| | | 10 | 420.34 | 424.29 | 1,811.27 | 1,811.84 | 4,127.05 | 4,159.02 |
| | | 100 | 424.16 | 428.91 | 1,787.82 | 1,796.72 | 3,611.10 | 3,749.56 |
| | 400 | 3 | 446.10 | 453.04 | 2,584.06 | 2,602.60 | 1,500.95 | 1,542.49 |
| | | 10 | 472.71 | 480.51 | 2,632.29 | 2,642.64 | 989.45 | 987.88 |
| | | 100 | 494.58 | 507.82 | 2,567.03 | 2,560.71 | 713.15 | 712.91 |
| 60 | 200 | 3 | 1,043.12 | 1,063.39 | 1,854.86 | 1,860.17 | 3,851.10 | 3,898.28 |
| | | 10 | 1,115.25 | 1,143.96 | 1,949.03 | 1,953.80 | 3,019.58 | 2,863.58 |
| | | 100 | 1,165.34 | 1,198.22 | 1,950.89 | 1,952.51 | 2,656.65 | 2,661.63 |
| | 400 | 3 | 1,425.22 | 1,451.14 | 2,953.36 | 2,969.49 | 2,194.63 | 1,701.78 |
| | | 10 | 1,497.34 | 1,554.39 | 2,818.67 | 2,825.45 | 4,175.38 | 4,406.43 |
| | | 100 | 1,613.93 | 1,696.99 | 2,310.29 | 2,313.46 | 6,270.70 | 6,026.75 |
| 100 | 200 | 3 | 1,252.47 | 1,280.29 | 3,356.19 | 3,360.01 | 110.38 | 111.84 |
| | | 10 | 1,386.87 | 1,414.46 | 3,549.66 | 3,551.13 | 92.85 | 93.56 |
| | | 100 | 1,517.46 | 1,543.80 | 3,520.80 | 3,521.71 | 98.00 | 97.92 |
| | 400 | 3 | 1,766.55 | 1,789.38 | 3,332.02 | 3,333.50 | 3,171.93 | 2,753.72 |
| | | 10 | 1,976.12 | 2,003.44 | 3,600.00 | 3,600.00 | 451.03 | 457.74 |
| | | 100 | 2,025.83 | 2,071.39 | 3,600.00 | 3,600.00 | 256.63 | 257.68 |
| Average | | | 1,274.59 | 1,304.20 | 2,739.33 | 2,743.74 | 2,293.31 | 2,239.34 |

**Table B.19**
Results for DBLF with RTree — w/o LIFO.

| Instances | | | ∅ ttd | | ∅ time [s] | | ∅ iterations | |
|---|---|---|---|---|---|---|---|---|
| n | m | Item types | best | avg. | best | avg. | best | avg. |
| 20 | 200 | 3 | 408.38 | 408.61 | 984.83 | 992.25 | 6,169.15 | 6,239.68 |
| | | 10 | 414.69 | 415.68 | 1,782.06 | 1,805.08 | 4,201.20 | 4,231.15 |
| | | 100 | 420.93 | 422.44 | 1,367.34 | 1,371.89 | 4,269.55 | 4,315.68 |
| | 400 | 3 | 436.90 | 439.20 | 2,235.03 | 2,256.21 | 2,520.10 | 2,328.57 |
| | | 10 | 447.88 | 452.47 | 2,608.75 | 2,611.99 | 1,831.70 | 1,922.01 |
| | | 100 | 469.65 | 475.92 | 2,547.27 | 2,561.16 | 2,127.85 | 2,165.85 |
| 60 | 200 | 3 | 1,012.61 | 1,023.80 | 1,759.85 | 1,777.77 | 4,051.08 | 3,959.32 |
| | | 10 | 1,036.71 | 1,062.54 | 1,931.42 | 1,938.81 | 2,801.40 | 2,678.86 |
| | | 100 | 1,077.29 | 1,106.44 | 1,993.17 | 1,993.77 | 2,785.73 | 2,627.81 |
| | 400 | 3 | 1,355.40 | 1,375.43 | 2,968.12 | 2,972.43 | 1,415.23 | 1,319.17 |
| | | 10 | 1,390.20 | 1,430.49 | 2,923.16 | 2,930.62 | 2,244.55 | 2,048.95 |
| | | 100 | 1,509.19 | 1,570.97 | 2,441.35 | 2,437.34 | 3,951.83 | 3,761.41 |
| 100 | 200 | 3 | 1,213.02 | 1,236.55 | 3,358.97 | 3,361.01 | 124.68 | 124.42 |
| | | 10 | 1,296.55 | 1,319.25 | 3,474.56 | 3,477.12 | 98.43 | 161.79 |
| | | 100 | 1,385.49 | 1,412.18 | 3,519.67 | 3,520.48 | 107.60 | 106.71 |
| | 400 | 3 | 1,658.49 | 1,694.96 | 3,303.74 | 3,315.40 | 2,363.25 | 2,318.42 |
| | | 10 | 1,807.75 | 1,843.33 | 3,446.43 | 3,452.90 | 574.85 | 569.48 |
| | | 100 | 1,839.21 | 1,875.84 | 3,600.00 | 3,600.00 | 286.55 | 289.99 |
| Average | | | 1,192.07 | 1,217.26 | 2,698.87 | 2,705.13 | 2,091.00 | 2,037.85 |

**Table B.20**
Results for DBLF with RTree — w/o Minimal Supporting Area.

| Instances | | | ∅ ttd | | ∅ time [s] | | ∅ iterations | |
|---|---|---|---|---|---|---|---|---|
| n | m | Item types | best | avg. | best | avg. | best | avg. |
| 20 | 200 | 3 | 409.02 | 411.93 | 1,217.62 | 1,221.52 | 5,653.15 | 5,459.69 |
| | | 10 | 418.78 | 422.33 | 1,811.86 | 1,812.65 | 4,151.65 | 4,140.28 |
| | | 100 | 422.05 | 424.68 | 1,662.07 | 1,679.56 | 3,950.30 | 3,873.98 |
| | 400 | 3 | 442.26 | 448.75 | 2,600.08 | 2,592.97 | 1,500.20 | 1,493.55 |
| | | 10 | 465.88 | 473.27 | 2,633.38 | 2,638.83 | 1,075.75 | 1,139.83 |
| | | 100 | 478.64 | 484.34 | 2,565.56 | 2,570.81 | 1,312.30 | 1,230.11 |
| 60 | 200 | 3 | 1,024.66 | 1,046.45 | 1,937.71 | 1,938.96 | 3,568.85 | 3,440.33 |
| | | 10 | 1,067.60 | 1,098.87 | 1,972.29 | 1,974.16 | 2,572.03 | 2,508.18 |
| | | 100 | 1,106.67 | 1,135.34 | 2,073.62 | 2,073.06 | 2,520.93 | 2,513.09 |
| | 400 | 3 | 1,389.93 | 1,413.70 | 2,954.96 | 2,972.97 | 1,064.75 | 1,243.63 |
| | | 10 | 1,422.27 | 1,465.33 | 2,998.13 | 2,996.04 | 2,555.25 | 2,676.65 |
| | | 100 | 1,512.42 | 1,571.53 | 2,440.92 | 2,439.39 | 3,224.65 | 3,796.01 |
| 100 | 200 | 3 | 1,241.36 | 1,271.63 | 3,355.11 | 3,358.37 | 109.15 | 107.84 |
| | | 10 | 1,355.34 | 1,377.03 | 3,550.76 | 3,551.14 | 93.03 | 91.66 |
| | | 100 | 1,449.49 | 1,472.96 | 3,522.94 | 3,519.58 | 100.75 | 98.55 |
| | 400 | 3 | 1,721.49 | 1,754.70 | 3,402.28 | 3,407.15 | 1,672.33 | 1,651.29 |
| | | 10 | 1,874.06 | 1,912.97 | 3,600.00 | 3,600.00 | 310.43 | 308.31 |
| | | 100 | 1,904.12 | 1,932.25 | 3,600.00 | 3,600.00 | 161.18 | 159.53 |
| Average | | | 1,225.85 | 1,252.36 | 2,776.93 | 2,779.27 | 1,785.00 | 1,817.59 |

**Table B.21**
Results for DBLF with RTree — w/o Fragility.

| Instances | | | ∅ ttd | | ∅ time [s] | | ∅ iterations | |
|---|---|---|---|---|---|---|---|---|
| n | m | Item types | best | avg. | best | avg. | best | avg. |
| 20 | 200 | 3 | 408.95 | 411.27 | 1,212.20 | 1,218.75 | 5,699.95 | 5,795.96 |
| | | 10 | 418.33 | 421.58 | 1,808.73 | 1,809.52 | 4,195.30 | 4,171.03 |
| | | 100 | 422.74 | 425.69 | 1,605.23 | 1,612.80 | 4,070.10 | 4,202.12 |
| | 400 | 3 | 442.81 | 447.28 | 2,595.09 | 2,601.50 | 1,530.70 | 1,535.40 |
| | | 10 | 465.30 | 474.08 | 2,632.39 | 2,638.59 | 1,168.60 | 1,067.19 |
| | | 100 | 486.02 | 493.46 | 2,550.61 | 2,560.04 | 1,125.20 | 1,131.14 |
| 60 | 200 | 3 | 1,031.36 | 1,055.23 | 1,920.20 | 1,930.07 | 4,056.83 | 3,880.08 |
| | | 10 | 1,088.32 | 1,118.37 | 1,953.63 | 1,958.13 | 2,963.55 | 2,899.51 |
| | | 100 | 1,140.98 | 1,169.42 | 2,039.41 | 2,041.04 | 2,698.05 | 2,661.66 |
| | 400 | 3 | 1,385.14 | 1,412.25 | 2,958.28 | 2,963.86 | 1,356.23 | 1,523.61 |
| | | 10 | 1,446.68 | 1,505.91 | 2,938.83 | 2,927.42 | 4,452.03 | 3,824.71 |
| | | 100 | 1,568.61 | 1,640.17 | 2,424.07 | 2,424.59 | 5,741.78 | 5,304.28 |
| 100 | 200 | 3 | 1,243.45 | 1,270.47 | 3,354.67 | 3,362.08 | 131.60 | 133.46 |
| | | 10 | 1,379.64 | 1,398.32 | 3,547.80 | 3,551.19 | 96.50 | 105.78 |
| | | 100 | 1,479.96 | 1,503.80 | 3,521.26 | 3,521.37 | 167.65 | 119.22 |
| | 400 | 3 | 1,735.86 | 1,766.28 | 3,366.69 | 3,367.99 | 2,875.40 | 2,627.00 |
| | | 10 | 1,910.17 | 1,946.92 | 3,600.00 | 3,600.00 | 451.20 | 457.35 |
| | | 100 | 1,964.84 | 1,999.43 | 3,600.00 | 3,600.00 | 253.68 | 259.18 |
| Average | | | 1,246.47 | 1,274.88 | 2,761.80 | 2,764.56 | 2,275.96 | 2,183.15 |

**Table B.22**
Results for DBLF with RTree — w/o Load Capacity.

| Instances | | | ∅ ttd | | ∅ time [s] | | ∅ iterations | |
|---|---|---|---|---|---|---|---|---|
| n | m | Item types | best | avg. | best | avg. | best | avg. |
| 20 | 200 | 3 | 407.17 | 410.38 | 1,122.76 | 1,123.53 | 6,353.60 | 5,735.38 |
| | | 10 | 419.02 | 421.77 | 1,621.74 | 1,621.82 | 4,372.05 | 4,417.89 |
| | | 100 | 424.27 | 426.92 | 2,016.84 | 2,018.54 | 4,467.60 | 4,506.84 |
| | 400 | 3 | 440.77 | 446.19 | 2,369.67 | 2,375.68 | 2,166.85 | 2,580.01 |
| | | 10 | 466.56 | 474.48 | 2,632.50 | 2,639.59 | 1,942.75 | 1,990.15 |
| | | 100 | 491.67 | 501.74 | 2,558.14 | 2,567.58 | 1,895.55 | 1,987.62 |
| 60 | 200 | 3 | 1,034.36 | 1,045.17 | 2,812.16 | 2,809.04 | 3,131.78 | 3,229.33 |
| | | 10 | 1,095.67 | 1,115.99 | 2,816.16 | 2,816.85 | 2,660.38 | 2,571.07 |
| | | 100 | 1,152.61 | 1,174.51 | 2,790.89 | 2,795.01 | 2,320.85 | 2,342.47 |
| | 400 | 3 | 1,392.88 | 1,413.55 | 2,572.47 | 2,580.20 | 2,182.58 | 2,257.93 |
| | | 10 | 1,467.76 | 1,493.34 | 2,385.76 | 2,400.64 | 1,639.05 | 1,688.41 |
| | | 100 | 1,606.68 | 1,629.68 | 2,144.16 | 2,153.80 | 944.90 | 925.72 |
| 100 | 200 | 3 | 1,229.01 | 1,244.98 | 2,845.52 | 2,844.13 | 4,037.95 | 3,899.78 |
| | | 10 | 1,384.05 | 1,396.98 | 2,984.17 | 2,983.59 | 2,524.25 | 2,439.00 |
| | | 100 | 1,495.08 | 1,516.26 | 3,247.58 | 3,248.27 | 1,698.25 | 1,704.71 |
| | 400 | 3 | 1,739.13 | 1,761.02 | 3,111.58 | 3,113.11 | 3,029.63 | 2,693.61 |
| | | 10 | 1,952.74 | 1,981.10 | 3,589.12 | 3,587.19 | 603.68 | 601.40 |
| | | 100 | 2,013.95 | 2,044.17 | 3,600.00 | 3,600.00 | 319.30 | 317.77 |
| Average | | | 1,259.24 | 1,277.17 | 2,737.36 | 2,740.35 | 2,379.45 | 2,352.01 |

**Table B.23**
Results for Zhang et al. (2017) Instances.

| Algorithm | Constraint set | ∅ ttd | | ∅ time [s] | | ∅ iterations | |
|---|---|---|---|---|---|---|---|
| | | best | avg. | best | avg. | best | avg. |
| DBLF with Points | All Constraints | 784.50 | 788.53 | 500.52 | 503.75 | 17,160.67 | 17,749.56 |
| | w/o Rotation | 790.17 | 797.09 | 299.02 | 297.49 | 17,417.59 | 17,468.40 |
| | w/o LIFO | 728.22 | 733.14 | 294.48 | 293.31 | 16,512.89 | 15,740.81 |
| | w/o Minimal S. Area | 741.43 | 746.89 | 281.62 | 282.15 | 18,127.56 | 17,035.43 |
| | w/o Fragility | 769.22 | 776.54 | 274.02 | 278.05 | 17,087.70 | 17,037.24 |
| | w/o Load Capacity | 746.34 | 750.87 | 284.17 | 281.21 | 17,271.48 | 17,488.07 |
| **Average** | | **759.98** | **765.51** | **322.30** | **322.66** | **17,262.98** | **17,086.59** |
| DBLF with Spaces | All Constraints | 778.19 | 782.43 | 476.40 | 467.67 | 17,276.37 | 17,338.45 |
| | w/o Rotation | 786.36 | 791.10 | 255.27 | 253.55 | 17,969.11 | 17,031.30 |
| | w/o LIFO | 723.02 | 728.62 | 294.41 | 284.69 | 17,459.63 | 16,342.47 |
| | w/o Minimal S. Area | 734.72 | 739.33 | 247.04 | 244.50 | 16,645.52 | 16,373.58 |
| | w/o Fragility | 763.95 | 766.22 | 268.38 | 271.03 | 17,333.59 | 16,733.05 |
| | w/o Load Capacity | 738.42 | 743.78 | 288.18 | 292.40 | 17,615.37 | 17,119.99 |
| **Average** | | **754.11** | **758.58** | **304.95** | **302.31** | **17,383.27** | **16,823.14** |
| DBLF with RTree | All Constraints | 874.95 | 874.91 | 1,962.35 | 1,947.49 | 7,668.78 | 7,772.15 |
| | w/o Rotation | 869.32 | 880.53 | 2,077.42 | 2,080.08 | 7,960.70 | 7,454.01 |
| | w/o LIFO | 785.39 | 797.30 | 2,067.01 | 2,074.40 | 7,227.78 | 6,929.02 |
| | w/o Minimal S. Area | 823.46 | 841.06 | 2,079.90 | 2,085.14 | 7,567.52 | 7,244.15 |
| | w/o Fragility | 852.32 | 857.95 | 2,066.63 | 2,076.38 | 8,095.26 | 7,945.50 |
| | w/o Load Capacity | 826.78 | 840.67 | 2,123.77 | 2,122.91 | 5,423.78 | 5,276.93 |
| **Average** | | **838.70** | **848.74** | **2,062.85** | **2,064.40** | **7,323.97** | **7,103.63** |

## References


Allen, S., Burke, E., Kendall, G., 2011. A hybrid placement strategy for the three-dimensional strip packing problem. European J. Oper. Res. 209, 219–227. http://dx.doi.org/10.1016/j.ejor.2010.09.023.

Araújo, A., Özcan, E., Atkin, J., Baumers, M., Ashcroft, I., 2019. An experimental analysis of deepest bottom-left-fill packing methods for additive manufacturing. Int. J. Prod. Res. 1–17. http://dx.doi.org/10.1080/00207543.2019.1686187.

Baker, B., Coffman, E., Rivest, R., 1980. Orthogonal packings in two dimensions. SIAM J. Comput. 9, 846–855. http://dx.doi.org/10.1137/0209064.

Clarke, G., Wright, J.W., 1964. Scheduling of vehicles from a central depot to a number of delivery points. Oper. Res. 12, 568–581. http://dx.doi.org/10.1287/opre.12.4.568, http://www.jstor.org/stable/167703.

Feng, X., Moon, I., Shin, J., 2015. Hybrid genetic algorithms for the three-dimensional multiple container packing problem. Flex. Serv. Manuf. J. 27, 451–477. http://dx.doi.org/10.1007/s10696-013-9181-8.

Gendreau, M., Iori, M., Laporte, G., Martello, S., 2006. A tabu search algorithm for a routing and container loading problem. Transp. Sci. 40, 342–350. http://dx.doi.org/10.1287/trsc.1050.0145, http://pubsonline.informs.org/doi/abs/10.1287/trsc.1050.0145.

Guttman, A., 1984. R-trees: A dynamic index structure for spatial searching. SIGMOD Rec. 14, 47–57. http://dx.doi.org/10.1145/971697.602266.

Hopper, E., Turton, B., 2001. An empirical investigation of meta-heuristic and heuristic algorithms for a 2d packing problem. European J. Oper. Res. 128, 34–57. http://dx.doi.org/10.1016/S0377-2217(99)00357-4.

Jamrus, T., Chien, C.F., 2016. Extended priority-based hybrid genetic algorithm for the less-than-container loading problem. Comput. Ind. Eng. 96, 227–236. http://dx.doi.org/10.1016/j.cie.2016.03.030, http://www.sciencedirect.com/science/article/pii/S036083521630105X.

Kang, K., Moon, I., Wang, H., 2012. A hybrid genetic algorithm with a new packing strategy for the three-dimensional bin packing problem. Appl. Math. Comput. 219, 1287–1299. http://dx.doi.org/10.1016/j.amc.2012.07.036, http://www.sciencedirect.com/science/article/pii/S0096300312007369.

Karabulut, K., İnceoğlu, M.M., 2005. A hybrid genetic algorithm for packing in 3d with deepest bottom left with fill method. In: Yakhno, T. (Ed.), Advances in Information Systems. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 441–450.

Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P., 1983. Optimization by simulated annealing. Science 220 4598, 671–680.

Koch, H., 2018. Vehicle Routing Problems with Three-Dimensional Loading Constraints and Backhauls (Ph.D. thesis). Otto-von-Guericke-Universität Magdeburg, http://dx.doi.org/10.25673/13615.

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, http://dx.doi.org/10.1007/s00291-018-0506-6.

Koch, H., Schlögell, M., Bortfeldt, A., 2019. A hybrid algorithm for the vehicle routing problem with three-dimensional loading constraints and mixed backhauls. J. Sched. http://dx.doi.org/10.1007/s10951-019-00625-7.

Krebs, C., Ehmke, J.F., 2021a. Axle weights in combined vehicle routing and container loading problems. EURO J. Transp. Logist. 10, 100043. http://dx.doi.org/10.1016/j.ejtl.2021.100043, URL: https://www.sciencedirect.com/science/article/pii/S2192437621000157.

Krebs, C., Ehmke, J.F., 2021b. Vertical stability constraints in combined vehicle routing and 3d container loading problems. In: Mes, M., Lalla-Ruiz, E., Voß, S. (Eds.), Computational Logistics. Springer International Publishing, Cham., pp. 442–455.

Krebs, C., Ehmke, J.F., Koch, H., 2021. Advanced loading constraints for 3d vehicle routing problems. http://dx.doi.org/10.1007/s00291-021-00645-w, OR Spectrum.

Ma, H.w., Zhu, W., Xu, S., 2011. Research on the algorithm for 3l-cvrp with considering the utilization rate of vehicles. In: Chen, R. (Ed.), Intelligent Computing and Information Science. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 621–629.

Mak-Hau, V., Moser, I., Aleti, A., 2018. An exact algorithm for the heterogeneous fleet vehicle routing problem with time windows and three-dimensional loading constraints. In: Sarker, R., Abbass, H.A., Dunstall, S., Kilby, P., Davis, R., Young, L. (Eds.), Data and Decision Sciences in Action. Springer International Publishing, Cham, pp. 91–101. http://dx.doi.org/10.1007/978-3-319-55914-8.

Moon, I., Nguyen, L., 2013. Container packing problem with balance constraints. OR Spectrum 36, http://dx.doi.org/10.1007/s00291-013-0356-1.

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 Comput. Sci. 51, 2257–2266. http://dx.doi.org/10.1016/j.procs.2015.05.382, http://www.sciencedirect.com/science/article/pii/S1877050915011904. international Conference On Computational Science, ICCS 2015.

Ropke, S., Pisinger, D., 2006. A unified heuristic for a large class of vehicle routing problems with backhauls. Eur. J. Oper. Res. 171, 750–775. http://dx.doi.org/10.1016/j.ejor.2004.09.004, URL: http://www.sciencedirect.com/science/article/pii/S0377221704005831.

Silva, E.F., Toffolo, T.A.M., Wauters, T., 2019. Exact methods for three-dimensional cutting and packing: a comparative study concern- ing single container problems. Comput. Oper. Res. 109, 12–27. http://dx.doi.org/10.1016/j.cor.2019.04.020, URL: https://www.sciencedirect.com/science/article/pii/S0305054819301030.

Solomon, M.M., 1987. Algorithms for the vehicle routing and scheduling problems with time window constraints. Oper. Res. 35, 254–265. http://dx.doi.org/10.1287/opre.35.2.254.

Wauters, T., Verstichel, J., Vanden Berghe, G., 2013. An effective shaking procedure for 2d and 3d strip packing problems. Comput. Oper. Res. 40, 2662–2669. http://dx.doi.org/10.1016/j.cor.2013.05.017.

Wu, B., Lin, J.g., Dong, M., 2013. Artificial bee colony algorithm for three-dimensional loading capacitated vehicle routing problem. In: Qi, E., Shen, J., Dou, R. (Eds.), Proceedings of 20th International Conference on Industrial Engineering and Engineering Management. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 815–825.

Zhang, D., Cai, S., Ye, F., Si, Y.W., Nguyen, T.T., 2017. A hybrid algorithm for a vehicle routing problem with realistic constraints. Inf. Sci. 394-395, 167–182. http://dx.doi.org/10.1016/j.ins.2017.02.028.

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. Comput. Oper. Res. 39, 2178–2195. http://dx.doi.org/10.1016/j.cor.2011.11.001.